Developing Performant Neural Network Architectures for Processing Medical Data

Thesis for the Degree of Doctor of Philosophy (PhD)

by Gergő Bogacsovics

Supervisor: Dr. András Hajdu

University of Debrecen

Doctoral Council of Natural Sciences and Information Technology

Doctoral School of Informatics

Debrecen, 2024

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a PhD Degree in Informatics at the University of Debrecen.

The results published in the thesis are not reported in any other PhD theses.

Debrecen, …… 2024.

.....
signature of the candidate

Hereby I confirm that Gergő Bogacsovics candidate conducted his studies with my supervision within the Data science and visualization Doctoral Programme of the Doctoral School of Informatics between 2020 and 2024. The independent studies and research work of the candidate significantly contributed to the results published in the thesis.
I also declare that the results published in the thesis are not reported in any other theses.
I support the acceptance of the thesis.

Debrecen, …… 2024. *.....*
signature of the supervisor

Developing Performant Neural Network Architectures for Processing Medical Data

Dissertation submitted in partial fulfilment of the requirements
for the doctoral (PhD) degree in Informatics

Written by Gergő Bogacsovics certified Computer Scientist

Prepared in the framework of
the Doctoral School of Informatics of University of Debrecen
(Data science and visualization programme)

Dissertation advisor: Dr. András Hajdu

The official opponents of the dissertation:

Dr. ....

Dr. ....

Dr. ....

The evaluation committee:

chairperson: Dr. ....

members: Dr. ....

Dr. ....

Dr. ....

Dr. ....

The date of the dissertation defence: ..... 2024.

Contents

1 Introduction

The fields of clinical research, healthcare, and medicine have experienced a major change in the last years. Devices capable of performing high-quality image acquisition became wide-spread across the world, with more and more hospitals investing in modern imaging equipments in an effort to increase the standard of clinical workflows. This gave birth to the widespread application of different imaging modalities, like positron emission tomography (PET), X-rays, computed tomography (CT), and magnetic resonance imaging (MRI). These imaging techniques, however, result in huge amounts of image data being generated for each patient and medical examination. This has led to never-before-seen amounts of data being generated each passing day, which clinicians struggle to handle, resulting in slower clinical trials, longer patient queues, and a significantly overworked hospital staff.

To counter these issues, several revolutionary methods have been proposed in the last decades, which attempted to automatize the most time-consuming parts of the clinicians' workflow. The earlier versions of these systems applied traditional image processing methods, such as thresholding [1, 2, 3] and bandpass filters [4, 5] among other techniques. Thresholding-based approaches relied on calculating a given threshold value T to generate a binary mask of the input image. They achieved this by organizing the pixels of the image into the desired groups (e.g., a pixel belonging to a tumor versus a pixel containing other information) by checking if the intensity of the given pixel was higher than the threshold T. Methods using a bandpass filter followed a rather similar approach, where only the desired frequencies were kept from the input image, while others were filtered out. Traditional systems relying on similar techniques often combined multiple of these approaches and hence usually required cumbersome and time-consuming fine-tuning from the clinical experts, e.g., to configure the parameters for the given system to work optimally.

Ultimately, the rise of novel artificial intelligence-based techniques, and the integration of neural networks in particular, made it possible to overcome these issues. These solutions required little to no fine-tuning or interaction from the clinicians due to the considerably lower number of input parameters and the more robust operation with better generalization capabilities. Most of these improvements could be attributed to the appearance of a new type of neural network architecture, called

the convolutional neural network (CNN). As the name implies, this family of algorithms used a technique called convolution, which extracts important local features from a pre-defined patch of the image. The size of this area depends on a tunable hyperparameter, called a kernel, which is used to perform matrix multiplication between said kernel and the patch of the image. This operation gets executed for each patch of the image for a given kernel, which a modern CNN can have several of. The role of a given kernel is to specialize for the detection of a given feature present in the image or in the outputs of the previous layer. Therefore, a kernel (in the first layer of a CNN) can essentially be thought of as the equivalent of a given traditional image processing method, such as edge detection. Modern CNNs also have multiple layers, making it possible for the model to learn more complex patterns as well. Since the kernels get their values assigned during the training process, there is no longer a need for clinicians to carefully fine-tune them depending on the given clinical problem. Instead, these architectures can learn to extract the most meaningful and most important features from the perspective of the given problem, drastically reducing the complexity of developing Computer Aided Diagnosis (CAD) systems. As a result, highly advanced CAD systems, capable of automated and reliable clinical diagnosis, have been proposed which helped in tackling some of the most common clinical tasks. These tasks included the detection of multiple types of cancer [6, 7, 8, 9], various disorders, such as attention deficit hyperactivity disorder (ADHD) [10, 11] or autism spectrum disorder (ASD) [12], and diseases, such as Alzheimer's [13, 14] or chronic kidney disease (CKD) [15, 16].

A subset of the experiments focused on further increasing the overall reliability and accuracy of the developed systems by merging multiple distinct models into one single model, called an ensemble. Ensemble models have the theoretical advantage of having better generalization capabilities when compared to regular machine- or deep learning models. This is due to the fact that when building ensembles, multiple distinct models are considered, that, in theory, should work and operate in vastly different and diverse ways. Given that all the models inside the ensemble are sufficiently good, meaning that they can solve the task to a sufficiently good degree, the ensemble should have a better generalization capability than any of its member models. This is due to the fact that by incorporating more and more models that operate in different ways, and that all perform sufficiently well on the given

task, the chances of all of these good-performing models misclassifying a given input or making a bad prediction at the same time should drastically decrease. Moreover, combining the outputs of multiple different models should, in theory, also result in a better ability to detect outliers, i.e., inputs that are vastly different than those in the training dataset used to train the models. This is because with more models the probability of any of these models detecting such abnormalities also increases.

Traditional ensemble models are constructed using already trained models. That is, each member model needs to be trained on the given dataset before the ensemble is constructed. Then, the individual models can be merged into one singular ensemble model by using a wide range of possible approaches. One of the most popular techniques in the current literature for combining the outputs of the individual classifier models is called majority voting. This approach simply takes the outputs of each model and outputs the one with the most votes, which is the prediction that most of the models predicted. In spite of its simple nature, this technique has been used with great success in the field of clinical diagnosis [17, 18, 19]. Another approach for building a traditional ensemble model is to combine the probabilities predicted by each model by calculating their average or weighted average for each class [20, 21, 22]. Then, the final output of the ensemble will be the class with the highest average probability.

The main reason behind the popularity of these traditional ensemble methods is that they generally result in an increased performance due to the statistical foundation, while still being fairly easy to implement. However, the biggest problem with these approaches is also rooted in their simplicity: when using these techniques, the overall quality of the member models constructing the ensemble is usually not checked. That is, it is generally not examined how different the models are. This is a serious short-coming, as it has been shown multiple times [23, 24] that diversity, in other words, if the models constructing the ensemble are different enough, is a vital metric to consider in order to build accurate and performant ensemble models. Moreover, using models that are similar, or in worse cases, almost identical to each other will also lead to sub-par results. The reason behind this is that the generalization capability of the ensemble will also be lacking due to the similarity between the member models, as similar models will also produce similar predictions.

In our research, we focused on developing novel solutions to increase the performance of neural network models by using the previously discussed traditional techniques and proposing new methods for constructing and training ensemble architectures. The objective of this dissertation is to develop novel neural network-based algorithms to further facilitate the implementation and integration of solutions using artificial intelligence into the clinical workflow. To this end, multiple neural network-based solutions and techniques are presented that make it possible to solve the given clinical task with sufficient accuracy, and reliability.

The approaches detailed in this dissertation can be divided into two groups: i) techniques that combine traditional methods with neural networks, and ii) techniques that combine various neural network models in the form of an ensemble. The first group of techniques discussed in the dissertation deal with the effective combination of traditional and neural network-based solutions, showing that combining these two paradigms can lead to a more accurate, and also more robust model. This dissertation presents two different architectures to solve this problem. The first one, used for the efficient and accurate prediction of COVID-19 and influenza cases in various countries, shows that it is possible to approximate a given traditional model with a neural network and then use that as a starting point for further training the network on real data. It is shown that this results in an overall better model that can be used even when the training data is small. The network can learn the most important aspects of the given phenomenon using the traditional model, which can generate almost unlimited training data, and then we can use the real, small dataset to fine-tune the network. The other method proposed in this dissertation examines the possibility of using hand-crafted features, extracted by traditional methods, in conjunction with neural networks. We show that we can encapsulate the hand-crafted features with those extracted by the network to reach even better performance.

The second group of methods deal with the practical and efficient combination of multiple models into one single ensemble model. The first proposed technique shows a way of re-using a fully convolutional neural network (FCN) [25] as the base of combining the outputs of several other FCN models previously trained on the given dataset in an effort to improve performance. We show that this way, the ensemble can take into account both the original input image and the

outputs of the pre-trained models as well, and process every related information jointly, from layer to layer. The other method presents two novel frameworks for training performant ensembles, that directly include the diversity of the member models as a metric during training. We show that these frameworks have several advantages as compared to other traditional ensemble architectures, and achieve great results when used for classifying different types of brain tumors on MRI images.

The main contributions of this dissertation are therefore as follows:

- a novel two-step architecture that combines traditional theoretical models with neural networks;

  we propose a two-step architecture for training reliable time-series models that can accurately predict the total number of people affected by influenza and COVID-19. The two-step architecture first fits a traditional epidemic model, such as an SIR model [26, 27], to the data and trains a neural network model to mimic the operation of said traditional model. Then, as the second step, the architecture trains the neural network model further on additional data to further increase the accuracy of the model. For influenza, we used data published by the World Health Organization (WHO), while for COVID-19, we used data available on the Humanitarian Data Exchange (HDX) platform.

  Related publication: [28].

- multiple novel approaches for the effective combination of hand-crafted features with features extracted by convolutional neural networks;

  we present multiple novel methods for combining the traditional, hand-crafted features with those extracted by neural networks. We show that there are multiple ways to accomplish this: by using the outputs of the linear layer before the output layer, the last convolutional layer, and building a mini-ensemble model, which combines the outputs of a simple model receiving the hand-crafted features with that of the neural network. We apply these solutions to accurately and reliably classify some of the most common diabetic eye disases, diabetic macular edema and diabetic retinopathy.

  Related publication: [29].

- a hybrid ensemble framework for accurate cell segmentation;

  we present a unique ensemble framework that processes the generated probability masks of several pre-trained FCNs and the input image at the same time, from convolutional layer to convolutional layer. We introduce several variants of the framework and use it on a dataset of manually annotated cell images, organized and created at the University of Debrecen. We then show that the resulting models surpass the performance of the original FCNs and other state-of-the-art approaches as well for the task of cell segmentation.

  Related publications: [30, 31].

- a novel ensemble architecture for the accurate classification of brain tumors;

  we propose a novel way of measuring diversity inside an ensemble during the training process and use it as a form of regularization to build robust and performant ensemble models to accurately classify some of the most common types of brain tumors. We show that it is possible to use the feature vector extracted by the last convolutional layer and directly measure the similarity between the vectors produced by each member model inside the ensemble to facilitate the training of diverse ensemble models. We propose multiple variants of the architecture, one that uses the cosine similarity function, and another that uses the histogram loss [32].

  Related publications: [33, 34].

In this dissertation, we will declare claims based on simple theoretical considerations and our observed experimental results. We will denote such claims as **Claim** in the text of the dissertation. Our motivations behind introducing this notation are to make identifying our main contributions easier and to improve the structuring of the dissertation. We will also use claims instead of theorems in cases when the justification or reasoning behind the given statement is simple and therefore does not warrant any in-depth proof. In the event that a short explanation would be beneficial, we will use the term *Reasoning* instead of *Proof* to give a short justification for the given **Claim**.

The rest of the dissertation is organized in the following structure. In section 2, the basic concepts and notations used in the dissertation

are introduced and the most crucial elements of the theoretical background are covered. In section 3, we present our two-step architecture that combines traditional theoretical models and neural networks, and showcase its performance for the prediction of influenza and COVID-19 cases. In section 4, we focus on the usage of hand-crafted features, which have been mostly neglected since the rise of deep learning-based solutions. Then, we introduce multiple architectures for the effective combination of hand-crafted features with those extracted by convolutional neural networks. In section 5, we introduce the custom dataset which was created at the University of Debrecen, and give an overview of its characteristics, including the methodology used during the digitization and annotation steps. Then, we present our ensemble framework which combines the outputs of multiple pre-trained FCN networks [25] with the given input image using one single architecture by utilizing an FCN as its backbone. In section 6, we investigate possible ways of building more effective ensembles by measuring the diversity of the member models. We present multiple frameworks to achieve this and contrast them with traditional ensemble methods such as majority voting and weighted averaging. Finally, in sections 7 and 8, we give a summary of the dissertation.

2 Background

This chapter aims to precisely define the basic concepts and notations used in this dissertation, as well as to provide the necessary theoretical background that will be the base of subsequent chapters. For the latter, first an overview of the common definitions and optimization techniques is given. Then, we extend the traditional deep learning training procedure to the area of ensemble models by introducing the most commonly used ensemble building techniques and detailing how they operate. We also summarize the most important key aspects of traditional methods that are relevant to the dissertation.

2.1 Basic concepts and notations

Machine- and deep learning-based algorithms need a dataset, containing examples of the problem at hand, to learn from. The methods covered in this dissertation all belong to the supervised algorithms, therefore, only datasets containing both the inputs and outputs will be considered. Throughout the dissertation, we will denote such datasets, containing the inputs and outputs as $\mathcal{D}$. For the effective training and evaluation of machine- and deep-learning-based approaches, it is also preferred to split the original dataset $\mathcal{D}$ into training, validation, and test parts, that will be used to train, fine-tune, and evaluate the final performance of the model. In this dissertation, we will refer to these datasets as $\mathcal{D}_{train}$, $\mathcal{D}_{val}$, and $\mathcal{D}_{test}$, respectively.

We will refer to the i-th elements inside a dataset as $x^{(i)}$ for the inputs, and as $y^{(i)}$ for the outputs for $i = 1, \ldots, m$ where $m \in \mathbb{N}$ is the number of training samples. Therefore, we define a dataset of m examples as a set of the input-output pairs, denoted as $\mathcal{D} := \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$. We will denote the dimension, in other words, the length of the input $x^{(i)}$ as $dim(x^{(i)})$ and that of the output $y^{(i)}$ as $dim(y^{(i)})$, respectively. For regression models, in other words, for models predicting a continuous value, unless otherwise specified, we will define each $y^{(i)} \in \mathbb{R}$ as a real number, while for classification models, i.e., models that predict a discrete output, we will define each $y^{(i)} \in \{1, 2, ..., n_c\}$ as discrete numbers, where n_c denotes the total number of classes.

For cases when a concatenation of any two vectors (e.g., several inputs or outputs) is needed, we will use $< a, b >$ to denote the

usage of said operation for any two vectors $a = [a_1, ..., a_{d_a}]^\top$, and $b = [b_1, ..., b_{d_b}]^\top$, where $d_a := dim(a)$ and $d_b := dim(b)$. The resulting vector will be defined as $< a, b >:= [a_1, a_2, ..., a_{d_a}, b_1, b_2, ..., b_{d_b}]^\top$. Similarly, we define the depth-wise concatenation of two matrices $A \in \mathbb{R}^{m \times n \times k}$ and $B \in \mathbb{R}^{m \times n \times l}$ using the same operator "<>" as $< A, B > \in \mathbb{R}^{m \times n \times k+l}$ where $< A, B >_{i,j,q} := A_{i,j,q}$ for each $q \leq k$ and $< A, B >_{i,j,q} := B_{i,j,q-k}$ for each $q > k$. We also define the depth-wise concatenation operator for multiple matrices. Let $A_1, A_2, ..., A_N$ denote some arbitrary matrices. The depth-wise concatenation of these matrices is defined as $< A_1, A_2, ..., A_N > :=< ... << A_1, A_2 >, A_3 > ..., A_N >$.

For the models, we will refer to any learnable parameters, in other words weights, of any – traditional, machine-, or deep learning – model as $\theta \in \mathbb{R}^{d_\theta}$, and will use M_θ to refer to the model itself. In this dissertation, we will incorporate the bias in the definition of the weights θ. In other words, throughout this dissertation, the equation $d_\theta = dim(\theta) = dim(x) + 1$ will hold. However, for the sake of brevity, we will still denote multiplying the input $x^{(i)}$ by the weights θ as $\theta^\top x^{(i)}$ instead of writing out the full form $\theta^\top([1, x_1^{(i)}, x_2^{(i)}, \ldots, x_{d_x}^{(i)}]^\top)$, where $d_x = dim(x)$. In some cases, where θ is not specifically relevant from the perspective of the given formula, we may also use the shorthand notation M to refer to M_θ, to keep the formulae shorter and more concise. Furthermore, we use σ to denote any activation function. Thus, a forward pass of a model M with weights θ and activation function σ for any given input $x^{(i)}$ will be denoted as $M_\theta(x^{(i)}) := \sigma(\theta^\top x^{(i)})$. Lastly, for a particular neural network model M with k hidden layers and their corresponding weights $\theta_1, \theta_2, ..., \theta_k$ and activation functions $\sigma_1, \sigma_2, ..., \sigma_k$, we may also use the shorthand notations $M_\theta(x^{(i)})$ to make the given formula more compact and concise.

The goal of supervised machine- and deep learning models is to minimize a cost function $J(\theta)$ that measures the performance of the given model M_θ: a better set of weights θ will have a lower $J(\theta)$, while a worse set of parameters will have a higher $J(\theta)$. The cost function is calculated for a batch of samples at once and is usually defined as the mean of the loss values calculated for each element of the given batch, computed between the ground truth labels and the predictions of the given model. For the latter, we will denote the predictions of a given model M for a particular input $x^{(i)}$ as $\widehat{y}_i := M(x^{(i)})$. In this dissertation, we will refer to the loss function, calculated for each $y^{(i)}$ and $\widehat{y}_i$ during the training process of a particular model M_θ as $L_i(\theta)$.

The goal of the optimization procedure will be to minimize the cost function

$$J(\theta) := \frac{1}{m} \sum_{i=1}^{m} L_i(\theta) \tag{1}$$

on the given dataset $\mathcal{D}$ and for the given model M_θ, usually achieved by calculating the gradient $\nabla_\theta J(\theta) := [\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, ..., \frac{\partial J}{\partial \theta_{d_\theta}}]^\top$.

This dissertation focuses on two main areas of machine- and deep learning: i) time series prediction, and ii) image processing. For time series prediction-related tasks, we will expand upon the previously introduced notation system to handle both multi-dimensional inputs and outputs. We will use the terms dimension and time window interchangeably to refer to the number of data points in the input data (t_{in}) and in the outputs (t_{out}), respectively. Consequently, for time series-related algorithms, we define each input $x^{(i)}$ and output $y^{(i)}$ as $x^{(i)} \in \mathbb{R}^{t_{in}}$ and $y^{(i)} \in \mathbb{R}^{t_{out}}$.

When dealing with image processing-related problems, each element $x^{(i)}$ will be an image in our dataset $\mathcal{D}$. In this case, we define the input image $x^{(i)} \in \mathbb{R}^{h \times w \times d}$ as an element in the $\mathbb{R}^{h \times w \times d}$ vector space, where h denotes the height, w the width, and d the depth (i.e., number of channels) of the images originating from this space. We will use the notation $x^{(i)}_{px,py}$ to refer to a given pixel at the (px, py) coordinates, where $1 \leq px \leq w$ and $1 \leq py \leq h$. Although there are many different ways to encode and store digital images depending on the device, procedure, format, and imaging modality (e.g., PET, MRI, etc.) that are used to record and store the images, in this dissertation, we will focus on images where the pixel values are in the $0, \ldots, 255$ range. We will also use this same notation when the pixel values of the input image have been standardised or normalized to the $[0, 1]$ interval. We will also refer to $x^{(i)} \in \mathbb{R}^{h \times w \times 1}$ images as grayscale, and use the phrase colored (RGB) image when referring to any image $x^{(i)} \in \mathbb{R}^{h \times w \times 3}$. For segmentation problems, where the outputs are also images, we will define the outputs $y^{(i)} \in \mathbb{R}^{h \times w \times n_c}$ as images as well, where n_c represents the number of possible classes for each pixel of the output image $y^{(i)}$.

Furthermore, in this dissertation, we will refer to any kind of ensemble model as Ens. Accordingly, let M_1, M_2, ..., M_n ($n \in \{2, 3, 4, ...\}$) be any traditional, machine-, or deep learning models following our previously introduced notation system. An ensemble constructed of these models will be defined as $Ens := \{M_1, M_2, ..., M_n\}$.

2.2 Theoretical background

The goal of any supervised machine-, and deep learning model M_θ is to minimize a given cost function $J(\theta)$ by minimizing the loss function $L_i(\theta)$ for each sample $(x^{(i)}, y^{(i)})$. To achieve this, a robust optimizer, such as Stochastic Gradient Descent (SGD) [35], RMSProp [36], or Adam [37] should be used. The common objective of these optimizers is to reduce the value of $J(\theta)$ incrementally, usually from batch to batch in practice. This is because, although considering all training samples at once may be more beneficial from a convergence standpoint, keeping all of the samples in the memory is usually unfeasible and impractical due to the sheer amount of memory required by each $x^{(i)}$. Due to this, the value of $J(\theta)$ and the gradient $\nabla_\theta J(\theta)$ is usually approximated by using only a smaller number of training samples, called a mini-batch to update the weights θ for each batch of samples. Moreover, the batches are chosen from the dataset $\mathcal{D}$ in a randomly shuffled manner to reduce any intrinsic dependencies or correlations between the samples inside each batch. This process of gradually lowering the total cost is then repeated for a number of steps, called epochs. An overview of an SGD optimizer for a simple machine learning model M_θ with weights θ can be seen in Algorithm 1.

Algorithm 1 Stochastic gradient descent

Inputs:
$\mathcal{D} \leftarrow$ Dataset
$m \leftarrow$ Number of elements inside $\mathcal{D}$
$N \leftarrow$ Number of epochs
$\alpha \leftarrow$ Learning rate
$B \leftarrow$ Batch size

Output:
The trained model M_θ

Initialize the weights θ randomly
for *epoch* $\leftarrow 1$ to N **do**
    batches $\leftarrow$ Get non-overlapping, randomly shuffled batches from $\mathcal{D}$
    for b in *batches* **do**
        for i in b **do**
            $\widehat{y}^{(i)} \leftarrow M_\theta(x^{(i)})$
        $J(\theta) \leftarrow \frac{1}{B} \sum_{i \in b} L_i(\theta)$
        $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$

The following is an in-depth overview of Algorithm 1:

- **Input** – $\mathcal{D}$: the dataset of training inputs $x^{(i)}$ and their corresponding outputs $y^{(i)}$,
- **Input** – m: the total number of input-output pairs inside the dataset $\mathcal{D}$,
- **Input** – N: the total number of epochs during the optimization process,
- **Input** – B: the number of training input and output pairs considered for one single update of the weights of the model,
- **Input** – M_θ: the machine learning model, parameterized by the randomly initialized weights θ,
- **Output** – M_θ: the machine learning model with the learned parameters θ after the optimization process has ended,
- *Line 1*: the weights of the model M_θ are initialized as follows: $\theta_i := rand(\mathbb{R})$, where the $rand(\mathcal{A})$ function returns a random element from the set $\mathcal{A}$,
- *Line 7*: calculate the cost function by taking the mean of the loss $L_i(\theta)$, calculated for every output $y^{(i)}$ and their corresponding predictions $\widehat{y}^{(i)}$ in the given batch,
- *Line 8*: update the weights θ by taking a step along the steepest decrease in the weight space $\mathbb{R}^{d_\theta}$, by calculating the negative gradient $-\nabla_\theta J(\theta)$.

When constructing traditional ensembles with n arbitrary models, each model M_j $(j \in 1, 2, ..., n)$ is trained individually with no interaction between the different models. After each model M_j has been trained, they are constructed into an $Ens := \{M_1, M_2, ..., M_n\}$ ensemble. For inference, i.e., for using the ensemble for performing predictions on real or test data, the outputs of the different models are combined and aggregated in a fixed and pre-defined manner. Some of the most common techniques for performing this aggregation are: i) majority voting, and ii) weighted averaging. In this dissertation, we will use Ens_M to denote a majority voting ensemble, and Ens_W to

denote weighted averaging. For majority voting, the outputs of each model M_j are taken, and the class label with the highest count (i.e., the mode of the outputs) is considered as the final prediction of the ensemble:

$$Ens_M(x^{(i)}) := majority(M_1(x^{(i)}), M_2(x^{(i)}), ..., M_n(x^{(i)})), \quad (2)$$

where $majority()$ returns the class label with the most occurrences, if any, otherwise, if the outputs do not differ, a default value is returned. In this dissertation, we will use the default value of zero, signaling that the ensemble could not produce any outputs if all of the members predicted different classes. When two or more classes receive the same number of votes, one of them is selected randomly. In the case of weighted averaging, a coefficient $\beta_j \in [0, 1]$ is computed for each model M_j where $\sum_{j=1}^{n} \beta_j = 1$. The coefficients are computed by first evaluating each model according to a chosen metric, then assigning higher β_j values to models achieving better results from the perspective of the given metric, and assigning lower coefficients to worse models. Then, the final output of the ensemble is calculated as the weighted average of the output of each model, formalized as:

$$Ens_W(x^{(i)}) := \beta_1 M_1(x^{(i)}) + \beta_2 M_2(x^{(i)}) + ... + \beta_n M_n(x^{(i)}). \quad (3)$$

For segmentation tasks, i.e., when both the inputs and the outputs are images, we apply formulae 2 and 3 to each pixel in the generated segmentation masks, defined as

$$Ens_M(x^{(i)})_{px,py} := majority(M_1(x^{(i)})_{px,py}, M_2(x^{(i)})_{px,py}, ..., M_n(x^{(i)})_{px,py}) \quad (4)$$

and

$$Ens_W(x^{(i)})_{px,py} := \beta_1 M_1(x^{(i)})_{px,py} + \beta_2 M_2(x^{(i)})_{px,py} + ... + \beta_n M_n(x^{(i)})_{px,py} \quad (5)$$

for Ens_M and Ens_W, respectively.

3 Designing a two-step architecture for the accurate prediction of diseases

3.1 Introduction

In this chapter, we outline a simple architectural solution regarding a two-step framework for training accurate and data-efficient time series models. Although we use the proposed method for the prediction of influenza and COVID-19 cases, the framework is applicable to any other time series-related problems as well due to its general formulation and simple-to-implement nature. The two-step architecture first trains a neural network to approximate the operation of a given theoretical model. Then, as the second step, the model is trained on real data to further improve its performance.

This architecture aims to combine the biggest advantages of traditional and deep learning algorithms by showing that theoretical models can be applied during the training of neural network models as a pre-training step, resulting in a more accurate model. We will show that if we train a neural network first on a theoretical model, then train it further on real data, it is possible to build solutions that outperform not only the original theoretical model, but also a neural network trained only on real data. We conclude that this is because this way the neural network not only has more time to learn the intrinsic properties of the given problem, but the initial training phase (approximating the theoretical model) is mathematically well-defined and the data points are not noisy, unlike the real data. This way, the neural network will have a solid set of weights, that are roughly equivalent to a theoretical model, before being trained on real data. This makes the second part of the training much smoother, easier, and more effective.

The structure of this chapter is as follows. In section 3.2, we explain our motivations behind building the proposed architecture, detailing the advantages and disadvantages of the traditional and deep learning algorithms and explain why combining the two approaches may be a good choice in certain circumstances. Then, in section 3.3, we give an overview of the datasets that were used during our research. In section 3.4, we introduce our proposed framework and in section 3.5 we detail our experimental setup and findings. Finally, in section 3.6, we provide our conclusions and list our key contributions.

The universal two-step architecture introduced in this chapter and the corresponding experimental results were published in [28].

3.2 Motivation

Researchers often use theoretical models, which provide a relatively simple, yet concise and effective way of modelling various phenomena. However, it is a well-known fact that the more complex the model, the more complex the mathematical description is. For this reason, theoretical models generally avoid large complexity and aim for the simplest possible definition, which although makes the model mathematically more manageable, in practice it also often leads to sub-optimal performance. This is mainly because even after this simplification step, theoretical models are usually still too rigid due to their sophisticated nature and therefore cannot really deal with sudden changes in the environment. Furthermore, the data collected during the observations usually contain confounding factors, for which a simple theoretical model cannot be prepared. The application of artificial intelligence provides a good opportunity to develop complex models that can combine the basic capabilities of the theoretical models with the ability to learn more complex relationships. It has been shown [38, 39] that with neural networks, we can build such models that can approximate mathematical functions. Trained artificial neural networks are thus able to behave like theoretical models, while still retaining their overall flexibility. This, in turn, guarantees an overall better performance in a complex real-world environment.

The aim of our study was to show our notion that we can create a framework that is able to combine the main benefits of both the theoretical and deep learning methods, resulting in a solution that is i) applicable even if only a small amount of training data is available, ii) is mathematically founded, and iii) achieves good performance. Our solution relies on using neural networks, which are able to approximate a given theoretical model [38, 39], and then further improve the model with the help of real data to suit the real world and its various aspects better. In order to validate the functionality of the architecture we developed, we have selected a simple theoretical model, namely the Kermack-McKendrick model [26] as the base of our research. This is an SIR [27] model, which is a relatively simple compartmental epidemic model, based on differential equations that can be used well for

infections that spread very similar to influenza or COVID-19.

SIR is a general virus spread model that can be interpreted easily. The model can be described with an ordinary differential equation and has only a few parameters. Furthermore, in terms of behaviour it is a non-linear system. It is primarily recommended to be used for viruses where infected individuals cannot develop long-lasting immunity after recovery. This theoretical approach with regard to the spread of viruses was first described by William Ogilvy Kermack and Anderson Gray McKendrick and became generally known as the Kermack – McKendrick theory [26]. We used this model in our research because currently, for both influenza and COVID-19, the scientific consensus is that, based on the behavioral characteristics of the virus, individuals cannot get sustained immunity after recovering. A classic SIR model considers the following parameters:

- t – a given moment in time,
- N – total population,
- $S(t)$ – number of susceptible individuals at time step t,
- $I(t)$ – number of infectious individuals at time step t,
- $R(t)$ – number of recovered individuals at time step t,
- β – potential exposure rate per capita, i.e., how many additional individuals a particular infected can infect at a given point in time,
- γ – rate of recovery, which is practically the recovery/death rate and $1/\gamma$ is the infectious period.

Definition 3.1. *According to the Kermack-McKendrick model [26], given the functions* $S\colon \mathbb{N}_0 \to \mathbb{R}$, $I\colon \mathbb{N}_0 \to \mathbb{R}$, $R\colon \mathbb{N}_0 \to \mathbb{R}$ *defined for each time step* $t \in \mathbb{N}_0$, *and parameters* $\beta \in \mathbb{R}$ *and* $\gamma \in \mathbb{R}$, *the classic SIR model is considered as a system determined by the following differential equations:*

$$
\begin{aligned}
\frac{dS}{dt} &= -\frac{\beta IS}{N}, \\
\frac{dI}{dt} &= \frac{\beta IS}{N} - \gamma I, \\
\frac{dR}{dt} &= \gamma I, \\
\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} &= 0.
\end{aligned} \tag{6}
$$

By definition, for any given problem, the total number of susceptible (S), infectious (I), and recovered individuals (R) at any given time step $t \in \mathbb{N}_0$ constitutes the total population N. In other words, the equation

$$S(t) + I(t) + R(t) = N \tag{7}$$

holds true for each time step t. Furthermore, in the case of a SIR model, the number of recovered individuals (R) at the first moment in time is related to the potential exposure rate β and the rate of recovery γ, resulting in the following formula:

$$R(0) = \frac{\beta}{\gamma}. \tag{8}$$

The main disadvantage of using the SIR model, however, is that it relies too heavily on its parameters, with slight changes in them leading to drastic overall changes of the S, I, and R curves. Moreover, the simplicity of the SIR model distorts its accuracy in many cases as the underlying theoretical model is too rigid. By using the SIR model, we have shown that the two-step framework described in the coming sections can be a valid approach for modeling the spread of diseases such as influenza or COVID-19.

3.3 Datasets

The datasets used during our research were obtained from the Flunet database [40] published by the World Health Organization (WHO), and the Humanitarian Data Exchange (HDX) platform [41] for influenza and COVID-19, respectively. All population-related data was obtained from the World Bank [42]. For influenza, the dataset we used

contained the weekly number of newly infected people for any given time step. For our research, we used the influenza data of Germany, Hungary and Romania for the influenza season 2019 (starting from the winter of 2018 and ending in the spring of 2019). We chose these particular countries because each of them had data roughly resembling the bell shape of an SIR curve, making them more suitable for our experiment. In the case of COVID-19, the dataset that we used contained the number of newly infected people for any given time step. The available data was aggregated at a daily level, meaning that each data point in the dataset corresponded to the number of COVID-19 cases on a given day in a particular country. During our research, we considered the data available for a wide range of countries, which were the following: Austria, Croatia, Germany, Hungary, Japan, Romania, Slovakia, Slovenia, and Switzerland.

3.4 Two-step architecture

For approximating the spread of infectious diseases, we apply a two-step architecture. We split the original training dataset $\mathcal{D}_{train}$ into two subsets $\mathcal{D}_{train,1}$ and $\mathcal{D}_{train,2}$. Let M_{SIR} denote the theoretical model predicting the number of infectious individuals for any given time step. In other words, let us define it as $M_{SIR}(t) := I(t)$ for each time step t. We use $\mathcal{D}_{train,1}$ for optimizing the parameters of the theoretical model M_{SIR}, and use $\mathcal{D}_{train,2}$ for fine-tuning our neural network M_θ. We first fit the neural network M_θ to the infected curve of the SIR model, in other words, the outputs of the model M_{SIR}. For this, we construct a synthesized dataset $\mathcal{D}_{Synth}$ by applying the theoretical model M_{SIR} for a range of synthesized inputs $\widehat{x}_i$, where each $\widehat{x}_i$ is a randomly sampled point from the possible range of input values.

Definition 3.2. *Let $\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_m$ denote some arbitrary synthesized (e.g., randomly sampled) inputs and M_{SIR} the theoretical model. The synthesized dataset $\mathcal{D}_{Synth}$ is defined as*

$$\mathcal{D}_{Synth} := \{(\widehat{x}_1, M_{SIR}(\widehat{x}_1), (\widehat{x}_2, M_{SIR}(\widehat{x}_2), ..., (\widehat{x}_m, M_{SIR}(\widehat{x}_m)\}. \quad (9)$$

Claim 3.1. *Given a SIR model M_{SIR}, we can generate an infinite number of synthesized samples to construct $\mathcal{D}_{Synth}$.*

Reasoning. By definition, the model M_{SIR}: $\mathbb{N}_0 \to \mathbb{R}$ can produce an output for any given synthethised input $\widehat{x}_i \in \mathbb{N}_0$. Therefore, given

an infinite set of synthesized inputs $\{\widehat{x}_1, \widehat{x}_2, ...\}$, we can generate an infinite number of data points $\mathcal{D}_{Synth} = \{(\widehat{x}_1, M_{SIR}(\widehat{x}_1), (\widehat{x}_2, M_{SIR}(\widehat{x}_2),$ $..., (\widehat{x}_m, M_{SIR}(\widehat{x}_m)\}$. □

After training the neural network model M_θ, we measure how close its predictions to the theoretical model M_{SIR} are by calculating the mean squared error (MSE) using each $M_{SIR}(x^{(i)})$ and $M_\theta(x^{(i)})$ as

$$MSE(M_{SIR}, M_\theta) := \frac{1}{m}\sum_{i=1}^{m}(M_{SIR}(x^{(i)}) - M_\theta(x^{(i)}))^2. \tag{10}$$

Theorem 3.1. *Let m denote the number of samples in the dataset $\mathcal{D}_{Synth}$, generated by using the model M_{SIR} and synthesized inputs $\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_m$. Given that the number of samples m is sufficiently large and that the neural network M_θ has the sufficient computational capacity, we can train M_θ such that it is able to approximate the original model M_{SIR}. Moreover, with a sufficient capacity and weights θ, $MSE(M_{SIR}, M_\theta) \mapsto 0$ as $m \mapsto \infty$.*

Proof. It has been shown in [39] that neural networks with at least one hidden layer can approximate any continuous function. Therefore, given the SIR model M_{SIR}, the neural network M_θ, and the dataset $\mathcal{D}_{Synth}$ with m elements, the only thing influencing the quality of the trained model M_θ is m. If the size of the dataset $\mathcal{D}$ is too small, that means that M_θ cannot get a comprehensive description of the nature of the data. However, we have shown previously that we can generate an infinite number of synthesized samples to construct $\mathcal{D}_{Synth}$ using the SIR model M_{SIR}. Since we have a dataset of adequate size, according to [39], as long as the neural network has the sufficient capacity, it can approximate the continuous function M_{SIR}. In this case, (10) falls below a given threshold. Moreover, as m $\mapsto \infty$, $\frac{1}{m} \mapsto 0$, resulting in (10) converging to 0. □

When the neural network model possesses the ability to approximate the theoretical model with a sufficiently small MSE, the next step of the framework starts. During this step, the neural network gets access to the second part of the original training data, denoted as $\mathcal{D}_{train,2}$ and the second round of training begins. The objective of this second step is to provide more data to the model in the form of the real, observed values, so that the model can become more accurate.

During this step, the neural network model is given the chance to deviate from the theoretical model in cases when the latter would make undesirable predictions by making adjustments to its own weights. If this deviation is not too significant, the resulting model should both be able to simulate the operation of the original theoretical model to a high degree and be more accurate as well, due to learning from real data. A detailed description of the proposed two-step architecture is given in Algorithm 2.

Algorithm 2 The proposed two-step model

Input:

$\mathcal{D} \leftarrow$ Dataset
$m \leftarrow$ The number of samples to synthesize
$N_{SIR} \leftarrow$ Number of epochs to approximate the SIR model
$N_{FT} \leftarrow$ Number of epochs to fine-tune the neural network
$\alpha \leftarrow$ Learning rate used for the synthesized dataset
$\alpha_{FT} \leftarrow$ Learning rate used during the fine-tuning step
$B \leftarrow$ Batch size

Output:

The trained model M_θ

$\mathcal{D}_{train,1}, \mathcal{D}_{train,2} \leftarrow SPLIT(\mathcal{D})$
$\beta, \gamma \leftarrow CONFIGURE(\mathcal{D}_{train,1})$
$M_{SIR} \leftarrow MAKE\_SIR(\beta, \gamma)$
$\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_m \leftarrow SAMPLE(\mathbb{R}^{d_x})$
$\mathcal{D}_{Synth} \leftarrow \{(\widehat{x}_1, M_{SIR}(\widehat{x}_1), (\widehat{x}_2, M_{SIR}(\widehat{x}_2), ..., (\widehat{x}_m, M_{SIR}(\widehat{x}_m)\}$
Initialize the weights θ randomly
for *epoch* $\leftarrow 1$ to N_{SIR} **do**
    batches $\leftarrow$ Get non-overlapping, randomly shuffled batches from $\mathcal{D}_{Synth}$
    for b in *batches* **do**
        for i in b **do**
            $\widehat{y}_i \leftarrow M_\theta(x^{(i)})$
        $J(\theta) \leftarrow \frac{1}{B} \sum_{i \in b} L_i(\theta)$
        Update θ using the gradient $\nabla_\theta J(\theta)$ and learning rate α
for *epoch* $\leftarrow 1$ to N_{FT} **do**
    batches $\leftarrow$ Get non-overlapping, randomly shuffled batches from $\mathcal{D}_{train,2}$
    for b in *batches* **do**
        for i in b **do**
            $\widehat{y}^{(i)} \leftarrow M_\theta(x^{(i)})$
        $J(\theta) \leftarrow \frac{1}{B} \sum_{i \in b} L_i(\theta)$
        Update θ using the gradient $\nabla_\theta J(\theta)$ and learning rate α_{FT}

The following is an in-depth overview of Algorithm 2:

- *Line 1*: the $SPLIT()$ function splits the original $\mathcal{D}$ dataset into two parts $\mathcal{D}_{train,1}$ and $\mathcal{D}_{train,2}$.
- *Line 2*: the $CONFIGURE()$ function is used to fit the parameters β and γ of the SIR model to the given dataset $\mathcal{D}_{train,1}$.
- *Line 3*: using the configured parameters β and γ, we define the theoretical model M_{SIR}.
- *Line 4*: the $SAMPLE()$ function is used to sample m synthesized inputs. The sampling process can follow any reasonable strategy, such as random sampling or by dividing the input space into even parts. In our work [28], we followed the latter strategy.
- *Lines 7 to 13*: train the neural network on the synthesized dataset $\mathcal{D}_{Synth}$ using SGD (from Algorithm 1).
- *Lines 14 to 20*: fine-tune the trained neural network M_θ on the real dataset $\mathcal{D}_{train,2}$ using SGD (from Algorithm 1).

3.5 Experimental setup and results

During our experiments, we used a simple dense network with three hidden layers of 20, 40, 20 neurons and the rectified linear unit (ReLU) activation function in each layer. We focused on this simpler architecture instead of using more sophisticated ones like recurrent neural network (RNN), long short-term memory (LSTM) or gated recurrent unit (GRU) to show that the proposed architecture can be used for a variety of problems. This time, we made the model function similar to a simple RNN by feeding it data containing several time steps as input but this is not required; the framework itself can be used for non-time series data as well. Additionally, for handling time series data, we have considered using a neural network that does not only receive the data for the previous day and predicts the next day, but receives a sequence of t_{in} days as input, and makes predictions regarding a sequence of t_{out} days. We hand-picked the potential values for t_{in} and t_{out}, respectively, according to public forecasts that focus on the recent past and near future, as well as filtering out impractical settings during an initial prototyping phase. We noticed that predicting more days than what

the network has information on is impractical, since the network has access to less information than what it needs to generate. Accordingly, we observed a stark decline in performance even for the plain networks when using t_{out} values that were greater than t_{in}. This would have made comparing them to our architecture impossible. Therefore, during our experiments, we kept t_{out} smaller than t_{in}.

3.5.1 Influenza

To demonstrate the basic idea behind the architecture, we first experimented with diseases that are simpler in their nature. Our first objective was therefore to show that the architecture can be used for known diseases, like influenza. For these diseases, there are already some mathematically defined models, which are used heavily in practice due to their simplicity and good overall performances. We experimented with the SIR model and observed how it performs on data for a few selected countries and how we can further improve the performance by using our proposed two-step architecture. To measure the efficiency of these models, we used the available data of Germany, Hungary and Romania for the influenza season 2019. We chose these countries specifically from a bigger pool of countries by selecting those that had a data roughly resembling an SIR curve (see Figure 1). That is, we deliberately selected countries that had data which especially favored the SIR model. This is because our aim was to show that our proposed architecture can improve the overall performance of the original theoretical model even in cases where the theoretical model performs relatively well.

Since the data was aggregated at a weekly level, we used the configuration $t_{in} \in \{2, 4\}, t_{out} = 1$ for the neural network model. This way, it could process some relatively recent information (the last 2 or 4 weeks) without relying too much on older information (where $t_{in} > 4$), while keeping the model relatively simple ($t_{out} = 1$) and suitable for showcasing the potential of the model. The predictions were evaluated by calculating the mean square error (MSE) and root mean square error (RMSE) metrics. Furthermore, for every country and configuration, we trained five different neural networks to calculate the spread of the errors. Table 1 shows the summarized results of both the SIR and the two-step architecture considering a confidence level of 95% ($p = 0.05, n = 5$, using t-statistics).

Country	Model	t_{in}	t_{out}	MSE	RMSE
Germany	SIR	-	-	603.88	24.57
Germany	Two-step	2	1	176.66 ± 49.96	13.23 ± 1.79
Germany	Two-step	4	1	170.67 ± 29.50	13.04 ± 1.14
Hungary	SIR	-	-	276.92	16.64
Hungary	Two-step	2	1	184.58 ± 21.57	13.57 ± 0.79
Hungary	Two-step	4	1	127.41 ± 15.36	11.28 ± 0.67
Romania	SIR	-	-	1452.93	38.12
Romania	Two-step	2	1	877.12 ± 124.16	29.58 ± 2.05
Romania	Two-step	4	1	1178.39 ± 175.33	34.28 ± 2.54

Table 1: The results of the SIR and the two-step architecture on the influenza dataset.

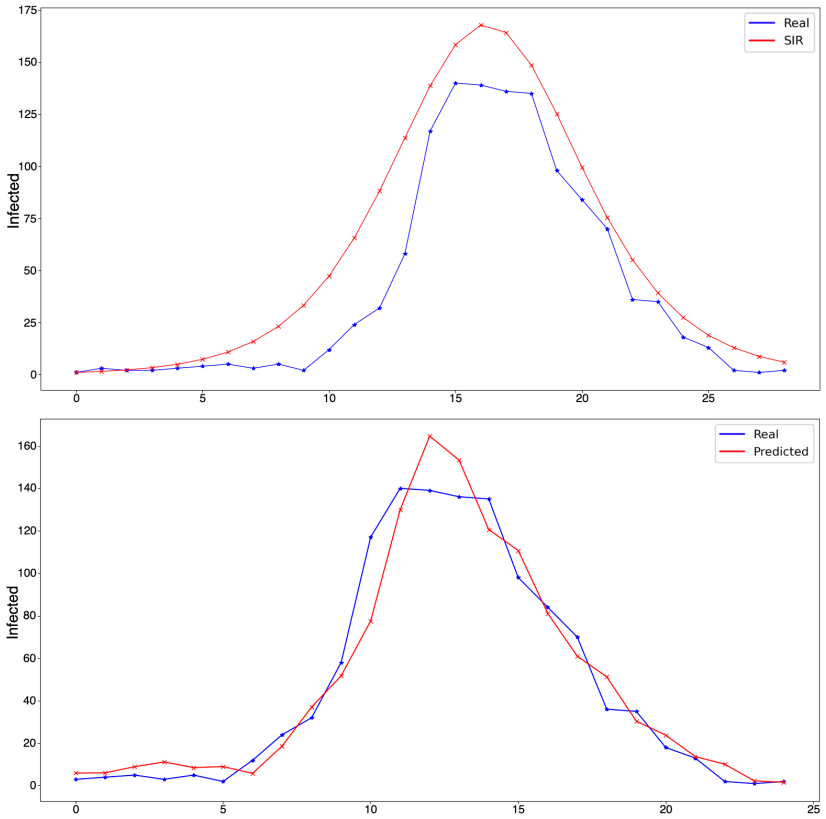

Figure 1: A comparison between the original SIR model (top) and one of the $t = 4, T = 1$ models (bottom) for Germany.

It can be seen that by using our proposed two-step neural network architecture, we were able to drastically decrease the overall error in our predictions. The improvements were the most drastic for the German and Romanian data. This is because, while the SIR model fit relatively well to the real data, there were still a number of data points that were far away from the curves produced by the models (see Figure 1). This shows that using our proposed architecture can further increase the overall performance even in cases where the original theoretical model performs well. Thus, it can be a plausible solution for tackling the spread of some diseases to achieve state-of-the-art results.

3.5.2 COVID-19

To fully demonstrate the capabilities of the proposed architecture, we ran some experiments on COVID-19 data. We think that choosing this disease can better showcase the performance and reliability of our architecture, since due to the nature of the disease, there are no theoretical models that can perform really well on COVID-19 data. As outlined previously, there are several factors, such as the numerous waves, noise in the data, and regulations that make predicting COVID-19 hard or impossible for a single simple theoretical model. Moreover, these factors make training a neural network harder, too, since the noise present in the dataset may mislead the networks during the training phase.

We used different values for t_{in} and t_{out}, respectively, to find out how many days' worth of data can better describe the disease, as well as to improve the overall performance. Namely, we used the configurations $\{(t_{in}, t_{out}) \,|\, t_{in} \in \{14, 7, 3\}$ and $t_{out} \in \{7, 3, 1\}$ and $t_{out} < t_{in}\}$, since the available data was aggregated at a daily level. This way, we experimented with how many days the model should take into account when making predictions and find out whether increasing the size of the input resulted in any substantial performance gains. We also tried changing the output size to experiment with whether doing so could make the model more reliable by having it constantly focus on a series of next days. Since due to the nature of the model, there will be multiple predictions for a given day when using $t_{out} > 1$, we also outline an aggregated solution that combines the outputs of a single model for a given day by taking the mean of the predicted values.

We found that using $t_{in} > 14$ made the training of the model much

harder and resulted in models that performed worse. This made the model more complex, which focused too much on days that were too far away, therefore not contributing to the current number of infected people. For a similar reason, we observed that when t_{out} was closer to t_{in}, the overall performance plummeted, since the model simply did not have enough information to make accurate long-term predictions. Moreover, we found that when using the setting $t_{out} > 3$, the quality of the predictions regarding the future started to deteriorate, suggesting that predictions with a larger output window size for the COVID-19 dataset were not feasible. Therefore, we suggest that it is better to use smaller t_{out} values instead of trying to make long-term predictions (see Figure 2).

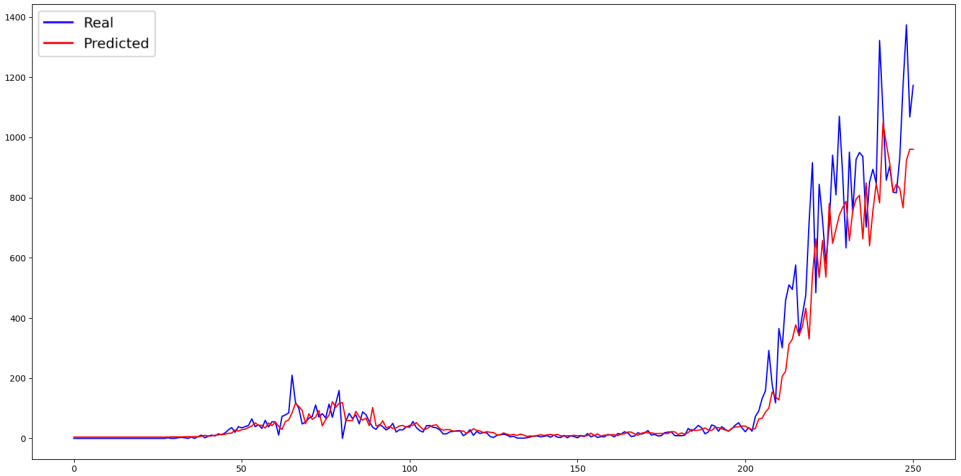

Figure 2: A model with the configuration $t_{in} = 14, t_{out} = 1$ and its predictions (red) for Hungary.

After the experiments, we concluded that our approach led to shorter training times and better convergence. After the neural network obtained a set of weights that was roughly equivalent to the given SIR model (the predictions were close to the curve I of the SIR model), we decreased the learning rate and trained the neural network on real data. This step is applied to make sure that the weights of the neural network do not change substantially, which could have resulted in a model that no longer resembles the original SIR model. Another advantage of the decreased learning rate is that the impact of the noise present in real data can be reduced in this way.

We trained all the models on the first wave of COVID-19 for any given country. This means that we first fit a SIR model to the data of the first wave, then approximated the SIR model with a neural network, then trained it further on the real data of the first wave. For the first wave, the data was also split into training and test parts. After evaluating the networks on the test part of the first wave, we tested the models on the second wave of COVID-19 for the given country. To test the overall reliability and performance of our proposed architecture, we compared its results with a plain neural network that was not initialized with weights similar to an SIR model but simply trained on the available COVID-19 data. These plain neural networks were also trained in the exact same way: first they were fitted to the first wave of the real data and then tested on the second wave. We repeated each experiment for a given (t_{in}, t_{out}) pair a total of n times to measure how the results fluctuated. During our research, we considered multiple values when choosing n. We found the number 10 to be the best for our purpose: the samples gathered by training each model 10 times proved to be representative enough to reliably calculate the overall error while the time required to train the models was still manageable.

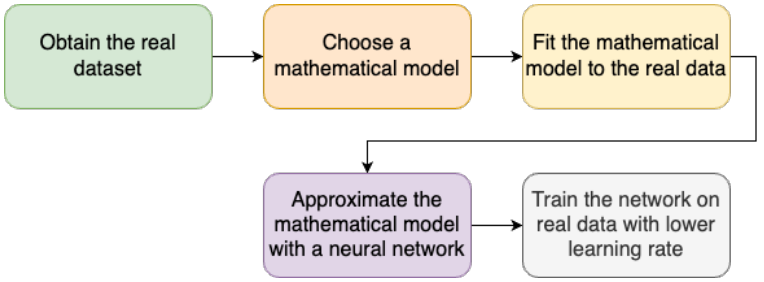

Figure 3: The basic workflow of the two-step architecture.

This approach (see Figure 3) has quite a few benefits compared to training a neural network directly on real data. First of all, the amount of noise generated throughout the training phase is considerably reduced. This is thanks to the model being taught on a much smoother and mathematically well-defined function, which is the output of the SIR model. The shape of the infected curve (I) for a given SIR model is bell-shaped, with no irregularities and noise. Hence, it is easier for neural networks to be trained on this simpler data. Moreover, since the network has a solid set of weights after the first phase is finished and the learning rate is smaller during the second phase, the

irregularities present in the real data do not affect the training as much as they would normally. This fact, along with the ability of an SIR model to approximate the original data fairly well, leads to a more controlled training process. This way, the network can first extract meaningful information about the nature of the disease, and then it has access to a more irregular dataset for further training.

Another huge advantage of this framework is the increased amount of data available during the learning phase. This is beneficial when the available dataset contains only a handful of records; as in the case of the influenza dataset in our study. However, by pre-training on a theoretical model that behaves roughly the same, we can generate the necessary data points to obtain such a starting set of weights that only needs to be further refined on real data. This ultimately leads to a smoother and more controlled training process. Tables 2, 3, 4, and 5 contain our experimental results for Hungary and Germany. All values shown in these tables were rounded to the nearest whole number to make the interpretation of the results easier. We have also included additional results for other countries in Appendix A.

Table 2: Hungary - first wave errors.

t_{in}–t_{out}	Model	First wave errors			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	87 ± 20	-	9 ± 1	-
	Plain	272 ± 53	-	16 ± 2	-
7–1	Two-step	108 ± 19	-	10 ± 1	-
	Plain	186 ± 32	-	14 ± 1	-
3–1	Two-step	99 ± 12	-	10 ± 1	-
	Plain	141 ± 16	-	12 ± 1	-
14–7	Two-step	99 ± 31	79 ± 25	10 ± 1	9 ± 1
	Plain	302 ± 70	210 ± 58	17 ± 2	14 ± 2
14–3	Two-step	85 ± 12	67 ± 15	9 ± 1	8 ± 1
	Plain	291 ± 41	272 ± 104	17 ± 1	16 ± 3
7–3	Two-step	93 ± 20	81 ± 25	10 ± 1	9 ± 1
	Plain	214 ± 70	169 ± 74	14 ± 2	12 ± 3

It can easily be seen that the results of the proposed architecture are generally way better than that of a simple, randomly initialized neural network. This shows that having the model learn a less complex and theoretically defined function that roughly resembles the target data

Table 3: Hungary - second wave errors.

t_{in}–t_{out}	Model	Second wave errors			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	17249 ± 5150	-	129 ± 17	-
	Plain	31268 ± 7438	-	175 ± 21	-
7–1	Two-step	17882 ± 2884	-	133 ± 11	-
	Plain	25712 ± 5262	-	159 ± 16	-
3–1	Two-step	12987 ± 448	-	114 ± 2	-
	Plain	27127 ± 3578	-	164 ± 11	-
14–7	Two-step	12826 ± 1247	30126 ± 9607	113 ± 5	170 ± 27
	Plain	37160 ± 27294	51982 ± 5450	181 ± 51	227 ± 12
14–3	Two-step	14273 ± 2688	36985 ± 18003	119 ± 11	182 ± 46
	Plain	54496 ± 43076	42304 ± 11570	211 ± 75	202 ± 28
7–3	Two-step	13117 ± 2039	25896 ± 9038	114 ± 9	157 ± 28
	Plain	76712 ± 51466	53238 ± 23835	247 ± 94	221 ± 49

Table 4: Germany - first wave errors.

t_{in}–t_{out}	Model	First wave errors (Germany)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	153053 ± 58837	-	380 ± 72	-
	Plain	207112 ± 60990	-	446 ± 68	-
7–1	Two-step	129619 ± 35072	-	354 ± 49	-
	Plain	204516 ± 43125	-	447 ± 48	-
3–1	Two-step	112976 ± 13019	-	335 ± 20	-
	Plain	156238 ± 44880	-	387 ± 57	-
14–7	Two-step	111460 ± 39019	73770 ± 24834	325 ± 59	265 ± 44
	Plain	151080 ± 46847	218403 ± 112174	379 ± 66	441 ± 116
14–3	Two-step	113580 ± 29820	105878 ± 41615	332 ± 42	316 ± 59
	Plain	250274 ± 97146	243238 ± 132332	485 ± 92	460 ± 134
7–3	Two-step	116271 ± 26964	138752 ± 56923	337 ± 40	358 ± 77
	Plain	165840 ± 64845	184525 ± 85200	394 ± 79	413 ± 89

may be a beneficial pre-training step and could yield potentially better results when trained further on real data compared to models that are trained only on the latter. Another important note is that the target mathematical function does not need to match precisely with the real data, as it was the case for our research regarding COVID-19. The only important part is that it should contain some key information (in this case the bell-shape curve hinting that the number of diseases

Table 5: Germany - second wave errors.

t_{in}–t_{out}	Model	Second wave errors (Germany)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	386211 ± 55594	-	619 ± 44	-
	Plain	350296 ± 59203	-	588 ± 50	-
7–1	Two-step	373298 ± 40520	-	609 ± 32	-
	Plain	378416 ± 36601	-	614 ± 30	-
3–1	Two-step	396793 ± 23701	-	629 ± 19	-
	Plain	538980 ± 393801	-	685 ± 84	-
14–7	Two-step	244445 ± 55773	268252 ± 90832	489 ± 54	508 ± 75
	Plain	595381 ± 453948	268209 ± 66891	680 ± 274	511 ± 65
14–3	Two-step	284439 ± 34150	365277 ± 121480	532 ± 32.42	593 ± 88
	Plain	336348 ± 60559	404303 ± 128238	576 ± 52	625 ± 89
7–3	Two-step	286050 ± 27532	352900 ± 38206	534 ± 26	592 ± 33
	Plain	593034 ± 519253	326211 ± 56222	677 ± 276	567 ± 50

should keep increasing until a certain point and then start decreasing from then on) that can provide a strong foundation for the network to build upon in the second phase of the training. This approach also makes it harder for the model to focus on dispensable features due to the first step containing the differential equations in its loss function. Another interesting point is that training the network on a SIR model for the first wave has proved to be really beneficial for predicting even the second wave. The network surpassed the original theoretical model, which also proves that the network can learn important features present in the theoretical model, which it can use to recognize similar patterns in future data and remarkably surpass the performance of plain neural networks.

Overall, this two-step approach made the training of the model easier and more manageable, since it is always easier to fine-tune a network to fit to a mathematically well-defined function. This also reduced the amount of noise the networks faced during training thanks to first being trained on a theoretical model and then switching to the real data with a smaller learning rate and an already robust set of weights instead of random ones. Moreover, we did not need any pre-configured network even though we basically pre-train the model, since the theoretical model can be relatively easily defined. This in turn provided a fast and cheap, yet effective way of using transfer learning.

3.6 Conclusions

In this chapter, we have introduced a two-step framework for training performant models for time series prediction. We explained that there are certain scenarios, e.g., when the dataset is small, when training a neural network model on the data may lead to sub-optimal performance. We also highlighted the fact that theoretical models, due their rigid nature, may not perform optimally either, if the operation of said model does not fully reflect the nature of the observed data. To overcome these issues, we trained neural networks to first approximate the given theoretical model and then trained them further on real data. We showed that following this approach, the models could first grasp the most important aspects of the data (spread, nature, bell-like shape etc.) without being affected by the outliers and noise present in the real data. Once their set of weights was solid enough, they could learn further on real data.

We evaluated the proposed framework using influenza and COVID-19 data. We summarized the results of the architecture for a number of countries and various configurations by changing the input and output size of the model. We showed that the proposed approach performed better than theoretical models for predicting the weekly number of influenza patients. We also tested how this approach fares with much noisier COVID-19 data, which currently no theoretical models can predict reliably. Our experimental results show that the proposed architecture performs better than simple neural networks that are only trained on real data. We also highlighted how this approach can combine the benefits of the two main pillars, which were the theoretical models and the neural networks. Namely, we showed how one model trained using this architecture can not only surpass plain neural networks that are initialized randomly but how the features regarding the spread of the disease extracted from the first wave can help with making predictions for the second wave, surpassing the original theoretical model, too, which could only predict a single wave.

4 Combining hand-crafted features with features extracted by CNNs

4.1 Introduction

Eye diseases such as diabetic retinopathy and diabetic macular edema pose a major threat in today's world as they affect a significant portion of the global population. Therefore, it is of utmost importance to develop robust solutions that can accurately detect these diseases, especially in their early stages. However, current methods, based on hand-crafted features devised by experts, are not sufficiently accurate. Several solutions have been proposed that use deep learning techniques to improve the performance of such systems. However, they usually ignore the highly valuable hand-crafted features, that could contribute to more accurate predictions.

This chapter presents a novel way of combining the hand-crafted features with those extracted by a CNN. We detail the main motivation behind the solution, showing the potential limitations and drawbacks of traditional methods relying purely on hand-crafted features, and highlight the potential benefits of our method. We introduce multiple variants of our proposed framework and show that all of them can solve the problem of detecting various diabetes-related eye diseases using digital fundus images as input. We systematically study several state-of-the-art neural networks and methods, and propose a number of ways to integrate them into our framework. We show that it is possible to achieve significantly better results and outperform networks that do not consider hand-crafted features using the proposed methods.

The structure of this chapter is as follows. In section 4.2, we explain the main motivations and reasoning behind the proposed architecture. We introduce the main idea and detail the biggest advantages of incorporating the use of hand-crafted features during the training procedure. In section 4.3, we give an overview of the dataset that was used during our research. In section 4.4, we present our proposed framework and introduce different variants, highlighting their advantages and disadvantages. In section 4.6, we detail our experimental setup, and in section 4.7, we summarize our experimental results. Lastly, in section 4.8, we provide our conclusions and list our most notable contributions.

The architecture described in this chapter and all corresponding experimental results were published in [29].

4.2 Motivation

Nowadays, diabetic retinopathy (DR) is the most common cause of blindness in developed countries. It is an eye disease caused by long-standing diabetes. Moreover, about 1 in 15 people with diabetes will develop diabetic macular edema (DME). DME occurs when blood vessels of the retina leak fluid into the macula which causes blurry vision. In 2019, an estimated 1.5 million deaths were directly caused by diabetes and the World Health Organization estimates that 422 million people worldwide have the disease [43]. Progression to vision impairment can be slowed down if DR/DME is detected at an early stage. DR is a diabetes complication, with possible consequences ranging from mild visual impairment to blindness. Currently, detecting the signs of DR/DME is a time-consuming and manual process that requires a trained clinician to examine and evaluate digital color fundus photographs of the retina. During the annual screening, a large number of digital images are taken and evaluated by an ophthalmologist. Only in the UK, the screening programs result in around two million retinal images for evaluation each year [44]. Repeated screening of DR is therefore not only costly but also exhausting, so there is a significant need to automate this process.

Expectations for trustworthy automated screening systems are high in the case of DR and DME. Recently, deep learning has been increasingly used in the field of medical image analysis, and many such methods have been proposed that use CNNs to detect microaneurysms (MAs), hemorrhages (HEs), hard exudates (EXs), or soft exudates (SEs). [45] proposed a solution in which the entire fundus image was divided into patches that were considered as input without any further preprocessing steps. Then, the applied Stacked Sparse Autoencoder automatically extracted the distinguishing features to classify these patches. [46] showed how the authors used a single CNN to automatically segment and discriminate lesions in fundus images. They also divided the input image into 51×51 pixel patches and used their convolutional neural network to classify them as background, EXs, HEs, or MAs. In addition to the publications mentioned above, there are many other papers in which the relevant features are extracted using

conventional digital image processing tools. [47] used morphological processes and kernel density estimation to segment MAs and evaluated the input images based on the number of detected lesions. [48] adapted optimal wavelet transform and template matching to perform automatic segmentation of MAs in retinal images. [49] proposed the use of multiscale amplitude modulation-frequency modulation (AM-FM) as a feature extraction method to discriminate between normal and pathological retinal images.

[50] proposed a solution that combines the powerful, self-extracted, CNN-based features with traditional, hand-crafted ones into a single framework to enhance classification performance. The idea derived from [51], where the authors claimed that a CNN can automatically extract many important local, textual features from images by convolving with a sliding window and forming a filter. However, besides the local features, the global image descriptors also have played an important role in many image processing tasks. While the local features can be called textural features, the global features usually mean contour features and structural ones. In the case of DR and DME, the presence of diseases is characterized by detecting one or more retinal lesions like MAs, HEs, EXs, and SEs. These signs can be described well by their contour feature and we can successfully apply them to improve the final accuracy of a CNN-based screening system.

In this chapter, we extend the solutions proposed in [50] by investigating the applicability of some additional state-of-the-art neural networks. We show that our experimental results support our former statement, namely, that we can improve the final classification results of a CNN-based solution by using hand-crafted features besides deep learning ones. Moreover, we investigate the advantages of the proposed methodology and also its limitations by a comprehensive comparison, where we test different ways to concatenate the automatically extracted textural features with the hand-crafted ones.

4.3 Dataset

In this section, we give a brief description of the publicly available datasets that were used to train and evaluate the methods described in this chapter. The Kaggle DR and Messidor datasets and the training part of the Indian Diabetic Retinopathy Image Dataset (IDRiD) were used for training, while all evaluations were performed using the test

part of the IDRiD dataset.

4.3.1 Indian Diabetic Retinopathy Image Dataset

The IDRiD dataset [52] consists of 516 color fundus images divided into a training and test part with 413 and 103 images, respectively. The images have a resolution of 4 288 × 2 848 pixels and a 50° field of view. Each image is categorized according to the severity of DR (5 classes) and the risk of DME (3 classes). Regarding DR, the following classes are defined: no DR (DR0), mild DR (DR1), moderate DR (DR2), severe DR (DR3), and proliferative DR (DR4). The training and test sets contain 134, 20, 136, 74, 49, 33, and 33, 5, 32, 19, 13 images for DR0, DR1, DR2, DR3, and DR4, respectively. Concerning DME, the classes are defined based on the presence of hard exudates near the macular center. These classes are: no risk of macular edema (DME0), moderate risk of macular edema (DME1), and high risk of macular edema (DME2). The training set includes 177 DME0, 41 DME1, and 195 DME2 images, while the test set contains 44 DME0, 10 DME1, and 48 DME2 images.

4.3.2 Kaggle DR dataset

The Kaggle DR dataset is the training portion of the dataset provided by EyePACS for a DR grading challenge [53] held by Kaggle. It contains 35 126 color fundus images with resolutions ranging from 400 × 315 to 5 184 × 3 456 pixels and different fields of view. For each image, a DR grading score is provided: 25 810 of these images are categorized as DR0, 2443 as DR1, 5292 as DR2, 873 as DR3, and 708 as DR4. In addition, 7 806 of the images in this dataset were labeled by an experienced local ophthalmologist for the risk of DME: 5 949 of the images are categorized as DME0, 1 033 as DME1, and 824 as DME2. It is important to note that several images in this dataset are affected by imaging artifacts, blurring, under- or overexposure.

4.3.3 Messidor dataset

The Messidor dataset [54] comprises 1 200 color fundus images with three different resolutions (1 440 × 960, 2 240 × 1 488, and 2 304 × 1 536 pixels) and a 45° field of view. Both DR and DME grading scores are available for the images of this dataset. However, DR scoring differs

slightly from the method used for the other two datasets. Using the DR severity classes of IDRiD and Kaggle DR, 546 of the images in this dataset are categorized as DR0, 153 as DR1, 247 as DR2, and 254 as DR4. Regarding DME, 974 images are categorized as DME0, 75 as DME1, and 151 as DME2.

4.4 Extracting the hand-crafted features

The image-level and lesion-specific approaches used to extract the traditional features considered in our methods are described in this section.

4.4.1 Image-level feature extraction

For image-level feature extraction, we employed an amplitude and frequency modulation-based approach [49], which extracts various features from a retinal image by decomposing its green channel into AM-FM components at different scales [55]. A collection of twenty-five bandpass channel filters, coupled with four frequency scales, is used to generate the different scales for feature extraction. After the image features are extracted, the information is clustered into 30 groups using k-means clustering. As a result, a 30-element feature vector is generated that reflects the intensity, shape, and texture of the structures of the image.

4.4.2 Lesion-specific feature extraction

We used two detector ensembles consisting of ⟨preprocessing method, candidate extractor⟩ pairs (⟨PP, CE⟩ for short) organized into a voting system to extract lesion-specific features associated with microaneurysms (MAs) and exudates (EXs). By applying a PP to the input retinal image and a CE to the PP output, a ⟨PP, CE⟩ pair is formed. A ⟨PP, CE⟩ pair extracts a set of candidate lesions from the input image and functions as a single detector method in this way.

Number of MAs Because MAs are the earliest manifestations of DR and indicators of its progression, their number is an important factor in DR classification. MAs are capillary swellings that appear as tiny red dots; however, their similarity to vascular fragments makes them difficult to detect.

The combined output of the MA detector ensemble is obtained in the following way: if the Euclidean distances of the candidates in the result sets of ⟨PP, CE⟩ pairs are smaller than a given value, they are merged. The ratio of the number of ⟨PP, CE⟩ pairs suggesting this candidate to the total number of pairs in the ensemble is used to assign a confidence value to each common candidate in the ensemble.

We used the results of [56] to form the ⟨PP, CE⟩ pairs of the ensemble. Five PPs (contrast-limited adaptive histogram equalization (CLAHE) [57], illumination equalization (IE) [58], vessel removal with inpainting (VR) [59], Walter-Klein contrast enhancement (WK) [60], and "no preprocessing" (NP) for formal reasons) and three CEs (the methods of [61], [62], and [63]) were selected. Table 6 (a) lists the ⟨PP, CE⟩ pairs in our MA detector ensemble. After the MA candidate set of the ensemble is obtained, it is thresholded at six confidence levels and the number of candidates at each level is counted to obtain six features.

EX-specific features EXs have properties useful in determining the severity of nonproliferative DR and the risk of DME. EXs are lipid residues of serous leakage from injured capillaries that appear as bright spots of various shapes in retinal images.

The combined output of the EX detector ensemble is obtained as follows: each ⟨PP, CE⟩ pair produces a binary mask containing EX candidates. The probability that a pixel belongs to an EX is determined by the ratio of the number of pairs that marked the pixel as EX to the total number of pairs, which is then used to construct a probability map using the output of the different pairs.

The results of [64] were used to create the ensemble described above. Four PPs (gray-world normalization (GN) [65], illumination equalization (IE) [58], morphological contrast enhancement (MC) [66], and vessel removal with inpainting (VR) [59]) and three CEs (the method of [67], [68], and [69]) were selected. Table 6 (b) lists the eight ⟨PP, CE⟩ pairs that were used in the ensemble. As the last step, the result set of the EX ensemble is thresholded at eight different confidence levels, resulting in a total of 32 features: the ratio of all EX pixels to region of interest (ROI) pixels, the number of EXs (8-connected components), the ratio of the largest EX (8-connected component) to ROI, and the average EX size to ROI.

Table 6: The ⟨PP, CE⟩ pairs of the (a) MA, and (b) EX detector ensembles.

(a)

	PP	CE
1	NP	Lazar *et al.*
2	CLAHE	Lazar *et al.*
3	IE	Lazar *et al.*
4	WK	Lazar *et al.*
5	NP	Walter *et al.*
6	CLAHE	Walter *et al.*
7	NP	Zhang *et al.*
8	VR	Zhang *et al.*
9	WK	Zhang *et al.*

(b)

	PP	CE
1	GN	Sopharak *et al.*
2	MC	Sopharak *et al.*
3	VR	Sopharak *et al.*
4	IE	Walter *et al.*
5	MC	Walter *et al.*
6	VR	Walter *et al.*
7	IE	Welfer *et al.*
8	MC	Welfer *et al.*

4.5 Combining the hand-crafted features with deep learning techniques

[50] showed that state-of-the-art accuracy could be attained for this problem by merging the hand-crafted features with the features extracted by a CNN. To achieve this, [50] extended the last fully connected (FC) layer (4 096 neurons) with additional neurons (68 in total) and then reduced the weights to the given class probabilities. More precisely, for any given image $x^{(i)}$, the feature vectors extracted by an AlexNet [70] model and the traditional extractors were calculated. The hand-crafted features were normalized to bring them to the same scale as the features extracted by AlexNet. Then, these two feature vectors were concatenated and passed through an additional fully connected layer.

In subsequent sections, we present three different versions for improving the effectiveness and performance of the described method. We will denote each version as V1, V2, and V3, respectively.

4.5.1 Extending the original method by using other architectures

It can be seen how optimizing for more descriptive $y_1^{(i)}$, in other words, choosing the architecture, can affect performance. This is why we extended the original idea to several commonly used networks other than

AlexNet [71] and objectively compared the results of those networks and their variants that use the hand-crafted features to demonstrate that our solution greatly improves the classification accuracy. All the architectures received the input images in the shape that the original networks operated on. This meant that each RGB input image $x^{(i)}$ was resized to the size of $224 \times 224 \times 3$ before being given to the given network. During the research, we chose AlexNet as our baseline network and tried to improve its accuracy. To this end, we used several state-of-the-art networks, such as MobileNetv2 [72] and Resnet-50 [73] and compared their results to that of the baseline – both with and without the hand-crafted features. In the case of the first one, for any given input image $x^{(i)}$ the hand-crafted features were calculated and then concatenated with the extracted CNN features of the given network. For the latter, only the input image was given to the algorithm. For this version V1 of the algorithm, we kept the original structure [50] but swapped the feature extracting part with the given network (AlexNet, MobileNetv2, and Resnet-50, respectively), as can be seen in Figure 4.

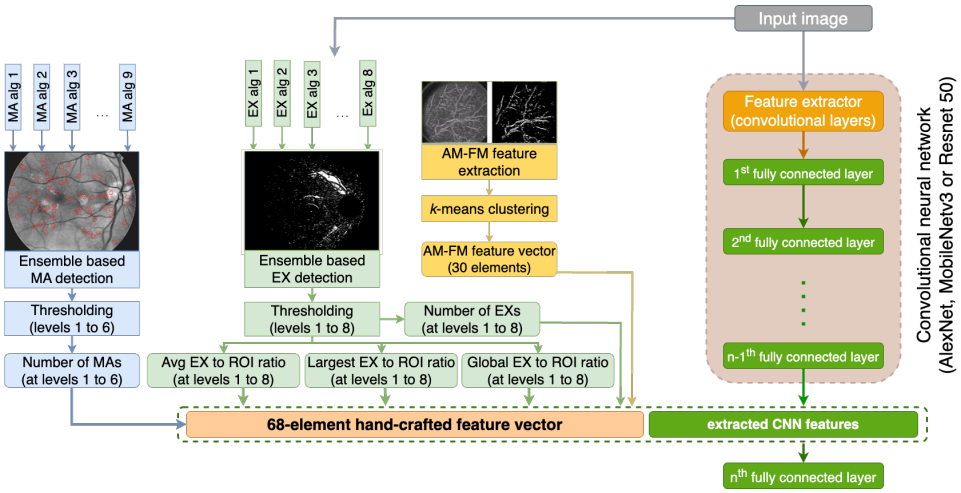

Figure 4: Fusing the hand-crafted features with those extracted by a CNN in the last layer of the given neural network.

Definition 4.1. *For any given input image $x^{(i)}$, features $y_1^{(i)}$ extracted by a CNN and $y_2^{(i)}$ normalized vector extracted by traditional extractors, the output of the architecture V1 considering both features is defined as*

$$\widehat{y}^{(i)} = \sigma\left(\theta^\top y_c^{(i)}\right) \tag{11}$$

where $y_c^{(i)}$ represents the concatenated feature vector computed as $y_c^{(i)} := < y_1^{(i)}, y_2^{(i)} >$ and σ is the softmax function.

While by substituting AlexNet with the other variants we could greatly increase the computational capacity of the architecture, this method had some shortcomings. Namely, we can clearly see that although we take into account both the hand-crafted and CNN features, we only pass them through one set of weights θ after calculating $\widehat{y}^{(i)} = \sigma(\theta^\top y_c^{(i)})$, where $y_c^{(i)} =< y_1^{(i)}, y_2^{(i)} >$. This makes combining $y_1^{(i)}$ and $y_2^{(i)}$ hard since we need to do that with only the weights θ.

Theorem 4.1. *Given the weights θ, features $y_1^{(i)}$ extracted by a CNN and $y_2^{(i)}$ normalized vector extracted by traditional extractors, the formula introduced in (11) for the V1 architecture is not guaranteed to be able to approximate the real outputs $y^{(i)}$ if the relationship between $y_c^{(i)}$ and $y^{(i)}$ is non-linear.*

Proof. Let θ denote the weights of the V1 architecture, and let $y_1^{(i)}$ and $y_2^{(i)}$ denote the CNN and normalized hand-crafted features, respectively. Furthermore, let us indirectly assume that Theorem 4.1 is false, meaning that the V1 architecture is in fact guaranteed to be able to approximate the real outputs $y^{(i)}$ even if the relationship between $y_c^{(i)}$ and $y^{(i)}$ is non-linear.

Let us construct a theoretical example, where the output is the square of the first element of the concatenated feature vector, meaning that $y^{(i)} = {y_c^{(i)}}_1^2$. This would require the algorithm to find weights θ such that

$$y^{(i)} = {y_c^{(i)}}_1^2 = \sigma(\theta^\top y_c^{(i)}) \tag{12}$$

holds true. From this equation, $\theta^\top y_c^{(i)}$ is by definition a linear transformation, as applying the transformation $v := \theta^\top y_c^{(i)}$ can only produce vectors v that are linearly dependent on $y_c^{(i)}$. Therefore, the only way for this equation to be true is if the activation function σ is a quadratic function. However, the activation function used in Definition 4.1 is the softmax function. Ultimately, we arrive at a contradiction, meaning that our initial assumption is false, showing that there is no guarantee

that the V1 architecture is able to approximate the real outputs $y^{(i)}$ if the relationship between $y_c^{(i)}$ and $y^{(i)}$ is non-linear.

□

For this reason, we also explore new architectural solutions that may more effectively combine the information extracted by these hand-crafted features with that of the features extracted by a neural network.

4.5.2 Learning the features in parallel

One reasonable solution is to divide the computational graph of the network into two branches as can be seen in Figure 5. One branch can be responsible for processing hand-crafted features and another one can process the input image. These parts would calculate their predictions separately and we could merge the results to get our final predictions. Namely, instead of calculating the feature vectors $y_1^{(i)}$ and $y_2^{(i)}$, we could calculate the approximate predictions $\bar{y}_1^{(i)}$ and $\bar{y}_2^{(i)}$ for each part and then average the outputs.

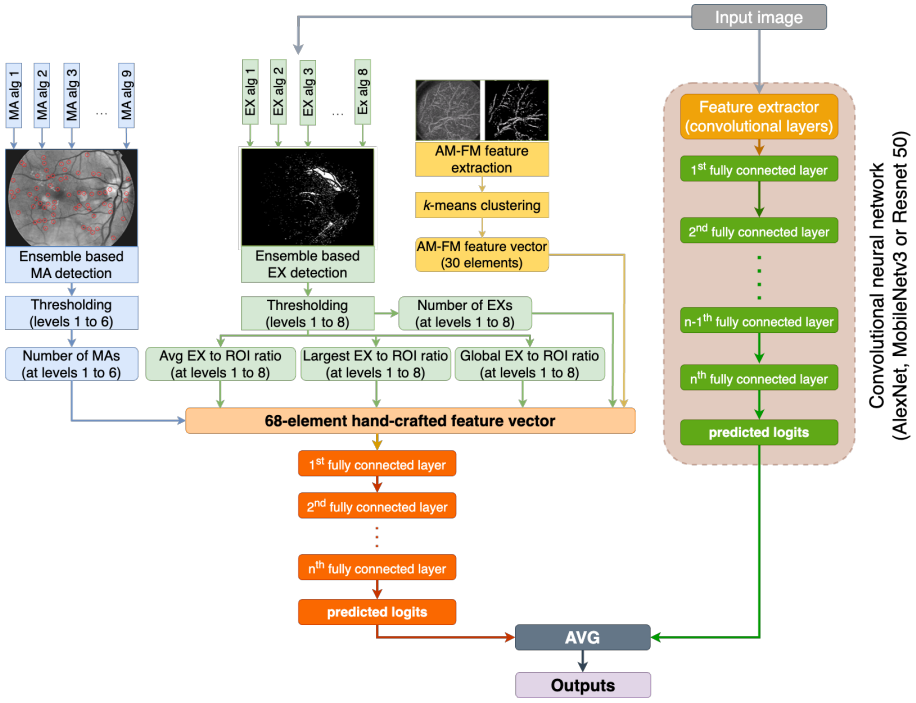

Figure 5: Fusing the hand-crafted features with those extracted by a CNN by learning in two separate paths.

Definition 4.2. *Given the feature vectors $y_1^{(i)}$ extracted by a CNN, $y_2^{(i)}$ normalized vector extracted by traditional extractors, two sets of weights $\theta_1^1, \theta_1^2, ..., \theta_1^k$ and $\theta_2^1, \theta_2^2, ..., \theta_2^l$, two sets of activation functions $\sigma_1^1, \sigma_1^2, ..., \sigma_1^{k-1}$ and $\sigma_2^1, \sigma_2^2, ..., \sigma_2^{l-1}$ for the CNN and hand-crafted features, respectively, and the softmax function σ, a more complex combination of the feature vectors is defined as*

$$\begin{aligned} \bar{y}_1^{(i)} &= {\theta_1^k}^\top ... \sigma_1^2({\theta_1^2}^\top \sigma_1^1({\theta_1^1}^\top y_1^{(i)})), \\ \bar{y}_2^{(i)} &= {\theta_2^l}^\top ... \sigma_2^2({\theta_2^2}^\top \sigma_2^1({\theta_2^1}^\top y_2^{(i)})), \\ \widehat{y}^{(i)} &= \sigma\left(\frac{\bar{y}_1^{(i)} + \bar{y}_2^{(i)}}{2}\right). \end{aligned} \tag{13}$$

This would in fact act like a "mini" ensemble network that uses averaging, but where each path of the computational graph is trained at the same time. This is possible since the backpropagation of the predicted error is executed with one step from the calculated predictions in 13.

This approach has many strong points. First of all, the features are processed separately and undergo more steps, making it possible to derive more high-level information. Moreover, we could benefit from the advantages of using an ensemble solution, which often yields better and more stable results. Finally, the network could also potentially learn when it should emphasize the hand-crafted features $y_1^{(i)}$ and when the CNN features $y_2^{(i)}$, depending on the input image $x^{(i)}$ by adjusting its weights accordingly. This can be especially important if sometimes the first one and sometimes the latter one is more accurate, and when thus having the ability to weight these predictions depending on the input image $x^{(i)}$ could result in more stable predictions.

4.5.3 A deeper combination of the hand-crafted and CNN features

The method described in section 4.5.2 guarantees that both the features extracted by the CNN and the hand-crafted ones undergo a longer processing step due to the increased number of fully connected layers. However, in the long run, it may not always be beneficial to entirely separate the processing of the hand-crafted and CNN features. This is because doing so may make the learning process much harder due to the increased number of parameters and by having the network concentrate

on essentially two different things: making sense of the hand-crafted features and making sense of the input image. This could disrupt the learning process, leading to poor results in some cases.

Claim 4.1. *Using two completely separate sets of weights* $\theta_1^1, \theta_1^2, ..., \theta_1^k$ *and* $\theta_2^1, \theta_2^2, ..., \theta_2^l$ *to combine the features* $y_1^{(i)}$ *extracted by a CNN and* $y_2^{(i)}$ *normalized vector extracted by traditional extractors may lead to sub-optimal performance, as the two branches of the architecture can only focus on one feature set at any given time and completely ignore the features extracted by the other branch.*

Reasoning. During the forward pass, each of the branches of the algorithm starts by calculating the features $y_1^{(i)}$ and $y_2^{(i)}$ for the given input image $x^{(i)}$. Denoting the branch extracting the CNN features as b_1 and the one extracting the hand-crafted features as b_2, this means that any weight inside b_1 only has access to features extracted by b_1, but not to any features from b_2. The same constraint applies to b_2 as well, meaning that it is trivial to see that until the calculation of the features $y_1^{(i)}$ and $y_2^{(i)}$, the two branches b_1 and b_2 cannot share any information with each other. Then, following (13), each branch utilizes its own sets of weights to calculate $\bar{y}_1^{(i)}$ and $\bar{y}_2^{(i)}$, respectively, resulting in the complete isolation between b_1 and b_2 still standing. □

Instead, we could move the concatenation step $y_c^{(i)} =< y_1^{(i)}, y_2^{(i)} >$ up in the architecture to merge the features $y_1^{(i)}$ that have been extracted by purely the convolutional layers with the hand-crafted ones, denoted as $y_2^{(i)}$. Then, we can use more FC layers to process the joint features $y_c^{(i)}$. With this approach, we could get the benefits of both worlds: the network would have more capabilities to process the feature vectors thanks to the increased number of dense layers, and both the features extracted by the convolutional layers and the hand-crafted ones would be processed at the same time, layer by layer. This process can be observed in Figure 6.

Definition 4.3. *Given the feature vectors* $y_1^{(i)}$ *extracted by a CNN and* $y_2^{(i)}$ *normalized vector extracted by traditional extractors and weights* $\theta^1, \theta^2, ..., \theta^k$*, another possible combination of the feature vectors, denoted as the V3 architecture, is defined as*

$$\begin{aligned} y_c^{(i)} &= < y_1^{(i)}, y_2^{(i)} > \\ \widehat{y}^{(i)} &= \sigma^k({\theta^k}^\top ...\sigma^2({\theta^2}^\top \sigma^1({\theta^1}^\top y_c^{(i)}))). \end{aligned} \tag{14}$$

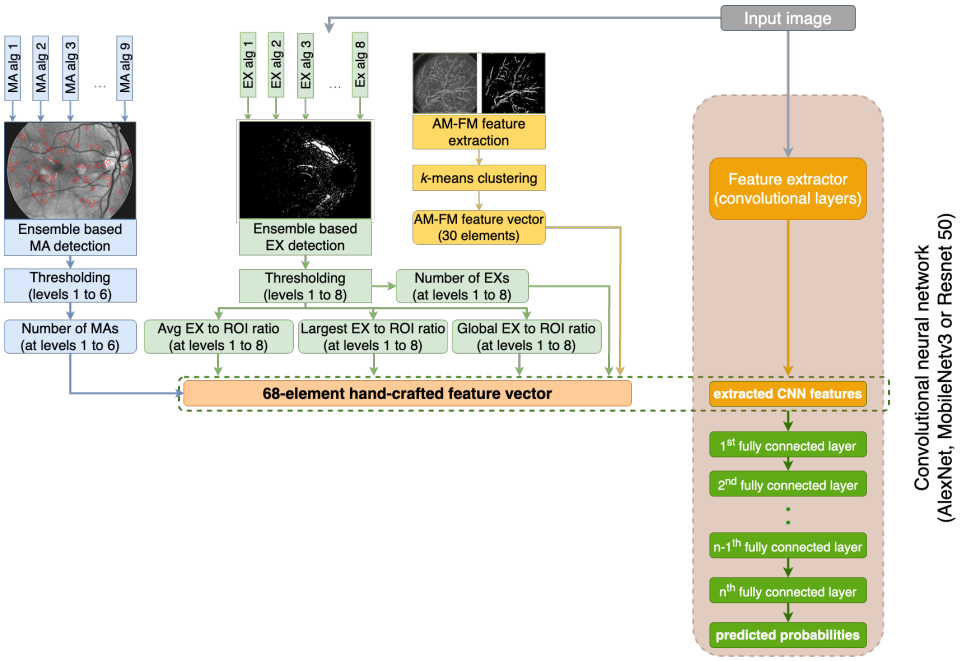

Figure 6: Fusing the hand-crafted features with those extracted by a CNN by moving the concatenation step up in the architecture.

Claim 4.2. *Using the same set of weights* $\theta^1, \theta^2, ..., \theta^k$ *to process the combined feature vector* $y_c^{(i)}$ *constructed from* $y_1^{(i)}$*, the features extracted by a CNN and* $y_2^{(i)}$*, the normalized vector extracted by traditional extractors, gives the V3 architecture the ability to process every piece of information at the exact same time. This makes it possible to learn and recognize more complex patterns that would require the simultaneous presence of both kinds of features.*

Reasoning. Let $\theta^1, \theta^2, ..., \theta^k$ denote the weights, and $y_1^{(i)}$ and $y_2^{(i)}$ the feature vectors. Following Definition 4.3, it is easy to see that both of the feature vectors are being processed at the same time, as after the concatenation step $y_c^{(i)} =< y_1^{(i)}, y_2^{(i)} >$, the step ${\theta^1}^\top y_c^{(i)}$ achieves, by definition, a linear combination of the two vectors, by having weights corresponding to each and every element of the feature vectors and summing the elements. Therefore, from this point on, each multiplication using the subsequent weights $\theta_2, \theta_3, ..., \theta_k$ has access to all of the feature information in both of the vectors. □

4.6 Experimental setup

As noted in section 4.3, we used a total of 36 739 images for training (413 from IDRiD, 35 126 from the Kaggle and 1 200 from the Messidor dataset) and the test part of the IDRiD dataset for testing. To get a better understanding of the results, we divided the dataset into smaller cross-validation sets. For each set and problem type (DME or DR), we divided the original data into training and validation parts in the ratio of 4:1 (80% training, 20% validation), while keeping the relative frequencies of the different classes the same. With the latter, we tried to mitigate the distribution shift between the training and validation set. Furthermore, there was no overlap between the cross-validation sets, meaning that the validation sets were disjoint. This meant that if an image appeared in the validation set of one cross-validation set, then it could no longer appear in the validation set of any subsequent cross-validation set. As for the classes themselves, we made sure that all annotations followed the same grading procedure. The manual annotations of the images were made by ophthalmologists according to the International Clinical Diabetic Retinopathy (ICDR) disease severity scale [74] for all three of the original image datasets (Messidor [54], Kaggle [53], and IDRiD [52] datasets) that we used. The ICDR scale is one of the most commonly used clinical scales and consists of a 5-point grade for DR: no, mild, moderate, severe, and proliferative.

We thoroughly searched the optimal hyperparameters for each algorithm. This search included looking for the optimal batch size, optimizer, learning rate, and the number of epochs. The hyperparameter search was naturally carried out purely on the training set. In the case of optimizers, we experimented with Stochastic Gradient Descent (SGD) [35] (both with and without momentum) and Adam [37] and looked for the one that made the loss decrease in the smoothest way possible, with minimal oscillations and continuous decreases over the epochs. For the batch size, we looked for the value that made the learning process the most stable and led to the smallest oscillation in the training loss. In the case of the learning rate, we used learning rate scheduling and examined the change of the loss. Then, we picked the learning rate α that had the lowest observed loss value and had a sufficiently large environment $[\alpha - \varepsilon, \alpha + \varepsilon]$ (where $\varepsilon \in (0, 1)$) in which the loss values did not oscillate and did not increase rapidly. Finally, we chose the number of epochs so that training was terminated when the

validation loss started to continuously increase. We found the batch size of 32 and the Adam optimizer to be the best for every algorithm, while the optimal learning rate and the number of epochs varied from network to network. Tables 7 and 8 summarize the best hyperparameter settings for each network and problem type (DR and DME). These were the ones that we used for evaluating the different algorithms.

Table 7: The list of optimal hyperparameters for the DR problem.

Network	*Type*	*Optimizer*	*Batch size*	*Learning rate*	*Epochs*
AlexNet	original	Adam	32	0.0001	100
MobileNetv2	original	Adam	32	0.001	150
ResNet-50	original	Adam	32	0.001	100
AlexNet	V1	Adam	32	0.0001	100
MobileNetv2	V1	Adam	32	0.001	150
ResNet-50	V1	Adam	32	0.0001	100
AlexNet	V2	Adam	32	0.001	150
MobileNetv2	V2	Adam	32	0.001	200
ResNet-50	V2	Adam	32	0.001	150
AlexNet	V3	Adam	32	0.0001	150
MobileNetv2	V3	Adam	32	0.0003	200
ResNet-50	V3	Adam	32	0.0001	100

All of the algorithms were evaluated on the test part of the IDRiD dataset. Each image $x^{(i)}$ had its corresponding DR and DME labels, as well as the hand-crafted features associated with the given image. As the test set that we had access to was heavily imbalanced and skewed towards certain classes, we took several metrics into account during the evaluation process to make it fairer. Additionally, since a standard accuracy value would give biased results in our case, we used a weighted average of the accuracies calculated at the class level, where each weight represented how many times a given class occurred in the test set. Accordingly, we calculated a number of metrics regarding the performance of the algorithms and their ability to differentiate healthy and non-healthy images, such as positive predictive value (PPV), sensitivity (SE), specificity (SP), F_1-score, and weighted accuracy (ACC_w). To calculate these, we considered the following quantities for every prediction: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). In our case TP meant

Table 8: The list of optimal hyperparameters for the DME problem.

Network	*Type*	*Optimizer*	*Batch size*	*Learning rate*	*Epochs*
AlexNet	original	Adam	32	0.0001	150
MobileNetv2	original	Adam	32	0.0001	200
ResNet-50	original	Adam	32	0.001	200
AlexNet	V1	Adam	32	0.0003	150
MobileNetv2	V1	Adam	32	0.0001	200
ResNet-50	V1	Adam	32	0.001	200
AlexNet	V2	Adam	32	0.001	150
MobileNetv2	V2	Adam	32	0.001	200
ResNet-50	V2	Adam	32	0.001	150
AlexNet	V3	Adam	32	0.0003	200
MobileNetv2	V3	Adam	32	0.0003	200
ResNet-50	V3	Adam	32	0.001	200

that both the prediction and the ground truth said the given image $x^{(i)}$ belonged to a non-healthy specimen, while TN meant the same but with healthy specimens. The number of FPs indicated how many times the algorithm predicted the non-healthy label, while the ground truth was healthy, and the number of FNs showed the exact opposite. The exact formulas of the used metrics are given as

$$PPV = \frac{TP}{TP + FP}, \tag{15}$$

$$SE = \frac{TP}{TP + FN}, \quad SP = \frac{TN}{TN + FP}, \tag{16}$$

$$F_1\text{-score} = \frac{2TP}{2TP + FP + FN}, \tag{17}$$

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}, \tag{18}$$

$$ACC_w = \sum_{c \in classes} w_c * ACC(c), \tag{19}$$

where w_c is the ratio of the samples in the class c to the total number of samples.

The experiments were carried out on a computer with an Nvidia RTX 3080 video card and a total of 128 GB RAM. The codebase was written purely in Python and we used the PyTorch framework for writing the experiment code.

4.7 Experimental results

As noted in section 4.6, we used a variety of metrics to reliably evaluate the performance of the different algorithms, which is why we have included multiple tables for each problem type (DR and DME). Furthermore, as previously discussed, we divided the dataset into cross-validation sets. Consequently, all the tables in this section show the results of the different algorithms measured after being trained on these cross-validation sets. The results shown are calculated at 95% confidence levels and are sorted by network and algorithm type. For the original networks, we refer to as *original*, while the proposed architectures are noted as V1, V2, and V3, respectively. We would also like to note that since the ResNet-50 architecture only had a total of one fully connected layer, the V3 version of the network is fully equivalent to the V1 version. Therefore, the tables presented contain the same results for these two versions of the ResNet-50 architecture.

We have also trained several baselines that only used the hand-crafted features $y_2^{(i)}$ but did not use the input images to measure the predictive capabilities of the hand-crafted features alone. We have considered three different approaches for this task. In the first approach, we optimized directly for the best weights θ to calculate $\bar{y}^{(i)} = \theta^\top y_2^{(i)}$, then, we used the softmax function to calculate the predicted class probabilities $\hat{y}^{(i)}$. We will refer to this approach as Softmax in the subsequent tables. In the second and third approaches, we used a support vector machine (SVM) and a neural network (MLP) respectively to calculate $\hat{y}^{(i)}$. For the SVM, we used the Radial Basis Function (RBF) kernel, while for the MLP, we used two hidden layers with 64 and 32 units respectively. Furthermore, we have also considered another set of architectures from [75] as additional baselines to compare our results with. We will refer to this last approach as LightCNN in all the subsequent tables while also specifying the type of architecture used as discussed in the original paper [75] by abbreviating random forest as RF and decision tree as DT.

4.7.1 Diabetic retinopathy

First, we measured the ability of the networks to differentiate between healthy and non-healthy images as a starting point. While the networks that used only the hand-crafted features performed surprisingly well, they achieved substantially worse results in terms of SE and F_1-score than the original networks that received only the images as inputs. In other words, – according to the formulas (15) and (16) – the number of FNs was substantially greater than the number of FPs for the networks that only used the hand-crafted features. This increase in FNs resulted in the model misclassifying numerous ill specimens as healthy. The architectures proposed in [75] achieved really good results similar to that of the original ResNet-50 model but were still outperformed by the networks that used the hand-crafted features. While all of the original networks were able to more or less determine these classes without the hand-crafted features, as can be seen in Table 9, the usage of hand-crafted features improved the performance greatly in all cases. The improvements were the most substantial for networks derived from the AlexNet architecture, while the best-performing algorithms were the V2 versions of the two newer networks.

Table 9: A summary of the metrics measured on the DR dataset.

Network	*Type*	PPV	SE	SP	F_1-score
LightCNN [75]	SVM	0.905 ± 0.044	0.804 ± 0.128	0.824 ± 0.115	0.849 ± 0.052
LightCNN [75]	RF	0.858 ± 0.017	0.826 ± 0.142	0.721 ± 0.086	0.840 ± 0.066
LightCNN [75]	MLP	0.909 ± 0.038	0.790 ± 0.071	0.838 ± 0.087	0.845 ± 0.025
LightCNN [75]	DT	0.801 ± 0.003	0.848 ± 0.043	0.574 ± 0.029	0.824 ± 0.019
Hand-crafted	Softmax	0.780 ± 0.012	0.616 ± 0.043	0.647 ± 0.001	0.688 ± 0.031
Hand-crafted	SVM	0.793 ± 0.042	0.638 ± 0.001	0.662 ± 0.086	0.707 ± 0.017
Hand-crafted	MLP	0.888 ± 0.017	0.746 ± 0.014	0.809 ± 0.029	0.811 ± 0.015
AlexNet	original	0.746 ± 0.053	0.783 ± 0.142	0.412 ± 0.173	0.774 ± 0.016
MobileNetv3	original	0.820 ± 0.017	0.855 ± 0.028	0.544 ± 0.086	0.828 ± 0.012
ResNet-50	original	0.908 ± 0.108	0.797 ± 0.085	0.824 ± 0.231	0.846 ± 0.001
AlexNet	V1	0.741 ± 0.005	0.833 ± 0.014	0.662 ± 0.087	0.800 ± 0.021
MobileNetv3	V1	0.893 ± 0.054	0.841 ± 0.000	0.794 ± 0.115	0.866 ± 0.025
ResNet-50	V1	0.932 ± 0.029	0.783 ± 0.028	0.882 ± 0.058	0.850 ± 0.005
AlexNet	V2	0.880 ± 0.030	0.783 ± 0.028	0.794 ± 0.058	0.809 ± 0.001
MobileNetv3	V2	0.876 ± 0.056	**0.877 ± 0.099**	0.750 ± 0.140	**0.873 ± 0.019**
ResNet-50	V2	**0.938 ± 0.013**	0.797 ± 0.028	**0.897 ± 0.029**	0.840 ± 0.021
AlexNet	V3	0.835 ± 0.013	0.848 ± 0.071	0.691 ± 0.029	0.824 ± 0.024
MobileNetv3	V3	0.924 ± 0.047	0.783 ± 0.000	0.868 ± 0.087	0.847 ± 0.020
ResNet-50	V3	0.932 ± 0.029	0.783 ± 0.028	0.882 ± 0.058	0.850 ± 0.005

Since all of the networks were able to differentiate between healthy and non-healthy specimens, we measured their performance with regard to their accuracies per class. These class-level accuracies can be seen in Table 10.

Table 10: A summary of the class-level accuracies measured on the DR dataset.

Network	*Type*	*DR0*	*DR1*	*DR2*	*DR3*	*DR4*
LightCNN [75]	SVM	0.811 ± 0.048	0.951 ± 0.001	0.651 ± 0.057	0.835 ± 0.019	0.859 ± 0.029
LightCNN [75]	RF	0.791 ± 0.067	0.918 ± 0.048	0.675 ± 0.086	0.835 ± 0.019	0.869 ± 0.010
LightCNN [75]	MLP	0.806 ± 0.019	0.947 ± 0.010	0.694 ± 0.029	0.835 ± 0.057	0.874 ± 0.019
LightCNN [75]	DT	0.757 ± 0.019	0.850 ± 0.048	0.617 ± 0.029	0.777 ± 0.019	0.835 ± 0.001
Hand-crafted	Softmax	0.626 ± 0.029	0.951 ± 0.001	0.452 ± 0.010	0.816 ± 0.001	**0.874 ± 0.001**
Hand-crafted	SVM	0.646 ± 0.029	0.951 ± 0.001	0.495 ± 0.095	0.840 ± 0.010	**0.874 ± 0.001**
Hand-crafted	MLP	0.767 ± 0.019	0.951 ± 0.001	0.612 ± 0.019	0.835 ± 0.001	0.864 ± 0.001
AlexNet	original	0.695 ± 0.028	0.947 ± 0.009	0.534 ± 0.020	0.753 ± 0.105	0.762 ± 0.086
MobileNetv3	original	0.763 ± 0.028	0.937 ± 0.028	0.631 ± 0.057	0.840 ± 0.028	0.830 ± 0.010
ResNet-50	original	0.806 ± 0.020	0.942 ± 0.019	0.709 ± 0.095	**0.884 ± 0.019**	0.855 ± 0.019
AlexNet	V1	0.777 ± 0.038	0.951 ± 0.000	0.549 ± 0.085	0.840 ± 0.028	0.850 ± 0.009
MobileNetv3	V1	0.796 ± 0.020	0.947 ± 0.009	0.641 ± 0.019	0.850 ± 0.028	0.869 ± 0.010
ResNet-50	V1	0.806 ± 0.020	0.947 ± 0.009	0.728 ± 0.114	0.864 ± 0.020	0.859 ± 0.010
AlexNet	V2	0.738 ± 0.020	0.947 ± 0.009	0.622 ± 0.019	0.869 ± 0.028	0.753 ± 0.008
MobileNetv3	V2	**0.830 ± 0.010**	**0.956 ± 0.010**	0.612 ± 0.095	0.821 ± 0.028	0.845 ± 0.019
ResNet-50	V2	0.806 ± 0.020	**0.956 ± 0.010**	0.675 ± 0.048	0.879 ± 0.028	0.788 ± 0.002
AlexNet	V3	0.758 ± 0.019	0.932 ± 0.037	0.622 ± 0.038	0.826 ± 0.019	0.731 ± 0.008
MobileNetv3	V3	0.811 ± 0.028	0.937 ± 0.028	**0.743 ± 0.028**	0.869 ± 0.010	0.831 ± 0.028
ResNet-50	V3	0.806 ± 0.020	0.947 ± 0.009	0.728 ± 0.114	0.864 ± 0.020	0.859 ± 0.010

The networks that used only the hand-crafted features performed surprisingly well yet again for some classes (DR1, DR3 and DR4) but performed really poorly for other classes (DR2). The reason behind their remarkably good results for DR4 was that they did not predict DR4 for any given sample; meaning that they only predicted 0 values for this class. Since the total number of DR4 specimens was really low compared to other classes, this resulted in only a handful of DR4 cases, leading to a great number of 0 values and only a few 1 values in the ground truth. This, combined with the fact that the networks predicted 0 values for each sample, resulted in a high accuracy value for this class. The architectures proposed in [75] delivered much better results, surpassing several networks that only used the input images but they were still outperformed by the networks that used both the hand-crafted features and image inputs. For these networks, while the same ones (the V2 version of MobileNetv2 and Resnet-50) seemed to perform the best for DR0 and DR1 as in Table 9, for the other classes,

there were better performing alternatives. Three out of five of these used hand-crafted features with DR4 being an exception due to its low cardinality, and even in the case of DR3 and DR4, the difference between the best and second-best network (which used the hand-crafted features) was negligible (0.005 and 0.005). On the other hand, in the case of the other classes, the difference between networks without and with the hand-crafted features was quite substantial, in favor of the latter. This demonstrates again that using the hand-crafted features is beneficial for any of these networks and can drastically increase the accuracy.

To get a general overview of how the different networks performed on the test set, we also measured their weighted accuracies. For this purpose, we calculated the accuracies on the class level and then weighted the results depending on the number of samples per class in the test set. The results are enclosed in Table 11.

Table 11: The weighted accuracies measured on the DR dataset.

Network	*Type*	ACC_w
LightCNN [75]	SVM	0.778 ± 0.002
LightCNN [75]	RF	0.779 ± 0.007
LightCNN [75]	MLP	0.792 ± 0.016
LightCNN [75]	DT	0.731 ± 0.014
Hand-crafted	Softmax	0.654 ± 0.012
Hand-crafted	SVM	0.678 ± 0.041
Hand-crafted	MLP	0.752 ± 0.001
AlexNet	original	0.687 ± 0.011
MobileNetv2	original	0.771 ± 0.022
ResNet-50	original	0.807 ± 0.030
AlexNet	V1	0.723 ± 0.025
MobileNetv2	V1	0.799 ± 0.007
ResNet-50	V1	0.805 ± 0.048
AlexNet	V2	0.725 ± 0.004
MobileNetv2	V2	0.763 ± 0.021
ResNet-50	V2	0.788 ± 0.002
AlexNet	V3	0.731 ± 0.008
MobileNetv2	V3	0.803 ± 0.034
ResNet-50	V3	0.805 ± 0.048

It can be seen that although the networks that only used the hand-crafted features performed well for some classes as we noted for Table

10, the overall accuracy of these networks was substantially lower than that of the networks that used the input images as well. The architectures proposed in [75] performed considerably better but were yet again outperformed by some of the networks that used both the hand-crafted features and the input images. We can once again see that by using the hand-crafted features, the overall accuracy has greatly improved for all of the networks. The best-performing networks were once again the ones based on the MobileNetv2 and Resnet-50 architectures, but even the ones using AlexNet benefitted greatly from the usage of the hand-crafted features.

4.7.2 Diabetic macular edema

For DME, we followed the exact same steps as outlined previously in section 4.7.1. First, we measured how reliably the various algorithms could differentiate the healthy specimens from the non-healthy ones using the PPV, SE, SP, and F_1-score metrics. As it can be seen in Table 12, all the networks were able to achieve this task.

Table 12: A summary of the metrics measured on the DME dataset.

Network	*Type*	PPV	SE	SP	F_1-score
LightCNN [75]	SVM	0.913 ± 0.001	0.724 ± 0.001	0.911 ± 0.001	0.808 ± 0.001
LightCNN [75]	RF	0.897 ± 0.002	0.750 ± 0.017	0.889 ± 0.001	0.817 ± 0.011
LightCNN [75]	MLP	0.922 ± 0.020	0.707 ± 0.001	0.922 ± 0.022	0.800 ± 0.008
LightCNN [75]	DT	0.898 ± 0.04	0.759 ± 0.034	0.889 ± 0.044	0.822 ± 0.037
Hand-crafted	Softmax	0.786 ± 0.027	0.285 ± 0.017	0.900 ± 0.022	0.418 ± 0.015
Hand-crafted	SVM	0.781 ± 0.142	0.595 ± 0.017	0.778 ± 0.174	0.674 ± 0.064
Hand-crafted	MLP	0.859 ± 0.003	0.733 ± 0.017	0.844 ± 0.001	0.791 ± 0.011
AlexNet	original	0.815 ± 0.023	0.681 ± 0.085	0.789 ± 0.022	0.757 ± 0.048
MobileNetv2	original	0.848 ± 0.007	0.785 ± 0.017	0.811 ± 0.022	0.831 ± 0.019
ResNet-50	original	0.938 ± 0.001	0.785 ± 0.017	**0.956 ± 0.044**	0.855 ± 0.011
AlexNet	V1	0.851 ± 0.078	0.724 ± 0.034	0.833 ± 0.109	0.782 ± 0.013
MobileNetv2	V1	0.900 ± 0.039	0.810 ± 0.034	0.889 ± 0.044	0.833 ± 0.036
ResNet-50	V1	0.940 ± 0.002	**0.836 ± 0.017**	0.922 ± 0.022	0.878 ± 0.006
AlexNet	V2	0.897 ± 0.068	0.724 ± 0.068	0.889 ± 0.087	0.806 ± 0.003
MobileNetv2	V2	0.859 ± 0.109	0.750 ± 0.017	0.845 ± 0.131	0.787 ± 0.069
ResNet-50	V2	0.832 ± 0.010	0.724 ± 0.034	0.811 ± 0.022	0.774 ± 0.015
AlexNet	V3	0.797 ± 0.014	0.802 ± 0.017	0.745 ± 0.022	0.786 ± 0.007
MobileNetv2	V3	**0.951 ± 0.019**	**0.836 ± 0.017**	0.945 ± 0.022	**0.890 ± 0.018**
ResNet-50	V3	0.940 ± 0.002	**0.836 ± 0.017**	0.922 ± 0.022	0.878 ± 0.006

The networks that only used the hand-crafted features had the exact same problem that we discussed previously: their SE scores

were substantially lower than that of the other networks, except for the network that used the two-layer neural network (MLP). As for the architectures proposed in [75], they achieved really good results, surpassing most (but not all) of the original networks but were still outperformed by the networks that used both the hand-crafted features and the input images. As for AlexNet, MobileNetv2 and ResNet-50, comparing the solutions that did not use the hand-crafted features with those that did, it is clearly visible that the results of the latter were substantially better. The network with the most notable and drastic improvements was MobileNetv2, which turned out to be the best-performing solution regarding these metrics.

Next, we measured how accurately the networks could classify the specimens into different classes. In the case of DME, there were a total of three classes: DME0, DME1, and DME2. Table 13 shows the measured class-level accuracies.

Table 13: A summary of the class-level accuracies measured on the DME dataset.

Network	*Type*	*DME0*	*DME1*	*DME2*
LightCNN [75]	SVM	0.806 ± 0.001	0.816 ± 0.019	0.777 ± 0.057
LightCNN [75]	RF	0.811 ± 0.010	0.806 ± 0.001	0.772 ± 0.086
LightCNN [75]	MLP	0.801 ± 0.010	0.816 ± 0.019	0.782 ± 0.067
LightCNN [75]	DT	0.816 ± 0.038	0.825 ± 0.001	0.786 ± 0.057
Hand-crafted	Softmax	0.553 ± 0.001	0.903 ± 0.001	0.563 ± 0.019
Hand-crafted	SVM	0.675 ± 0.085	0.903 ± 0.001	0.675 ± 0.085
Hand-crafted	MLP	0.782 ± 0.010	0.893 ± 0.001	0.752 ± 0.010
AlexNet	original	0.753 ± 0.028	0.859 ± 0.047	0.801 ± 0.029
MobileNetv2	original	0.816 ± 0.019	0.879 ± 0.009	**0.869 ± 0.010**
ResNet-50	original	0.850 ± 0.009	0.874 ± 0.038	0.840 ± 0.047
AlexNet	V1	0.772 ± 0.028	0.893 ± 0.020	0.762 ± 0.047
MobileNetv2	V1	0.826 ± 0.038	0.893 ± 0.020	0.840 ± 0.047
ResNet-50	V1	0.869 ± 0.010	**0.913 ± 0.019**	**0.869 ± 0.010**
AlexNet	V2	0.801 ± 0.010	0.908 ± 0.010	0.797 ± 0.038
MobileNetv2	V2	0.777 ± 0.076	0.893 ± 0.020	0.782 ± 0.105
ResNet-50	V2	0.762 ± 0.010	0.884 ± 0.019	0.748 ± 0.038
AlexNet	V3	0.762 ± 0.010	0.898 ± 0.029	0.719 ± 0.038
MobileNetv2	V3	**0.884 ± 0.019**	0.888 ± 0.010	0.864 ± 0.037
ResNet-50	V3	0.869 ± 0.010	**0.913 ± 0.019**	**0.869 ± 0.010**

Among the networks that only used the hand-crafted features, the Softmax and SVM versions performed substantially worse than the other networks in the case of DME0 and DME2. The version that used the MLP as its backbone performed remarkably well, almost surpassing the original AlexNet network in all aspects but was outperformed by both the other original networks and the ones that used the hand-crafted features as well as the input images. The LightCNN variants performed quite uniformly, meaning that the accuracy values were almost the same for each architecture but were now outperformed by not only the networks that used the hand-crafted features and the input images but also by the original networks as well. It can be also seen that the networks using the hand-crafted features performed substantially better than the networks that did not use these features. Only the MobileNetv2 architecture was a slight exception since it achieved good results for the DME2 class. However, the results of this network without the hand-crafted features were significantly lower in the case of the other classes. The best-performing network was the ResNet-50 network and its V1 and V3 versions, which achieved top 1 scores for two out of the three classes.

Finally, we measured the weighted accuracies (ACC_w) of the networks on the test set. As it can be seen in Table 14, we can once again confirm that the usage of the hand-crafted features led to significant improvements. The networks that only used the hand-crafted features performed significantly worse yet again. The only exception was the MLP variant which performed almost comparably to the AlexNet original network. The results of the LightCNN architectures were once again really uniform but they were outperformed by both the majority of the original networks and the ones that used both the hand-crafted features and the input images. The best-performing networks were the MobileNetv2 (V3) and ResNet-50 (V1 and V3) architectures, which corresponds to Tables 12 and 13 and other metrics as well.

Table 14: The weighted accuracies measured on the DME dataset.

Network	*Type*	ACC_w
LightCNN [75]	SVM	0.793 ± 0.029
LightCNN [75]	RF	0.792 ± 0.044
LightCNN [75]	MLP	0.793 ± 0.029
LightCNN [75]	DT	0.803 ± 0.043
Hand-crafted	Softmax	0.592 ± 0.009
Hand-crafted	SVM	0.697 ± 0.077
Hand-crafted	MLP	0.779 ± 0.009
AlexNet	original	0.783 ± 0.026
MobileNetv2	original	0.847 ± 0.012
ResNet-50	original	0.852 ± 0.022
AlexNet	V1	0.779 ± 0.037
MobileNetv2	V1	0.837 ± 0.043
ResNet-50	V1	0.874 ± 0.007
AlexNet	V2	0.809 ± 0.024
MobileNetv2	V2	0.789 ± 0.082
ResNet-50	V2	0.764 ± 0.018
AlexNet	V3	0.754 ± 0.028
MobileNetv2	V3	0.875 ± 0.011
ResNet-50	V3	0.874 ± 0.007

4.8 Conclusions

In this chapter, we presented a novel deep learning-based framework for the accurate and reliable detection of diabetes-related eye diseases, such as diabetic macular edema and diabetic retinopathy using digital fundus images. We emphasized the importance of the hand-crafted features, which, with the rise of AI, less and less research focus on and have shown that they may still hold value from the perspective of the overall performance of any CAD system. We showed that, by using the proposed framework, it is possible to combine the hand-crafted features with features extracted by convolutional neural networks.

We introduced several variants of the framework. We showed that all variants use the main idea of concatenating the hand-crafted and CNN features and then performing a transformation on them. The main difference between them is how they achieve this. The first variant that we presented simply fed the concatenated feature vector through a single weight and then applied the softmax function to get the outputs, while the second and third variants used more elaborate techniques. For the second variant, we separated the computational graph into two distinct paths: one that processes only the hand-crafted features and one that processes those extracted by the neural network. We noted that this separation may not always be optimal, as it totally isolates these features. To overcome this issue, we proposed the third variant, where both kinds of features are processed at the same time, making it possible for the framework to detect more complex patterns.

We have shown that each of the proposed versions has advantages and disadvantages, which makes choosing the best architecture a really difficult task. We measured the performance of all of these algorithms, as well as their original networks on our test set. We also compared these results to several baselines that only used the hand-crafted features as well as different architectures proposed in [75]. For this comparison, a variety of metrics were used to make the evaluation more precise and fair. We have shown that all of the algorithms that used the hand-crafted features achieved substantially better results than the networks that did not use them and outperformed the aforementioned baselines and architectures as well. We concluded that while the overall performance of the different versions of the proposed algorithm were close to each other, for our problem the V3 version of MobileNetv2 and the V1 version of ResNet 50 architectures performed the best.

5 Fully convolutional ensemble model for automatic cell segmentation

5.1 Introduction

This chapter presents a method for building accurate and reliable ensemble models that could be used as a part of an automatic screening system for cell segmentation. The proposed method utilizes a fully convolutional neural network (FCN) [76] architecture to build an ensemble model that, contrary to traditional ensembles, can automatically change the importance and weighting of each member model on its own. Our final goal during this research was the early detection of cervical cancer using digitized Pap smear images. Therefore, we carried out multiple experiments geared towards the proper and accurate segmentation of cells in an attempt to improve the overall reliability of the system. During the development of this segmentation subsystem, we have considered state-of-the-art machine learning methodologies to reach high segmentation accuracy.

The presented ensemble approach uses the segmentation predictions produced by other fully convolutional networks in addition to the original scanned image as its input. During training, the architecture therefore has the ability to derive features and patterns both in the input image and in the predictions of the member models. Based on these, it can automatically decide what weight to assign to each member. The main benefit of our approach is that the ensemble can not only aggregate the results of the member models, but also correct them for an input image where the prediction masks are not deemed correct. We show that our proposed method outperforms several other state-of-the-art segmentation algorithms and produces good quality segmentation masks that could be incorporated into our screening system.

The structure of this chapter is as follows. In section 5.2, we explain our motivations behind building the proposed architecture, focusing on why there may be a need for considering the outputs of multiple networks for a given image. For this, we show that aggregating the outputs of several base networks is not trivial and what disadvantages traditional ensemble methods may have. Then, in section 5.3, we give a detailed overview of the dataset used during our research, which was organized and manually annotated at the University of Debrecen. In

section 5.4, we introduce our proposed framework. In section 5.5, we include an overview of our experimental setup, while in section 5.6 we detail our results. Finally, in section 5.7, we provide our conclusions and list our most important contributions.

The ensemble architecture introduced in this chapter and all the corresponding experimental results were published in [30]. The newly created dataset established during our experiments was submitted for publication in [31].

5.2 Motivation

At the beginning of the 21st century, a few automated screening systems are available with limited applicability in the field of cytology. They are based on computer image analysis techniques for screening and try to exclude the surely negative samples from the consequent investigation procedure to decrease the number of specimens. To the best of our knowledge, the most widely used automatic solutions as of now are the Hologic ThinPrep Imaging System [77] and the Focal Point Slide Profiler [78]. These have been approved by the U.S. Food and Drug Administration (FDA) and operate in high-volume reference laboratories under human supervision. Unfortunately, unlike our system, the Hologic ThinPrep Imaging System can only analyze ThinPrep Pap Test slides which have much higher costs than the most commonly applied Papanicolaou smear test [79]. The other solution, the Focal Point Slide Profiler also has some notable drawbacks, as it can eliminate only up to 25% of the lowest-risk slides to allow the cytologists to focus on the highest-risk slides. Our automatic system could overcome the limitations of these two solutions, as it could process the most commonly applied Pap smear test images and could also rank the slides by the level of risk more accurately. The official procedure of taking the Papanicolaou smear test begins by opening the vaginal canal with a speculum and collecting cells at the outer opening of the cervix. After that, the collected cells are fixed on the glass slide and this specimen is placed into a large capacity whole slide scanner which can digitize it by generating a digital color image with a resolution of $100\,000 \times 220\,000$ pixels. The resulting image can contain more than $10\,000$ cells at a $40\times$ magnification as it can be seen in Figure 7.

Our final goal during our research was the creation of a fully auto-

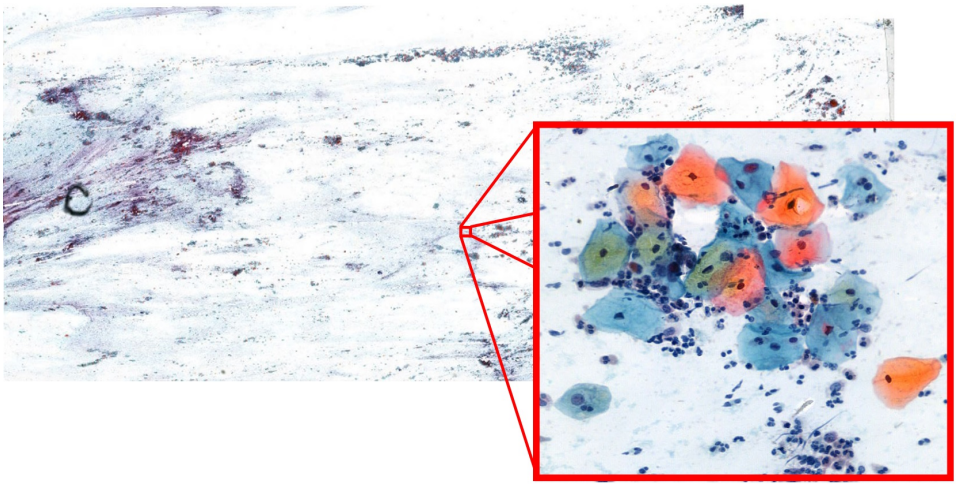

Figure 7: A sample scanned image about cytological specimen, where the red box shows cells with 40× magnification.

mated system that can recognize cancerous cells in digitized cytological specimens with sufficiently high reliability and is suitable for clinical applications. The final software is aimed to be able to automatically rank the smear images, thus allowing the practicing clinicians to always focus on patients with the most severe conditions, making the treatment process faster and more efficient. By using a similar automatic screening system, the governments could save money and human efforts besides the developments of its outpatient services. Our automatic screening system aims to localize and segment each cell from this high-resolution image with high sensitivity and specificity. The workflow of this type of system can be formed by the following steps: the proper segmentation of the individual cells; a pre-trained deep learning-based algorithm that classifies all of the segmented cells as healthy or unhealthy. In the case that any of the cells is considered pathologically diseased, the whole test will be investigated by a cytologist, as well. The high sensitivity is crucial because of the mortality of the Human Papillomavirus (HPV) and other cervical cancers. Based on the statistics, 2-3 million abnormal Pap smear results are found each year [80] in the United States.

5.3 Datasets

For our experiments, we constructed a new dataset in a collaboration with the Clinical Center of the University of Debrecen, Department of Pathology. The manual annotation of the cytological smears was carried out by a team of three annotators coordinated by a team leader. The concrete annotation work was preceded by an IT and cytology training. During the latter, annotators were trained by a medical expert (cytopathologist). Our dataset contains digital images derived from scanned results of the Pap smear tests. Every entry in the annotated dataset is thus made up of two elements: the scanned image $x^{(i)}$ and the manual segmentation $y^{(i)}$ corresponding to it (see Figure 8).

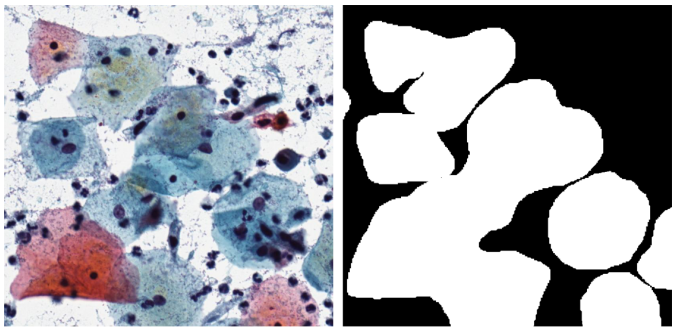

Figure 8: A sample input image (left) and its corresponding ground truth (right).

Pap test slides were digitized by a laboratory analyst trained for this task by the scanner vendor using a 3DHistech Pannoramic 1000 digital slide scanner [81] equipped with an Adimec Q-12A-180Fc brightfield camera. The laboratory analyst has properly prepared and cleaned the slides before digitizing them. The scanner was equipped with 3 different objectives (20× Plan-Apochromat, 40× Plan-Apochromat, and 40×w C-Apochromat) [81] , resulting in pixel sizes 0.25 µm, 0.12 µm, and 0.12 µm, respectively. A total magnification of 200× (20× objective with 10× eyepiece) was used during the scan. We have chosen this specific setting as a 200× total magnification is commonly used in digital pathology for scanning because it balances resolution and field of view, making it suitable for various diagnostic purposes. The digitized slides were saved in MRXS format, a proprietary WSI format with multiple resolution levels for faster navigation. The dimensions of the acquired digital slides are approximately 100 000 × 200 000 pixels,

and the size of a digitized slide is approximately 5 GB using JPEG compression.

To extract fields of view (FOVs) from the digital slides, the region of interest was first identified using the intensity features of the slide at the lowest resolution level. Then, non-overlapping FOVs with 2 000× 2 000 pixels were extracted from the 200× magnification level (approximately 0.25 µm/pixels) of the slide as PNG files using the libvips [82] library. This resolution and FOV size allow for examining the patterns at the cellular level, the context of a cell, and the arrangement of cells. The scanning analyst recorded and anonymized the non-personal clinical data (age, Bethesda classification scoring category, smear serial number) required for subsequent processing.

Due to the time-consuming nature of the manual annotation process, the creation of the dataset required a long time. Hence, each of our experiments could only use the most recent version of the dataset that was available and validated at the time of the research. The dataset introduced in [30] contained a total of 3 157 images with the size of 500×500 pixels, which were divided into three parts. We trained the FCN algorithms on the first part, our proposed combined network on the second one, and evaluated their respective performances on the third part. The three parts consisted of 1 284, 416 and 557 images, respectively, of size 500 × 500 pixels. This way, we have avoided excessive training of the combined network on the same data that the individual FCN algorithms were trained on.

In [31], we introduced the finalized and most comprehensive version of our dataset of 3 565 images, each containing 2 000×2 000 pixels. Image slices from 3 smears were included in the training data set (2 227 images), and image slices from 2 smears made up the test data set (1 338 images). Considering the number of cells, the training dataset contains circa 30 000 cells, while the test dataset contains nearly 7 000 cells. Therefore, the full downloadable dataset published in [31] includes a training and a test part as seen in Table 15. We have split the images into a training and a test set to ensure appropriate tools for training and evaluating the algorithms for the automated segmentation task. This clear splitting aims to be used in any further development related to the field for an official evaluation as a benchmark. Here, we retained 1 505 images as a private set, which can be used later to organize an international challenge to make a reliable evaluation and performance measurement for the participants.

	Training part	Test part
Number of different Pap smears	3	2
Number of cells	30 000	7 000
Inputs (RGB images)	2 227	1 338
Ground Truths (binary masks)	2 227	1 338

Table 15: Details of the dataset published in [31].

5.4 The fusion-based ensemble architecture

In this section, we introduce our fusion-based fully convolutional network denoted as Ens_{Fusion} for cell segmentation. Instead of taking solely the image $x^{(i)}$ as the input, the network receives both the original three-channel (RGB) image and the outputs of other FCN [76] algorithms, denoted as $M_{FCN\text{-}32}$, $M_{FCN\text{-}16}$, and $M_{FCN\text{-}8}$ for the FCN-32, FCN-16, and FCN-8 models, respectively. The reason behind this lies in our observation that there are many cases when even though the different FCN algorithms provided very different and disjoint outputs, these outputs could still be aggregated in such a way that the combined result would have had a higher accuracy. Such a common scenario occurred when the various algorithms found their distinct group of cells on a given smear image. This phenomenon can be observed in Figure 9.

Figure 9: (a) a sample input image, and the outputs of the (b) FCN-32, (c) FCN-16, and (d) FCN-8 algorithms.

It can easily be seen that a standard aggregation model (e.g., majority voting, statistical combination) would have problems in cases where each algorithm finds different parts of the cells or cell groups. This is because traditional ensemble models do not have the ability to dynamically put more emphasis on their member models $M_1, ..., M_n$ depending on the input image $x^{(i)}$. Therefore, we aim to provide an

efficient solution for the combination of the segmentation outputs, or in other words, we propose a fusion-based ensemble system Ens_{Fusion}.

Claim 5.1. *Using the traditional ensembles Ens_M and Ens_W leads to sub-optimal results if the segmentation masks $M_1(x^{(i)}), ..., M_n(x^{(i)})$ produced by the member models M_1, ..., M_n contain non-overlapping areas for a given input image $x^{(i)}$.*

Reasoning. Let Ens_M and Ens_W denote traditional ensembles using majority voting and weighted averaging, respectively, constructed of member models $M_1, M_2, ..., M_n$. If the outputs do not overlap, that means that $M_j(x^{(i)})_{px,py} \neq M_k(x^{(i)})_{px,py}$ for each model $j, k = 1, ..., n$, $j \neq k$ and pixel at the (px, py) coordinates.

In the case of majority voting, following (4), the output of the ensemble becomes zero (the default value) for each pixel at the (px, py) coordinates, as there are no overlapping parts (i.e., parts where a majority of the models "agree on"). This means that each segment is predicted only once, thus no majority of the outputs can be defined.

It is also easy to see that in this special case there is no merit in applying weighted averaging. If the outputs do not overlap, then for each pixel at the coordinates (px, py) $M_j(x^{(i)})_{px,py} > 0.5$ will only be true if $M_k(x^{(i)})_{px,py} < 0.5$ for each model $k = 1, ..., n, j \neq k$. This will result in

$$\begin{aligned} Ens_W(x^{(i)})_{px,py} &= \beta_1 M_1(x^{(i)})_{px,py} + \beta_2 M_2(x^{(i)})_{px,py} + ... \\ &\quad + \beta_n M_n(x^{(i)})_{px,py} \\ &= \frac{M_1(x^{(i)})_{px,py}}{n} + \frac{M_2(x^{(i)})_{px,py}}{n} + ... + \frac{M_n(x^{(i)})_{px,py}}{n} \\ &= 0 + 0 + ... + \frac{M_j(x^{(i)})_{px,py}}{n} + ... + 0 \\ &= \frac{M_j(x^{(i)})_{px,py}}{n}. \end{aligned} \tag{20}$$

Since each model can only predict the presence of a given class for any given pixel with a maximum probability of 1, (20) will be zero if $n > 2$. If $n = 2$, then the output will be the same as the output of the M_j model if $M_j(x^{(i)})_{px,py} = 1$, otherwise it will be zero. This is also undesired, as in this case, the ensemble model loses all of its advantages compared to the single model M_j.

□

Let us denote the given input image as x (i.e., omit the index) for the sake of brevity. In this case, the member algorithms assign probabilities $M_{FCN\text{-}i}(x_{px,py}) \in [0,1]$ (where $i \in \{8, 16, 32\}$) to each $x_{px,py}$ pixel of the input image x to indicate their confidence whether the given pixel belongs to a cell (plasm or nucleus) or not. Instead of using pixel-wise combination after thresholding, like an element-wise multiplication or majority voting, we consider region-based combination. Consequently, our method uses the information gathered by the individual segmentation algorithms, while still being able to make its own decision by involving the original image as well. To achieve this, first, we train some FCN algorithms, namely the $M_{FCN\text{-}32}$, $M_{FCN\text{-}16}$ and $M_{FCN\text{-}8}$ networks. As pointed out in [76], these algorithms differ in their main architectures and the amount of upsampling they use. The FCN-8 architecture is based on AlexNet [70], while the other two use the architecture of the VGG-16 [83] network. The numbers in the names of these models represent the amount of upsampling used for each respective network.

After training the FCN networks, we combine their outputs. To avoid using only their (sometimes) improper segmentation results, we also use the original image as input. In this way, the final ensemble model could consider the original input image and the segmentation results together and learn how it should combine them to reach the most accurate output. This was achieved by implementing Ens_{Fusion} as a regular FCN-32 architecture, where we increased the number of input channels. Thus both the outputs of the FCN algorithms and the input image can pass through each convolutional and transposed convolutional layer at the same time. This ensures region-based combination by using convolutional operators instead of pixel-wise ones.

Definition 5.1. *Given the models $M_{FCN\text{-}32}$, $M_{FCN\text{-}16}$, and $M_{FCN\text{-}8}$, we define the concatenated matrix C as*

$$C =< x^{(i)}, M_{FCN\text{-}32}(x^{(i)}), M_{FCN\text{-}16}(x^{(i)}), M_{FCN\text{-}8}(x^{(i)}) >, \qquad (21)$$

where each element $C_{px,py}$ of the matrix contains the input image and the outputs of each FCN model, concatenated into a single vector, and where $x^{(i)}_{px,py} \in [0,1] \times [0,1] \times [0,1]$ is a given pixel of the input image, treated as a 3D vector that contains the normalized intensity values of the original input image regarding the red, green and blue channels.

C is provided as an input to the ensemble Ens_{Fusion} (see Figure 10) which results in the required region-based combination by applying the

appropriate convolutional filters with weights found during the second stage of the training.

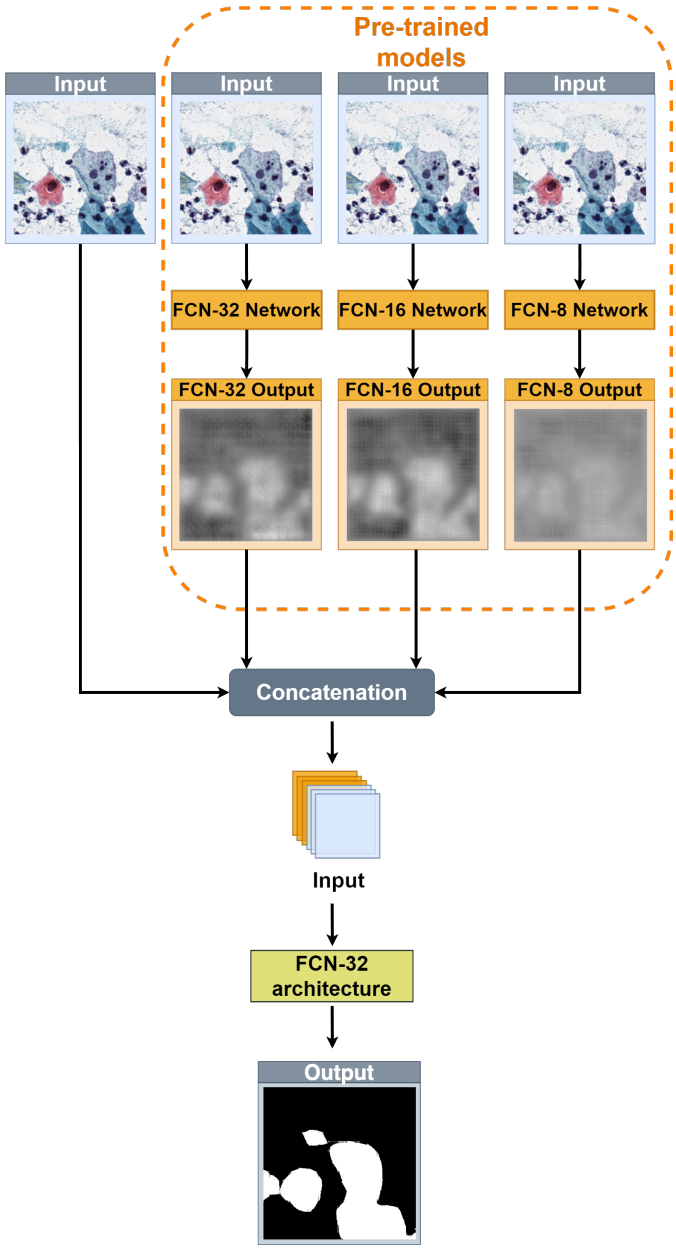

Figure 10: Our fusion-based system receives the heatmaps of the individual pre-trained FCN algorithms and the original RGB image as input (top) to generate the segmented output image (bottom).

Definition 5.2. *Given the models $M_{FCN\text{-}32}$, $M_{FCN\text{-}16}$, and $M_{FCN\text{-}8}$ trained on a given dataset $\mathcal{D}$ and their corresponding weights $\theta_{FCN\text{-}32}$, $\theta_{FCN\text{-}16}$, and $\theta_{FCN\text{-}8}$, respectively, we define the output of the fusion-based ensemble model Ens_{Fusion} with weights $\theta_1, ..., \theta_k$ and activation functions $\sigma_1, ..., \sigma_k$ as*

$$Ens_{Fusion}(x^{(i)}) := \sigma_k({\theta_k}^\top ... \sigma_2({\theta_2}^\top \sigma_1({\theta_1}^\top C))), \tag{22}$$

where the concatenated matrix C is calculated according to (21) for the given input image $x^{(i)}$.

It can be seen how our algorithm can receive the outputs of multiple pre-trained models with the original input image to determine the final segmentation output, meaning that based on the formulation of the concatenated matrix C, multiple different variants of the architecture can be derived. In [30], we showcased the results of several actual implementations of the proposed algorithm. One difference between these is the number of pre-trained models that the variants use to construct C. Our reasoning for trying out multiple variants is that we wanted to experiment with minimizing the amount of extra information that is being given to the model and how this reduction affects performance. Consequently, we use the abbreviation Ens_{Fusion} to refer to our combined network with some suffixes, which denote the extra inputs used. For example, $Ens_{Fusion\text{-}16\text{-}8}$ means that the combined network receives the input image, as well as the outputs of the trained FCN-16 and FCN-8 algorithms, respectively.

5.5 Experimental setup

To evaluate the trained networks, we used the indicators true positive (TP), false positive (FP), true negative (TN) and false negative (FN) that were previously introduced in section 4.6. These indicators were this time calculated at the pixel level. Based on their values, we have calculated the accuracy (ACC) following (18), as well as the intersection over union (IoU) and dice score (DSC) metrics, which were calculated as

$$IoU = \frac{TP}{TP + FP + FN}, \tag{23}$$

$$DSC = \frac{2TP}{2TP + FP + FN}. \tag{24}$$

During the experiments, our dataset had a total of 3 157 images with a size of 500 × 500 pixels, which were grouped into training, validation and test sets, with 1 284, 416, and 557 images, respectively. To evaluate the algorithms, we used cross-validation, during which we shuffled the previously mentioned three parts of the data around so that we could evaluate the performance of the networks on different test sets. We systematically searched for the optimal hyperparameters for each algorithm: both the baselines and the proposed one. During this process, we explored a number of different combinations and ranges by running several manual experiments and noted the best-performing variations. For the batch size, we could only use a maximum of 4 due to hardware limitations as we carrioud out the experiments on a computer with a GTX 1070 graphics card. In the case of our proposed model, we found that using lower learning rates enabled us to reduce the oscillation of the learning loss and make the learning more stable. For learning rate, the oscillation became noticably smaller under 0.001, and in the end, 0.0001 produced the best results, which we used to train the proposed models. To compensate for this low learning rate, we had to increase the number of epochs. We found that 200 epochs worked best for our experiments. For the other parameters, such as stride, padding, etc. we mostly used the ones recommended by [76]. The parameters of the first convolutional layer were however changed to compensate for the bigger input images.

5.6 Experimental results

After carefully tuning the hyperparameters of each architecture and splitting up the dataset as described in section 5.5, we evaluated and recorded the performance of each model following the methodology described in section 5.5. Table 16 shows the overall results at 95% confidence level. It can be seen that the combined networks yielded better results than any of the member FCN architectures which were used originally as a cell segmentation algorithm in [89]. They also outperformed other state-of-the-art re-implemented methods based on [84] and [86] for the most important metrics of segmentation, namely IoU and DSC. For example, a 5% improvement can be observed when compared to even the best-performing FCN algorithm (FCN-8) for IoU and 4% for DSC. The reason for our focus on these two metrics is that the manually annotated dataset used to train the algorithms contained

Table 16: Results on the test dataset.

Algorithm	ACC	IoU	DSC
FCN-32 [76]	0.915 ± 0.054	0.497 ± 0.161	0.660 ± 0.144
FCN-16 [76]	0.913 ± 0.063	0.503 ± 0.143	0.666 ± 0.127
FCN-8 [76]	0.919 ± 0.037	0.507 ± 0.180	0.668 ± 0.158
sota [84]	0.775	0.343	
Ens_1 [85]	0.923 ± 0.022	0.534 ± 0.239	0.688 ± 0.205
Ens_2 [85]	0.923 ± 0.020	0.534 ± 0.243	0.688 ± 0.208
DeepLabv3[86]	0.889 ± 0.117	0.487 ± 0.039	0.655 ± 0.035
U-Net [87]	0.917 ± 0.063	0.504 ± 0.224	0.662 ± 0.199
GSCNN [88]	0.909 ± 0.091	0.514 ± 0.185	0.674 ± 0.162
$Ens_{Fusion\text{-}32\text{-}8}$	0.926 ± 0.034	0.530 ± 0.195	0.688 ± 0.168
$Ens_{Fusion\text{-}32\text{-}16}$	0.928 ± 0.036	0.534 ± 0.191	0.691 ± 0.163
$Ens_{Fusion\text{-}16\text{-}8}$	**0.928 ± 0.031**	**0.537 ± 0.203**	**0.693 ± 0.173**
$Ens_{Fusion\text{-}32\text{-}16\text{-}8}$	0.927 ± 0.040	0.531 ± 0.175	0.687 ± 0.147

some imperfect annotations, meaning that in some cases the outlines of the cells were bigger and rougher than needed. Consequently, there were cases when the algorithms produced even more accurate masks than the ground truth, resulting in a higher number of FNs instead of TNs when comparing them. This phenomenon motivated us to focus on measures that do not take into account the TNs. It is also imperative to note that as our test set contained 557 images of size 500×500, the total number of pixels in the test set was 139 250 000. By this logic, e.g., in terms of accuracy even a 1% improvement leads to an increase of 1 392 500 correctly labeled pixels. It is also important to note that the accuracies of the baseline models were all well over 90%, making these improvements even more substantial and valuable (since in this range even tiny improvements require a lot of effort, engineering and extra computation). Some comparisons regarding the outputs of the standard FCN models and our proposed architecture can be seen in Figure 11.

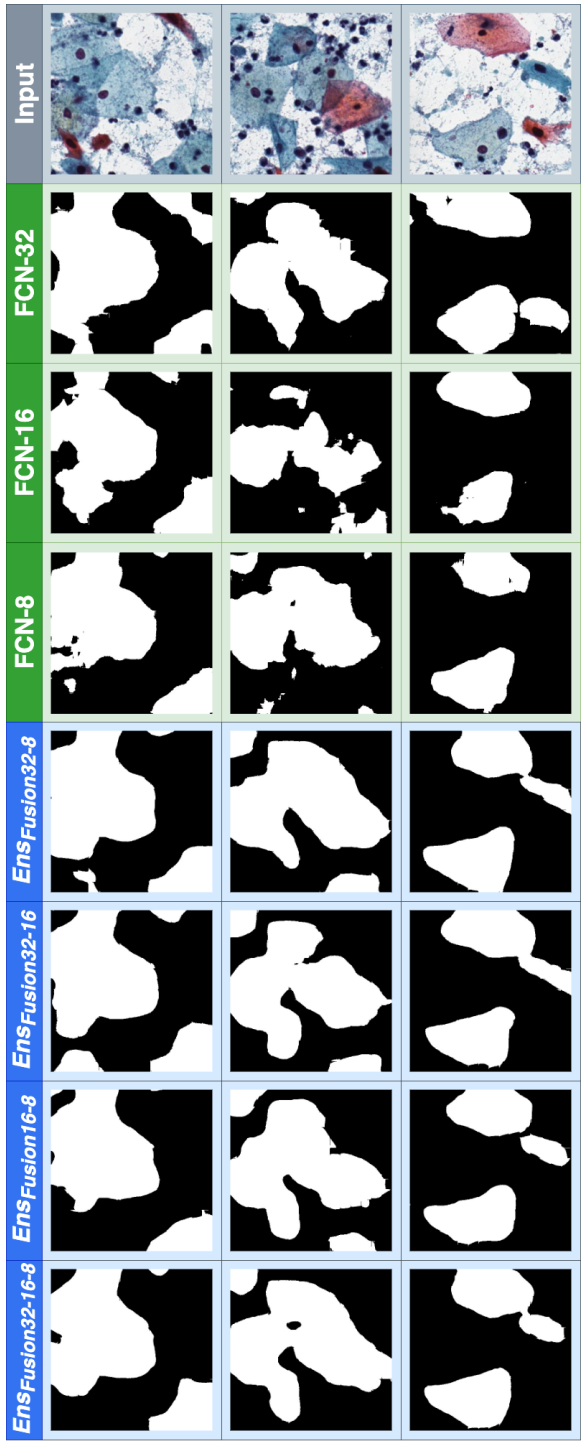

Figure 11: A comparison between the performances of the individual FCN algorithms and the combined architectures.

5.7 Conclusions

In this chapter, we have proposed an ensemble-based cell segmentation approach to combine multiple trained fully convolutional networks to obtain a model that exceeds the performances of all of these individual models. We worked on the problem of segmentation of cells on digitized Pap smear images, and compared the results of the traditional FCN algorithms with several different implementations of our proposed method. We also introduced our own dataset developed and constructed as a foundation for our research. We emphasized that there are currently no other cell segmentation datasets that we are aware of with these many specimens and images that are publicly available.

Regarding the proposed architecture, we showed that we can concatenate the input image and the outputs of some pre-trained FCN networks and use our proposed method as an ensemble model that can not only aggregate the outputs of the member models but also override their predictions. By doing so, the ensemble can derive its own conclusions to further improve the segmentation accuracy. We introduced multiple variants of our solution, and used different combinations of pre-trained FCNs. We showed that any combination proved to be more accurate than any of these standard algorithms and yielded better segmentation results. Moreover, we also noted that giving both the outputs of these FCN algorithms and the input to the combined networks can be a usable solution to achieve higher segmentation accuracy. Using our proposed method not only can the model combine the different outputs, but it can also come up with its own decisions on how to combine the different segments of these outputs (e.g., link them assuming that cells are connecting them), thus further improving the preciseness of the model. This improvement could be seen from the increase in the number of cells found and from the more accurate extraction of the cells when compared to the traditional methods.

6 Building diverse ensemble models by penalizing the similarity between the member models

6.1 Introduction

This chapter presents a novel method for constructing diverse ensemble models. We first give an overview of traditional ensemble solutions, such as majority voting and weighted averaging, and highlight their biggest drawbacks. Then, we introduce our proposed framework, which, contrary to traditional ensemble methods, directly measures the diversity of the member models during the training process. We show that the presented framework can be used for a wide range of tasks due to its flexible and generic nature. We also showcase that using the proposed method, we can re-use the exact same neural network architecture and build diverse ensembles that surpass the performance of the original architectures.

We show that the main idea of our method is to treat the features extracted by the convolutional layers of each CNN as a low-dimensional vector representation of the original input image. Then, we can measure the similarity between the generated latent vectors and force them to be dissimilar. By doing so, we can sufficiently isolate the vector spaces of each model that generate these vectors, resulting in models that generate diverse feature vectors and hence use different sets of features to solve the given task. In this chapter, we also present our experimental results for detecting some of the most common types of brain tumors (meningioma, glioma, and pituitary tumors). Moreover, we introduce two distinct similarity measures that can be integrated into the framework, such as cosine similarity and histogram loss.

This chapter is structured as follows. In section 6.2, we showcase the biggest drawbacks of traditional ensemble models and introduce the main idea behind our proposed framework. In section 6.3, we include an overview of the dataset that we used during our research. In section 6.4, we present our proposed frameworks, highlighting their advantages and disadvantages from a practical and theoretical aspect. In section 6.5, we detail our experimental setup, and in section 6.6, we present our experimental results. Lastly, in section 6.7, we provide our conclusions and list our contributions.

The preliminary research results for the architecture were published in [33]. The final research material and results were submitted for publication in [34].

6.2 Motivation

Deep learning and CNN-based approaches provided very efficient solutions for several challenging and crucial tasks in the medical workflows, like in breast cancer [90, 91] and brain tumor [92, 93] detection and also in automatically identifying skin cancer [94]. Nevertheless, the performance of these deep learning-based techniques is rather volatile; it may vary from application to application. We can easily experience that a specific model is very efficient in one task while its performance falls back in another scenario. We can fuse the different models relying on various architectures into an ensemble as a remedy for such cases. In this way, we can raise the overall performance since member models with weak performance for a given input can be compensated with others performing better there. Naturally, we can expect improvement only if the better-performing models are in a majority within the ensemble. To address this issue, there is strong ongoing research on organizing the individual member models into an efficient ensemble [95, 96, 97]. Applying ensemble-based systems has numerous advantages in the clinical domain. A major benefit is that they can increase the overall stability and performance of the model since their statistically more robust internal mechanisms make them more resistant to outliers or irregularities in the inputs. This feature leads to a more efficient solution and better generalization characteristics.

This chapter presents a novel ensemble-based method for the reliable and accurate classification of brain tumors from T1-weighted contrast-enhanced MRI images. The improved methods we introduce can benefit routine healthcare by reducing the time needed to diagnose patients and raising the quality of the clinical workflow. Our approach can be taken into consideration to integrate it in a (semi-)automated CAD system to provide better healthcare services, reduce the burden on clinical experts, accelerate routine jobs, and, perhaps most importantly, enable the detection of brain tumors at an early stage. For building the ensemble models, we use some of the most commonly used state-of-the-art CNN architectures as the member models of the ensembles, such as AlexNet [98, 99], MobileNetv2 [72, 100, 101], Effi-

cientNet [102, 103], and ShuffleNet v2 [101, 104].

We present multiple versions of our framework: one that uses the cosine similarity and another that uses the histogram loss [32]. We also show that the latter is a natural and logical extension of our original framework with a theoretically better-founded formalization. We also describe the theoretical uncertainty when using the cosine similarity function to compare the features extracted by the ensemble members used in our previous framework. To this end, we present a novel solution using the histogram loss that further extends our previously proposed framework while being more robust and theoretically better founded. We also expand on using a weighted loss function and show its effect on our framework. We conclude that using such a weighted loss function does not hinder the optimization of diversity and show that our framework can still converge to a local minimum.

Cancer poses a serious threat in our modern society, affecting a significant portion of the population. Recent research has shown that it affects even the younger population [105] and that the incidence and mortality rates increase drastically with aging [105]. The same phenomenon – the increased risk of being diagnosed with said tumors with age – has been observed in the case of brain or central nervous system (CNS) tumors[106]. Brain tumors can develop when the brain's cells grow irregularly and abnormally, forming a mass inside the patient's skull. This results in various symptoms, including nausea, headaches, seizures, altered mental status, and even death [107], since the skull only has limited space. The work [108] reported approximately 300 000 global cases of brain and CNS cancer in 2019, which, according to [108], meant a total of 94.35% increase since 1990. The Central Brain Tumor Registry of the United States (CBTRUS) [109] reported 84 264 cases of death between 2015 and 2019, where the cause of death was a malignant brain or CNS tumor. CBTRUS also reported [109] an estimated amount of 93 470 new cases of malignant and non-malignant (benign) brain and CNS tumors for 2022, only in the U.S. alone. According to the latest reports, brain and other nervous system-related cancers resulted in more than 18 000 estimated cases of death in the U.S. in 2023 [110], while the number of new cases was estimated to be more than 24 000 [110]. These statistics all highlight the importance of developing tools that could help with the accurate and reliable detection of these tumors.

The problem, however, lies in the fact that these new cases and

the sheer number of patients currently being treated, coupled with the newly diagnosed patients, generate an increasing amount of image data that the limited number of clinicians struggle to handle. An automatic Computer-Aided Diagnosis (CAD) system could help not only process and evaluate this image data but could potentially discover tumors in their earlier stages due to the increased speed of the clinical workflows. To this end, we present two deep-learning-based frameworks that can process MRI images and automatically classify different types of brain tumors with high accuracy and reliability. We primarily focus on three of the most commonly occurring types of brain tumors [109]: meningioma, glioma, and pituitary tumors.

6.3 Dataset

We used the brain tumor dataset published by Jun Cheng [111] for our research to devise solutions capable of automatically detecting and classifying brain tumors. We considered this dataset a good choice for a pragmatic examination and evaluation of the overall performance of our proposed solutions and to test whether they are robust enough for the following reasons. First, the dataset contains more than 3 000 images, which can be sufficient to train deep learning-based solutions that need huge amounts of data. It has also been tested and verified in the clinical setting, and was used as the foundation of much research in the last few years [112, 113, 114, 115, 116]. Moreover, it contains five pre-defined cross-validation splits, making it straightforward to compare our results with that of other research since the data we used for training and testing is the same as what was used in other works. Due to this, we can directly take the metrics reported by other works and use them as baselines during the evaluation phase. The five pre-defined cross-validation splits make drawing statistically significant and valid conclusions easier, making it harder for the evaluation process to be affected or swayed by outliers or models that only perform well on a few of these splits.

The dataset [111] contains a total of 3 064 T1-weighted contrast-enhanced MRI images, which are divided into the previously mentioned five cross-validation splits. The images were obtained from 233 patients who suffered from one of the three brain tumors meningioma, glioma, or pituitary tumor. Although there were some differences in terms of the total number of images for each category, at first glance, it

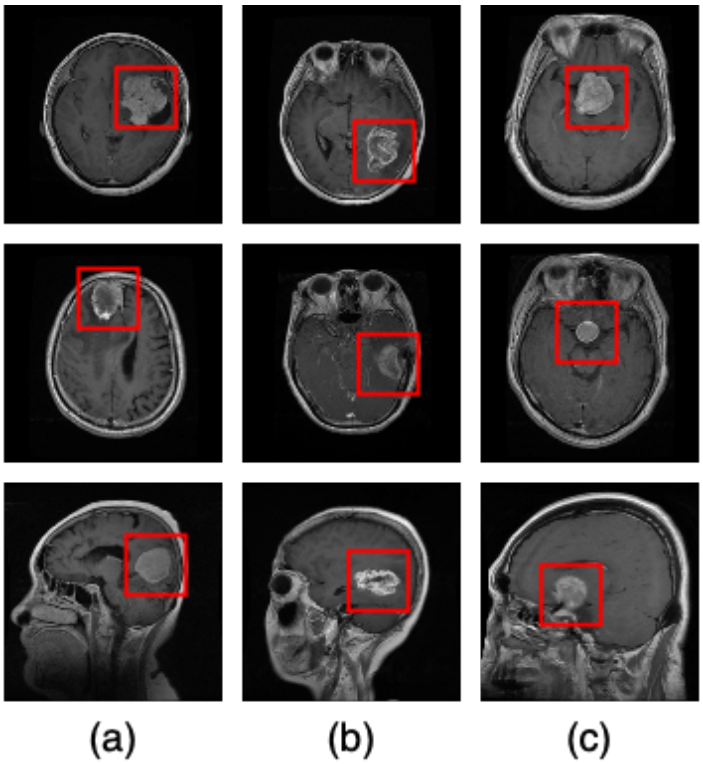

Figure 12: Input images sampled from the three classes present in the dataset: (a) meningioma, (b) glioma, and (c) pituitary tumor. Images in the same column belong to the same class, while the red boxes indicate the locations of the tumors.

did not seem to warrant any up- or downsampling before training the models. Due to this, we first experimented with non-weighted loss functions, but later we expanded our experiments by using a weighted loss function to test the robustness of our proposed framework. All images in the dataset originate from one of the three anatomical planes: the axial, coronal, and sagittal. The exact number of images for each type of tumor and anatomical plane in the dataset can be seen in Table 17, while Figure 12 shows some images sampled randomly from the dataset.

Table 17: The total number of images in the original dataset [111] grouped by (a) tumor type, and (b) anatomical plane.

(a)

Tumor type	Number of images
Meningioma	708
Glioma	1,426
Pituitary tumor	930

(b)

Anatomical plane	Number of images
Axial	994
Coronal	1,025
Sagittal	1,045

6.4 The ensemble framework

In this section, we present the various frameworks for training diverse ensemble models. We start by introducing the cosine similarity-based framework, which we will denote as Ens_{cos}. Then, we showcase some potential disadvantages and scenarios in which Ens_{cos} may perform sub-optimally. Finally, we introduce our extended version of the framework using the histogram loss, which we will denote as Ens_{hist}.

6.4.1 Using the cosine similarity

The framework presented in [33] for training diverse ensembles was based on the cosine similarity function. The core idea of our method is that instead of considering the direct outputs of the member models, we can use the features extracted by the models to measure their similarity. This is a viable option to measure similarity as models operating on different sets of features should behave differently; hence, it is worth combining them in an ensemble. Our proposed framework Ens_{cos}, therefore, uses the features extracted by each model and directly optimizes diversity as a function of similarity among the member models of the ensemble during the training process. This, in practice, means a completely different internal operation compared to regular methods like majority voting or weighted average ensembles. In the case of traditional methods, each member model of the ensemble is trained individually first, without any interaction between them, and the ensemble is only built after all the models have been trained already. In contrast, our method trains all the members simultaneously, where the same inputs are fed to all the different models, batch by batch. This makes it possible for the member models to interact with each other while they are being trained, resulting in highly abstract information, like the similarity of the features extracted, to be able to flow between the members during the training process. In [33], we

showed that our solution can be applied to all CNN-based architectures. We proposed a general method that breaks down any given CNN into two main blocks, denoted by E and D.

Definition 6.1. *For any given CNN model M that uses convolutional and dense layers, we define the forward pass of the model for a given input image $x^{(i)}$ as*

$$M(x^{(i)}) := D(E(x^{(i)})), \tag{25}$$

where E is constructed from convolutional layers and D from dense ones.

In (25), E is responsible for extracting some abstract, high-level features from the input image x_i, while D is responsible for transforming the features extracted by E into probability scores, or in other words, predictions.

We make the following claims and use the following notations to generalize our solution. Let us enclose $n \in \mathbb{N}$ members $M_1, M_2, \ldots, M_n$ in our ensemble with corresponding sets of weights $\theta_1, \theta_2, \ldots, \theta_n$. Let our training set $\mathcal{D}_{train}$ contain a total of m elements, with $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ denoting the inputs (images), and $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$ denoting the ground truth labels. Furthermore, let the cost function to be optimized be denoted as $J(\theta_1, \theta_2, \ldots, \theta_n)$, or $J(\theta)$ for short. Moreover, let E_j and D_j denote the feature extraction (convolutional layers) and prediction (dense layers) steps for each model M_j for $j = 1, \ldots, n$ in our ensemble, respectively. Using these notations, we can declare that E_j is responsible for extracting the most important features from the input image, and then D_j uses these highly abstract features, represented by the extracted feature vector, and apply some non-linear transformations to it to get the predictions. Therefore, we can treat each $E_j(x^{(i)})$ as a compact, low-dimensional representation of the original input image $x^{(i)}$. This representation holds all the crucial information regarding the most important features extracted by M_j about the image $x^{(i)}$ that may be important for the classification procedure. Therefore, if our objective is measuring the similarity between any two models M_j and M_k inside our ensemble, we can determine the similarity between $E_j(x^{(i)})$, and $E_k(x^{(i)})$. These two vectors will be highly similar if the models have extracted similar features and will be dissimilar otherwise.

The main benefit of this solution can be easily seen in the following scenario. Suppose that we are dealing with brain tumor classification, and one of the models only extracts information regarding the tumor's

shape while not considering the texture at all, while the other model only extracts information regarding the tumor's texture but does not consider its shape. Then, we can consider them to work differently, as they operate on different sets of features. Furthermore, their predictions will change independently, resulting in less statistical dependence between the models (i.e., their outputs are not correlated) and an overall more robust ensemble that may resist outliers better.

The last important building block of our proposed solution is the similarity function that measures how similar the encoded latent vectors of the different member models are. For this, we can use any given $\mathcal{S}$ function that is suitable for measuring the similarity between any two arbitrary vectors in a high-dimensional latent vector space. In [33], we focused on the cosine similarity measure, which is not only widely used to compare high-dimensional vectors in natural language processing [117, 118, 119], but very recent research has also shown that it can be used with great success for CNNs as well [120, 121, 122]. We will treat $E_j(x_i)$ and $E_k(x_i)$ as some arbitrary vectors in a latent vector space and use the cosine similarity function to measure the overall similarities between the encoded latent vectors produced by the member models. There are two smaller technical problems with the cosine similarity for our specific use case. The first one is that it cannot directly be built into the cost function J as a simple addition, as the output values can fall anywhere in the $[-1, 1]$ range. The second problem is that the cosine similarity of two vectors at an angle of $180°$ is -1. This is a particularly undesired property in our case, as $E_j(x^{(i)})$ and $-E_j(x^{(i)})$ carry the same information from the perspective of D despite the two vectors pointing to opposing directions.

Definition 6.2. *Given an input image $x^{(i)}$ and two feature vectors u and v extracted by the models M_1 and M_2, respectively, the similarity of the two models is defined as*

$$\mathcal{S}(u, v) = \left(\frac{u \cdot v}{\|u\| \|v\|} \right)^2. \tag{26}$$

By using the similarity function (26), we can eliminate both of the previously mentioned problems: $\mathcal{S}(u, v) \in [0, 1]$ for all vectors u and v, and $\mathcal{S}(u, v) = 1$ if $u = -v$. $\mathcal{S}(u, v)$ will be minimal if the two vectors are orthogonal. For our use case, this is only possible when the members use different sets of features, which inherently means they are dissimilar and make up a diverse ensemble.

Definition 6.3. *Given $E_1, E_2, ..., E_n$ feature extractors, the loss function L_i, and $\lambda \in \mathbb{R}$ the cost function for the framework Ens_{cos} is defined as*

$$J_{cos}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \mathcal{S}(E_j(x^{(i)}), E_k(x^{(i)})) \right]. \quad (27)$$

The first part of (27) focuses on minimizing the original loss function L_i. The second part of the addition is a regularization term that penalizes members that have extracted similar features and hence have constructed similar latent vectors for any input $x^{(i)}$. This second part calculates the similarity between M_j and the other members and adds the sum of the similarities to the loss function multiplied by a $\lambda \in \mathbb{R}$. λ acts as a controlling parameter: smaller λ values result in a weaker regularization effect, while the larger its value is, the stronger the regularization effect becomes. Hence, λ adjusts how much emphasis we put on the diversity of the members during the optimization process. As we will see in section 6.6, λ plays a crucial role in the overall performance of the ensemble. Figure 13 shows an overview of our framework.

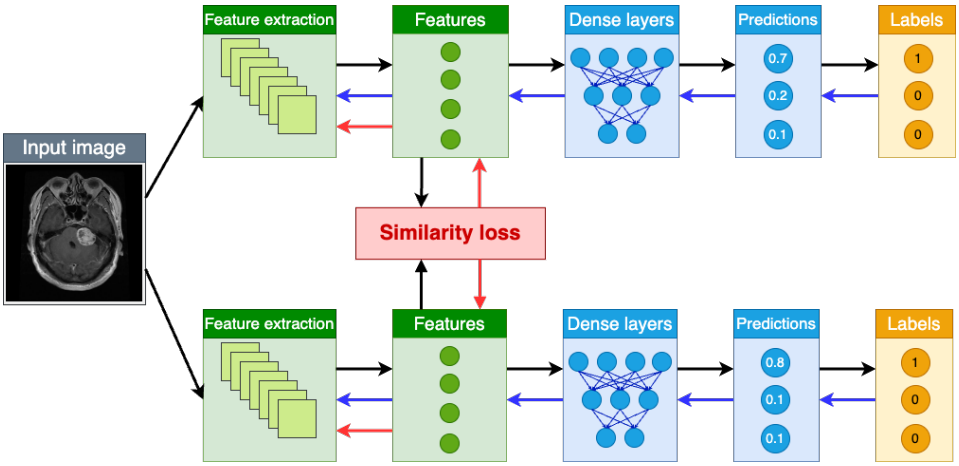

Figure 13: An overview of our framework that calculates the similarity of the members to measure the diversity of the ensemble. The framework considers both the original optimization objective (blue arrows), as well as the similarity loss (red arrows) to update the weights of the members of the ensemble model.

Definition 6.4. *Given the ensemble* $Ens_{cos} = \{M_1, M_2, ..., M_n\}$ *and the input image* $x^{(i)}$*, the output of the ensemble is defined as*

$$Ens_{cos}(x^{(i)}) := majority(M_1(x^{(i)}), M_2(x^{(i)}), ..., M_n(x^{(i)})). \tag{28}$$

Corollary 6.1. *The framework presented in Definitions 6.3 and 6.4 defaults to simple majority voting behaviour when using* $\lambda = 0$*, resulting in an ensemble that does not measure diversity at all.*

Proof. Given $E_1, E_2, ..., E_n$ for the models $M_1, M_2, ..., M_n$, respectively, the loss function L_i, and $\lambda = 0$, the cost function becomes

$$\begin{aligned} J_{cos}(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n} L_i(\theta_j) + 0\sum_{j=1}^{n-1}\sum_{k=j+1}^{n}\mathcal{S}(E_j(x^{(i)}), E_k(x^{(i)}))\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} L_i(\theta_j). \end{aligned} \tag{29}$$

By (29), there is no interaction between the member models during the optimization process, resulting in models trained in complete isolation. Furthermore, there is no metric defined that would measure the diversity of the models. After combining these models into an ensemble $Ens = \{M_1, M_2, ..., M_n\}$, the output of the ensemble will be exactly the same as that of an ensemble using majority voting according to (28). □

Based on Corollary 6.1, we will refer to majority voting as $\lambda = 0$ in the subsequent parts. On the other hand, using $\lambda = 1$ means that diversity is just as important as the original optimization objective, which is to minimize the cross-entropy loss. As we will show later, in some cases using this setting may also hinder the solution's overall performance. This is because diversity should usually be a secondary objective, with the primary objective of the optimization process being the minimization of the cross-entropy loss. The reason is that diversity, while playing an important role in the overall performance of the ensemble, can only lead to better results if the member models can solve the original optimization task. If the members cannot solve that, there is usually no merit in maximizing the diversity between them.

6.4.2 Problems with the cosine similarity

Although [33] showed that the proposed framework performs quite well, there are still some theoretical limitations that, depending on the area where the framework is being applied, may hinder the training process or make the proposed solution work in a sub-optimal fashion. We show that such problems are not only theoretical but may also arise in practice and should be addressed directly. We use the MNIST [123] and Medical MNIST [124] datasets as simple yet expressive examples and show that different models, even when using different architectures, may extract highly similar sets of features and hence work in a really similar way. We chose these specific datasets because they contain enough images to train neural networks and have been used extensively as benchmarks in the literature [125, 126, 127, 128, 129]. Moreover, the Medical MNIST dataset contains medical images with a diverse set of classes (abdomen CT, breast MRI, chest CT, chest X-ray, hand CT, and head CT).

For our first experiments, we used the MNIST [123] and Medical MNIST [130] datasets, as well as a very basic neural network architecture as simple examples to show that the problem of not recognizing permutations in the feature vectors does happen in practice. We used a very basic neural network architecture with only two convolutional and three linear layers (shown in Figure 14) to avoid overfitting to the relatively simple datasets. In the case of the MNIST dataset, we used the original train and test splits proposed in [123] for training and evaluating the models. For the Medical MNIST dataset, there are no such splits defined. Therefore, we split the original dataset into training, validation, and test splits in a 3:1:1 ratio for each class, respectively, resulting in 60% of the original data being used for training, 20% for validating, and 20% for testing purposes.

For the MNIST dataset, each model was trained for 10 epochs. In the case of the Medical MNIST dataset, the models were trained for 5 epochs due to the lower number of classes. We used the standard learning rate of 0.001 for both datasets, an SGD optimizer with a momentum of 0.9, and the cross-entropy loss. Each model converged to a (local) minimum by the end of the training, achieving at least 98% accuracy on the training and test sets for the MNIST, and 99% in the case of the Medical MNIST dataset, respectively. For a more in-depth summary of the results of each model, see Table 18.

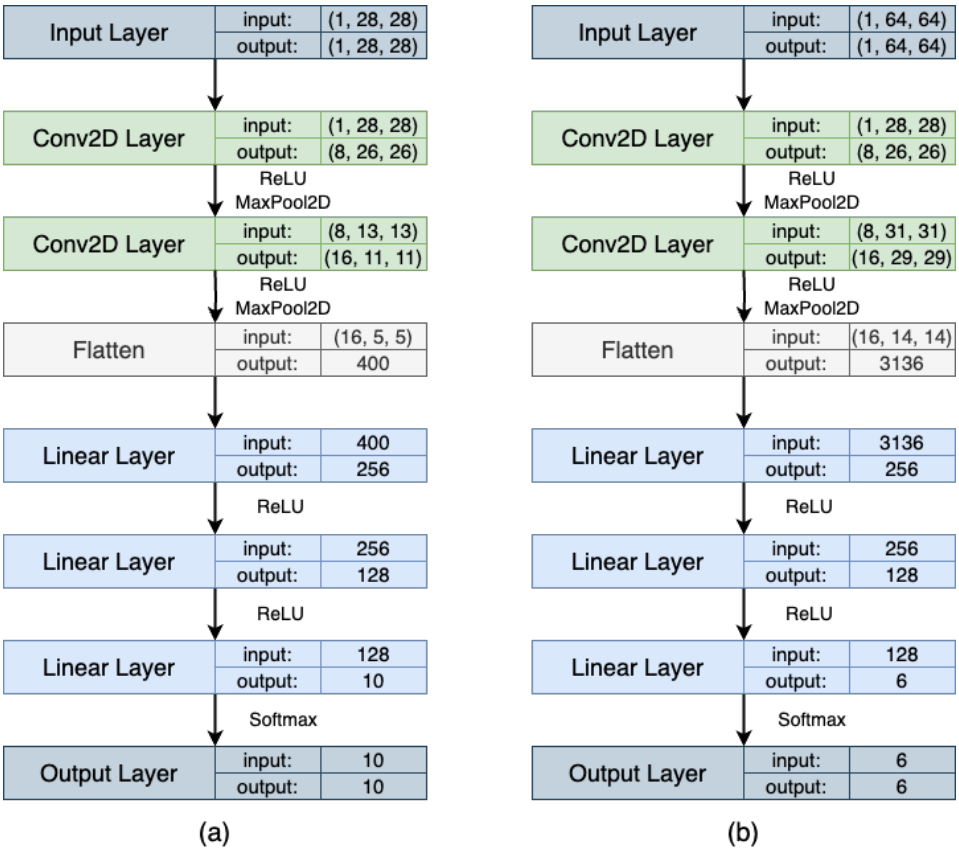

Figure 14: An overview of the simple architecture used in our experiments for the (a) MNIST and (b) Medical MNIST datasets.

Table 18: A summary of the results of the different models trained on the MNIST and Medical MNIST datasets.

Dataset	Model	Loss (train)	ACC (train)	Loss (test)	ACC (test)
MNIST	M_1	0.038	0.988	0.005	0.986
MNIST	M_2	0.037	0.988	0.003	0.989
MNIST	M_3	0.038	0.989	0.037	0.986
MNIST	M_4	0.039	0.988	0.129	0.987
MNIST	M_5	0.039	0.988	0.003	0.987
Medical MNIST	M_1	0.006	0.998	0.000	0.998
Medical MNIST	M_2	0.014	0.996	0.003	1.000
Medical MNIST	M_3	0.012	0.996	0.001	0.998
Medical MNIST	M_4	0.012	0.996	0.001	1.000
Medical MNIST	M_5	0.010	0.997	0.001	1.000

Our goal with this experiment was to see how likely it is for different models sharing the same architecture to rely on highly similar sets of features after being trained and observe how the cosine similarity handles these cases. For this, we trained five different versions of the previously mentioned basic architecture on the MNIST dataset. Each version of the model M_j ($j = 1, \ldots, 5$) was trained independently from any of the other models, using weights θ_j that were randomly initialized. After each model M_j was trained, we compared the outputs of their last convolutional layers on a set of test images, which outputs correspond to $E_j(x^{(i)})$ for our framework. Then, we compared these extracted latent vectors directly by examining their values and comparing their heatmap representations between the different models M_j. Our goal was to observe how random the activations (i.e., the higher values) and their positions inside the latent vectors u_j are in practice.

As seen in Figure 15, the results may seem relatively random and not correlated at first glance. That is, the extracted latent vectors' heatmap representations seem diverse enough. However, when looking at the histogram representations of these same vectors, some slight similarities can be seen, which may indicate similarities between the extracted feature vectors. For example, the histograms of the first, second, and third models and those of the fourth and fifth ones seem similar. If we measure the means and standard deviations of these models (shown in Table 19), then we can see that indeed this is the case; models M_1, M_2, and M_3, as well as models M_4, and M_5 seem to produce feature vectors with highly similar means and standard deviations. The results of this experiment, therefore, seem to support

our previous claim that there may be cases when simply aggregating some already trained networks may result in a sub-optimal ensemble, as there may be some similarities between the member models. This, in turn, could lower the overall robustness and decrease the abilities of the ensemble to generalize well.

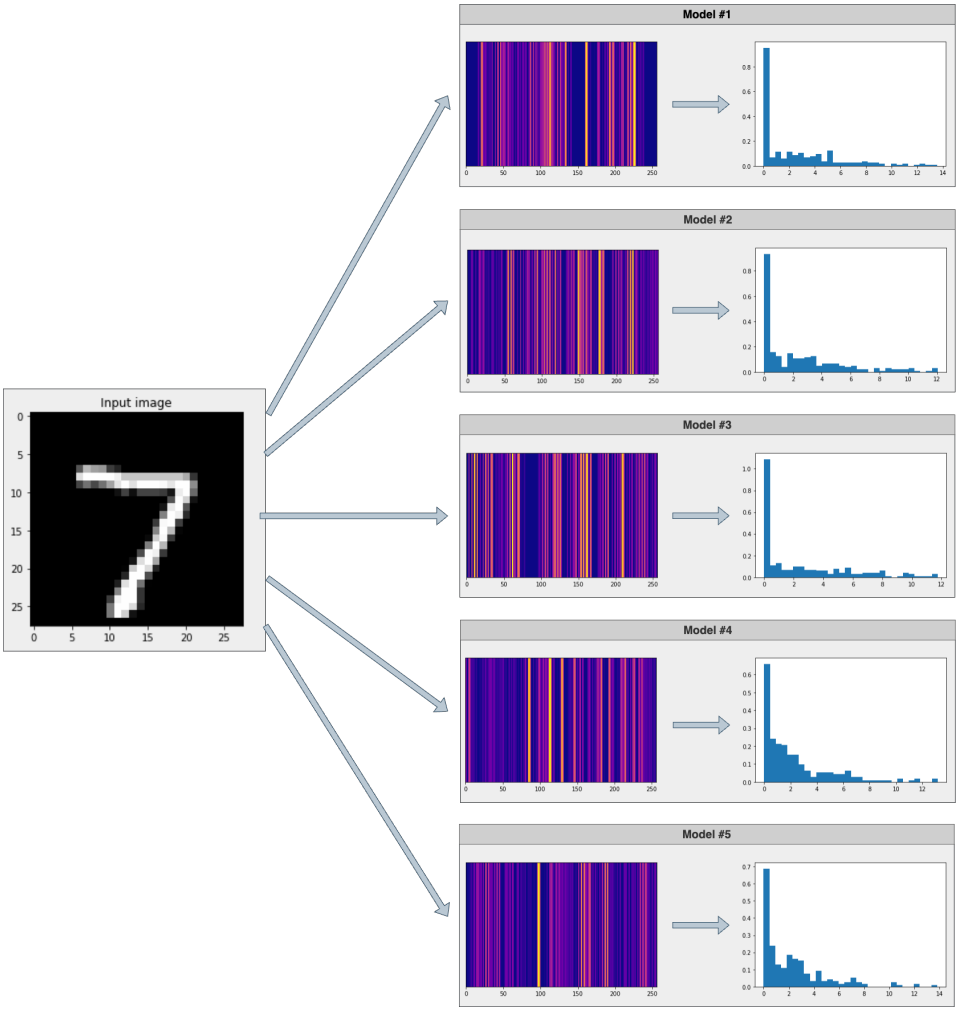

Figure 15: A heatmap representation of the latent vectors (right) generated by each model for a given input image (left).

Even though the cosine similarity has been used with great results both for the area of natural language processing [117, 118, 119] and computer vision as well [120, 121, 122], it inherently has some theoretical drawbacks that may result in sub-optimal performance in some

Table 19: The means and standard deviations of the latent vectors extracted by each model M_j for the test images shown in Figures 15 and 17, respectively.

Dataset	Model	Mean	Std. dev.
MNIST	M_1	2.4837	3.0861
MNIST	M_2	2.4214	2.8950
MNIST	M_3	2.4739	3.0619
MNIST	M_4	2.2324	2.6339
MNIST	M_5	2.2031	2.5594
Medical MNIST	M_1	0.7732	1.4526
Medical MNIST	M_2	0.7141	0.9595
Medical MNIST	M_3	0.8317	0.8939
Medical MNIST	M_4	0.6305	0.6060
Medical MNIST	M_5	0.7386	0.9838

cases using the Ens_{cos} framework. One of the biggest drawbacks of the cosine similarity is its inability to take spatial relationships into account. This is highly problematic for our use case, where we would like to notice any similarities between two latent vectors extracted by different model. In this case, the order of the features is not guaranteed to be the same for every model.

For example, let us take two hypothetical feature vectors u and v. Let $u := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and $v := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. In this setup, the extracted feature vectors coincide; therefore, the similarity metric defined in section 6.4.1 will return

$$\mathcal{S}(u, v) = \left(\frac{u \cdot v}{\|u\| \|v\|} \right)^2 = 1, \tag{30}$$

meaning that the framework recognizes that the latent vectors are the same. However, if we randomly shuffle the elements of any of these vectors, the result will be vastly different. For example, for $u := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and $u' := [9, 4, 8, 10, 1, 6, 7, 3, 5, 2]$, the result will be

$$\mathcal{S}(u, u') = \left(\frac{u \cdot u'}{\|u\| \|u'\|} \right)^2 = 0.4561, \tag{31}$$

which would signal a significantly lower relationship between u and u'. However, that is not the case, as the feature vectors u and u' encode the exact same features, just in different order. [131] has also drawn attention to the possibility of such permutations for CNNs.

For a more in-depth comparison, we ran many experiments examining how much this may affect the calculation of the similarities between the two vectors. During these experiments, we considered vectors with varying dimensions, ranging from 10 to 10 000. This range, according to the current literature [72, 98, 102, 104], seems to cover the dimension of the possibly extracted feature vectors of all of the most commonly used CNNs. Then, we ran multiple tests for each setting, calculating the squared cosine similarity between the original randomly generated latent vector u and its permutated version u'. To observe the variance of the similarities, we repeated these experiments several times, with different repetitions ranging from 10 to 1 000 000. Our motivation behind using different repetitions was that by increasing the sample size, we should get statistically more accurate results. Therefore, by conducting these experiments, we should be able to observe the overall trend indicated by the orange line in Figure 16.

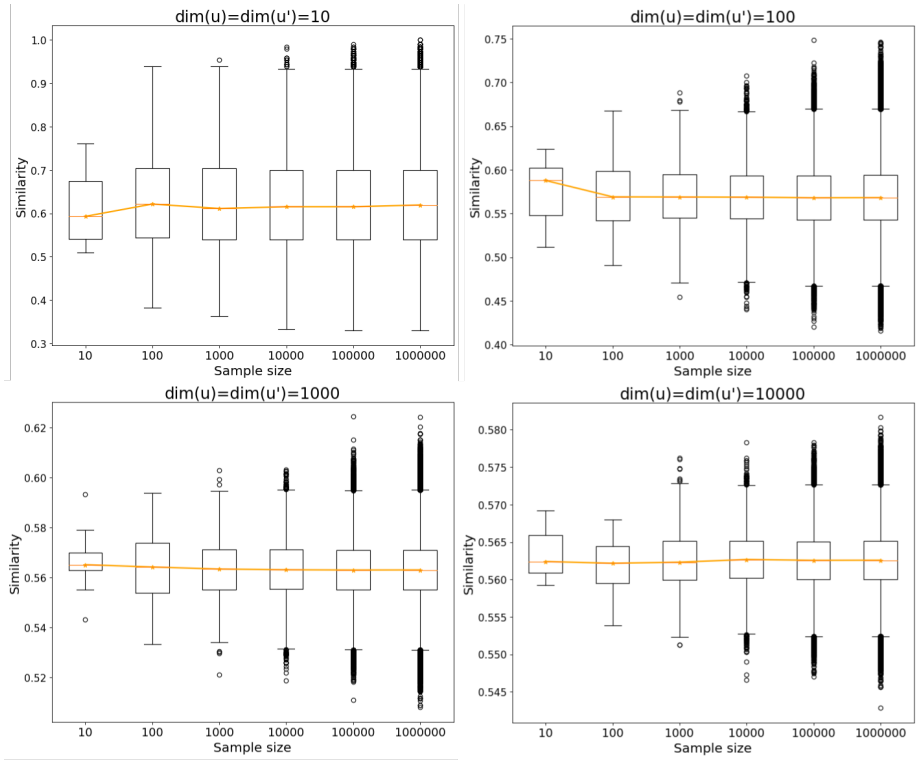

Figure 16: The squared cosine similarities measured between a latent vector u and its randomly shuffled variant u', calculated for various sample sizes (horizontal axis) and varying vector dimensions.

It can be observed that the average cosine similarity for any randomly permutated vector u' and the original vector u tends to be close to 0.6 for low-dimensional feature vectors ($dim(u) = dim(u') < 100$). However, as we increase the dimensions of these feature vectors, the effectiveness of the cosine similarity for recognizing permutated vectors decreases. Current state-of-the-art CNN architectures [72, 98, 102, 104] tend to use feature vectors with dimensions falling between 1 024 and 4 096. In this range (1 000 to 10 000, the bottom row of Figure 16), we can see that usually only roughly 56% of the similarity is being recognized, although both vectors contain the same sets of features. This is highly problematic due to the random nature of the training process of neural networks and the randomized weight initialization procedure. In theory, this randomness can lead to two models that learn to extract the same sets of features in a slightly different order, i.e., the activations are the same, but their order is different. In this case, the framework proposed in [33] will fail to recognize these similarities and will, therefore, not perform optimally. As we have shown, this can lead to simply failing to recognize roughly 50% of the similarities, ultimately leading to an ensemble less diverse than possible.

We show that the problem of failing to recognize the similarity between a vector and its permutated version shown in the first part of this section may also occur in practice. We use again the simple MNIST and Medical MNIST datasets to show that when training multiple models of the same architecture, we may end up with similar, slightly permutated feature vectors. We used the already trained models $M_1, \ldots, M_5$ and extracted their corresponding feature vectors $u_1, \ldots, u_5$ for a test image (see Figure 17).

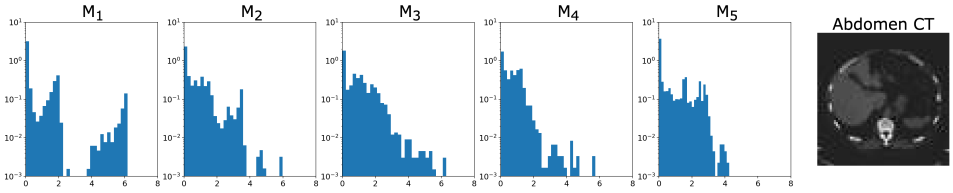

Figure 17: A sample input image from the Medical MNIST dataset (right) and an overview of the distribution of the extracted features by different models (left). The horizontal axis represents the values inside the feature vectors, and the vertical one shows their corresponding frequencies inside the vector on a logarithmic scale.

As the next step, we measured the cosine similarity between the extracted u_j vectors. Even though, in this case, the similarity could be easily seen with the naked eye in Figure 17, the extracted cosine similarity values shown in Figure 18 are really low and fail to capture this. This is highly problematic, as even though we can observe a clear and indistinguishable correlation between the histograms of the u_j vectors, these similarities will be lost when we optimize our cost function J as introduced in section 6.4.1. To solve this issue, we can use a similarity metric that is not sensitive to the order of the elements when comparing the two feature vectors. Earlier, we showed that one intuitive way of measuring similarity is by comparing the distributions of the two latent vectors. One theoretically founded way of measuring this is to use a histogram representation of the similarities and then compare these higher-level representations to penalize vectors that are similar but come from different sources.

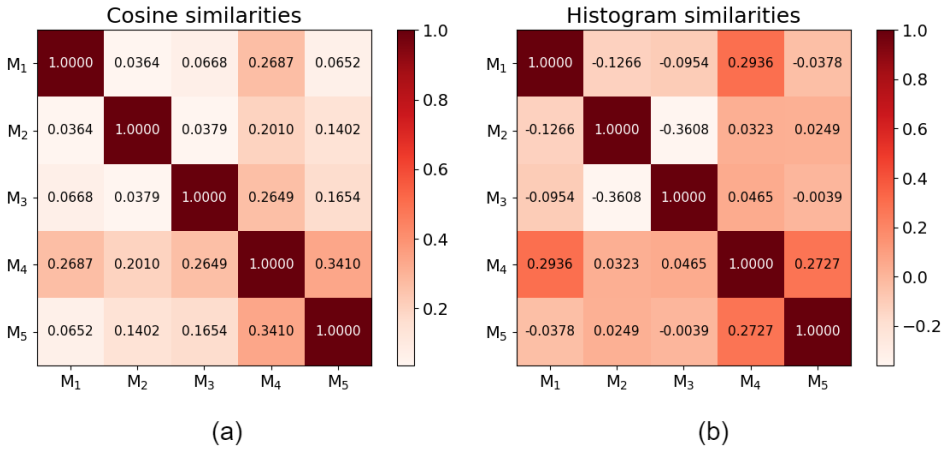

Figure 18: The correlations measured between the (a) cosine, and (b) histogram similarities calculated between the feature vectors extracted by the five models trained on the Medical MNIST dataset for the sample image shown in Figure 17.

There are a variety of possible solutions for measuring this similarity, ranging from calculating the intersection between the histograms, calculating their correlation, or using a chi-square test. The main idea, however, is that a larger overlap between two histograms indicates a higher degree of similarity, while a small overlap means a lower degree of similarity. By measuring using histograms instead of the similari-

ties between the latent feature vectors directly, we lose all information regarding the order of the elements that could negatively affect the similarity calculation, as the improved measure only considers the aggregated, higher-level representation. In Figure 18, we can see that despite having no information about the order of the elements, we can get better and more realistic similarity values for the correlation metric when we only consider the histogram representation of the latent feature vectors. We can also see that using the histogram similarities, we could still retain the most important and striking similarities (e.g., between models M_1 and M_4, or M_4 and M_5) while recognizing that models M_2 and M_4 are not similar (which were deemed as similar by the cosine similarity).

6.4.3 Histogram-based similarity

In section 6.4.2, we have shown that the cosine similarity measure fails to capture some intricate similarities in the following cases: a) when the same feature vector is permutated, and b) when even though the order of the feature vectors is not the same, their distributions are similar. To solve these shortcomings, we proposed a new, improved version of our framework denoted as Ens_{hist} that uses the histogram loss [32] instead of the original cosine similarity when calculating the $\mathcal{S}(u, v)$ similarity between any two latent vectors u and v. We use a modified version of the histogram loss which is differentiable and can be used with neural networks as the base of our solution to measure the similarities between the histogram representations of the extracted latent vectors of the member models in the ensemble. This way, the similarity metric does not rely on the order of the elements of the feature vectors, resulting in an overall more robust and theoretically founded solution.

For the framework Ens_{hist}, instead of measuring the similarities between any two pairs of latent vectors directly, we reformulate our previous problem as measuring the probability $p_{reverse}$ [32] that any two random latent vectors u_j and u_k extracted by two different models M_j and M_k (negative pair) are more similar to each other than latent vectors extracted by the same models (positive pair). We will use this probability as a form of regularization when training the models, penalizing those that produce latent vectors more similar to other models than those produced by the same model. This should, in theory, result

in a more efficient and "tighter" distribution of the latent vectors generated by each model inside the ensemble, since we penalize any overlap between said distributions. Due to the fewer overlaps, we should also be able to reduce the variance of the distributions, as the models are forced to generate latent vectors with more compact and concentrated distributions (see Figure 19).

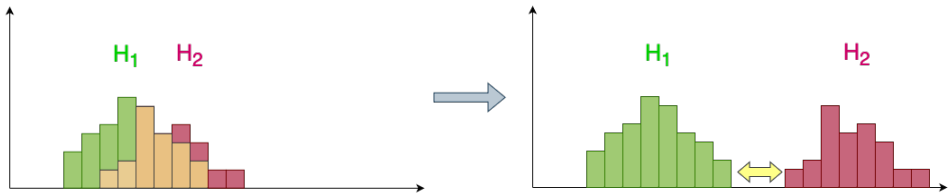

Figure 19: By using the histogram representation, we can directly penalize any overlap (orange) between two histograms H_1 (green) and H_2 (red).

The benefit of this approach is that the overall distribution of the similarities is taken into account during training. This holds more information than simply relying on the individual similarities of some arbitrarily sampled latent vector pairs. This is because, for each similarity measuring step, a larger batch of latent vectors u_j originating from all of the ensemble models and their similarities are considered by calculating the inner products. During the optimization process, each model M_j is forced to generate latent vectors with a higher probability of being sampled from the distribution of the similarities belonging to the same model M_j than any other model M_k. To calculate this probability, which the authors of [32] refer to as "the probability of reverse" $p_{reverse}$, we need to approximate the two probability distributions p^+ and p^-. These refer to the estimate of the probability distributions of the similarities of latent vectors produced by the same model M_j and by a distinct model M_k, respectively.

For our proposed framework Ens_{hist}, we feed the same batch of size b of inputs $\{x_1, x_2, \ldots, x_b\}$ into the different models M_j of the ensemble to extract the latent vectors $E_j(x^{(i)})$. For the proposed framework to work, the member models' outputs must also be normalized due to the limitations discussed in [32]. For this, we follow the recommendations of [32] and apply the L2 norm to each latent vector as $E^{*(j)}(x_i) = ||E^{(j)}(x_i)||$. For estimating p^+ and p^-, following the conventions of [32], we construct sample sets denoted by $\mathcal{S}^+$ and $\mathcal{S}^-$ that contain the

similarity of the sample latent vector pairs, calculated using the inner product, produced by the same models ($\mathcal{S}^+$) and by different models ($\mathcal{S}^-$). We construct the sample sets $\mathcal{S}^+$ and $\mathcal{S}^-$ by defining each sample $s_{j,k}$ using the normalized latent vectors extracted by each member M_j.

Definition 6.5. *Given $E_1^*, E_2^*, ..., E_n^*$, we formulate the sample sets $\mathcal{S}^+$ and $\mathcal{S}^-$ using the following formulae:*

$$\begin{aligned} \mathcal{S}^+ &= \left\{ s_{k,k} = u \cdot v \ : \ u = E_k^*(x^{(i)}), v = E_k^*(x^{(j)}), i \neq j \right\}, \\ \mathcal{S}^- &= \left\{ s_{k,l} = u \cdot v \ : \ u = E_k^*(x^{(i)}), v = E_l^*(x^{(i)}), k \neq l \right\}. \end{aligned} \tag{32}$$

At this point, it can be observed that this method is a natural and logical extension of our framework Ens_{cos} [33], as it also builds on the core idea of using the inner product to measure the similarities of the sample latent vector pairs. However, instead of using these similarities directly, it generates a histogram representation using multiple observed similarities to approximate their distributions. For calculating the similarity between any two models inside the ensemble, we approximate the p^+ and p^- distributions with histograms H^+ and H^-, each with R bins, respectively, following the workflow described in [32], with some slightly modified formulae.

Definition 6.6. *Given the sample sets S^+ and S^-, we define the histograms H^+ and H^- with uniformly spaced bins, with their nodes $-1 = t_1, \ldots, t_R = 1$ filling the $[-1, 1]$ interval and with the step size $\Delta = \frac{2}{R-1}$, according to the following formulae*

$$\begin{aligned} H^+ &= \left[h_r^+ = \frac{1}{|\mathcal{S}^+|} \sum_{j=1}^{n} \delta_{j,j,r} \ : \ r = 1, \ldots, R \right], \\ H^- &= \left[h_r^- = \frac{1}{|\mathcal{S}^-|} \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} \delta_{j,k,r} \ : \ r = 1, \ldots, R \right], \end{aligned} \tag{33}$$

where each weight $delta_{j,k,r}$ is defined as

$$\delta_{j,k,r} = \begin{cases} (s_{j,k} - t_{r-1})/\Delta, & \text{if } s_{j,k} \in [t_{r-1}; t_r] \ , \\ (t_{r+1} - s_{j,k})/\Delta, & \text{if } s_{j,k} \in [t_r; t_{r+1}] \ , \\ 0, & \text{otherwise.} \end{cases} \tag{34}$$

As seen in (33), when calculating H^+, we only consider the similarities between the same model, while when calculating H^-, we only consider the similarities between distinct models. The histograms H^+ and H^- approximate the real distributions p^+ and p^- of the similarities between the latent vectors extracted by each model. Using them, we can directly look for any overlap (see Figure 19) between the similarities of the latent vectors extracted by each model M_j inside the ensemble.

Definition 6.7. *Given $H^+ = [h_1^+, ..., h_R^+]$ and $H^- = [h_1^-, ..., h_R^-]$, we define their similarities using the following formula introduced in [32]*

$$\mathcal{S}(H^+, H^-) = \sum_{r=1}^{R} \left(h_r^- \sum_{q=1}^{r} h_q^+ \right). \tag{35}$$

Since both the forward pass and the histogram loss depend on the normalized E_j^* vectors as well as the histograms H^+ and H^-, the cost function also needs to be slightly modified.

Definition 6.8. *Given the histograms H^+ and H^-, we define the cost function $J_{hist}(\theta)$ for the framework Ens_{hist} as*

$$J_{hist}(\theta) \quad = \quad \frac{1}{m} \sum_{i=1}^{m} \left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \mathcal{S}(H^+, H^-) \right]. \tag{36}$$

6.4.4 Using a weighted cost function

In this section, we extend both the Ens_{cos} and the Ens_{hist} frameworks to handle imbalanced datasets where there is a significant disparity between the total occurrences of the labels. In such cases, some labels appear disproportionally more often in the dataset than others. This is especially common in the medical field, where there are usually orders of magnitude more healthy medical records than records containing any lesion, further increasing the difficulty of training solutions that can recognize chronic lesions or diseases. A common technique for training efficient solutions for such imbalanced datasets is using a weighted loss function that weights the loss calculated for the given class in inverse proportion to its frequency in the dataset. This ultimately makes the training process smoother, providing models that can recognize rare classes.

When using a weighted cost function, the weights $\beta_i \in \mathbb{R}$ are calculated for all inputs $x^{(i)}$ in the dataset $\mathcal{D}$ to increase the influence of the label $y^{(i)}$ on the overall cost during the training procedure. To define a weighted configuration for our framework, it is imperative to note that our cost functions (27) and (36) are constructed of two parts: the first part optimizes the original loss L_i and the second part optimizes the similarity $\mathcal{S}$. This is extremely important because, using a weighted cost function, the objective is to weight each L_i depending on the rarity of the class $y^{(i)}$. However, the second part of the cost function measures the similarity between the ensemble members, which is independent of the labels $y^{(i)}$. Since only the second part of the frameworks depend on the latent vectors $E_j(x^{(i)}))$ derived by each model M_j for the corresponding input $x^{(i)}$, it is only necessary to change the first parts of formulae (27) and (36).

Claim 6.1. *Let $\beta_1, \beta_2, ..., \beta_m$ denote the weights corresponding to each sample in the dataset $\mathcal{D}_{train} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$. Furthermore, let L_i denote the original loss to be optimized. For the frameworks Ens_{cos} and Ens_{hist}, the weights $\beta_1, \beta_2, ..., \beta_m$ are responsible for weighting the original loss L_i but not the similarities computed between the different models of the ensemble.*

Reasoning. Given the weights $\beta_1, \beta_2, ..., \beta_m$ for the dataset $\mathcal{D}_{train} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$, the objective of a weighted cost function is to weight each sample based on the rarity of the sample's label, resulting in rare classes having higher weights and more frequent classes having lower weights.

For the framework Ens_{cos}, it can be seen that in (27), only the first part depends on the outputs $\widehat{y}^{(i)}$, which can be used to calculate the frequency of the given class. The second part, however, only relies on the similarities of the extracted feature vectors between different models. It can be seen that each model M_j is compared to every other model M_k where $k = j + 1, ..., n$. This part does not rely on the class labels $y^{(i)}$ in any way, and its purpose is to compare the features $E_j(x^{(i)})$ and $E_k(x^{(i)})$ for each model M_j and M_k, irrespective of the class labels, treating each model as equal. Therefore, not only does this second part not need to be adjusted by the weights $\beta_1, \beta_2, ..., \beta_m$, if we were to do so, that would mean that the diversity of models with higher β_j coefficients are more important, which is not the case for this framework.

Following the same logic, the same statements are true for the Ens_{hist} framework as well. In (36), similarly to Ens_{cos}, only the first part depends on the class labels $y^{(i)}$. The second part only measures the similarities between the calculated histograms based on the extracted feature vectors of the different ensemble members. This part hence only relies on the extracted (normalized) feature vectors and not on the original class labels $y^{(i)}$.

□

Definition 6.9. *Given the frameworks Ens_{cos} and Ens_{hist} and the original loss function L_i, their corresponding weighted cost function using the weights $\beta_1, \beta_2, \ldots, \beta_m \in \mathbb{R}$ is defined as*

$$\begin{aligned} J^*_{cos}(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\beta_i L_i(\theta_j) + \lambda\sum_{j=1}^{n-1}\sum_{k=j+1}^{n}\mathcal{S}(E_j(x^{(i)}), E_k(x^{(i)}))\right], \\ J^*_{hist}(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\beta_i L_i(\theta_j) + \lambda\mathcal{S}(H^+, H^-)\right] \end{aligned} \tag{37}$$

for Ens_{cos} and Ens_{hist}, respectively.

Although no change would directly affect the calculation of the similarities, the overall behavior and the outputs of the new weighted cost functions shown in (37) changed due to the introduced weights β_i. Therefore, in section 6.6, we also examine the effect of using a weighted cost function for our frameworks using the dataset published by Jun Cheng [111]. In section 6.3 and Table 17, we have shown a slight imbalance between the different classes. Namely, there were almost as many images belonging to the glioma class as in the other two classes (meningioma and pituitary tumor), combined. Therefore, this dataset presents a good opportunity to study the effects of using a weighted loss function following our proposed frameworks.

6.4.5 Using different architectures inside the ensemble

This section extends the Ens_{cos} and Ens_{hist} frameworks to function with different architectures. So far, we have only discussed re-using the exact same architecture, which was a limitation. It stemmed from formulae (27) and (36), which both relied on calculating the similarity $\mathcal{S}$ of the two feature vectors produced by the models M_j and M_k. During the

similarity calculation (see (26) and (32) for Ens_{cos} and Ens_{hist}, respectively), both methods utilized the inner product to calculate the similarity. Since the inner product can only be used for two vectors u and v of the same dimension $dim(u) = dim(v)$, this limits the usage of our framework to neural networks and members M_j and M_k that have the exact same feature vector dimension $dim(E_j(x^{(i)})) = dim(E_k(x^{(i)}))$.

To solve this problem, it is required to bring the feature vectors to a common dimension before measuring the similarity between them. This is not a trivial problem to solve, as reducing the dimension to the value of the smallest feature vector may result in losing important features in longer feature vectors. It is also unclear which components to drop from longer feature vectors, as some of them encode special features and are activated only when these particular features are present in the input image. In this case, dropping such a component, which usually contains 0 values, may decrease the performance. Another approach can be to expand all feature vectors to be of the same dimension as the one with the maximum length. During our experiments, we found that this does not lead to either worse results or instabilities. Therefore, in [34], we propose a simple way to address this issue by applying zero padding to the feature vectors $E_k(x^{(i)})$ to bring them to the same dimension as the ones produced by the model M_j that has the largest dimension. An overview of this method can be seen in Figure 20.

Definition 6.10. *Given the frameworks Ens_{cos} and Ens_{hist} constructed from the member models $M_1, M_2, ..., M_n$, with their corresponding $E_1, ..., E_n$ feature extracting parts, we define the maximum feature vector length $d_{E_{max}}$ for any given input image $x^{(i)}$ as*

$$d_{E_{max}} := max(dim(E_1(x^{(i)})), dim(E_2(x^{(i)})), , ..., dim(E_n(x^{(i)}))) \quad (38)$$

and apply zero-padding to all of the feature vectors $E_1, E_2, ..., E_n$, such that

$$E_j(x^{(i)})_q := \begin{cases} 0, & \text{if } q > dim(E_j(x^{(i)})) \ , \\ E_j(x^{(i)})_q, & \text{otherwise} \end{cases} \quad (39)$$

for every $j = 1, ..., n$ and $q = 1, ..., d_{E_{max}}$, where $E_j(x^{(i)})_q$ denotes the q-th element of the feature vector $E_j(x^{(i)})$.

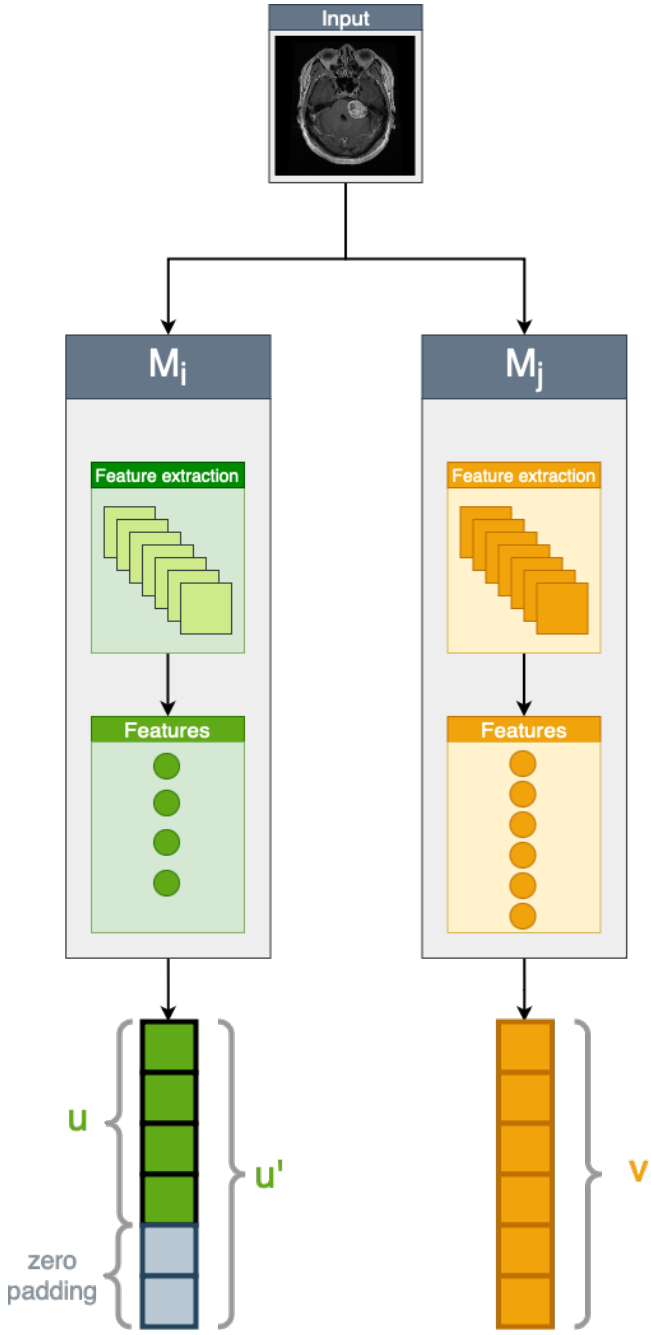

Figure 20: We can bring latent vectors u and v of different dimensions to a common dimension by applying zero padding on the vector with fewer elements. The resulting u' vector will have the same dimension as v, with the last elements being zero.

6.5 Experimental setup

Before training the models, we tuned the hyperparameters of each architecture separately. During this process, except for the total number of epochs, we only used the training set to configure the hyperparameters to avoid any possible bias towards the validation sets. We searched for the optimal value of the batch size, learning rate, and number of epochs and looked for the best optimizer. When searching for the optimal batch size, we considered multiple factors, such as substantial oscillations in the loss – from batch to batch and from epoch to epoch – that would make the learning process unstable. For finding the optimal learning rate $\alpha \in (0, 1)$, we followed a similar approach as in section 4.6. Namely, we used a schedule for a total of 100 epochs on the training set, evaluated the effectiveness and performance of a wide range of learning rates, and picked the one that i) had the lowest loss value and ii) had a sufficiently large environment $[\alpha - \varepsilon, \alpha + \varepsilon]$ with $\varepsilon \in (0, 1)$, where the loss values neither excessively oscillated nor increased substantially. The number of epochs was determined using the validation set; each model was trained until the observed validation loss started increasing rapidly. During the training process, the weights with the lowest validation loss were saved, which were used in the testing phase to evaluate the performance of each model.

After the hyperparameter tuning phase, we found that the batch size of 32 gave the best results for all models, leading to minimal oscillations and smooth decreases in the loss values. For the optimizer, we found Adam [37] to be the most efficient, while for the learning rate, the value of 0.0001 yielded the best overall performance for each architecture: it led to continuous and smooth decreases in the loss value while also being relatively fast, compared to lower learning rates. Furthermore, it did not seem to get stuck in bad-performing local optima during our experiments. For the Ens_{hist} framework, we also considered multiple settings when determining the optimal number of bins. After evaluating multiple different options for R ranging from 65 to 257 (due to Δ relying on $R - 1$) following the findings of [32] and considering the change of training loss, as well as validation loss, we determined that $R = 129$ was the most suitable option for the number of bins in our use case, resulting in $\Delta = 0.015625$. This was also in line with the findings of the original paper [32], which also showed $\Delta = 0.02$ (where Δ was rounded to two decimal places) as a suitable choice, balancing

the level of detail (i.e., number of bins) and performance.

During the evaluation procedure, we considered several of the most important metrics to give an in-depth overview of the performance of the different methodologies and architectures. Namely, we calculated the overall accuracy (ACC), as well as the per class accuracy (ACC_{class_name}), sensitivity (SE), and F_1-score for each architecture following the equations from section 4.6. We also evaluated each model using the precision ($PREC$) metric, defined as

$$PREC = \frac{TP}{TP+FP}. \tag{40}$$

Additionally, our datasets contained multiple classes, three in total. Since the metrics introduced so far are calculated at the class level, it was also important to decide how to calculate a unified metric that merged the results for the different classes into one metric. For this, we calculated the macro- and weighted average of each metric.

As previously mentioned, we used the original folds and splits provided by the authors of the dataset [111] to train and test our models. The official splits divide the dataset of 3 064 MRI images into training and test parts and contain five folds. One drawback of the official splits is that they only contain the training and test parts. To measure and detect the degree of over- and underfitting in our models during training and to monitor the behavior of the models, i.e., to stop the training procedure if the validation loss keeps increasing while the training loss decreases in case of overfitting, we defined a custom validation split using the training data from the five original folds. During this procedure, we split the original training part of the five folds into training and validation parts in a 4:1 ratio, respectively. After this, each model was trained using the training parts of the five newly generated folds, validated using the validation part of the current fold, and then tested using the official test part of the given fold defined in [111].

The experiments were conducted in the cloud using the Azure Machine Learning service. The training took place on a virtual machine with 6 cores, a total of 112GB RAM, and a Tesla V100 GPU. All the related code was written in the Python programming language and the PyTorch framework.

6.6 Experimental results

In this section, we give an exhaustive and in-depth overview of the performance of the various algorithms and report our experimental results. We calculated various metrics using the pre-defined folds and splits to evaluate the overall performance of each model. All models were trained on each of the five training folds and were then evaluated using the original test sets defined by the authors of [111]. We have also compared our results with several other state-of-the-art approaches and standard techniques, such as majority voting, which we used as baselines to evaluate and compare with our results. With the $\lambda = 0$ setup, both our original and improved frameworks apply the standard majority voting, as discussed in section 6.4. Therefore, our tables refer to the standard majority voting approach as $\lambda = 0$. We also considered several other works that used the same dataset, both traditional [115] and deep learning-based ones [113, 114, 116], and used them as simple baselines and compared their reported results with the performance of our frameworks. Cells containing a "-" symbol in one of the columns belonging to a metric contain values that were not reported in the corresponding original study. For columns corresponding to a given setting, the same "-" symbol indicates a setting that does not apply to the given architecture.

We calculated the macro- and weighted average for each previously introduced metric during testing and provided dedicated tables for each setting. Macro average calculates the class-level metrics and then averages them without taking the imbalance into account. For the weighted average, the weights reflect how often a given class occurs in the test set. The results shown are calculated at 95% confidence levels and sorted by the type of model and training methodology in the following order: state-of-the-art baseline approaches and weighted averaging ($WeightedAvg.$), followed by our previous and new frameworks, evaluated using different values of λ. We also included the results of the hybrid ensemble architecture presented in section 6.4.5, where we selected the best-performing base networks MobileNetv2, EfficientNet, and ShuffleNet v2. Using the aforementioned three models, for the sake of brevity, we refer to these approaches as Ens in our tables. For the other methods that used the same member models, we refer to them using the name of the given architecture in the tables. First, we measured how Ens and Ens_{hist} compares with Ens_{cos}. In Table

20, we compare our newly proposed approaches' accuracy – global and class-level – with our previous framework based on the cosine similarity and several state-of-the-art baselines. ACC indicates the overall global accuracy, while the columns ACC_G, ACC_M, and ACC_{PT} correspond to the class-level accuracies belonging to the glioma, meningioma, and pituitary tumor classes, respectively.

Table 20: A summary of the measured global and class-level accuracies (values are shown in %).

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
Afshar et al. [113]	-	-	90.9	-	-	-
Abiwinanda et al. [114]	-	-	84.2	-	-	-
Fasihi et al. [115]	-	-	84.1	-	-	-
MIXCAPS-BoxCaps [116]	-	-	91.3	-	-	-
Weighted Avg.	-	-	91.2 ± 1.7	92.9 ± 1.2	94.0 ± 1.7	95.6 ± 1.7
AlexNet	base	-	86.8 ± 4.2	89.4 ± 3.7	91.0 ± 1.3	93.2 ± 4.0
MobileNetv2	base	-	89.3 ± 3.2	91.9 ± 1.6	92.2 ± 2.6	94.5 ± 2.7
ShuffleNet v2	base	-	86.5 ± 2.9	88.7 ± 2.5	90.5 ± 2.7	93.9 ± 2.0
EfficientNet	base	-	89.8 ± 1.8	91.7 ± 1.4	93.3 ± 1.6	94.5 ± 1.3
AlexNet	cosine	0.0	87.5 ± 3.6	90.2 ± 2.7	90.9 ± 1.7	93.9 ± 3.2
MobileNetv2	cosine	0.0	88.8 ± 2.6	91.1 ± 1.8	92.0 ± 1.8	94.5 ± 2.7
ShuffleNet v2	cosine	0.0	87.7 ± 3.8	89.7 ± 2.8	91.3 ± 2.7	94.5 ± 2.5
EfficientNet	cosine	0.0	89.6 ± 2.2	91.6 ± 1.8	92.9 ± 1.2	94.8 ± 2.3
Ens	cosine	0.0	90.3 ± 1.9	91.6 ± 1.2	93.9 ± 1.5	95.2 ± 1.9
AlexNet	histogram	0.0	87.5 ± 3.6	90.2 ± 2.7	90.9 ± 1.7	93.9 ± 3.2
MobileNetv2	histogram	0.0	88.8 ± 2.6	91.1 ± 1.8	92.0 ± 1.8	94.5 ± 2.7
ShuffleNet v2	histogram	0.0	87.7 ± 3.8	89.7 ± 2.8	91.3 ± 2.7	94.5 ± 2.5
EfficientNet	histogram	0.0	89.6 ± 2.2	91.6 ± 1.8	92.9 ± 1.2	94.8 ± 2.3
Ens	histogram	0.0	90.3 ± 1.9	91.6 ± 1.2	93.9 ± 1.5	95.2 ± 1.9
AlexNet	cosine	0.5	88.2 ± 3.1	90.3 ± 2.6	92.2 ± 1.3	93.8 ± 3.1
MobileNetv2	cosine	0.5	91.5 ± 1.8	93.1 ± 1.4	**94.5 ± 1.2**	95.4 ± 1.9
ShuffleNet v2	cosine	0.5	89.2 ± 3.0	90.7 ± 2.5	92.8 ± 1.8	94.9 ± 2.4
EfficientNet	cosine	0.5	90.4 ± 2.3	92.0 ± 1.4	94.0 ± 1.3	94.9 ± 2.6
Ens	cosine	0.5	90.8 ± 1.9	92.2 ± 1.5	94.2 ± 1.0	95.2 ± 1.9
AlexNet	histogram	0.5	88.7 ± 3.4	90.9 ± 3.0	91.9 ± 1.3	94.6 ± 3.1
MobileNetv2	histogram	0.5	90.8 ± 1.3	92.9 ± 0.9	93.5 ± 0.9	95.3 ± 1.7
ShuffleNet v2	histogram	0.5	89.5 ± 2.1	91.2 ± 1.4	92.9 ± 1.4	94.9 ± 1.9
EfficientNet	histogram	0.5	90.9 ± 1.5	92.3 ± 1.3	94.2 ± 0.9	95.4 ± 2.0
Ens	histogram	0.5	90.1 ± 1.7	91.7 ± 1.2	93.7 ± 0.7	94.8 ± 2.2
AlexNet	cosine	1.0	88.1 ± 4.5	90.9 ± 2.9	91.7 ± 2.2	93.7 ± 4.1
MobileNetv2	cosine	1.0	90.4 ± 2.7	92.2 ± 2.0	93.6 ± 1.7	95.1 ± 2.3
ShuffleNet v2	cosine	1.0	89.3 ± 2.8	90.1 ± 1.9	91.8 ± 2.6	94.4 ± 2.5
EfficientNet	cosine	1.0	90.5 ± 2.1	91.9 ± 1.4	93.7 ± 1.0	95.3 ± 3.0
Ens	cosine	1.0	90.1 ± 3.2	91.7 ± 2.9	93.9 ± 1.5	94.5 ± 2.7
AlexNet	histogram	1.0	87.7 ± 4.2	90.2 ± 2.7	91.3 ± 2.2	93.9 ± 4.0
MobileNetv2	histogram	1.0	**92.1 ± 0.5**	**93.5 ± 0.5**	**94.5 ± 1.1**	**96.3 ± 1.7**
ShuffleNet v2	histogram	1.0	90.2 ± 1.9	92.3 ± 1.4	92.8 ± 1.7	95.2 ± 2.2
EfficientNet	histogram	1.0	90.4 ± 2.4	92.1 ± 2.2	93.9 ± 0.4	94.7 ± 2.4
Ens	histogram	1.0	90.6 ± 1.8	91.6 ± 1.8	94.3 ± 0.5	95.2 ± 2.0

After getting an overview of the overall accuracy of each model, we evaluated them using the sensitivity (SE), precision ($PREC$), and F_1-score. For our first set of experiments, we used the macro averaging method when aggregating the per-class results of each metric (see Table 21). In other words, during these experiments, we treated each class equally during the evaluation procedure and calculated the results for all metrics. As seen in Table 21, Ens_{hist} achieved results similar to Ens_{cos} for the $\lambda = 0.5$ setup; however, it greatly outperformed it

with a larger value of $\lambda = 1.0$. The best-performing model was the same that achieved the highest accuracies in Table 20: the ensemble that was constructed using three MobileNetv2 models. This model achieved over 90% for each metric: 90.5% sensitivity, 92.1% precision, and 91.1% F_1-score.

Table 21: A summary of the measured sensitivity, precision, and F_1-scores using macro averaging (values are shown in %).

Algorithm	Type	λ	SE	$PREC$	F_1
Weighted Avg.	-	-	89.8 ± 1.5	91.0 ± 1.7	90.2 ± 1.7
AlexNet	base	-	85.7 ± 3.7	87.1 ± 4.3	85.6 ± 4.5
MobileNetv2	base	-	87.6 ± 3.2	89.4 ± 2.7	88.1 ± 3.3
ShuffleNet v2	base	-	84.4 ± 3.1	86.3 ± 2.8	85.0 ± 3.0
EfficientNet	base	-	88.5 ± 1.8	89.3 ± 1.8	88.6 ± 1.8
AlexNet	cosine	0.0	85.4 ± 4.0	87.4 ± 3.7	86.1 ± 4.0
MobileNetv2	cosine	0.0	86.9 ± 2.6	89.0 ± 2.8	87.5 ± 2.9
ShuffleNet v2	cosine	0.0	85.9 ± 4.3	87.5 ± 3.3	86.4 ± 4.1
EfficientNet	cosine	0.0	88.5 ± 3.0	89.0 ± 2.2	88.5 ± 2.8
Ens	cosine	0.0	88.8 ± 1.5	89.8 ± 1.9	89.1 ± 1.9
AlexNet	histogram	0.0	85.4 ± 4.0	87.4 ± 3.7	86.1 ± 4.0
MobileNetv2	histogram	0.0	86.9 ± 2.6	89.0 ± 2.8	87.5 ± 2.9
ShuffleNet v2	histogram	0.0	85.9 ± 4.3	87.5 ± 3.3	86.4 ± 4.1
EfficientNet	histogram	0.0	88.5 ± 3.0	89.0 ± 2.2	88.5 ± 2.8
Ens	histogram	0.0	88.8 ± 1.5	89.8 ± 1.9	89.1 ± 1.9
AlexNet	cosine	0.5	86.6 ± 3.3	87.9 ± 3.3	86.9 ± 3.5
MobileNetv2	cosine	0.5	90.2 ± 1.9	91.0 ± 1.7	90.5 ± 1.9
ShuffleNet v2	cosine	0.5	87.2 ± 3.8	88.8 ± 2.9	87.7 ± 3.5
EfficientNet	cosine	0.5	89.1 ± 2.0	90.1 ± 2.3	89.3 ± 2.5
Ens	cosine	0.5	89.6 ± 1.8	90.2 ± 1.8	89.7 ± 1.9
AlexNet	histogram	0.5	87.1 ± 3.6	88.6 ± 3.6	87.5 ± 3.7
MobileNetv2	histogram	0.5	89.3 ± 1.0	90.8 ± 1.4	89.8 ± 1.3
ShuffleNet v2	histogram	0.5	87.7 ± 2.1	89.2 ± 2.3	88.2 ± 2.3
EfficientNet	histogram	0.5	89.4 ± 1.3	90.7 ± 1.9	89.8 ± 1.7
Ens	histogram	0.5	88.6 ± 1.4	89.8 ± 1.8	88.8 ± 1.8
AlexNet	cosine	1.0	86.2 ± 4.8	88.4 ± 4.3	86.7 ± 4.9
MobileNetv2	cosine	1.0	88.4 ± 2.8	90.4 ± 3.0	89.2 ± 3.0
ShuffleNet v2	cosine	1.0	88.0 ± 3.3	88.8 ± 2.5	88.0 ± 3.1
EfficientNet	cosine	1.0	88.8 ± 1.5	90.4 ± 2.5	89.2 ± 2.2
Ens	cosine	1.0	88.7 ± 3.5	89.7 ± 3.4	88.9 ± 3.6
AlexNet	histogram	1.0	86.0 ± 3.8	87.8 ± 3.9	86.3 ± 4.5
MobileNetv2	histogram	1.0	**90.5 ± 1.1**	**92.1 ± 0.9**	**91.1 ± 0.8**
ShuffleNet v2	histogram	1.0	88.6 ± 1.8	90.1 ± 1.8	89.1 ± 1.9
EfficientNet	histogram	1.0	89.2 ± 2.2	90.0 ± 2.7	89.3 ± 2.7
Ens	histogram	1.0	89.1 ± 1.9	90.0 ± 2.2	89.4 ± 2.0

To get a better overview of the performance of the different models, we calculated the weighted sensitivity, precision, and F_1-score (see Table 22). During this evaluation, each weight was calculated using

the support value of the given class, meaning that classes that appear more often would get assigned higher weights. As seen in Table 22, the previously top-performing MobileNetv2 ensemble architecture that used our newly proposed histogram-based framework achieved the best overall results. It achieved over 92% for each of the three metrics.

Table 22: A summary of the measured sensitivity, precision, and F_1-scores using weighted averaging (values are shown in %).

Algorithm	Type	λ	SE	$PREC$	F_1
Weighted Avg.	-	-	91.2 ± 1.7	91.6 ± 1.4	91.2 ± 1.7
AlexNet	base	-	86.8 ± 4.2	88.2 ± 3.0	86.8 ± 4.1
MobileNetv2	base	-	89.3 ± 3.2	89.9 ± 2.5	89.2 ± 3.2
ShuffleNet v2	base	-	86.5 ± 2.9	87.2 ± 2.7	86.5 ± 2.9
EfficientNet	base	-	89.8 ± 1.8	90.3 ± 1.7	89.8 ± 1.8
AlexNet	cosine	0.0	87.5 ± 3.6	88.0 ± 3.3	87.4 ± 3.7
MobileNetv2	cosine	0.0	88.8 ± 2.6	89.5 ± 2.0	88.7 ± 2.7
ShuffleNet v2	cosine	0.0	87.7 ± 3.8	88.4 ± 3.3	87.7 ± 3.8
EfficientNet	cosine	0.0	89.6 ± 2.2	90.2 ± 1.8	89.6 ± 2.3
Ens	cosine	0.0	90.3 ± 1.9	90.8 ± 1.4	90.3 ± 1.8
AlexNet	histogram	0.0	87.5 ± 3.6	88.0 ± 3.3	87.4 ± 3.7
MobileNetv2	histogram	0.0	88.8 ± 2.6	89.5 ± 2.0	88.7 ± 2.7
ShuffleNet v2	histogram	0.0	87.7 ± 3.8	88.4 ± 3.3	87.7 ± 3.8
EfficientNet	histogram	0.0	89.6 ± 2.2	90.2 ± 1.8	89.6 ± 2.3
Ens	histogram	0.0	90.3 ± 1.9	90.8 ± 1.4	90.3 ± 1.8
AlexNet	cosine	0.5	88.2 ± 3.1	89.0 ± 2.6	88.2 ± 3.1
MobileNetv2	cosine	0.5	91.5 ± 1.8	91.8 ± 1.7	91.5 ± 1.9
ShuffleNet v2	cosine	0.5	89.2 ± 3.0	89.6 ± 2.8	89.1 ± 3.2
EfficientNet	cosine	0.5	90.4 ± 2.3	91.1 ± 1.6	90.4 ± 2.3
Ens	cosine	0.5	90.8 ± 1.9	91.2 ± 1.7	90.8 ± 1.9
AlexNet	histogram	0.5	88.7 ± 3.4	89.3 ± 3.2	88.7 ± 3.5
MobileNetv2	histogram	0.5	90.8 ± 1.3	91.2 ± 1.2	90.8 ± 1.3
ShuffleNet v2	histogram	0.5	89.5 ± 2.1	90.0 ± 1.8	89.5 ± 2.1
EfficientNet	histogram	0.5	90.9 ± 1.5	91.4 ± 1.2	90.9 ± 1.5
Ens	histogram	0.5	90.1 ± 1.7	90.7 ± 1.2	90.1 ± 1.7
AlexNet	cosine	1.0	88.1 ± 4.5	88.9 ± 3.9	88.0 ± 4.6
MobileNetv2	cosine	1.0	90.4 ± 2.7	90.8 ± 2.4	90.4 ± 2.7
ShuffleNet v2	cosine	1.0	88.1 ± 3.1	89.0 ± 2.6	88.2 ± 3.1
EfficientNet	cosine	1.0	90.5 ± 2.1	91.0 ± 1.7	90.4 ± 2.1
Ens	cosine	1.0	90.1 ± 3.2	90.7 ± 2.8	90.1 ± 3.2
AlexNet	histogram	1.0	87.7 ± 4.2	88.7 ± 3.2	87.6 ± 4.3
MobileNetv2	histogram	1.0	**92.1 ± 0.5**	**92.3 ± 0.4**	**92.1 ± 0.5**
ShuffleNet v2	histogram	1.0	90.2 ± 1.9	90.5 ± 1.6	90.1 ± 1.9
EfficientNet	histogram	1.0	90.4 ± 2.4	91.0 ± 1.8	90.4 ± 2.3
Ens	histogram	1.0	90.6 ± 1.8	90.7 ± 2.0	90.5 ± 1.9

Next, we examined if using a weighted cost function led to any significant changes or problems during the training procedure. During training, each sample was weighted based on the frequency of its label.

We calculated the weights β_G, β_M, β_{PT} corresponding to the given class before training the given architecture. We assigned lower weights to classes with larger cardinalities and higher weights to classes with smaller cardinalities in the training set. As seen in Table 23, we can state that using a weighted cost function did not lead to any instabilities or problems regarding convergence for the examined methods.

Table 23: A summary of the measured global and class-level accuracies using a weighted cost function (values are shown in %).

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
Afshar et al. [113]	-	-	90.9	-	-	-
Abiwinanda et al. [114]	-	-	84.2	-	-	-
Fasihi et al. [115]	-	-	84.1	-	-	-
MIXCAPS-BoxCaps [116]	-	-	91.3	-	-	-
Weighted Avg.	-	-	91.5 ± 1.9	92.5 ± 1.5	94.9 ± 0.6	95.6 ± 2.3
AlexNet	base	-	87.9 ± 3.8	89.8 ± 3.2	91.2 ± 1.8	94.7 ± 3.3
MobileNetv2	base	-	88.9 ± 2.6	90.3 ± 2.1	92.7 ± 1.6	94.8 ± 2.1
ShuffleNet v2	base	-	88.1 ± 1.4	89.0 ± 1.8	92.8 ± 1.5	94.3 ± 2.2
EfficientNet	base	-	89.8 ± 2.3	91.1 ± 1.8	94.0 ± 1.0	94.4 ± 2.3
AlexNet	cosine	0.0	88.4 ± 4.3	90.7 ± 2.4	91.6 ± 3.0	94.4 ± 3.4
MobileNetv2	cosine	0.0	90.2 ± 2.3	92.2 ± 1.2	92.9 ± 2.0	95.4 ± 2.4
ShuffleNet v2	cosine	0.0	89.0 ± 2.5	90.6 ± 2.0	93.0 ± 1.9	94.5 ± 2.3
EfficientNet	cosine	0.0	89.7 ± 1.9	91.2 ± 1.4	93.5 ± 0.8	94.7 ± 2.9
Ens	cosine	0.0	91.2 ± 2.1	92.1 ± 1.8	94.7 ± 0.4	95.6 ± 2.4
AlexNet	histogram	0.0	88.4 ± 4.3	90.7 ± 2.4	91.6 ± 3.0	94.4 ± 3.4
MobileNetv2	histogram	0.0	90.2 ± 2.3	92.2 ± 1.2	92.9 ± 2.0	95.4 ± 2.4
ShuffleNet v2	histogram	0.0	89.0 ± 2.5	90.6 ± 2.0	93.0 ± 1.9	94.5 ± 2.3
EfficientNet	histogram	0.0	89.7 ± 1.9	91.2 ± 1.4	93.5 ± 0.8	94.7 ± 2.9
Ens	cosine	0.0	91.2 ± 2.1	92.1 ± 1.8	94.7 ± 0.4	95.6 ± 2.4
AlexNet	cosine	0.5	89.2 ± 3.2	91.4 ± 2.2	92.0 ± 2.1	95.0 ± 2.7
MobileNetv2	cosine	0.5	91.2 ± 1.7	92.8 ± 1.2	94.3 ± 1.1	95.4 ± 1.7
ShuffleNet v2	cosine	0.5	90.4 ± 3.3	91.9 ± 2.6	93.7 ± 1.6	95.1 ± 2.9
EfficientNet	cosine	0.5	91.8 ± 2.3	**93.2 ± 2.0**	95.0 ± 0.7	95.5 ± 2.2
Ens	cosine	0.5	**92.1 ± 1.6**	**93.2 ± 1.0**	95.0 ± 0.9	**95.9 ± 1.7**
AlexNet	histogram	0.5	89.0 ± 2.8	90.7 ± 2.0	92.2 ± 1.6	95.1 ± 2.9
MobileNetv2	histogram	0.5	90.9 ± 2.2	92.6 ± 1.3	94.1 ± 1.0	95.2 ± 2.6
ShuffleNet v2	histogram	0.5	91.3 ± 2.3	92.7 ± 1.4	94.1 ± 1.3	95.7 ± 2.4
EfficientNet	histogram	0.5	91.2 ± 2.0	92.4 ± 1.8	94.3 ± 1.4	95.6 ± 2.1
Ens	histogram	0.5	90.6 ± 2.6	92.2 ± 1.7	94.3 ± 1.6	94.8 ± 2.5
AlexNet	cosine	1.0	88.6 ± 3.2	90.8 ± 2.2	92.1 ± 1.6	94.3 ± 3.2
MobileNetv2	cosine	1.0	90.2 ± 1.7	91.9 ± 1.4	93.5 ± 1.4	95.0 ± 1.7
ShuffleNet v2	cosine	1.0	89.6 ± 3.1	91.3 ± 2.3	93.4 ± 2.3	94.5 ± 2.6
EfficientNet	cosine	1.0	91.9 ± 1.6	92.9 ± 1.4	**95.2 ± 0.8**	95.6 ± 1.8
Ens	cosine	1.0	91.4 ± 1.8	92.7 ± 1.9	94.5 ± 1.3	95.6 ± 2.0
AlexNet	histogram	1.0	89.6 ± 2.7	91.3 ± 2.7	92.8 ± 0.4	95.2 ± 2.7
MobileNetv2	histogram	1.0	90.9 ± 2.4	92.4 ± 1.9	94.1 ± 1.7	95.3 ± 2.0
ShuffleNet v2	histogram	1.0	90.7 ± 2.6	91.9 ± 2.0	94.0 ± 2.0	95.6 ± 2.2
EfficientNet	histogram	1.0	91.4 ± 2.3	92.8 ± 2.1	94.6 ± 0.9	95.5 ± 2.2
Ens	histogram	1.0	90.2 ± 2.0	91.5 ± 1.5	93.8 ± 1.4	95.1 ± 2.2

Although using this approach did not lead to better results as compared with the MobileNetv2 architecture in Table 20, we can observe a drastic improvement for the EfficientNet architecture, which achieved 1.5% higher accuracy as compared to the reported results in Table 20. These results highlight the importance of the framework being able to handle a weighted cost function, as, depending on the architecture, some models, like EfficientNet, may respond exceptionally well to introducing class weights in the loss function, leading to better results

for the given architecture.

We also evaluated the effect of using a weighted cost function on each architecture using the macro average of the sensitivity, precision, and F_1-scores; the results can be seen in Table 24. Similarly to the previous results, the EfficientNet-based ensemble architecture achieved good results, while the hybrid method using three different architectures (denoted by *Ens*) had the best results.

Table 24: A summary of the measured sensitivity, precision, and F_1-scores using macro averaging and a weighted cost function (values are shown in %).

Algorithm	Type	λ	SE	$PREC$	F_1
Weighted Avg.	-	-	90.9 ± 1.9	90.7 ± 2.0	90.5 ± 2.2
AlexNet	base	-	86.8 ± 4.0	87.4 ± 3.6	86.7 ± 4.0
MobileNetv2	base	-	87.9 ± 2.0	88.4 ± 2.5	87.7 ± 2.6
ShuffleNet v2	base	-	87.6 ± 0.6	87.4 ± 1.4	86.9 ± 1.3
EfficientNet	base	-	88.8 ± 2.8	88.8 ± 2.2	88.5 ± 2.6
AlexNet	cosine	0.0	87.2 ± 4.3	88.1 ± 3.7	87.2 ± 4.5
MobileNetv2	cosine	0.0	89.0 ± 2.3	90.0 ± 2.0	89.3 ± 2.3
ShuffleNet v2	cosine	0.0	88.3 ± 2.5	88.4 ± 2.2	88.0 ± 2.5
EfficientNet	cosine	0.0	88.2 ± 1.8	89.3 ± 2.1	88.4 ± 2.1
Ens	cosine	0.0	90.6 ± 1.9	90.4 ± 2.2	90.1 ± 2.4
AlexNet	histogram	0.0	87.2 ± 4.3	88.1 ± 3.7	87.2 ± 4.5
MobileNetv2	histogram	0.0	89.0 ± 2.3	90.0 ± 2.0	89.3 ± 2.3
ShuffleNet v2	histogram	0.0	88.3 ± 2.5	88.4 ± 2.2	88.0 ± 2.5
EfficientNet	histogram	0.0	88.2 ± 1.8	89.3 ± 2.1	88.4 ± 2.1
Ens	histogram	0.0	90.6 ± 1.9	90.4 ± 2.2	90.1 ± 2.4
AlexNet	cosine	0.5	87.7 ± 3.4	89.1 ± 2.9	88.1 ± 3.4
MobileNetv2	cosine	0.5	90.1 ± 1.8	90.9 ± 1.7	90.2 ± 1.9
ShuffleNet v2	cosine	0.5	89.3 ± 3.3	90.1 ± 3.2	89.3 ± 3.7
EfficientNet	cosine	0.5	90.5 ± 2.6	91.4 ± 2.5	90.8 ± 2.6
Ens	cosine	0.5	**91.0 ± 1.6**	**91.6 ± 1.7**	**91.1 ± 1.7**
AlexNet	histogram	0.5	87.6 ± 2.7	88.5 ± 2.8	87.8 ± 3.0
MobileNetv2	histogram	0.5	90.0 ± 2.0	90.5 ± 2.0	89.9 ± 2.4
ShuffleNet v2	histogram	0.5	90.2 ± 2.5	90.7 ± 2.1	90.2 ± 2.6
EfficientNet	histogram	0.5	90.1 ± 1.8	90.9 ± 2.2	90.2 ± 2.1
Ens	histogram	0.5	89.3 ± 2.4	90.2 ± 2.5	89.4 ± 2.8
AlexNet	cosine	1.0	88.9 ± 1.4	89.7 ± 1.7	89.1 ± 1.6
MobileNetv2	cosine	1.0	88.9 ± 1.4	89.7 ± 1.7	89.1 ± 1.6
ShuffleNet v2	cosine	1.0	88.3 ± 3.6	89.1 ± 2.7	88.4 ± 3.4
EfficientNet	cosine	1.0	90.6 ± 1.4	91.5 ± 2.2	90.8 ± 1.7
Ens	cosine	1.0	90.3 ± 1.7	90.9 ± 2.0	90.4 ± 1.9
AlexNet	histogram	1.0	88.3 ± 2.8	89.2 ± 3.1	88.5 ± 3.1
MobileNetv2	histogram	1.0	89.6 ± 2.1	90.6 ± 2.6	89.9 ± 2.4
ShuffleNet v2	histogram	1.0	89.7 ± 2.7	90.1 ± 2.4	89.7 ± 2.7
EfficientNet	histogram	1.0	90.3 ± 2.5	91.0 ± 2.5	90.4 ± 2.6
Ens	histogram	1.0	89.0 ± 1.8	89.7 ± 2.0	89.0 ± 2.1

Finally, we used the weighted averaging method to measure the same metrics SE, $PREC$, and F_1-score for the models trained using a weighted cost function. As shown in Table 25, the best-performing architectures were the ones using the EfficientNet network, achieving close to 92% sensitivity, precision, and F_1-score, and the hybrid architecture denoted by Ens, which achieved the best results with 92.1% sensitivity, 92.5% precision, and 92.1% F_1-score, respectively.

Table 25: A summary of the measured sensitivity, precision, and F_1-scores using weighted averaging and a weighted cost function (values are shown in %).

Algorithm	Type	λ	SE	$PREC$	F_1
Weighted Avg.	-	-	91.5 ± 1.9	92.2 ± 1.4	91.6 ± 1.8
AlexNet	base	-	87.9 ± 3.8	88.6 ± 3.4	87.9 ± 3.8
MobileNetv2	base	-	88.9 ± 2.6	89.8 ± 1.7	89.0 ± 2.4
ShuffleNet v2	base	-	88.1 ± 1.4	89.5 ± 1.0	88.3 ± 1.3
EfficientNet	base	-	89.8 ± 2.3	90.5 ± 2.4	89.8 ± 2.4
AlexNet	cosine	0.0	88.4 ± 4.3	89.2 ± 3.3	88.4 ± 4.2
MobileNetv2	cosine	0.0	90.2 ± 2.3	90.7 ± 1.9	90.2 ± 2.3
ShuffleNet v2	cosine	0.0	89.0 ± 2.5	89.9 ± 2.2	89.1 ± 2.5
EfficientNet	cosine	0.0	89.7 ± 1.9	90.4 ± 1.6	89.7 ± 1.8
Ens	cosine	0.0	91.2 ± 2.1	91.9 ± 1.5	91.3 ± 2.0
AlexNet	histogram	0.0	88.4 ± 4.3	89.2 ± 3.3	88.4 ± 4.2
MobileNetv2	histogram	0.0	90.2 ± 2.3	90.7 ± 1.9	90.2 ± 2.3
ShuffleNet v2	histogram	0.0	89.0 ± 2.5	89.9 ± 2.2	89.1 ± 2.5
EfficientNet	histogram	0.0	89.7 ± 1.9	90.4 ± 1.6	89.7 ± 1.8
Ens	cosine	0.0	91.2 ± 2.1	91.9 ± 1.5	91.3 ± 2.0
AlexNet	cosine	0.5	89.2 ± 3.2	89.8 ± 2.7	89.2 ± 3.3
MobileNetv2	cosine	0.5	91.2 ± 1.7	91.7 ± 1.4	91.2 ± 1.7
ShuffleNet v2	cosine	0.5	90.4 ± 3.3	91.1 ± 2.4	90.4 ± 3.3
EfficientNet	cosine	0.5	91.8 ± 2.3	92.2 ± 2.3	91.8 ± 2.3
Ens	cosine	0.5	**92.1 ± 1.6**	**92.5 ± 1.3**	**92.1 ± 1.6**
AlexNet	histogram	0.5	89.0 ± 2.8	89.5 ± 2.3	89.0 ± 2.8
MobileNetv2	histogram	0.5	90.9 ± 2.2	91.5 ± 1.5	90.9 ± 2.2
ShuffleNet v2	histogram	0.5	91.3 ± 2.3	91.7 ± 2.0	91.3 ± 2.4
EfficientNet	histogram	0.5	91.2 ± 2.0	91.8 ± 1.7	91.2 ± 2.0
Ens	histogram	0.5	90.6 ± 2.6	91.2 ± 2.1	90.6 ± 2.6
AlexNet	cosine	1.0	88.6 ± 3.2	89.2 ± 2.8	88.5 ± 3.4
MobileNetv2	cosine	1.0	90.2 ± 1.7	90.6 ± 1.6	90.2 ± 1.7
ShuffleNet v2	cosine	1.0	89.6 ± 3.1	90.3 ± 2.7	89.6 ± 3.1
EfficientNet	cosine	1.0	91.9 ± 1.6	92.3 ± 1.6	91.8 ± 1.6
Ens	cosine	1.0	91.4 ± 1.8	91.7 ± 1.8	91.4 ± 1.8
AlexNet	histogram	1.0	89.6 ± 2.7	90.1 ± 2.6	89.7 ± 2.7
MobileNetv2	histogram	1.0	90.9 ± 2.4	91.4 ± 2.1	90.9 ± 2.4
ShuffleNet v2	histogram	1.0	90.7 ± 2.6	91.2 ± 2.3	90.8 ± 2.6
EfficientNet	histogram	1.0	91.4 ± 2.3	91.9 ± 2.1	91.5 ± 2.3
Ens	histogram	1.0	90.2 ± 2.0	90.9 ± 1.4	90.3 ± 1.9

6.7 Conclusions

In this chapter, we proposed multiple frameworks for building diverse ensemble architectures, which penalize the similarities between the feature vectors extracted by the last convolutional layers of the member models. First, we introduced a framework that used the cosine similarity as a metric of diversity inside the ensemble. Even though the framework achieved significantly better results than other approaches, we showed that there can be instances when it may perform suboptimally. We noted that these relate to the cosine similarity not recognizing out-of-order sequences or permutations inside the feature vectors. According to our experimental findings, this may lead to the framework failing to perceive roughly 50% of the similarities between two vectors. Therefore, we introduced a modified framework that used the histogram loss. We showed that, due to its more robust theoretical background, this version does not rely on the order of the elements inside the feature vectors, ultimately solving the issues of the framework that uses the cosine similarity.

We evaluated both of the proposed frameworks on a dataset containing brain MRI images with some of the most common brain tumors: glioma, meningioma, and pituitary tumor. We compared our proposed frameworks not only with the original architectures that were used as member models inside the frameworks, but with other state-of-the-art models and methods as well. Moreover, we also compared our experimental results with traditional ensemble methods, such as majority voting and weighted average ensembles. During these investigations, we utilized several of the most common metrics, such as accuracy, sensitivity, precision and F_1-score. In all of our experiments, we showed that our proposed methods greatly outperformed all of these solutions. Our findings therefore highlight the importance of diversity in ensemble models.

7 Summary

In this dissertation, we presented numerous novel solutions for increasing the reliability and accuracy of neural network-based algorithms and used them to solve various problems in the clinical domain. We showed that the solutions included in this dissertation could be divided into two groups: i) combining traditional methods with neural networks, and ii) building performant ensembles. The objective of both groups was to further improve the performance of currently available state-of-the-art approaches.

For the first group, the first presented method aimed to combine the benefits of traditional theoretical models with those of modern neural network approaches. To achieve this, we introduced a novel two-step architecture. The architecture first trains a neural network model to approximate a given theoretical model by sampling synthetic training data using the original theoretical model, and then the model is trained further on the real dataset. We showed that by utilizing this two-step framework, neural network models can be trained even for small datasets, since a huge amount of synthetic data can be generated using the given theoretical model. We also noted that after this first round of training, the neural network can not only retain the positive properties of the theoretical model, but it can also fine-tune its own weights on the real dataset. This makes the neural network models more accurate than the original theoretical model for the given dataset.

Next, we introduced the idea of combining the hand-crafted feature vectors with the features extracted by a wide range of convolutional neural networks. We detailed the problems of hand-crafted features, namely that they require time-consuming fine-tuning and their individual usage often leads to sub-optimal results. Then, we showed that by using these features in conjunction with the features extracted by convolutional neural networks, we could develop algorithms that surpassed not only the original CNN networks, but also other state-of-the-art approaches. Finally, we detailed that the combination of the two types of features is not straightforward, and hence presented three different variants. Each variant utilized a different strategy for the combination, such as shallow (one layer), deeper (multi-layer) combination and processing the features individually before a final aggregation step.

Regarding the second group of the solutions presented in this dissertation, we first focused on the task of cell segmentation on digitized

Pap smear images. We explained that a reliable CAD system, responsible for the segmentation of cells from any other noise on the given smear, could potentially help in the early detection of cervical cancer. To this end, we presented our own dataset, containing a total of 3 565 image-annotation pairs. Then, we introduced our novel ensemble approach, which, utilizing an FCN-32 as its backbone, receives the generated segmentation masks of some pre-trained FCN models in addition to the original input. We discussed that the main benefit of this approach lies in the fact that it can not only aggregate the individual predictions, but it can even override them using the ensemble's own weights, ultimately coming up with its own predictions. We also showed that by using this approach, the ensemble was able to surpass any of the FCN networks and other state-of-the-art approaches as well.

Lastly, we focused on the problem of diversity in ensemble models. We discussed that traditional ensemble methods, such as majority voting and weighted averaging do not consider the diversity of the member models, which, according to recent literature, would be greatly beneficial and should considerably improve the performance of the ensemble. To this end, we presented multiple novel frameworks that directly optimize the diversity of the base models during the training of said models by using the similarity of the feature vectors generated by the models for each input image as a form of regularization. We presented two versions of our framework: one using the cosine similarity and one using the histogram loss to measure the previously mentioned similarity between the feature vectors. Next, we extended both of these frameworks so that using a weighted cost function and integrating different architectures with feature vectors of different dimensions become possible. Finally, we compared our solutions with other state-of-the-art approaches, the member models, and traditional ensemble methods as well by using a comprehensive list of different metrics, and have shown that our framework can greatly outperform all of them.

8 Összefoglaló

A disszertációban számos újszerű megoldást mutattunk be a neurális háló alapú algoritmusok megbízhatóságának, valamint pontosságának növelésére és alkalmaztuk azokat problémák megoldására a klinikai területen. Megmutattuk, hogy a disszertációban szereplő megoldások két csoportba sorolhatók: i) a hagyományos módszerek neurális hálókkal való kombinálása, és ii) a hatékony és megbízható együttes (ensemble) modellek építése. Mindkét csoport legfőbb célja az volt, hogy tovább javítsuk a jelenleg elérhető megközelítések, módszerek teljesítményét.

Az első csoport esetében az elsőként bemutatott módszer az elméleti modellek és a modern neurális háló alapú megközelítések előnyeit kívánta ötvözni. Ennek érdekében egy újszerű, kétlépcsős architektúrát vezettünk be. Az architektúra először egy neurális hálót tanít be egy adott elméleti modell közelítésére az eredeti elméleti modell felhasználásával mintavételezett szintetikus tanító adatok használatával, majd a modellt tovább tanítjuk a valós adathalmazon. Megmutattuk, hogy e kétlépcsős keretrendszer alkalmazásával a neurális háló alapú modellek kis adathalmazokra is betaníthatók, mivel hatalmas mennyiségű szintetikus adat generálható az adott elméleti modell segítségével. Továbbá megjegyeztük, hogy a neurális háló az első betanítási kör után nemcsak az elméleti modell pozitív tulajdonságait képes megtartani, hanem saját súlyait is finomhangolni tudja a valós adathalmazon. A megközelítést alkalmazva így a neurális háló alapú modellek pontosabbak lehetnek, mint az eredeti elméleti modell az adott adathalmazt tekintve.

Ezután bemutattuk a hagyományos képfeldolgozással és a különféle konvolúciós neurális hálók által kinyert jellemzők kombinálásának ötletét. Kitértünk a hagyományos módszerekkel kinyert jellemzők problémáira, nevezetesen, hogy időigényes finomhangolást igényelnek, és egyedi használatuk gyakran nem optimális eredményekhez vezet. Ezután megmutattuk, hogy e jellemzőknek a konvolúciós neurális hálók által kinyert jellemzőkkel együtt történő használatával olyan algoritmusokat tudtunk kifejleszteni, amelyek nemcsak az eredeti CNN hálókat, hanem más, korszerűbb megközelítéseket is felülmúltak. Végül részleteztük, hogy a kétféle jellemző megfelelő kombinációja egyáltalán nem triviális kérdés, ezért három különböző változatot mutattunk be. Mindegyik változat más stratégiát alkalmazott a kombináció kivitelezésére, úgymint a sekély (egy réteg), mélyebb (többrétegű) kombinációt

és a jellemzők egyenkénti, külön feldolgozását egy végső aggregációs lépés előtt.

A disszertációban bemutatott algoritmusok második csoportjából először a sejtek szegmentálásának feladatával foglalkoztunk digitalizált Pap-keneteken. Megemlítettük, hogy egy megbízható CAD rendszer, amely képes a sejtek szegmentálására és azok egyéb zajtól történő elkülönítésére az adott keneten, potenciálisan segíthet a méhnyakrák korai felismerésében. Ebből a célból bemutattuk a saját adathalmazunkat, amely összesen 3 565 kép-annotáció párt tartalmazott. Ezután bemutattuk az újszerű ensemble megközelítésünket, amely egy FCN-32 architektúrát használt gerinceként, és amely az eredeti bemenettel együtt megkapja több előre betanított FCN modell által generált szegmentációs maszkokat is. Megvitattuk, hogy ennek a megközelítésnek a fő előnye abban rejlik, hogy nem csak aggregálni tudja az egyes előrejelzéseket, hanem az architektúra képes azokat felül is írni saját súlyainak felhasználásával, így pedig végül saját előrejelzésekkel állhat elő. Azt is megmutattuk, hogy ennek a megközelítésnek a használatával az ensemble képes volt felülmúlni bármelyik FCN hálózatot és egyéb korszerű megközelítéseket is.

Zárásként az együttes modellek diverzitásának problémájával foglalkoztunk. Részleteztük, hogy a hagyományos ensemble módszerek, mint például a többségi szavazás és a súlyozott átlagolás nem veszik figyelembe az együttes tagjainak diverzitását, ami az aktuális szakirodalom szerint kifejezetten hasznos lenne és jelentősen javítaná az ensemble teljesítményét. Ebből a célból több olyan újszerű keretrendszert mutattunk be, amelyek közvetlenül optimalizálják a tagmodellek sokféleségét az említett modellek tanítása során azáltal, hogy a modellek által az egyes bemeneti képekre generált jellemzővektorok hasonlóságát használják egyfajta regularizációként. Keretrendszerünk két változatát mutattuk be: az egyik a koszinusz hasonlóságot, a másik pedig a hisztogram veszteséget használja a jellemzővektorok közötti hasonlóság mérésére. Ezt követően mindkét keretrendszert kibővítettük, így lehetővé vált a súlyozott költségfüggvény használata és különböző architektúrák integrálása különböző dimenziójú jellemzővektorok esetén is. Végül a megoldásainkat összehasonlítottuk más korszerű megközelítésekkel, a tagmodellekkel és a hagyományos ensemble módszerekkel is, különböző metrikák átfogó listáját használva, és megmutattuk, hogy a keretrendszerünk képes mindegyiket jelentősen felülmúlni.

Acknowledgements

I would like to thank everyone who contributed to my dissertation in any way.

First of all, I would like to thank my supervisor Dr. András Hajdu for his continuous professional guidance and help in the preparation of this dissertation. I would like to thank him for his professional knowledge, experience and advice in raising the quality of my dissertation.

I would also like to thank the faculty members who have contributed to my development with their professional expertise and experience during my studies. I am grateful to Dr. Magda Várterész, who encouraged me to start demonstrator activities at the faculty. I am also grateful to the late Dr. Norbert Bátfai. It was thanks to him that I was introduced to research and writing publications at the beginning of my studies. Last but not least, I would like to thank Dr. István Fazekas, who helped me with his professional knowledge, experience and encouragement.

I would also like to thank Dr. Balázs Harangi, who enabled me to participate in several research projects and who contributed greatly to my publication activities with his professional advice and experience.

I am immensely grateful to my mother and father, who have been by my side throughout the years, encouraging and supporting me to achieve my goal.

Some elements of this dissertation were supported by the New National Excellence Programme (tender identifiers ÚNKP-21-3-I-DE-99, ÚNKP-22-3-II-DE-109, and ÚNKP-23-3-II-DE-119) and the Open Clouds for Research Environments project (OCRE).

References

[1] A. Albrecht, E. Hein, K. Steinhöfel, M. Taupitz, and C. Wong, "Bounded-depth threshold circuits for computer-assisted ct image classification," *Artificial Intelligence in Medicine*, vol. 24, no. 2, pp. 179–192, 2002.

[2] L. Bi, J. Kim, L. Wen, and D. D. Feng, "Automated and robust percist-based thresholding framework for whole body pet-ct studies," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5335–5338, IEEE, 2012.

[3] J. Zhang, C.-H. Yan, C.-K. Chui, and S.-H. Ong, "Fast segmentation of bone in ct images using 3d adaptive thresholding," *Computers in biology and medicine*, vol. 40, no. 2, pp. 231–236, 2010.

[4] S. Kumar, R. Moni, and J. Rajeesh, "Contourlet transform based computer-aided diagnosis system for liver tumors on computed tomography images," in *2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies*, pp. 217–222, IEEE, 2011.

[5] S. Kumar, R. Moni, and J. Rajeesh, "An automatic computer-aided diagnosis system for liver tumours on computed tomography images," *Computers & Electrical Engineering*, vol. 39, no. 5, pp. 1516–1526, 2013.

[6] Y. Zhu, Q.-C. Wang, M.-D. Xu, Z. Zhang, J. Cheng, Y.-S. Zhong, Y.-Q. Zhang, W.-F. Chen, L.-Q. Yao, P.-H. Zhou, *et al.*, "Application of convolutional neural network in the diagnosis of the invasion depth of gastric cancer based on conventional endoscopy," *Gastrointestinal endoscopy*, vol. 89, no. 4, pp. 806–815, 2019.

[7] W. K. Moon, Y.-W. Lee, H.-H. Ke, S. H. Lee, C.-S. Huang, and R.-F. Chang, "Computer-aided diagnosis of breast ultrasound images using ensemble learning from convolutional neural networks," *Computer methods and programs in biomedicine*, vol. 190, p. 105361, 2020.

[8] N. E. Benzebouchi, N. Azizi, and K. Ayadi, "A computer-aided diagnosis system for breast cancer using deep convolutional neural networks," in *Computational Intelligence in Data Mining: Proceedings of the International Conference on CIDM 2017*, pp. 583–593, Springer, 2019.

[9] H. Sun, X. Zeng, T. Xu, G. Peng, and Y. Ma, "Computer-aided diagnosis in histopathological images of the endometrium using a convolutional neural network and attention mechanisms," *IEEE journal of biomedical and health informatics*, vol. 24, no. 6, pp. 1664–1676, 2019.

[10] A. Ahmadi, M. Kashefi, H. Shahrokhi, and M. A. Nazari, "Computer aided diagnosis system using deep convolutional neural networks for adhd subtypes," *Biomedical Signal Processing and Control*, vol. 63, p. 102227, 2021.

[11] R. Holker and S. Susan, "Computer-aided diagnosis framework for adhd detection using quantitative eeg," in *International Conference on Brain Informatics*, pp. 229–240, Springer, 2022.

[12] H. Haghighat, M. Mirzarezaee, B. N. Araabi, and A. Khadem, "An age-dependent connectivity-based computer aided diagnosis system for autism spectrum disorder using resting-state fmri," *Biomedical Signal Processing and Control*, vol. 71, p. 103108, 2022.

[13] V. Sathiyamoorthi, A. Ilavarasi, K. Murugeswari, S. T. Ahmed, B. A. Devi, and M. Kalipindi, "A deep convolutional neural network based computer aided diagnosis system for the prediction of alzheimer's disease in mri images," *Measurement*, vol. 171, p. 108838, 2021.

[14] S. Saravanakumar and P. Thangaraj, "A computer aided diagnosis system for identifying alzheimer's from mri scan using improved adaboost," *Journal of Medical Systems*, vol. 43, pp. 1–8, 2019.

[15] A. Sobrinho, A. C. D. S. Queiroz, L. D. Da Silva, E. D. B. Costa, M. E. Pinheiro, and A. Perkusich, "Computer-aided diagnosis of chronic kidney disease in developing countries: A comparative

analysis of machine learning techniques," *IEEE Access*, vol. 8, pp. 25407–25419, 2020.

[16] J. Liu, Z. Li, X. Fan, X. Hu, J. Yan, B. Li, Q. Xia, J. Zhu, and Y. Wu, "Crt-net: A generalized and scalable framework for the computer-aided diagnosis of electrocardiogram signals," *Applied Soft Computing*, vol. 128, p. 109481, 2022.

[17] K. Raza, "Improving the prediction accuracy of heart disease with ensemble learning and majority voting rule," in *U-Healthcare Monitoring Systems*, pp. 179–196, Elsevier, 2019.

[18] R. Atallah and A. Al-Mousa, "Heart disease detection using machine learning majority voting ensemble method," in *2019 2nd international conference on new trends in computing sciences (ictcs)*, pp. 1–6, IEEE, 2019.

[19] G. S. Tandel, A. Tiwari, and O. Kakde, "Performance optimisation of deep learning models using majority voting algorithm for brain tumour classification," *Computers in Biology and Medicine*, vol. 135, p. 104564, 2021.

[20] R. K. Bania and A. Halder, "R-hefs: Rough set based heterogeneous ensemble feature selection method for medical data classification," *Artificial Intelligence in Medicine*, vol. 114, p. 102049, 2021.

[21] M. S. Sharif, M. Abbod, A. Al-Bayatti, A. Amira, A. S. Alfakeeh, and B. Sanghera, "An accurate ensemble classifier for medical volume analysis: Phantom and clinical pet study," *IEEE Access*, vol. 8, pp. 37482–37494, 2020.

[22] R. Khasha, M. M. Sepehri, and S. A. Mahdaviani, "An ensemble learning method for asthma control level detection with leveraging medical knowledge-based classifier and supervised learning," *Journal of medical systems*, vol. 43, pp. 1–15, 2019.

[23] L. Liu, W. Wei, K.-H. Chow, M. Loper, E. Gursoy, S. Truex, and Y. Wu, "Deep neural network ensembles against deception: Ensemble diversity, accuracy and robustness," in *2019 IEEE 16th international conference on mobile ad hoc and sensor systems (MASS)*, pp. 274–282, IEEE, 2019.

[24] S. Zhang, M. Liu, and J. Yan, "The diversified ensemble neural network," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16001–16011, 2020.

[25] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[26] W. O. Kermack and A. G. McKendrick, "A contribution to the mathematical theory of epidemics," *Proceedings of the Royal Society of London. Series A, Containing papers of a mathematical and physical character*, vol. 115, no. 772, pp. 700–721, 1927.

[27] T. Harko, F. S. Lobo, and M. Mak, "Exact analytical solutions of the susceptible-infected-recovered (sir) epidemic model and of the sir model with equal death and birth rates," *Applied Mathematics and Computation*, vol. 236, pp. 184–194, 2014.

[28] G. Bogacsovics, A. Hajdu, R. Lakatos, M. Beregi-Kovács, A. Tiba, and H. Tomán, "Replacing the sir epidemic model with a neural network and training it further to increase prediction accuracy," in *Annales Mathematicae et Informaticae*, vol. 53, pp. 73–91, Eszterházy Károly Egyetem Líceum Kiadó, 2021.

[29] G. Bogacsovics, J. Toth, A. Hajdu, and B. Harangi, "Enhancing cnns through the use of hand-crafted features in automated fundus image classification," *Biomedical Signal Processing and Control*, vol. 76, p. 103685, 2022.

[30] G. Bogacsovics, A. Hajdu, and B. Harangi, "Cell segmentation in digitized pap smear images using an ensemble of fully convolutional networks," in *2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, pp. 1–6, IEEE, 2021.

[31] B. Harangi, G. Bogacsovics, J. Toth, I. Kovacs, E. Dani, and A. Hajdu, "Pixel-wise segmentation of cells in digitized pap smear images." Submitted for publication to *Scientific Data*.

[32] E. Ustinova and V. Lempitsky, "Learning deep embeddings with histogram loss," *Advances in neural information processing systems*, vol. 29, 2016.

[33] G. Bogacsovics, B. Harangi, and A. Hajdu, "Increasing the diversity of ensemble members for accurate brain tumor classification," in *2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 529–534, IEEE, 2023.

[34] G. Bogacsovics, B. Harangi, and A. Hajdu, "Developing diverse ensemble architectures for automatic brain tumor classification." Submitted for publication to *Multimedia Tools and Applications*.

[35] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[36] T. Tieleman, G. Hinton, *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] Z. Zainuddin and O. Pauline, "Function approximation using artificial neural networks," *WSEAS Transactions on Mathematics*, vol. 7, no. 6, pp. 333–338, 2008.

[39] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[40] "Flunet — who.int." `https://www.who.int/tools/flunet`. [Accessed 22-10-2023].

[41] "Welcome - Humanitarian Data Exchange — data.humdata.org." `https://data.humdata.org`. [Accessed 22-10-2023].

[42] "World Bank Open Data — data.worldbank.org." `https://data.worldbank.org/indicator/SP.POP.TOTL?name_desc=false`. [Accessed 22-10-2023].

[43] World Health Organization, *Global report on diabetes*. World Health Organization, 2016.

[44] A. D Fleming, S. Philip, K. A Goatman, G. J Prescott, P. F Sharp, and J. A Olson, "The evidence for automated grading in diabetic retinopathy screening," *Current diabetes reviews*, vol. 7, no. 4, pp. 246–252, 2011.

[45] J. Shan and L. Li, "A deep learning method for microaneurysm detection in fundus images," in *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*, pp. 357–358, IEEE, 2016.

[46] J. H. Tan, H. Fujita, S. Sivaprasad, S. V. Bhandary, A. K. Rao, K. C. Chua, and U. R. Acharya, "Automated segmentation of exudates, haemorrhages, microaneurysms using single convolutional neural network," *Information Sciences*, vol. 420, pp. 66–76, 2017.

[47] T. Walter, P. Massin, A. Erginay, R. Ordonez, C. Jeulin, and J.-C. Klein, "Automatic detection of microaneurysms in color fundus images," *Medical image analysis*, vol. 11, no. 6, pp. 555–566, 2007.

[48] G. Quellec, M. Lamard, P. M. Josselin, G. Cazuguel, B. Cochener, and C. Roux, "Optimal wavelet transform for the detection of microaneurysms in retina photographs," *IEEE transactions on medical imaging*, vol. 27, no. 9, pp. 1230–1241, 2008.

[49] C. Agurto, V. Murray, E. Barriga, S. Murillo, M. Pattichis, H. Davis, S. Russell, M. Abramoff, and P. Soliz, "Multiscale AM-FM methods for diabetic retinopathy lesion detection," *IEEE Transactions on Medical Imaging*, vol. 29, pp. 502–512, Feb 2010.

[50] B. Harangi, J. Toth, A. Baran, and A. Hajdu, "Automatic screening of fundus images using a combination of convolutional neural network and hand-crafted features," in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 2699–2702, IEEE, 2019.

[51] T. Zhang, Y. Zeng, and B. Xu, "Hcnn: A neural network model for combining local and global features towards human-like classification," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 30, no. 01, p. 1655004, 2016.

[52] P. Porwal, S. Pachade, R. Kamble, M. Kokare, G. Deshmukh, V. Sahasrabuddhe, and F. Meriaudeau, "Indian diabetic retinopathy image dataset (idrid): A database for diabetic retinopathy screening research," *Data*, vol. 3, no. 3, 2018.

[53] Kaggle Inc., "Diabetic Retinopathy Detection." Accessed: 2021-08-29.

[54] E. Decencière, X. Zhang, G. Cazuguel, B. Lay, B. Cochener, C. Trone, P. Gain, R. Ordonez, P. Massin, A. Erginay, B. Charton, and J.-C. Klein, "Feedback on a publicly distributed database: the messidor database," *Image Analysis & Stereology*, vol. 33, pp. 231–234, Aug 2014.

[55] J. P. Havlicek, *AM-FM image models*. PhD thesis, The University of Texas at Austin, 1996.

[56] B. Antal and A. Hajdu, "Improving microaneurysm detection using an optimally selected subset of candidate extractors and preprocessing methods," *Pattern Recognition*, vol. 45, no. 1, pp. 264 – 270, 2012.

[57] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in *Graphics Gems IV* (P. S. Heckbert, ed.), pp. 474–485, Academic Press Professional, Inc., 1994.

[58] A. Youssif, A. Ghalwash, and A. Ghoneim, "Comparative study of contrast enhancement and illumination equalization methods for retinal vasculature segmentation," in *Cairo International Biomedical Engineering Conference*, pp. 1–5, 2006.

[59] A. Criminisi, P. Perez, and K. Toyama, "Object removal by exemplar-based inpainting," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 721–728, 2003.

[60] T. Walter and J.-C. Klein, "Automatic detection of microaneurysms in color fundus images of the human retina by means of the bounding box closing," in *Medical Data Analysis: Third International Symposium (ISMDA)*, pp. 210–220, 2002.

[61] I. Lazar and A. Hajdu, "Retinal microaneurysm detection through local rotating cross-section profile analysis," *IEEE Transactions on Medical Imaging*, vol. 32, no. 2, pp. 400–407, 2013.

[62] T. Walter, P. Massin, A. Erginay, R. Ordonez, C. Jeulin, and J.-C. Klein, "Automatic detection of microaneurysms in color fundus images," *Medical Image Analysis*, vol. 11, no. 6, pp. 555–566, 2007.

[63] B. Zhang, X. Wu, J. You, Q. Li, and F. Karray, "Detection of microaneurysms using multi-scale correlation coefficients," *Pattern Recognition*, vol. 43, no. 6, pp. 2237–2248, 2010.

[64] B. Nagy, B. Harangi, B. Antal, and A. Hajdu, "Ensemble-based exudate detection in color fundus images," in *Symposium on Image and Signal Processing and Analysis*, pp. 700–703, 2011.

[65] G. D. Finlayson, B. Schiele, and J. L. Crowley, "Comprehensive colour image normalization," in *Computer Vision — ECCV'98* (H. Burkhardt and B. Neumann, eds.), (Berlin, Heidelberg), pp. 475–490, Springer Berlin Heidelberg, 1998.

[66] P. Soille, *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 2004.

[67] A. Sopharak, B. Uyyanonvara, S. Barman, and T. H. Williamson, "Automatic detection of diabetic retinopathy exudates from non-dilated retinal images using mathematical morphology methods," *Comp. Med. Im. Grap.*, vol. 32, no. 8, pp. 720–727, 2008.

[68] T. Walter, J.-C. Klein, P. Massin, and A. Erginay, "A contribution of image processing to the diagnosis of diabetic retinopathy-detection of exudates in color fundus images of the human retina," *Transactions on Medical Imaging*, vol. 21, no. 10, pp. 1236–1243, 2002.

[69] D. Welfer, J. Scharcanski, and D. Marinho, "A coarse-to-fine strategy for automatically detecting exudates in color eye fundus images," *Computerized Medical Imaging and Graphics*, vol. 34, no. 3, pp. 228–235, 2010.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[72] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[73] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[74] "American academy of ophthalmology. international clinical diabetic retinopathy disease severity scale, detailed table." http://www.icoph.org/dynamic/attachments/resources/diabetic-retinopathy-detail.pdf. Accessed: Oct 14, 2016.

[75] S. Gayathri, V. P. Gopi, and P. Palanisamy, "A lightweight cnn for diabetic retinopathy classification from fundus images," *Biomedical Signal Processing and Control*, vol. 62, p. 102115, 2020.

[76] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[77] C. V. Biscotti, A. E. Dawson, B. Dziura, L. Galup, T. Darragh, A. Rahemtulla, and L. Wills-Frank, "Assisted primary screening using the automated thinprep imaging system," *American journal of clinical pathology*, vol. 123, no. 2, pp. 281–287, 2005.

[78] D. Chute, H. Lim, and C. S. Kong, "BD focalpoint slide profiler performance with atypical glandular cells on surepath papanicolaou smears.," *Cancer cytopathology*, vol. 118, no. 2, pp. 68–74, 2010.

[79] G. N. Papanicolaou and H. F. Traut, "The diagnostic value of vaginal smears in carcinoma of the uterus," *Am. J. of Obstet. and Gynec.*, vol. 42, no. 2, pp. 193 – 206, 1941.

[80] R. P. Insinga, A. G. Glass, and B. B. Rush, "Diagnoses and outcomes in cervical cancer screening: A population-based study," *Am. J. of Obstet. and Gynec.*, vol. 191, no. 1, pp. 105 – 113, 2004.

[81] 3DHistech Ltd., "Pannoramic 1000 digital slide scanner." `https://www.3dhistech.com/research/pannoramic-digital-slide-scanners/pannoramic-1000/`. [Online; accessed on March 20, 2022].

[82] libvips contributors, "A fast image processing library with low memory needs." `https://libvips.org`. [Online; accessed on March 20, 2022].

[83] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[84] Z. Lu, G. Carneiro, and A. P. Bradley, "Automated nucleus and cytoplasm segmentation of overlapping cervical cells," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 452–460, Springer, 2013.

[85] B. Harangi, J. Toth, G. Bogacsovics, D. Kupas, L. Kovacs, and A. Hajdu, "Cell detection on digitized pap smear images using ensemble of conventional image processing and deep learning techniques," *11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pp. 38–42, 2019.

[86] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[87] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[88] T. Takikawa, D. Acuna, V. Jampani, and S. Fidler, "Gated-scnn: Gated shape cnns for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5229–5238, 2019.

[89] P. Naylor, M. Laé, F. Reyal, and T. Walter, "Nuclei segmentation in histopathology images using deep neural networks," in *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pp. 933–936, April 2017.

[90] F. Gao, T. Wu, J. Li, B. Zheng, L. Ruan, D. Shang, and B. Patel, "Sd-cnn: A shallow-deep cnn for improved breast cancer diagnosis," *Computerized Medical Imaging and Graphics*, vol. 70, pp. 53–62, 2018.

[91] A. Titoriya and S. Sachdeva, "Breast cancer histopathology image classification using alexnet," in *2019 4th International conference on information systems and computer networks (ISCON)*, pp. 708–712, IEEE, 2019.

[92] M. O. Khairandish, M. Sharma, V. Jain, J. M. Chatterjee, and N. Jhanjhi, "A hybrid cnn-svm threshold segmentation approach for tumor detection and classification of mri brain images," *Irbm*, vol. 43, no. 4, pp. 290–299, 2022.

[93] H. Kibriya, M. Masood, M. Nawaz, and T. Nazir, "Multiclass classification of brain tumors using a novel cnn architecture," *Multimedia Tools and Applications*, vol. 81, no. 21, pp. 29847–29863, 2022.

[94] N. Zhang, Y.-X. Cai, Y.-Y. Wang, Y.-T. Tian, X.-L. Wang, and B. Badami, "Skin cancer diagnosis based on optimized convolutional neural network," *Artificial intelligence in medicine*, vol. 102, p. 101756, 2020.

[95] A. A. Hekal, H. E.-D. Moustafa, and A. Elnakib, "Ensemble deep learning system for early breast cancer detection," *Evolutionary Intelligence*, pp. 1–10, 2022.

[96] N. Gupta, P. Bhatele, and P. Khanna, "Glioma detection on brain mris using texture and morphological features with ensemble learning," *Biomedical Signal Processing and Control*, vol. 47, pp. 115–125, 2019.

[97] Q. H. Nguyen, T. T. Do, Y. Wang, S. S. Heng, K. Chen, W. H. M. Ang, C. E. Philip, M. Singh, H. N. Pham, B. P. Nguyen, *et al.*, "Breast cancer prediction using feature selection and ensemble voting," in *2019 International Conference on System Science and Engineering (ICSSE)*, pp. 250–254, IEEE, 2019.

[98] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[99] S. Lu, S.-H. Wang, and Y.-D. Zhang, "Detection of abnormal brain in mri via improved alexnet and elm optimized by chaotic bat algorithm," *Neural Computing and Applications*, vol. 33, pp. 10799–10811, 2021.

[100] T. H. Arfan, M. Hayaty, and A. Hadinegoro, "Classification of brain tumours types based on mri images using mobilenet," in *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, pp. 69–73, IEEE, 2021.

[101] R. Roslidar, K. Saddami, F. Arnia, M. Syukri, and K. Munadi, "A study of fine-tuning cnn models based on thermal imaging for breast cancer classification," in *2019 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, pp. 77–81, IEEE, 2019.

[102] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[103] H. A. Shah, F. Saeed, S. Yun, J.-H. Park, A. Paul, and J.-M. Kang, "A robust approach for brain tumor detection in magnetic resonance images using finetuned efficientnet," *IEEE Access*, vol. 10, pp. 65426–65438, 2022.

[104] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

[105] K. D. Miller, M. Fidler-Benaoudia, T. H. Keegan, H. S. Hipp, A. Jemal, and R. L. Siegel, "Cancer statistics for adolescents and

young adults, 2020," *CA: a cancer journal for clinicians*, vol. 70, no. 6, pp. 443–459, 2020.

[106] K. D. Miller, Q. T. Ostrom, C. Kruchko, N. Patil, T. Tihan, G. Cioffi, H. E. Fuchs, K. A. Waite, A. Jemal, R. L. Siegel, *et al.*, "Brain and other central nervous system tumor statistics, 2021," *CA: a cancer journal for clinicians*, vol. 71, no. 5, pp. 381–406, 2021.

[107] A. Alentorn, K. Hoang-Xuan, and T. Mikkelsen, "Presenting signs and symptoms in brain tumors," *Handbook of clinical neurology*, vol. 134, pp. 19–26, 2016.

[108] Y. Fan, X. Zhang, C. Gao, S. Jiang, H. Wu, Z. Liu, and T. Dou, "Burden and trends of brain and central nervous system cancer from 1990 to 2019 at the global, regional, and country levels," *Archives of Public Health*, vol. 80, no. 1, pp. 1–14, 2022.

[109] Q. T. Ostrom, M. Price, C. Neff, G. Cioffi, K. A. Waite, C. Kruchko, and J. S. Barnholtz-Sloan, "Cbtrus statistical report: Primary brain and other central nervous system tumors diagnosed in the united states in 2015–2019," *Neuro-oncology*, vol. 24, no. Supplement_5, pp. v1–v95, 2022.

[110] R. L. Siegel, K. D. Miller, N. S. Wagle, and A. Jemal, "Cancer statistics, 2023," *CA: a cancer journal for clinicians*, vol. 73, no. 1, pp. 17–48, 2023.

[111] J. Cheng, "Brain tumor dataset." `https://doi.org/10.6084/m9.figshare.1512427.v5`. Accessed: 2023-06-30.

[112] J. Cheng, W. Yang, M. Huang, W. Huang, J. Jiang, Y. Zhou, R. Yang, J. Zhao, Y. Feng, Q. Feng, *et al.*, "Retrieval of brain tumors by adaptive spatial pooling and fisher vector representation," *PloS one*, vol. 11, no. 6, p. e0157112, 2016.

[113] P. Afshar, K. N. Plataniotis, and A. Mohammadi, "Capsule networks for brain tumor classification based on mri images and coarse tumor boundaries," in *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 1368–1372, IEEE, 2019.

[114] N. Abiwinanda, M. Hanif, S. T. Hesaputra, A. Handayani, and T. R. Mengko, "Brain tumor classification using convolutional neural network," in *World Congress on Medical Physics and Biomedical Engineering 2018: June 3-8, 2018, Prague, Czech Republic (Vol. 1)*, pp. 183–189, Springer, 2019.

[115] M. S. Fasihi and W. B. Mikhael, "Mri brain tumor classification employing transform domain projections," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1020–1023, IEEE, 2020.

[116] P. Afshar, F. Naderkhani, A. Oikonomou, M. J. Rafiee, A. Mohammadi, and K. N. Plataniotis, "Mixcaps: A capsule network-based mixture of experts for lung nodule malignancy prediction," *Pattern Recognition*, vol. 116, p. 107942, 2021.

[117] S. Pattnaik and A. K. Nayak, "Summarization of odia text document using cosine similarity and clustering," in *2019 International Conference on Applied Machine Learning (ICAML)*, pp. 143–146, IEEE, 2019.

[118] R. Singh and S. Singh, "Text similarity measures in news articles by vector space model using nlp," *Journal of The Institution of Engineers (India): Series B*, vol. 102, pp. 329–338, 2021.

[119] S. Pal, M. Chang, and M. F. Iriarte, "Summary generation using natural language processing techniques and cosine similarity," in *International Conference on Intelligent Systems Design and Applications*, pp. 508–517, Springer, 2021.

[120] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660, 2021.

[121] W. Chen, Y. Liu, W. Wang, E. M. Bakker, T. Georgiou, P. Fieguth, L. Liu, and M. S. Lew, "Deep learning for instance retrieval: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[122] G. Somepalli, V. Singla, M. Goldblum, J. Geiping, and T. Goldstein, "Diffusion art or digital forgery? investigating data replication in diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6048–6058, 2023.

[123] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[124] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[125] O. Kaziha and T. Bonny, "A comparison of quantized convolutional and lstm recurrent neural network models using mnist," in *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pp. 1–5, IEEE, 2019.

[126] R. F. Alvear-Sandoval, J. L. Sancho-Gómez, and A. R. Figueiras-Vidal, "On improving cnns performance: The case of mnist," *Information Fusion*, vol. 52, pp. 106–109, 2019.

[127] K. Cheng, R. Tahir, L. K. Eric, and M. Li, "An analysis of generative adversarial networks and variants for image synthesis on mnist dataset," *Multimedia Tools and Applications*, vol. 79, pp. 13725–13752, 2020.

[128] E. Slany, Y. Ott, S. Scheele, J. Paulus, and U. Schmid, "Caipi in practice: towards explainable interactive medical image classification," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 389–400, Springer, 2022.

[129] L. Bhatia and S. Samet, "A decentralized data evaluation framework in federated learning," *Blockchain: Research and Applications*, p. 100152, 2023.

[130] apolanco3225, "Medical mnist classification." `https://github.com/apolanco3225/Medical-MNIST-Classification`, 2017.

[131] F. Chollet, *Deep Learning with Python*. Manning, 2nd ed., Oct. 2021.

List of publications related to the dissertation

Journal articles in English

G. Bogacsovics, B. Harangi, and A. Hajdu, "Developing diverse ensemble architectures for automatic brain tumor classification." Submitted for publication to Multimedia Tools and Applications. (SJR: Q1, IF: 3.6)

B. Harangi, G. Bogacsovics, J. Toth, I. Kovacs, E. Dani, and A. Hajdu, "Pixel-wise segmentation of cells in digitized Pap smear images." Submitted for publication to Scientific Data. (SJR: D1, IF: 4.996)

G. Bogacsovics, J. Toth, A. Hajdu, and B. Harangi, "Enhancing CNNs through the use of hand-crafted features in automated fundus image classification," Biomedical Signal Processing and Control, vol. 76, p. 103685, 2022. (SJR: Q1, IF: 5.1)

G. Bogacsovics, A. Hajdu, R. Lakatos, M. Beregi-Kovács, A. Tiba, and H. Tomán, "Replacing the SIR epidemic model with a neural network and training it further to increase prediction accuracy," in Annales Mathematicae et Informaticae, vol. 53, pp. 73–91, Eszterházy Károly Egyetem Líceum Kiadó, 2021. (SJR: Q4)

Conference papers

G. Bogacsovics, B. Harangi, and A. Hajdu, "Increasing the diversity of ensemble members for accurate brain tumor classification," in 2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS), pp. 529–534, IEEE, 2023.

G. Bogacsovics, A. Hajdu, and B. Harangi, "Cell segmentation in digitized Pap smear images using an ensemble of fully convolutional networks," in 2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), pp. 1–6, IEEE, 2021.

List of other publications

Journal articles in English

R. Lakatos, G. Bogacsovics, B. Harangi, I. Lakatos, A. Tiba, J. Tóth, M. Szabó, and A. Hajdu, "A machine learning-based pipeline for the extraction of insights from customer reviews," Big Data and Cognitive Computing, vol. 8, no. 3, p. 20, 2024. (SJR: Q2)

N. Bátfai, D. Papp, R. Besenczi, G. Bogacsovics, and D. Veres, "Benchmarking cognitive abilities of the brain with the event of losing the character in computer games," Studia Universitatis Babes-Bolyai Informatica, vol. 64, pp. 15–25, 2019.

N. Bátfai, D. Papp, G. Bogacsovics, M. Szabó, V. S. Simkó, M. Bersenszki, G. Szabó, L. Kovács, F. Kovács, and E. S. Varga, "Object file system software experiments about the notion of number in humans and machines.," Cognition, Brain, Behavior, vol. 23, no. 4, 2019. (SJR: Q4)

Journal articles in Hungarian

G. Bogacsovics, A. Hajdu, B. Harangi, I. Lakatos, R. Lakatos, M. Szabó, A. Tiba, J. Tóth, and Á. Tarcsi, "Adatelemzési folyamat és keretrendszer a közigazgatás számára," KözigazgatásTudomány, vol. 1, no. 2, pp. 146–158, 2021.

G. Bogacsovics, A. Hajdu, B. Harangi, I. Lakatos, R. Lakatos, M. Szabó, A. Tiba, and J. Tóth, "Napelemfarmok Magyarország területén történő elhelyezését segítő döntéstámogató rendszer fejlesztése," KözigazgatásTudomány, vol. 1, no. 2, pp. 134–145, 2021.

N. Bátfai, R. Besenczi, J. Szabó, P. Jeszenszky, A. Buda, L. Jármi, R. B. Lovas, M. K. Pál, G. Bogacsovics, and E. Tóthné Kovács, "Deachackers: játszó hackerek, hackelő játékosok, "INFORMÁCIÓS TÁRSADALOM: TÁRSADALOMTUDOMÁNYI FOLYÓIRAT, vol. 18, no. 1, pp. 132–146, 2018. (SJR: Q4, IF: 0.022)

N. Bátfai, G. Bogacsovics, R. Paszerbovics, A. Antal, I. Czevár, and R. Besenczi, "E-sportolók mérése, "INFORMÁCIÓS TÁRSADALOM: TÁRSADALOMTUDOMÁNYI FOLYÓIRAT, vol. 18, no. 1, pp. 147–155, 2018. (SJR: Q4, IF: 0.022)

N. Bátfai, M. Bersenszki, M. Lukács, R. Besenczi, and G. Bogacsovics, “Az e-sport és a robotpszichológia közös jövője, ”INFORMÁCIÓS TÁRSADALOM: TÁRSADALOMTUDOMÁNYI FOLYÓIRAT, vol. 16, no. 4, pp. 26–39, 2016. (SJR: Q3, IF: 0.023)

Conference papers

G. Bogacsovics, B. Harangi, M. Beregi-Kovács, D. Kupás, R. Lakatos, N. D. Serbán, A. Tiba, and J. Tóth, “Assessing conventional and deep learning-based approaches for named entity recognition in unstructured Hungarian medical reports,” in 2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 000077–000082, IEEE, 2024.

R. Lakatos, G. Bogacsovics, and A. Hajdu, “Predicting the direction of the oil price trend using sentiment analysis,” in 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS), pp. 177–182, IEEE, 2022.

B. Harangi, J. Toth, G. Bogacsovics, D. Kupas, L. Kovacs, and A. Hajdu, “Cell detection on digitized Pap smear images using ensemble of conventional image processing and deep learning techniques,” in 2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA), pp. 38–42, IEEE, 2019.

Appendix A: Experimental results for the prediction of COVID-19 cases

This section contains additional experimental results for the proposed two-step architecture mentioned in section 3 for predicting the total number of COVID-19 cases for additional countries. For these tables, we used $n = 3$, meaning that we ran the experiments a total of 3 times, unlike for Germany and Hungary, where we used $n = 10$. This is because while we mainly focused on those countries, we still experimented with others. The summarized results were calculated with a confidence level of 95% ($p = 0.05, n = 3$, using t-statistics) and were rounded to the nearest whole number to allow for easier interpretation.

t_{in}–t_{out}	Model	First wave errors (Austria)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	1515 ± 1319	-	39 ± 17	-
	Plain	12726 ± 11505	-	112 ± 49	-
7–1	Two-step	2295 ± 1322	-	48 ± 14	-
	Plain	8658 ± 11595	-	91 ± 61	-
3–1	Two-step	1599 ± 760	-	40 ± 9	-
	Plain	2746 ± 2390	-	52 ± 22	-
14–7	Two-step	2048 ± 578	2661 ± 1620	45 ± 7	51 ± 16
	Plain	15399 ± 33313	13125 ± 23691	114 ± 146	106 ± 130
14–3	Two-step	2321 ± 2669	1035 ± 1041	47 ± 27	32 ± 17
	Plain	11627 ± 8161	12458 ± 8198	107 ± 39	111 ± 36
7–3	Two-step	1321 ± 1098	2027 ± 1670	36 ± 15	45 ± 18
	Plain	3613 ± 1323	3141 ± 8950	60 ± 11	50 ± 80
t_{in}–t_{out}	Model	Second wave errors (Austria)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	18279 ± 20982	-	132 ± 85	-
	Plain	24285 ± 13590	-	155 ± 43	-
7–1	Two-step	12567 ± 5289	-	112 ± 24	-
	Plain	19281 ± 14585	-	138 ± 51	-
3–1	Two-step	8441 ± 1662	-	91.82 ± 9.00	-
	Plain	9971 ± 1728	-	100 ± 9	-
14–7	Two-step	19226 ± 13849	36166 ± 9576	138 ± 48	190 ± 25
	Plain	54092 ± 102160	44141 ± 19431	221 ± 216	210 ± 45
14–3	Two-step	18191 ± 16938	49582 ± 41319	133 ± 61	221 ± 90
	Plain	22559 ± 7546	27781 ± 15107	150 ± 25	166 ± 47
7–3	Two-step	13648 ± 5141	16498 ± 531	117 ± 22	128 ± 2
	Plain	50651 ± 161764	40733 ± 78861	194 ± 345	192 ± 86

t_{in}–t_{out}	Model	First wave errors (Croatia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	221 ± 106	-	15 ± 4	-
	Plain	393 ± 243	-	20 ± 6	-
7–1	Two-step	225 ± 50	-	15 ± 2	-
	Plain	473 ± 436	-	22 ± 10	-
3–1	Two-step	190 ± 30	-	14 ± 1	-
	Plain	342 ± 211	-	18 ± 6	-
14–7	Two-step	196 ± 65	177 ± 6	14 ± 2	13 ± 0
	Plain	544 ± 796	486 ± 572	23 ± 19	22 ± 12
14–3	Two-step	171 ± 75	159 ± 60	13 ± 3	13 ± 2
	Plain	687 ± 476	393 ± 342	26 ± 10	20 ± 9
7–3	Two-step	197 ± 120	186 ± 55	14 ± 4	14 ± 2
	Plain	655 ± 521	288 ± 174	25 ± 11	17 ± 5
t_{in}–t_{out}	Model	Second wave errors (Croatia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	4975 ± 5811	-	69 ± 43	-
	Plain	4166 ± 301	-	65 ± 2	-
7–1	Two-step	4749 ± 2851	-	69 ± 20	-
	Plain	3324 ± 1122	-	58 ± 10	-
3–1	Two-step	3577 ± 421	-	60 ± 4	-
	Plain	3999 ± 1064	-	63 ± 8	-
14–7	Two-step	4858 ± 1388	7928 ± 2046	70 ± 10	89 ± 12
	Plain	9508 ± 19872	4748 ± 3964	91 ± 108	68 ± 31
14–3	Two-step	4142 ± 2596	5817 ± 3388	64 ± 20	76 ± 22
	Plain	15894 ± 25961	6649 ± 5852	119 ± 124	81 ± 37
7–3	Two-step	3484 ± 986	5307 ± 2669	59 ± 8	73 ± 18
	Plain	15098 ± 26632	8455 ± 6992	115 ± 134	91 ± 37
t_{in}–t_{out}	Model	First wave errors (Japan)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	31174 ± 12836	-	176 ± 37	-
	Plain	26445 ± 7112	-	162 ± 22	-
7–1	Two-step	2173 ± 2016	-	46 ± 23	-
	Plain	3967 ± 2946	-	63 ± 23	-
3–1	Two-step	1163 ± 903	-	34 ± 14	-
	Plain	1789 ± 940	-	42 ± 12	-
14–7	Two-step	1135 ± 984	1175 ± 415	33 ± 15	34 ± 6
	Plain	3862 ± 9162	5228 ± 6190	58 ± 71	71 ± 42
14–3	Two-step	1846 ± 2940	1679 ± 2506	42 ± 32	40 ± 31
	Plain	5591 ± 6236	4202 ± 10618	74 ± 40	59 ± 79
7–3	Two-step	2215 ± 359	3049 ± 681	47 ± 4	55 ± 6
	Plain	2938 ± 4028	3394 ± 3026	52 ± 42	58 ± 25
t_{in}–t_{out}	Model	Second wave errors (Japan)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	31174 ± 12836	-	176 ± 37	-
	Plain	26445 ± 7112	-	162 ± 22	-
7–1	Two-step	36346 ± 14877	-	190 ± 38	-
	Plain	27978 ± 11085	-	167 ± 34	-
3–1	Two-step	30484 ± 3488	-	175 ± 10	-
	Plain	28975 ± 5630	-	170 ± 17	-
14–7	Two-step	34384 ± 11578	97623 ± 257667	185 ± 31	285 ± 392
	Plain	174278 ± 577843	104793 ± 185062	352 ± 683	311 ± 272
14–3	Two-step	43934 ± 16623	149252 ± 46095	209 ± 40	386 ± 60
	Plain	46122 ± 13903	59817 ± 37494	215 ± 32	243 ± 77
7–3	Two-step	37621 ± 9176	44685 ± 5813	194 ± 24	211 ± 14
	Plain	170817 ± 583602	41983 ± 30684	346 ± 686	204 ± 72

t_{in}–t_{out}	Model	First wave errors (Romania)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	2003 ± 1241	-	45 ± 14	-
	Plain	2855 ± 2958	-	53 ± 27	-
7–1	Two-step	1892 ± 1361	-	43 ± 16	-
	Plain	3562 ± 1737	-	60 ± 14	-
3–1	Two-step	2335 ± 751	-	48 ± 8	-
	Plain	2831 ± 1388	-	53 ± 13	-
14–7	Two-step	2094 ± 1347	4507 ± 4763	45 ± 5	66 ± 39
	Plain	2778 ± 2285	2596 ± 1951	52 ± 21	51 ± 20
14–3	Two-step	2121 ± 741	9233 ± 14948	46 ± 8	91 ± 93
	Plain	5256 ± 2532	4891 ± 3756	72 ± 17	69 ± 27
7–3	Two-step	1877 ± 1313	2117 ± 1139	43 ± 15	46 ± 13
	Plain	4089 ± 1796	3371 ± 2518	64 ± 14	58 ± 23
t_{in}–t_{out}	Model	Second wave errors (Romania)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	71878 ± 45012	-	267 ± 86	-
	Plain	98789 ± 96282	-	311 ± 148	-
7–1	Two-step	85973 ± 45136	-	292 ± 77	-
	Plain	110135 ± 55831	-	331 ± 86	-
3–1	Two-step	76013 ± 22300	-	275 ± 40	-
	Plain	111524 ± 45812	-	333.22 ± 67.23	-
14–7	Two-step	47787 ± 31708	235030 ± 303607.15	217 ± 73	471 ± 352
	Plain	57212 ± 8566	190334 ± 274742	239 ± 18	425 ± 299
14–3	Two-step	70061 ± 21365	437849 ± 772868	264 ± 40	617 ± 724
	Plain	118355 ± 92843	128156 ± 64753	341 ± 133	357 ± 88
7–3	Two-step	92618 ± 59595	162631 ± 270817	303 ± 95	390 ± 318
	Plain	109022 ± 48828	175689 ± 294939	329 ± 73	404 ± 334
t_{in}–t_{out}	Model	First wave errors (Slovakia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	364 ± 364	-	19 ± 10	-
	Plain	850 ± 277	-	29 ± 5	-
7–1	Two-step	254 ± 106	-	16 ± 3	-
	Plain	418 ± 232	-	20 ± 6	-
3–1	Two-step	245 ± 43	-	16 ± 1	-
	Plain	273 ± 75	-	17 ± 2	-
14–7	Two-step	365 ± 513	420 ± 208	19 ± 13	20 ± 5
	Plain	910 ± 693	753 ± 723	30 ± 12	27 ± 14
14–3	Two-step	356 ± 381	372 ± 569	19 ± 10	18 ± 17
	Plain	918 ± 334	1429 ± 244	30 ± 5	38 ± 3
7–3	Two-step	359 ± 102	333 ± 262	19 ± 3	18 ± 7
	Plain	391 ± 252	322 ± 302	20 ± 6	18 ± 9
t_{in}–t_{out}	Model	Second wave errors (Slovakia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	14913 ± 7673	-	122 ± 32	-
	Plain	16968 ± 14892	-	129 ± 56	-
7–1	Two-step	17242 ± 6597	-	131 ± 25	-
	Plain	19466 ± 10696	-	139 ± 38	-
3–1	Two-step	16636 ± 1603	-	129 ± 6	-
	Plain	26667 ± 8118	-	163 ± 25	-
14–7	Two-step	4511 ± 1665	9543 ± 6423	67 ± 13	97 ± 33
	Plain	6347 ± 3010	12430 ± 5152	79 ± 19	111 ± 23
14–3	Two-step	12988 ± 3376	17976 ± 13028	114 ± 15	133 ± 47
	Plain	16989 ± 6788	17756 ± 6476	130 ± 26	133 ± 24
7–3	Two-step	13406 ± 951	19663 ± 12020	116 ± 4	140 ± 42
	Plain	30501 ± 60495	25223 ± 15568	166 ± 164	158 ± 49

t_{in}–t_{out}	Model	First wave errors (Slovenia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	78 ± 26	-	9 ± 1	-
	Plain	195 ± 190	-	14 ± 7	-
7–1	Two-step	95 ± 35	-	10 ± 2	-
	Plain	164 ± 142	-	13 ± 6	-
3–1	Two-step	70 ± 6	-	8 ± 0	-
	Plain	120 ± 37	-	11 ± 2	-
14–7	Two-step	84 ± 41	84 ± 36	9 ± 2	9 ± 2
	Plain	169 ± 140	125 ± 140	13 ± 5	11 ± 6
14–3	Two-step	67 ± 38	65 ± 44	8 ± 2	8 ± 3
	Plain	247 ± 106	148 ± 132	16 ± 3	12 ± 6
7–3	Two-step	85 ± 41	93 ± 32	9 ± 2	10 ± 2
	Plain	116 ± 51	133 ± 71	11 ± 2	11 ± 3
t_{in}–t_{out}	Model	Second wave errors (Slovenia)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	1022 ± 930	-	32 ± 15	-
	Plain	1248 ± 457	-	35 ± 7	-
7–1	Two-step	1267 ± 701	-	35 ± 10	-
	Plain	1350 ± 894	-	37 ± 12	-
3–1	Two-step	1107 ± 194	-	33 ± 3	-
	Plain	1506 ± 601	-	39 ± 8	-
14–7	Two-step	543 ± 221	1123 ± 974	23 ± 5	33 ± 14
	Plain	527 ± 137	1117 ± 832	23 ± 3	33 ± 13
14–3	Two-step	786 ± 32	1328 ± 1376	28 ± 1	36 ± 18
	Plain	1202 ± 524	2295 ± 3309	35 ± 8	47 ± 35
7–3	Two-step	1165 ± 829	1303 ± 445	34 ± 12	36 ± 6
	Plain	1545 ± 935	1536 ± 172	39 ± 12	39 ± 2
t_{in}–t_{out}	Model	First wave errors (Switzerland)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	1650 ± 1237	-	40 ± 16	-
	Plain	26768 ± 31289	-	160 ± 25	-
7–1	Two-step	1751 ± 1830	-	41 ± 24	-
	Plain	3134 ± 3334	-	55 ± 29	-
3–1	Two-step	1434 ± 1916	-	37 ± 28	-
	Plain	1830 ± 184	-	43 ± 2	-
14–7	Two-step	10433 ± 28717	19846 ± 30025	91 ± 142	135 ± 121
	Plain	38317 ± 37304	81279 ± 58801	193 ± 101	283 ± 103
14–3	Two-step	4861 ± 3698	8565 ± 6612	69 ± 27	92 ± 37
	Plain	15490 ± 34452	23939 ± 68420	109 ± 181	137 ± 221
7–3	Two-step	2204 ± 1266	1480 ± 1492	47 ± 14	38 ± 21
	Plain	2454 ± 4001	6122 ± 12600	47 ± 45	69 ± 111
t_{in}–t_{out}	Model	Second wave errors (Switzerland)			
		Test set MSE		Test set RMSE	
		Next Day	Aggregated	Next Day	Aggregated
14–1	Two-step	114979 ± 33096	-	339 ± 49	-
	Plain	127892 ± 13207	-	358 ± 19	-
7–1	Two-step	128959 ± 59863	-	358 ± 81	-
	Plain	118014 ± 4629	-	344 ± 7	-
3–1	Two-step	143138 ± 17756	-	378 ± 23	-
	Plain	128063 ± 15330	-	358 ± 21	-
14–7	Two-step	33776 ± 18394	35901 ± 19553	183 ± 49	189 ± 53
	Plain	46882 ± 19284	70863 ± 43875	216 ± 45	265 ± 82
14–3	Two-step	43197 ± 23294	45180 ± 10374	207 ± 56	212 ± 24
	Plain	65517 ± 73767	56636 ± 30540	252 ± 138	237 ± 67
7–3	Two-step	47442 ± 12858	45677 ± 9910	218 ± 30	214 ± 24
	Plain	58207 ± 89426	47090 ± 39493	234 ± 176	215 ± 88