Retinal disease classification based on optical coherence tomography images using convolutional neural networks

Maša Stanojević, Dražen Drašković,* and Boško Nikolić
University of Belgrade, School of Electrical Engineering, Department of Computer Science and Information Technology, Belgrade, Serbia

Abstract. The challenges in today's medicine are progressively more related to the application of artificial intelligence and supervised learning techniques. Optical coherence tomography (OCT) is a noninvasive imaging technology used to obtain high-resolution cross-sectional images of the retina. The layers within the retina can be differentiated and retinal thickness can be measured to facilitate early detection and diagnosis of retinal diseases and conditions. This research paper is aimed at exploring different possibilities of applying deep learning, specifically convolutional neural networks in retinal diseases. During the research, several different architectures—AlexNet, VGG, Inception, and residual network, were evaluated, and the convolutional neural network that proved to be the most successful in the classification was based on the Inception architecture. Hyperparameter tuning was applied as the main method to find the most optimal solution. The key contributions of this research refer to the analysis of different architectures that can be applied in the classification of retinal diseases based on OCT images, as well as the evaluation of the test set obtained by comparing different models with different hyperparameters. This research yielded the best results obtained with Inception1, when training by means of the root mean square propagation optimizer with a batch size of 32, learning rate of $1e^{-6}$, momentum of 0.99, and L2 regularization rate of 0.001. This model achieved an accuracy of 0.95528. In the conclusion of the paper, the advantages of the proposed and implemented solutions were discussed, and a proposal for further improvements was proposed. [DOI: 10.1117/1.JEI.32.3.032004]

Keywords optical coherence tomography; images analysis; deep learning; convolutional neural networks; retinal disease.

Paper 220681SS received Jul. 8, 2022; accepted for publication Nov. 15, 2022; published online Nov. 30, 2022.

1 Introduction

Retinal diseases range from fairly common and easily treatable to quite rare and complex.[1] Early identification and treatment are crucial for preventing vision impairment. Diabetic macular edema (DME) and choroidal neovascularization (CNV) are among the most common blinding retinal diseases.[2] These conditions require urgent referral to an ophthalmologist; if treatment is delayed, there is an increased risk of complications that cause irreversible vision impairment. Drusen, which are lipid deposits present in the dry form of macular degeneration, is a less urgent condition.[3]

Optical coherence tomography (OCT) is a noninvasive imaging technology used to obtain high-resolution cross-sectional images of the retina.[2,4] The layers within the retina can be differentiated and retinal thickness can be measured to facilitate early detection and diagnosis of retinal diseases and conditions.

Machine learning has been used for years in the medical field for aiding medical professionals in diagnosing and classifying diseases.[4] To obtain the most optimal solution the problem solved within the research refers to dataset preparation of more than 77,000 images based on the available source,[3] and the realization of different models based on four most important architectures—AlexNet, VGG, Inception, residual network (ResNet), and the tuning of

*Address all correspondence to Dražen Drašković, drazen.draskovic@etf.bg.ac.rs

hyperparameters. The research is aimed at illustrating a concrete application of convolutional neural networks (CNNs) on classifying OCT images into four categories, namely CNV, DME, drusen, and normal. The main research contributions refer to the analysis of different architectures that can be applied in the classification of retinal diseases based on OCT images, as well as the evaluation of the test set obtained by comparing 64 models with different hyperparameters.

The second section gives a brief introduction of the covered retinal diseases and examines the existing solutions in this field. The dataset used to develop the solution is also mentioned in this section. Section 3 describes the main concepts behind CNNs while covering the most popular CNN network architectures. Section 4 describes the development of the system, which includes dataset preprocessing, network architectures, hyperparameter tuning, training, and evaluation of the models. Section 5 presents results for each family of networks and the overall best-performing model. The conclusion outlines the contribution of this paper and provides some guidelines for future work.

2 Problem Description

This section represents a survey of the nature of the problem in loose medical terms. Additionally, it examines the related work and provides information about the dataset utilized in developing the solution.

The macular edema (ME) occurs when there is abnormal leakage and accumulation of fluid in the macula from damaged blood vessels in the nearby retina.[2] The DME is caused by a complication of diabetes called diabetic retinopathy. It is the most common diabetic eye disease and the leading cause of irreversible blindness in working-age Americans.[5]

Age-related macular degeneration (AMD) is an eye disease that damages the retina, causing vision loss.[3] The most common form of AMD is called the dry form. In the early stages, tiny deposits called drusen begin to appear under the retina. The presence of drusen indicates possible vision loss in the future, either from slowly progressing atrophy or from rapid new blood vessel growth. The risk of vision loss is tied to the number of drusen. The more and the bigger drusen are the likelier vision loss is. Dry AMD can progress to wet AMD, which is more likely to cause a relatively sudden change in vision resulting in serious vision loss.

The CNV involves the growth of new blood vessels that originate from the choroid, which is a vessel-containing layer under the retina. The new vessels, unlike normal ones, are leaky and they allow the fluid from the blood to enter the retina. This fluid distorts the vision and damages the retina, killing the light-sensing cells, called photoreceptors. The CNV occurs in wet AMD.

2.1 *Related Work*

Due to its advancement in recent years, artificial intelligence (AI) has increasingly been applied in the medical field for tasks such as disease classification. There have been numerous studies aimed at detecting and classifying retinal diseases.

Kang et al.[6] developed a deep learning model to detect treatment-requiring retinal vascular diseases using multimodal imaging. The dataset used included images obtained through retinal fundus photography, OCT, and fluorescein angiography. Hong et al.[7] created a hierarchical deep learning framework for classifying multiple visual impairment diseases using a coarse to fine approach. The hierarchy was derived from a predefined hierarchical eye disease taxonomy. The training was carried out using ocular surface and retinal images independently. Perdomo et al.[8] implemented a model for classifying three diabetes-related retinal diseases based on OCT volumes, as well as highlighting relevant scan areas used by the model to classify a specific disease.

All analyzed studies applied exclusively one developed model based on a specific CNN architecture, without fine-tuning hyperparameters. Thus, the aim of this research was to evaluate different models, on different CNN architectures accompanied by fine-tuning. Also, in comparison to the analyzed studies in the field of retinal diseases, a larger dataset was used for the given analysis.

Though there are other contemporary studies dealing with retinal image analysis, still they do not classify retinal diseases. For example, Lal et al.[9] analyzed the adversarial attacks and defenses on the retinal fundus images for the diabetic retinopathy recognition problem. Their results obtained on the retinal fundus images, which are prone to adversarial attacks, represent very promising results.

2.2 *Dataset*

Large dataset of labeled OCT and chest x-ray images[3] is a dataset containing thousands of validated OCT and chest x-ray images. Images are labeled as (disease)—(randomized patient ID)—(image number by this patient) and split into four directories, such as CNV, DME, drusen, and normal.

The OCT images were selected from retrospective cohorts of adult patients from the Shiley Eye Institute of the University of California San Diego, the California Retinal Research Foundation, Medical Center Ophthalmology Associates, the Shanghai First People's Hospital, and Beijing Tongren Eye Center between July 1, 2013, and March 1, 2017. Before training, each image went through a tiered grading system consisting of multiple layers of trained graders of increasing expertise for verification and correction of image labels. Each image imported into the database initially contained a label matching the most recent diagnosis of the patient. The first tier of graders consisted of undergraduate and medical students who had taken and passed an OCT interpretation course review. The evaluators consisted of undergraduate and medical students, four ophthalmologists, and two senior independent retinal specialists.

3 Convolutional Neural Networks

The CNNs are a class of artificial neural networks that are particularly aimed at tasks related to computer vision. These tasks include image classification, object detection, and object segmentation.

One of the main advantages of CNNs is that there is no need for hand-engineered features, which removes the requirement for domain experts when creating networks. The network is capable of extracting significant features by itself, and therefore, problems in various fields can be solved by people having no prior knowledge of the subject matter.

CNNs work by assembling patterns of increasing complexity using small and simple patterns embossed in their filters. They consist of an input layer, several hidden layers, and an output layer. Hidden layers can be either convolutional, pooling, or fully connected layers.

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters, or kernels, which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a two-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Pooling layers reduce dimensionality, by summarizing features in regions of the feature map generated by a convolutional layer. All of the neurons in a particular cluster will be combined into a single neuron in the next layer. The most common types of pooling are max and average. On the one hand, max pooling takes up the maximum value in the region, whereas, on the other hand, average pooling takes up the average value of the region.

Fully connected layers refer to layers every neuron of which is connected to every neuron in the preceding layer. They are usually present in the final layers of a CNN. The flattened matrix goes through fully connected layers, the last of which contains a number of neurons equal to the number of classes. These layers are also called dense.

Nonlinear functions allow CNNs to learn more complex functions. When the activation function is nonlinear, then a two-layer neural network can be proven to be a universal function approximator. This is known as the universal approximation theorem. If all of the layers are linear, the entire network is equivalent to a single-layer model.

Popular functions include Sigmoid, Tanh, rectified linear unit (ReLU), and Leaky ReLU. ReLU is the most common activation function. It is often preferred because it trains the neural network several times faster without a significant penalty to generalization accuracy. It is also less susceptible to the vanishing gradient problem, but it still suffers from saturated neurons.

Hyperparameters are values that are determined prior to training and that greatly influence performance. In order for the model to optimally solve the given problem hyperparameters require tuning. Hyperparameters include: model architecture, batch size, learning rate, number of epochs, activation function (for example: ReLU, Sigmoid, Tanh, Leaky ReLU, etc.), weight initialization, and dropout.

3.1 *Popular CNN Architectures*

ImageNet large scale visual recognition challenge is a famous competition fostering the development and benchmarking of state-of-the-art algorithms. The objective is to classify images into 1000 categories. Some of the architectures that have achieved significant results in this competition include AlexNet, VGG, Inception, and ResNet, and they have been used in many kinds of research for face recognition[10] or image recognition,[11] also in medical decision systems.[12,13]

AlexNet was submitted to the ImageNet competition in 2012. The network achieved a top five error of 15.3%, >10.8 percentage points lower than that of the runner up. Some of the main contributions of the AlexNet paper[14] include using nonsaturating neurons and an efficient GPU implementation of the convolution operation, as well as implementing data augmentation and dropout to address overfitting. Up until AlexNet, the standard for nonlinear activations was either Sigmoid or Tanh. The benefit of using ReLU is reflected in the fact that it enables the training process to be accelerated.

Rather than using relatively large receptive fields (AlexNet used 11×11), VGG network uses a small receptive field of 3×3 consistently throughout the network.[15] The focus was on creating a deeper model, and to that end other parameters of the architecture were fixed. A stack of convolutional layers is followed by three fully connected layers. There are several versions of the architecture, depending on the depth, namely VGG16 and VGG19.

The Inception network was an important milestone in the development of CNN classifiers. The hallmark of the Inception network is the Inception module.[16] Rather than choosing a specific filter size, the Inception module simultaneously uses multiple filter sizes at each layer, concatenating the results of each convolution to create the output.

Before the appearance of Inception network, creating better models relied mostly on going bigger and deeper in terms of architecture and number of layers; however, this lead to overfitting models and increased training time, due to growing number of parameters.

The ResNet[17] addresses the problem of accuracy degradation. The ResNets use skip connections, or shortcuts, to jump over some layers. These shortcuts allow networks to be significantly deeper than previously, without suffering from decaying accuracy.

The fact that accuracy begins to degrade due to an increase in layers implies difficulties for multiple nonlinear layers in approximating the identity mapping. With skip connections, if identity mappings are optimal, the optimizer can drive the weights of the multiple nonlinear layers toward zero thereby approaching identity mappings.

4 Development of the System

This section will describe the used dataset analysis and preparation. Furthermore, the main challenges that appeared during implementation will also be outlined, among which are hyperparameter and model architecture choices. The reasoning behind evaluation metric choices is also explained.

4.1 *Dataset Preprocessing*

As already pointed out, the dataset is comprised of four subfolders, each representing a different class: CNV, DME, drusen, and normal, respectively. Initial dataset exploration showed that all

Table 1 Dataset class distribution.

Class	Original size	Duplicates	Size after
CNV	37,455	5806	31,649
DME	11,598	450	11,148
Drusen	8866	840	8026
Normal	26,465	161	26,304

Table 2 Dataset image dimensions.

Dimensions	Samples
512×496	34,781
768×496	19,553
1536×496	7559
1024×496	244
384×496	16
512×512	14,974

classes contained duplicates that were removed because they hold no value in terms of training. Table 1 shows the initial class distribution, number of duplicates in each class, and the final number of samples.

Image dimensions varied across samples, which presented a problem as traditional CNNs cannot handle images of varying sizes. Table 2 shows the different image dimensions and the number of samples for each. This issue was solved by cropping out the center of each image, and the size of the crop was equal to the smaller dimension of the image. After this step, all of the images were resized to 128×128 pixels.

The dataset was split into train, validation, and test set in the proportion $8:1:1$, using packages for randomly splitting a dataset. Images in the dataset are in the .JPEG format, which indicate three channels. However, the images are grayscale, meaning each channel carries the same information, therefore we can reduce them to a single channel. After being reduced to one channel, images are converted into tensors and normalized.

Standardization (or z-score normalization) is recommended for neural networks. The resulting dataset after standardization has features with a mean of 0 and a standard deviation of 1. These properties help while training, due to the nature of gradient descent. Standardization was performed by obtaining the mean and standard deviation on the training set, then subtracting the mean, and subsequently, dividing by the standard deviation for the whole dataset.

4.2 *Models*

Model architectures were based on the previously mentioned CNNs, namely AlexNet, VGG, Inception, and ResNet. The weight initialization will be explained first in this section.

Two types of initializations used in this research are Xavier and Kaiming initialization. Xavier initialization[18] sets weights to values chosen from a random uniform distribution that's bounded between $\pm \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}$, where n_i is the number of incoming network connections (fan-in) to the layer, and n_{i+1} is the number of outgoing network connections (fan-out). Xavier initialization is the recommended method for VGG networks. Kaiming initialization[19] takes into account the nonlinearity of activation functions, such as ReLU. This initialization function is

$$\text{std} = \sqrt{\frac{2}{(1+a^2) \times \text{fan\_in}}}, \tag{1}$$

where a is the negative slope of the rectifier used after this layer (0 for ReLU by default) and fan_in is the number of input dimensions. There is another mode, fan_out. The difference is that choosing fan_in preserves the magnitude of the variance of the weights in the forward pass, and choosing fan_out preserves the magnitudes in the backward pass.

When using the ReLU activation function, it is desirable that the mean of the weights increments slightly, i.e., layer by layer. This is due to the nature of ReLU: if the input is bigger than zero, it will return the input, otherwise it returns zero. This means that after ReLU, all negative values become zero, thus increasing the mean. With Kaiming initialization, the following two conditions are fulfilled: the mean increments slowly and the standard deviation is close to 1 in the feedforward phase.

Two networks were created inspired by the AlexNet architecture. The networks in this family have a simple structure: stacks of convolutional, ReLU, and max pooling layers, followed by fully connected linear layers with dropout at the end. Filter sizes are the same as the ones used in the original AlexNet, the only differences being the number of channels and the overall number of layers.

Three VGG-inspired architectures were implemented. Two of the architectures have the same number of convolutional layers, only varying in the number of fully connected layers, whereas the third one has two additional convolutions. These networks follow a simple sequential pattern of stacking two or more convolutional and ReLU layers, followed by a max pooling. Filter size is consistent throughout the network and is 3×3. The network ends with fully connected layers.

To create Inception Net inspired models, the Inception module should be created first. The Inception module implemented in this research comes from the original paper by Szegedy et al.[16] Because the appearance of the reference,[16] several other versions of the module have appeared, featuring various performance improvements.

The input of the Inception module goes through four separate computations, the result of which is concatenated. These computations include a 1×1, 3×3, and 5×5 convolution, as well as a max pooling. The 3×3 and 5×5 convolutions are preceded by 1×1 convolutions, which manipulate the number of channels and reduce dimensionality. A 1×1 convolution also follows the max pooling layer. The max pooling layer is padded so that it can be easily added to the output, otherwise its dimensions would be smaller and thus incompatible.

The Inception Net consists of input layers, inception layers, and output layers. The input layers are usually a few convolutional layers, after which come stacks of inception layers with max pooling in between. The number of output channels for each convolution and max pooling must be specified for each Inception module, as well as the number of reduction channels for 3×3 and 5×5 convolutions. The output layers consist of average pooling followed by fully connected layers with dropout.

The main concept behind ResNets is residual blocks. The residual blocks implemented in this research are bottleneck residual blocks, which mean there are three convolutions in the main block manipulating the channel number. The channels are first reduced by a 1×1 convolution before performing the 3×3 convolution, after which they are restored by another 1×1 convolution.

ResNets start with a few convolutional and batch normalization layers, followed by residual blocks. Residual blocks are added using the function which specifies the type and number of residual blocks that will be added. The network ends with an average pooling layer and fully connected layers with dropout.

4.3 *Training*

This section will go over the training process, including hyperparameter tuning. The performance of a neural network greatly depends on the combination of its architecture, optimizer, and learning rate, as well as the problem itself.

4.3.1 *Batch size*

Batch size refers to the number of samples utilized in a single iteration of the algorithm. In mini batch gradient descent (BGD), batch size falls somewhere between 1 [stochastic gradient descent (SGD)] and the entire dataset (BGD). On the one hand, BGD computes the gradient using the whole dataset in which case one moves directly towards an optimum solution, either a local or global. This method is good for convex or relatively smooth error manifolds. However, calculating the gradient over the whole dataset results in slower model updates and training speed.

On the other hand, SGD works better for error manifolds that have a lot of local maxima/minima. The noisier gradient calculated using a single sample can pull the model out of local minima and advances much faster due to the batch size. However, this approach tends to be quite noisy, making it hard for the algorithm to settle on an error minimum for the model.

A good balance is achieved when the minibatch size is small enough to avoid some of the poor local minima, but large enough that it does not avoid the global minima or better-performing local minima. Mini BGD has higher model update frequency than BGD, which allows for a more robust convergence thereby avoiding local minima. The batched updates provide a computationally more efficient process than stochastic gradient descent. Larger mini-batches increase computational parallelism; however, smaller batch sizes have been shown to provide improved generalization while having a significantly smaller memory footprint.

Some guidelines regarding batch size selection mention using power of 2 values, as well as using 32 as a default.[20] The paper by Masters and Luschi showed that ImageNet performs best with batches between 16 and 64 with values 16 and 32 performing best for a wide range of learning rates.[21] Backed by this analysis, this research uses the default batch size of 32, while experimenting with batch size of 16 where training time complexity allowed.

4.3.2 *Epochs*

An epoch defines the number of times that the learning algorithm works through the entire training dataset. This number was originally set to 100, which was enough for most networks to sufficiently minimize the error. However, VGG-inspired networks needed more epochs to reach convergence, so the number of epochs was increased to 200, because convergence was not achieved for a value of 100. Early stopping was implemented to prevent overfitting. Early stopping is a form of regularization.

4.3.3 *Optimizer*

There is no single optimizer that is best for a specific network architecture, and the optimal performance also depends on the learning rate. According to Choi et al.[22] more general optimizers never underperform special cases. They found that, when carefully tuned, Adam and other adaptive algorithms never underperformed momentum or SGD. It should be pointed out that Adam and root mean square propagation (RMSprop) optimizers were used in this research.

There is a plethora of optimization algorithms for CNNs, out of which RMSprop and Adam are among the most popular. RMSprop was proposed by Geoffrey Hinton in his course on neural networks.[23] RMSprop is a minibatch version of Rprop.[24] The idea behind Rprop, short for resilient backpropagation, is to use only the sign of the derivative when updating the weights. For each weight, if there was a sign change of the partial derivative of the total error function compared to the last iteration, the update value for that weight is multiplied by a factor $\eta-$ where $\eta- < 1$. If the last iteration produced the same sign, the update value is multiplied by a factor of $\eta+$ where $\eta+ > 1$. The update values are calculated for each weight in the above manner, and finally each weight is changed by its own update value. However, this approach does not work well with minibatches, due to frequent updating of the weights based on noisy gradients.

To solve this, RMSprop keeps a moving average of the squared gradient for each weight and then divides the learning rate by the square root of that average

$$v_t = \beta v_{t-1} + (1-\beta) g_t{}^2, \tag{2}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{v_t}} g_t, \tag{3}$$

where v_t is the moving average of squared gradients, g_t is the gradient of the cost function with respect to the weight, η is the learning rate, and β is the moving average parameter (good default value suggested by Hinton is 0.9).

The PyTorch implementation takes the square root of the gradient average before adding ε, which is a small term added to the denominator to improve numerical stability, making the effective learning rate $\eta/\sqrt{v_t + \varepsilon}$.

Adam, short for adaptive moment estimation, is an update of the RMSprop optimizer. Adam is an adaptive learning rate method, meaning it computes individual learning rates for different parameters. It is straightforward to implement, computationally efficient and has little memory requirements.[25]

In addition to storing an exponentially decaying average of past squared gradients, such as RMSprop, Adam also keeps an exponentially decaying average of past gradients. According to the source paper, good default settings are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$, and $\varepsilon = 10^{-8}$ where α is the learning rate; β_1 and β_2 are decay rates; and ε is a small rate that prevents division by zero. We compute the decaying averages of past and past squared gradients m_t and v_t as follows

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{4}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{5}$$

where m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients. As they are initialized as 0 vectors, the authors have observed them to be biased towards zero, especially during the initial timesteps and when the decay rates are small (β_1 and β_2 are close to 1). This initialization bias can be easily counteracted, resulting in bias-corrected estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{6}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{7}$$

These values are then used to update parameters

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t. \tag{8}$$

4.3.4 *Learning rate*

Learning rate is perhaps the most important hyperparameter.[20,26] It controls the speed at which a model learns by controlling the amount of the error that gets applied to the weights. The range of values to consider when tuning the learning rate is <1 and $>10^{-6}$. However, smaller batch sizes are usually more suitable to smaller learning rates, due to the noisy estimate of the gradient error.

4.3.5 *Weight decay*

According to the paper by Loshchilov and Hutter,[26] L2 regularization and weight decay are equivalent only for standard stochastic gradient descent, but this is not the case for adaptive algorithms such as Adam. Common implementations of these algorithms use L2 regularization, mistakenly referring to it as weight decay. Therefore, in the remainder of this paper, PyTorch optimizer weight decay will be referred to as L2 regularization.

In the course of the training process, regularization was added whenever a network showed tendency to overfit to data. The idea behind L2 regularization is that networks with smaller

weights are observed to overfit less and generalize better. The regularization rate determines the trade-off between minimizing the loss function and keeping weights small. L2 regularization consists in adding to the loss function the sum of the squares of all the weights of the model, multiplied by the regularization parameter.

4.3.6 *Momentum*

RMSprop takes momentum as an optional parameter. Momentum is a method that helps accelerate optimizers in the relevant direction and dampens oscillations. Momentum adds inertia to the updates by causing many past updates in one direction to continue in that direction. It works by adding a fraction γ of the update vector of the past time step to the current update vector.

4.3.7 *Loss function*

When it comes to multiclass classification problems, softmax cross-entropy loss is the recommended loss function. Cross-entropy loss measures the performance of a classification model, the output of which is a probability between 0 and 1. Moreover, it increases as the predicted probability diverges from the actual label.

The PyTorch implementation implicitly adds a softmax that normalizes the output layer into a probability distribution. The loss function is defined as

$$\text{loss}(x, \text{class}) = -\log\left(\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])}\right) = -x[\text{class}] + \log\left(\sum_j \exp(x[j])\right). \tag{9}$$

4.4 *Evaluation*

Evaluation metrics quantify the performance of a classifier and also play a crucial role in guiding further modeling. Metrics are calculated on an unseen test dataset to compare different combinations of models and hyperparameters.

When it comes to multiclass classification, accuracy is not a good choice of a metric, especially in unbalanced datasets. Accuracy as an evaluation metric makes sense only if the class labels are uniformly distributed. As previously mentioned, the dataset used in this research is unbalanced, which makes the choice of evaluation metric an important one.

In this case a confusion-matrix is a good technique to summarize the performance of a classification algorithm. A confusion-matrix C is a square matrix where C_{ij} indicates the number of instances, which are known to belong to class i (true label), but were classified as class j (predicted label).

Other precision metrics that work well for unbalanced datasets are precision, recall, and F-measure. Precision summarizes the fraction of examples assigned the positive class that belong to the positive class

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}. \tag{10}$$

Recall summarizes how well the positive class was predicted and is the same calculation as sensitivity

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}}. \tag{11}$$

Precision and recall can be combined into a single score that seeks to balance both concerns, called the F-score or the F-measure. It is a popular metric for imbalanced classification

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{12}$$

Precision, recall, and F-measure are calculated per class and then averaged in one of two ways: micro and macro. A microaverage aggregates the contributions of all classes to compute the average metric, whereas a macroaverage computes the metric for each class independently and then takes the average. Macroaveraging has been selected in this paper because it gives equal importance to each class, whereas microaverage assigns equal importance to each sample, thus preferring the majority class.

Another useful metric is ROC AUC score, which stands for area under the receiver operating characteristic curve. A ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate against the false positive rate at various threshold settings.

The ROC curve is applicable to binary classification, to apply it to multiclass classification we must choose either one-vs-rest or one-vs-one mode. One-vs-rest computes the AUC of each class against the rest. This mode is sensitive to class imbalance even when macro-averaging. One-vs-one computes the average AUC of all possible pairwise combinations of classes and is insensitive to class imbalance when using macroaveraging, which makes it the better choice for this problem.

5 Experimental Results

This section will outline the findings and resulting metrics for each family of networks separately. The best-performing instances from each class will be compared to determine the advantages/disadvantages of each. The final model will be described in this section as well.

Every model has a unique name based on the architecture, serial number, optimizer, batch size, learning rate, momentum (relevant only when using RMSprop), and L2 regularization rate. For example, model AlexNet1_Adam_16_lr0.001_wd0.001 is a model inspired by the AlexNet architecture, trained using Adam with a batch size of 16, learning rate of 0.001 and L2 regularization rate of 0.001.

There are two models for each combination of network architecture and hyperparameters. The first one is achieved at the lowest validation loss, whereas the second one is achieved at the end of training process. The better of the two is included in the final assessment.

5.1 *AlexNet*

Unsurprisingly, networks belonging to this family were among the worst performing models, but they are simple and fast to train, which allowed for a large number of different hyperparameter combinations to be tested.

Figure 1 demonstrates the advantage of using momentum with RMSprop. The momentum factor was set to 0.99, which is a standard value. AlexNet2 converged significantly faster with momentum than without.

From Fig. 2, we can conclude that AlexNet2, the deeper variation of the network, was able to learn and reach convergence faster than AlexNet1 for the same values of hyperparameters. Figure 3 compare train losses for AlexNet1 and AlexNet2 with batch sizes 16 and 32. For both networks, it can be concluded that a batch size of 32 achieves better results.

From Fig. 4, it can be concluded that both RMSprop and Adam benefited from smaller learning rates. Table 3 shows combinations of model architectures and hyperparameters, and Table 4 displays the achieved results for the tested models. Only models that were able to converge are listed. Both networks were unable to converge when trained with RMSprop with learning rate 0.0001, as well as when trained with Adam with a learning rate 0.01.

It can be concluded from this table that the best performing model on the test set, based on every metric, is AlexNet1, trained using Adam with a learning rate of 0.0001 and a L2 regularization rate of 0.001. It achieved a precision of 0.891, recall of 0.893, and

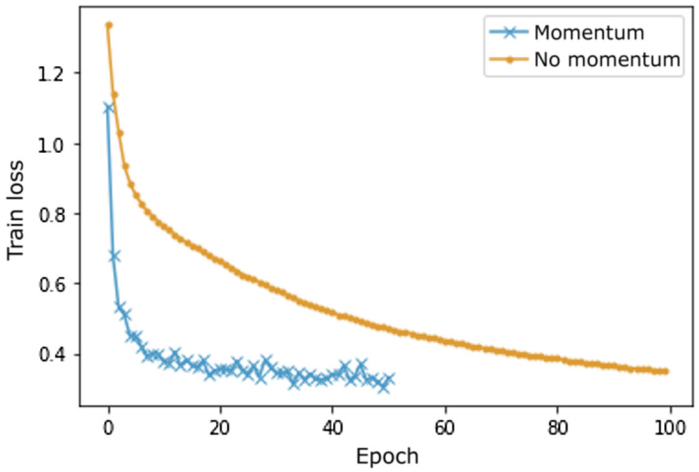

Fig. 1 Train loss comparison for AlexNet2 RMSprop_32_lr1e-5_wd0.001 with and without momentum.

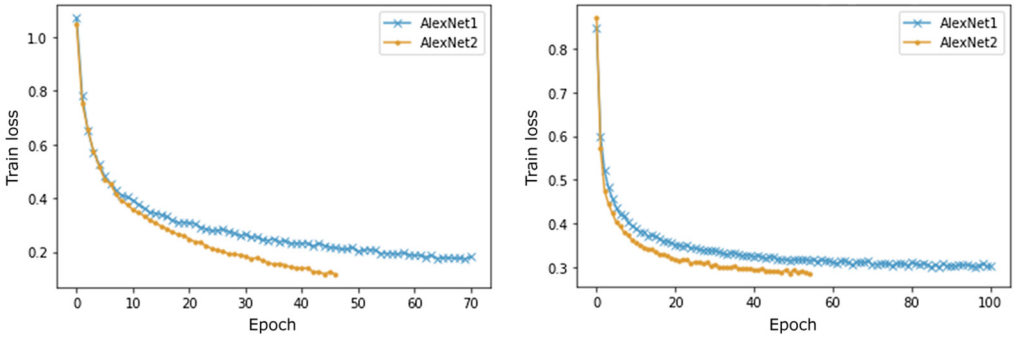

Fig. 2 Train loss comparison for AlexNet1 and AlexNet2—RMSprop_32_lr1e-06_m0.99_wd0.001 (left) and Adam_32_lr0.001_wd0.001 (right).

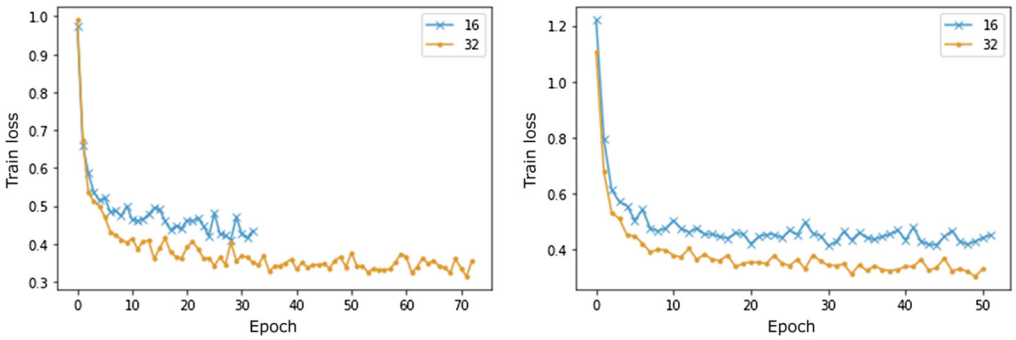

Fig. 3 Train loss comparison for different batch sizes—AlexNet1 RMSprop_lr1e-05_m0.99_wd0.001 (left) and AlexNet2 RMSprop_lr1e-05_m0.99_wd0.001 (right).

an F-score of 0.891. This model is followed closely by AlexNet2 trained with the same hyperparameters.

Based on the confusion matrix shown in Fig. 5, it can be observed that the network has the worst performance classifying drusen, which is most likely due to the fact that drusen is the minority class in the dataset and thus the network saw this class the least.

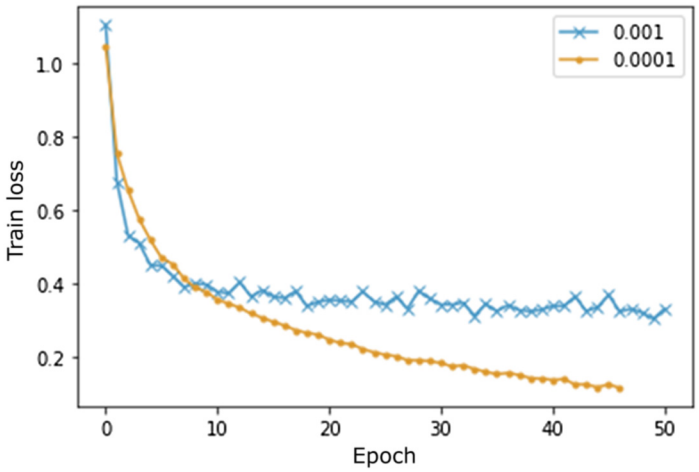

Fig. 4 Train loss comparison for different learning rates—AlexNet2 RMSprop_32_m0.99_wd0.001.

Table 3 Hyperparameters in the tested models based on the AlexNet architecture.

Model	Architecture	Optimizer	Batch size	Learning rate	Weight decay	Momentum
AlexNet1_Adam_32_lr0.0001_wd0.001	AlexNet1	Adam	32	0.0001	0.001	
AlexNet2_Adam_32_lr0.0001_wd0.001	AlexNet2	Adam	32	0.0001	0.001	
AlexNet2_RMSprop_32_lr1e-06_m0.99_wd0.001	AlexNet2	RMSprop	32	1.00E-06	0.001	0.99
AlexNet1_RMSprop_32_lr1e-06_m0.99_wd0.001	AlexNet1	RMSprop	32	1.00E-06	0.001	0.99
AlexNet2_Adam_32_lr0.001_wd0.001	AlexNet2	Adam	32	0.001	0.001	
AlexNet1_Adam_32_lr0.001_wd0.001	AlexNet1	Adam	32	0.001	0.001	
AlexNet1_Adam_16_lr0.001_wd0.001	AlexNet1	Adam	16	0.001	0.001	
AlexNet2_RMSprop_32_lr1e-05_m0.99_wd0.001	AlexNet2	RMSprop	32	1.00E-05	0.001	0.99
AlexNet1_RMSprop_32_lr1e-05_m0.99_wd0.001	AlexNet1	RMSprop	32	1.00E-05	0.001	0.99
AlexNet2_RMSprop_16_lr1e-05_m0.99_wd0.001	AlexNet2	RMSprop	16	1.00E-05	0.001	0.99
AlexNet2_RMSprop_32_lr1e-05_m0_wd0.001	AlexNet2	RMSprop	32	1.00E-05	0.001	0
AlexNet1_RMSprop_16_lr1e-05_m0.99_wd0.001	AlexNet1	RMSprop	16	1.00E-05	0.001	0.99

Table 4 AlexNet metrics.

Model	Precision	Recall	F-score	ROC AUC	Accuracy
AlexNet1_Adam_32_lr0.0001_wd0.001	0.89106	0.89307	0.8915	0.98404	0.92339
AlexNet2_Adam_32_lr0.0001_wd0.001	0.8841	0.88806	0.88578	0.98194	0.91755
AlexNet2_RMSprop_32_lr1e-06_m0.99_wd0.001	0.85703	0.89387	0.87226	0.98347	0.90407
AlexNet1_RMSprop_32_lr1e-06_m0.99_wd0.001	0.85333	0.89362	0.87007	0.98268	0.90161
AlexNet2_Adam_32_lr0.001_wd0.001	0.85327	0.88934	0.86869	0.98105	0.90096
AlexNet1_Adam_32_lr0.001_wd0.001	0.85419	0.88465	0.86695	0.98029	0.90018
AlexNet1_Adam_16_lr0.001_wd0.001	0.85284	0.8796	0.86434	0.97887	0.89837
AlexNet2_RMSprop_32_lr1e-05_m0.99_wd0.001	0.87228	0.84842	0.85664	0.9731	0.90018
AlexNet1_RMSprop_32_lr1e-05_m0.99_wd0.001	0.83108	0.8778	0.84953	0.97935	0.88411
AlexNet2_RMSprop_16_lr1e-05_m0.99_wd0.001	0.83107	0.86326	0.84445	0.97429	0.88229
AlexNet2_RMSprop_32_lr1e-05_m0_wd0.001	0.81596	0.85426	0.83133	0.97086	0.87179
AlexNet1_RMSprop_16_lr1e-05_m0.99_wd0.001	0.80738	0.85835	0.82465	0.97232	0.86349

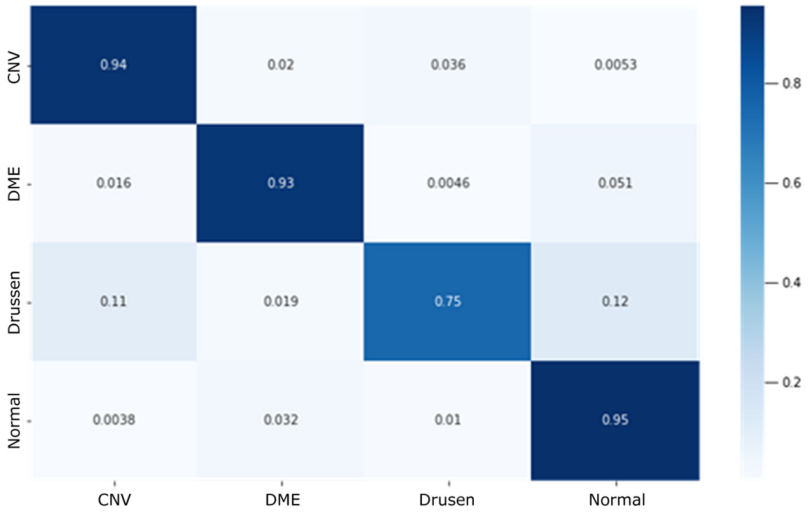

Fig. 5 AlexNet1_Adam_32_lr0.0001_wd0.001 confusion matrix.

5.2 *VGG*

These networks showed a great sensitivity to the combination of optimizer and learning rate. Namely, they were unable to converge using a learning rate of 0.001 when Adam was used. Next, for RMSprop they were unable to converge for both $1e^{-4}$ and $1e^{-5}$, only converging for $1e^{-6}$ in case of VGG1 and VGG2. However, when they did converge, their performance was superior to that of AlexNet inspired networks. In addition to this, they took significantly more epochs to train when compared with AlexNet networks.

Table 5 displays the number of parameters and the number of layers for each network in this family. The number of layers takes into account only layers with trainable parameters.

Figure 6 shows that VGG3, the deepest network, performed the best in terms of training. VGG2 outperformed VGG1 for both batch sizes, although it has approximately 2.6 times less

Table 5 Number of parameters and layers per network.

Model	Parameters	Layers
VGG1	470,276	7
VGG2	178,100	6
VGG3	304,052	9

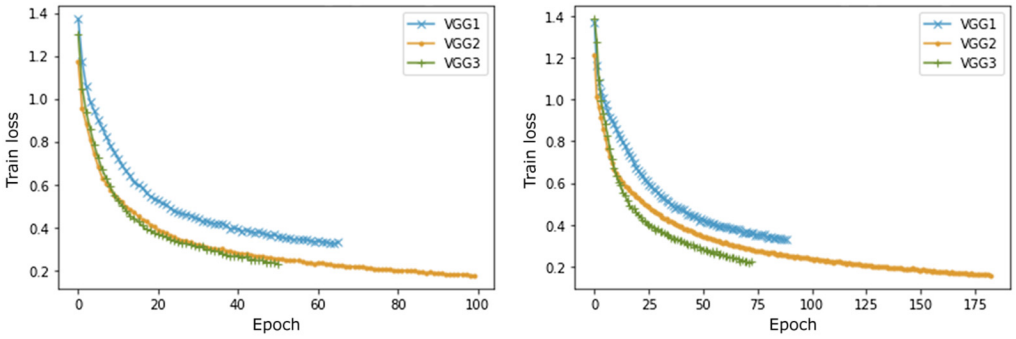

Fig. 6 Train loss comparison for VGG1, VGG2, and VGG3—Adam_16_lr0.0001_wd0 (left) and Adam_32_lr0.0001_wd0 (right).

Table 6 Hyperparameters in the tested models based on the VGG architecture.

Model	Architecture	Optimizer	Batch size	Learning rate	Weight decay	Momentum
VGG3_Adam_16_lr0.0001_wd0	VGG3	Adam	16	0.0001	0	
VGG2_Adam_32_lr0.0001_wd0	VGG2	Adam	32	0.0001	0	
VGG2_Adam_16_lr0.0001_wd0	VGG2	Adam	16	0.0001	0	
VGG3_Adam_32_lr0.0001_wd0	VGG3	Adam	32	0.0001	0	
VGG1_Adam_16_lr0.0001_wd0	VGG1	Adam	16	0.0001	0	
VGG1_Adam_32_lr0.0001_wd0	VGG1	Adam	32	0.0001	0	
VGG2_RMSprop_32_lr1e-06_m0.99_wd0.001	VGG2	RMSprop	32	1.00E-06	0.001	0.99

trainable parameters. This difference in parameters occurs mostly due to the fact that VGG2 contains one poorly connected layer, which is probably why it took the most epochs to train.

Table 6 shows the analyzed combinations of hyperparameters for VGG architecture. Table 7 shows metrics for every VGG-based model that converged for the given hyperparameters. The best performing model is VGG3, trained with Adam, using a batch size of 16, learning rate of 0.0001 and no L2 regularization.

From Fig. 7, it can be observed that the model has the worst performance on classification of the minority class.

5.3 *Inception*

Inception networks converged for every tried combination of optimizer and learning rate and generally performed better than both AlexNet and VGG inspired networks. Figure 8 shows that for Inception Nets as well, smaller learning rates perform better in terms of training.

Table 7 VGG metrics.

Model	Precision	Recall	F-score	ROC AUC	Accuracy
VGG3_Adam_16_lr0.0001_wd0	0.91152	0.93648	0.92297	0.98645	0.94361
VGG2_Adam_32_lr0.0001_wd0	0.91436	0.93066	0.92177	0.99164	0.94257
VGG2_Adam_16_lr0.0001_wd0	0.906	0.93408	0.91865	0.99239	0.93985
VGG3_Adam_32_lr0.0001_wd0	0.90318	0.93449	0.91721	0.98806	0.93842
VGG1_Adam_16_lr0.0001_wd0	0.90846	0.92264	0.91523	0.98805	0.93868
VGG1_Adam_32_lr0.0001_wd0	0.89958	0.92269	0.91	0.98834	0.93376
VGG2_RMSprop_32_lr1e-06_m0.99_wd0.001	0.88769	0.9214	0.90232	0.98965	0.9265

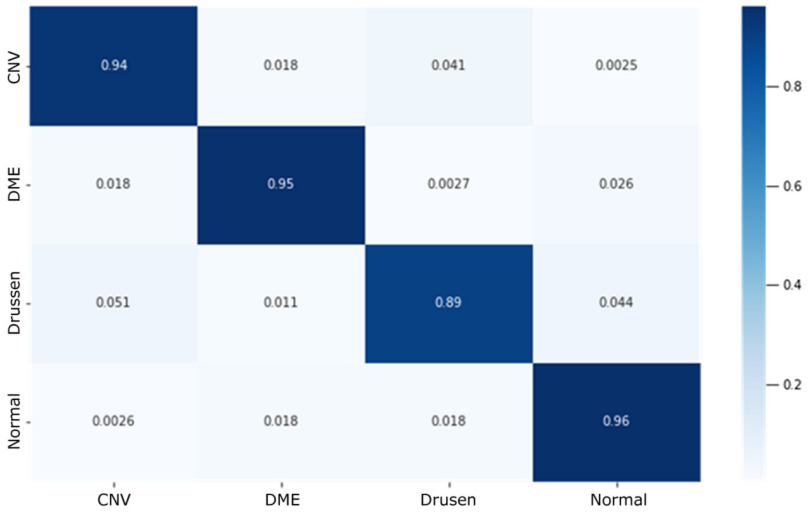

Fig. 7 Model confusion matrix.

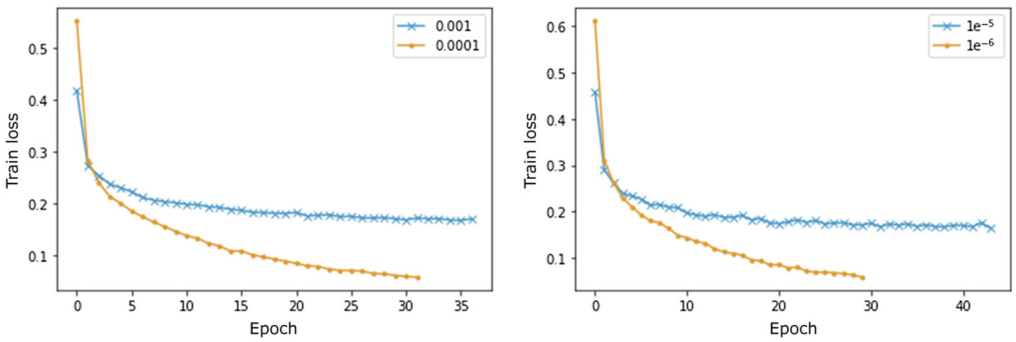

Fig. 8 Train loss comparison for different learning rates—Inception1 Adam_32_wd0.001 (left) and Inception1 RMSprop_32_m0.99_wd0.001 (right).

Based on Fig. 9, it can be concluded that Inception2 performed only marginally better than Inception1 in the process of training, despite having two additional Inception modules.

Tables 8 and 9 display the analyzed combinations of hyperparameters as well as the achieved metrics for each tried combination of Inception architecture. The network that performed the best

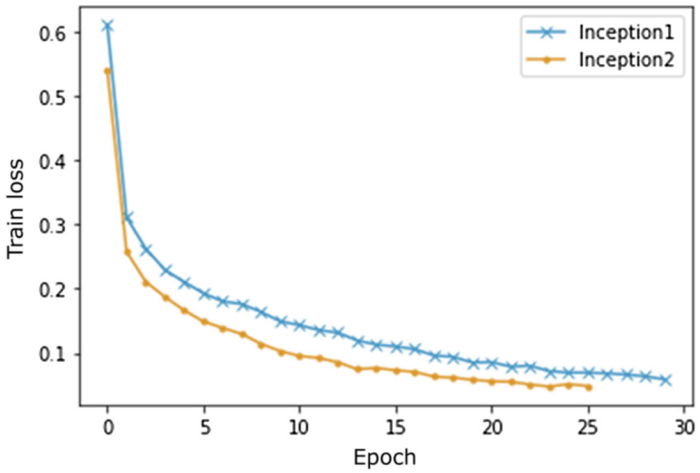

Fig. 9 Train loss comparison for Inception1 and Inception2—RMSprop_32_lr1e-6_m0.99_wd0.001.

Table 8 Hyperparameters in the tested models based on the Inception architecture.

Model	Architecture	Optimizer	Batch size	Learning rate	Weight decay	Momentum
Inception1_RMSprop_32_lr1e-06_m0.99_wd0.001	Inception1	RMSprop	32	1.00E-06	0.001	0.99
Inception2_RMSprop_32_lr1e-06_m0.99_wd0.001	Inception2	RMSprop	32	1.00E-06	0.001	0.99
Inception1_RMSprop_32_lr1e-05_m0.99_wd0.0001	Inception1	RMSprop	32	1.00E-05	0.0001	0.99
Inception2_RMSprop_32_lr1e-06_m0.99_wd0.01	Inception2	RMSprop	32	1.00E-06	0.01	0.99
Inception1_Adam_32_lr0.001_wd0.001	Inception1	Adam	32	0.001	0.001	
Inception1_Adam_32_lr0.0001_wd0.001	Inception1	Adam	32	0.0001	0.001	
Inception1_RMSprop_32_lr1e-05_m0.99_wd0.001	Inception1	RMSprop	32	1.00E-05	0.001	0.99

Table 9 Inception metrics.

Model	Precision	Recall	F-score	ROC AUC	Accuracy
Inception1_RMSprop_32_lr1e-06_m0.99_wd0.001	0.9357	0.9381	0.93687	0.99291	0.95528
Inception2_RMSprop_32_lr1e-06_m0.99_wd0.001	0.93109	0.94071	0.93576	0.99299	0.95359
Inception1_RMSprop_32_lr1e-05_m0.99_wd0.0001	0.92138	0.94736	0.93257	0.99396	0.95048
Inception2_RMSprop_32_lr1e-06_m0.99_wd0.01	0.92394	0.93931	0.93127	0.99202	0.95035
Inception1_Adam_32_lr0.001_wd0.001	0.92568	0.92941	0.92729	0.99166	0.9475
Inception1_Adam_32_lr0.0001_wd0.001	0.91285	0.93625	0.92295	0.99249	0.94296
Inception1_RMSprop_32_lr1e-05_m0.99_wd0.001	0.9055	0.93504	0.91803	0.99255	0.93842

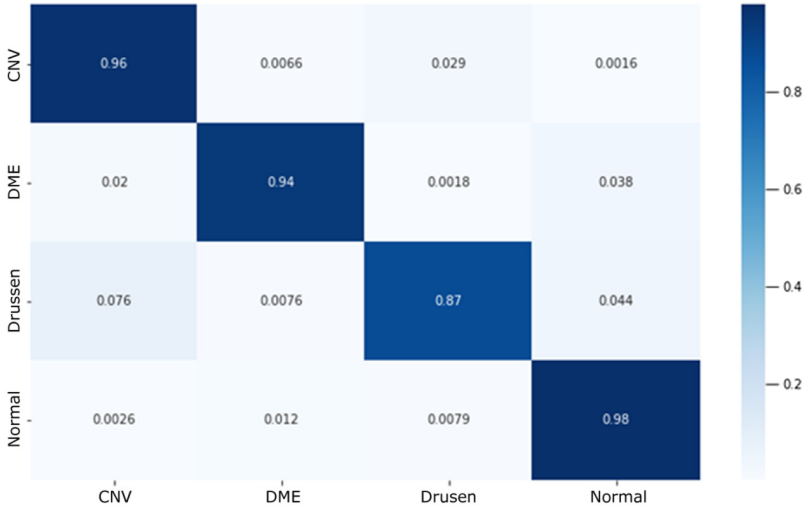

Fig. 10 Inception1_RMSprop_32_lr1 e-06_m0.99_wd0.001 confusion matrix.

is Inception1 trained using the RMSprop optimizer, with a batch size of 32, learning rate of $1e^{-6}$, momentum factor of 0.99 and a L_2 regularization rate of 0.001. Second best network is Inception2 with the same hyperparameters.

It can be seen from Fig. 10 that although the network has high metrics, it classifies the minority class quite poorly. This issue can be resolved by obtaining more samples for the minority class or, if this is not possible, then by means of oversampling the minority class while training.

5.4 *Residual Network*

The implemented ResNet was resilient to different combinations of optimizer and learning rate, and was able to converge every time, with similar results. From Fig. 11, it can be deduced that smaller learning rates train faster, for both Adam and RMSprop.

Each hyperparameter combination for ResNet architecture is shown in Table 10, followed by the metrics achieved by ResNet that are displayed in Table 11. The best performing network, by a small margin, is ResNet trained using the RMSprop optimizer, with a batch size of 32, learning rate of $1e^{-5}$, momentum factor of 0.99 and a L2 regularization rate of 0.0001.

Figure 12 shows that ResNet is much better at accurately classifying samples. It achieves 92% classification accuracy even for the minority class, without any data augmentation.

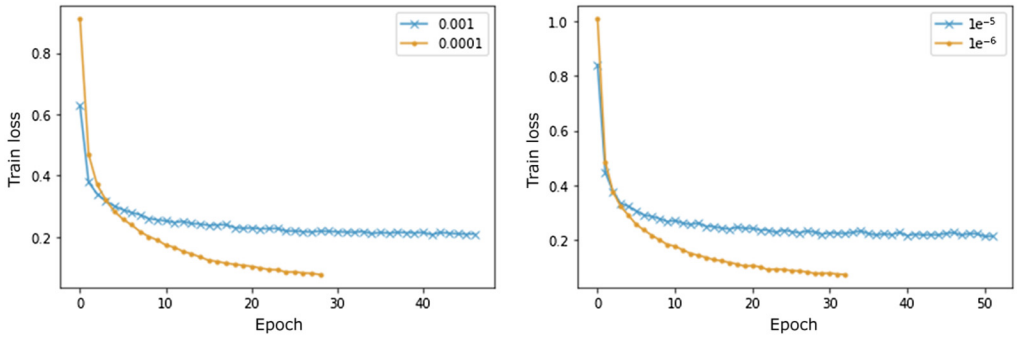

Fig. 11 Train loss comparison for different learning rates—ResNet Adam_32_wd0.001 (left) and ResNet RMSprop_32_m0.99_wd0.001 (right).

Table 10 Hyperparameters in the tested models based on the ResNet architecture.

Model	Architecture	Optimizer	Batch size	Learning rate	Weight decay	Momentum
ResNet_RMSprop_32_lr1e-05_ m0.99_wd0.0001	ResNet	RMSprop	32	1.00E-05	0.0001	0.99
ResNet_Adam_32_lr0.0001_wd0.01	ResNet	Adam	32	0.0001	0.01	
ResNet_Adam_32_lr0.0001_wd0.001	ResNet	Adam	32	0.0001	0.001	
ResNet_RMSprop_32_lr1e-06_ m0.99_wd0.001	ResNet	RMSprop	32	1.00E-06	0.001	0.99
ResNet_Adam_32_lr0.001_wd0.001	ResNet	Adam	32	0.001	0.001	
ResNet_RMSprop_32_lr1e-05_ m0.99_wd0.001	ResNet	RMSprop	32	1.00E-05	0.001	0.99

Table 11 ResNet metrics.

Model	Precision	Recall	F-score	ROC AUC	Accuracy
ResNet_RMSprop_32_lr1e-05_m0.99_wd0.0001	0.90358	0.93603	0.9176	0.99144	0.93881
ResNet_Adam_32_lr0.0001_wd0.01	0.90859	0.92695	0.91724	0.99019	0.9392
ResNet_Adam_32_lr0.0001_wd0.001	0.91271	0.91719	0.91436	0.98873	0.93803
ResNet_RMSprop_32_lr1e-06_m0.99_wd0.001	0.89502	0.92633	0.90893	0.98907	0.93285
ResNet_Adam_32_lr0.001_wd0.001	0.89624	0.92249	0.90654	0.99009	0.93065
ResNet_RMSprop_32_lr1e-05_m0.99_wd0.001	0.88806	0.92492	0.90389	0.99043	0.92715

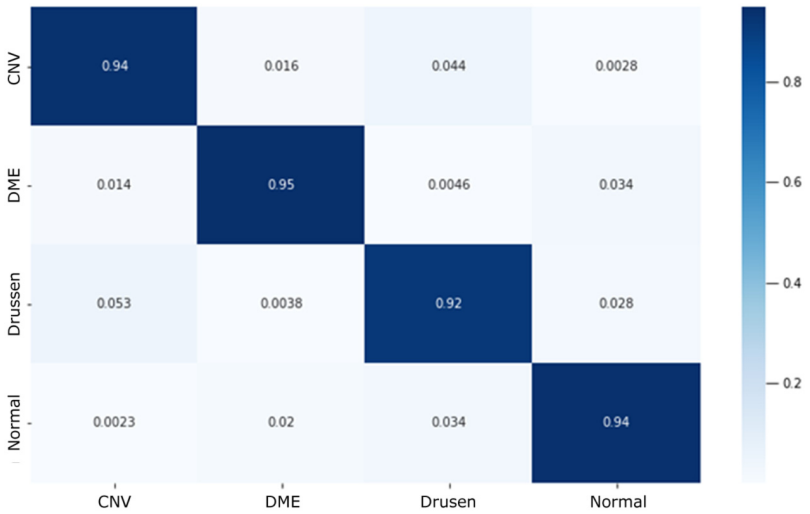

Fig. 12 ResNet_RMSprop_32_lr1e-05_m0.99_wd0.0001 confusion matrix.

5.5 *Final Model*

The best performing model is Inception1_RMSprop_32_lr1e-06_m0.99_wd0.001. It achieved a precision of 0.9357, recall of 0.9381, F-measure of 0.93687, ROC AUC score of 0.99291, and

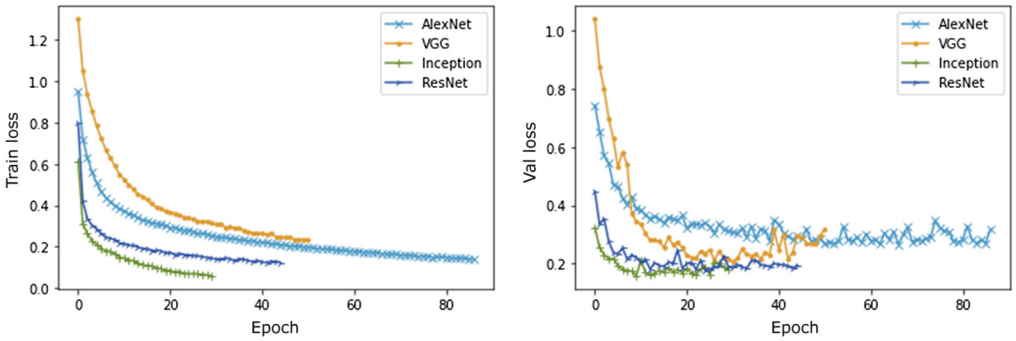

Fig. 13 Comparing loss on the train (left) and validation set (right).

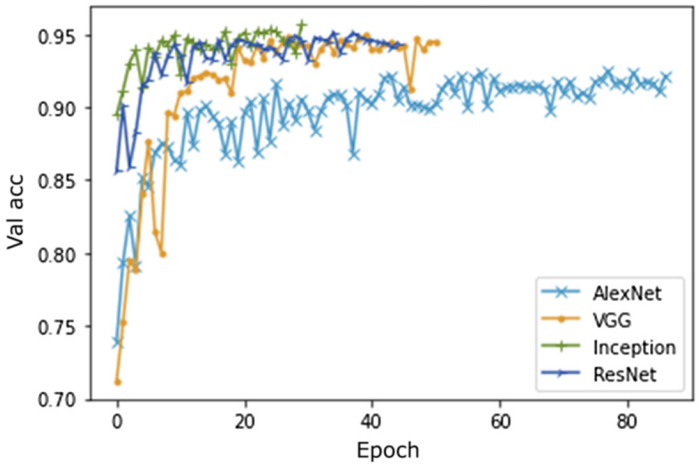

Fig. 14 Comparing validation accuracy.

an accuracy of 0.95528. However, its performance when classifying the minority class requires further improvement.

When compared with models from other families that achieved the best metrics, Inception1 is superior in terms of number of epochs it took to train, as well as in minimizing validation loss and maximizing validation accuracy.

Fig. 13 shows loss comparison on the training and validation sets for different model architectures, whereas Fig. 14 shows accuracy comparison.

The experiment was performed on a computer with AMD Ryzen 7 3700X processor, with 8 cores and 16 GB of RAM memory for training on central processing unit (CPU), NVIDIA GeForce GTX 1060 graphics card with 1280 CUDA cores, and 6 GB of video random access memory memory for GPU execution. For example, the VGG model underwent the average training that lasted several hours on the CPU to less than an hour on the GPU.

The experimental evaluation presents that the proposed architecture Inception1 with selected hyperparameters outperforms convolutional deep learning models from the state-of-the-art reported research based on retinal diseases classification, as shown in Table 12. The table indicates the year of the research, the models used, the datasets, the retinal anomalies detected, the results, and the main contributions. The only limitation of our approach is the usage of CNNs as slower networks, with many layers, so continued research will be oriented toward EfficientNet.[27]

With regard to the analyzed studies found in the related work section, this research showed the best results on the largest dataset taking into account evaluation on the largest number of models using different CNN architectures.

Table 12 Retinal disease state-of-the-art research and analysis of their datasets, results, and contributions.

Research	Year	Model(s)	Dataset	Detection	Results	Contributions
Perdomo et al.[8]	2019	OCT-NET (~VGG)	SERI + CUNK (SD-OCT)	Healthy, DME, DR-DME, and AMD	Precision = 0.93 Recall = 0.83 F-score = 0.85 AUC = 0.86	Classification of diabetes-related retinal diseases using OCT-NET.
Kang et al.[6]	2021	EfficientNetB4	5117 retinal fundus photos, 9316 OCT, and 20922 FA/ICGA	mCNV, DME, nAMD, BRUO, and CRUO	Accuracy = 0.93 AUC = 0.969	Deep learning model to detect treatment-requiring retinal vascular diseases using multimodal imaging.
Hong et al.[7]	2021	Inception v3	7100 clinical images, from 1600 patients with 100 diseases.	7 ocular diseases with 4 ocular surface diseases and 3 retinal fundus diseases in level 1 and 17 subclasses.	AUC from 0.743 to 0.989.	Deep learning framework for diagnosing multiple visual impairment diseases.
Our research	2022	Four different architectures (AlexNet, VGG, Inception, and ResNet)	77,127 OCT and x-ray images (without duplicates).[3]	CNV, DME, drusen, and HEALTY.	Precision = 0.936 Recall = 0.938 F-score = 0.937 AUC = 0.992 Accuracy = 0.955	Evaluation of 64 models based on 4 popular CNN architectures with different hyperparameter values, on a big dataset.

6 Conclusion

This research paper has explored the possibilities of applying deep learning on classifying retinal diseases. The main concepts behind CNNs have been explained, along with the importance of proper hyperparameter tuning.[28] Multiple architectures have been tested to find the optimal solution.

The analysis of different architectures that can be applied in the classification of retinal diseases based on OCT images as well as the evaluation of the test set obtained by comparing 64 models with different hyperparameters represents the main contributions of this research. The experimental evaluation presents the proposed architecture Inception1, with selected hyperparameters, as the best model, in relation to the state-of-the-art reported studies based on retinal disease classification. The best metrics were obtained with Inception1 when training using the RMSprop optimizer with a batch size of 32, learning rate of $1e^{-6}$, momentum of 0.99, and L2 regularization rate of 0.001.

This solution can be used as a decision support system by ophthalmologists in the field of medicine, particularly when determining retinal diseases. Such models recognize diseases with a high accuracy level. This same approach can be further generalized and applied to most multiclass classification problems.[29–31] However, a limitation of this study is the slow execution time since models based on CNNs with a larger number of layers converge slowly.

In the last few years, many researchers in the field of application of AI in medicine use the method of image segmentation.[32,33] Using image segmentation to identify significant areas in images that are classified as diseases can be one of the further directions of this research. The deep neural networks based on *U*-shaped architecture (U-net) and their variations have been widely applied in a variety of medical image analyses.[34–36] Also, it is possible to expand the current research to include EfficientNet. Further work could include dataset augmentation, undersampling or oversampling to reduce overfitting and increase accuracy when classifying the minority class.

Acknowledgments

This research was supported by the Science Fund of the Republic of Serbia (Grant No. 6526093), AI—AVANTES. We appreciate the effort devoted by Prof. Dr. Miloš Đurić (University of Belgrade, School of Electrical Engineering) to proofreading this manuscript.

Code Availability

The source code of the experimental results is available at the following link.[37]

References

1. A. Imran et al., "Comparative analysis of vessel segmentation techniques in retinal images," *IEEE Access* **7**, 114862–114887 (2019).
2. E. Y. K. Ng et al., *Ophthalmological Imaging and Applications*, CRC Press (2014).
3. D. S. Kermany et al., "Identifying medical diagnoses and treatable diseases by image-based deep learning," *Cell* **172**(5), 1122–1131 (2018).
4. A. S. Panayides et al., "AI in medical imaging informatics: current challenges and future directions," *IEEE J. Biomed. Health Inf.* **24**(7), 1837–1857 (2020).
5. National Eye Institue, "Macular Edema," https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/macular-edema (Accessed September 2021).
6. E. Y.-C. Kang et al., "A multimodal imaging-based deep learning model for detecting treatment-requiring retinal vascular diseases: model development and validation study," *JMIR Med. Inf.* **9**(5), e28868 (2021).
7. J. Hong et al., "A novel hierarchical deep learning framework for diagnosing multiple visual impairment diseases in the clinical environment," *Front. Med.* **8**, 654696 (2021).
8. O. Perdomo, et al., "Classification of diabetes-related retinal diseases using a deep learning approach in optical coherence tomography," *Comput. Methods Programs Biomed.* **178**, 181–189 (2019).

9. S. Lal et al., "Adversarial attack and defence through adversarial training and feature fusion for diabetic retinopathy recognition," *Sensors* **21**, 3922 (2021).
10. M. Antonijevic et al., "Robust encrypted face recognition robot based on bit slicing and Fourier transform for cloud environments," *J. Electron. Imaging* **31**(6), 061808 (2022).
11. W. Wang et al., "Development of convolutional neural network and its application in image classification: a survey," *Opt. Eng.* **58**(4), 040901 (2019).
12. S. Qummar et al., "Deep learning techniques for diabetic retinopathy detection," *Curr. Med. Imaging* **16**(10), 1201–1213 (2021).
13. M. Talo et al., "Convolutional neural networks for multi-class brain disease detection using MRI images," *Comput. Med. Imaging Graph.* **78**, 101673 (2019).
14. A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Syst.*, Vol. 25, pp. 1097–1105 (2012).
15. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR* (2015).
16. C. Szegedy et al., "Going deeper with convolutions," in *IEEE Conf. Comput. Vis. and Pattern Recognit. (CVPR)* (2015).
17. K. He et al., "Deep residual learning for image recognition" (2015).
18. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS* (2010).
19. K. He et al., "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification," in *IEEE Int. Conf. Comput. Vis.* (2015).
20. J. Bengio, "Practical recommendations for gradient-based training of deep architectures" (2012).
21. D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," ArXivabs/1804.07612 (2018).
22. G. Hinton, "Neural networks for machine learning," https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (accessed May 2021).
23. M. Riedmiller and H. Braun, "RPROP—a fast adaptive learning algorithm," in *Proc. ISCIS VII* (1992).
24. D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," CoRRabs/1412.6980 (2015).
25. I. Goodfellow, *Deep Learning*, MIT Press (2016).
26. I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR* (2019).
27. M. Tan and Q. Le, "EfficientNet: rethinking model scaling for convolutional neural networks," arXiv:1905.11946 (2020).
28. M. Karakaya, "Iris-ocular-periocular: toward more accurate biometrics for off-angle images," *J. Electron. Imaging* **30**(3), 033035 (2021).
29. Md. Fahimuzzman Sohan, A. Basalamah, and Md. Solaiman, "COVID-19 detection using machine learning: a large scale assessment of x-ray and CT image datasets," *J. Electron. Imaging* **31**(4), 041212 (2022).
30. S. Shekar, N. Satpute, and A. Gupta, "Review on diabetic retinopathy with deep learning methods," *J. Med. Imaging* **8**(6), 060901 (2021).
31. K. Venkatachalam et al., "Effective tensor based PCA machine learning techniques for glaucoma detection and ASPP—EffUnet classification," *Lect. Notes Comput. Sci.* **13079**, 181–192 (2021).
32. A. A. Abdulsahib et al., "Comprehensive review of retinal blood vessel segmentation and classification techniques: intelligent solutions for green computing in medical images, current challenges, open issues, and knowledge gaps in fundus medical images," *Netw. Model. Anal. Health Inf. Bioinf.* **10**, 20 (2021).
33. A. A. Abdulsahib et al., "An automated image segmentation and useful feature extraction algorithm for retinal blood vessels in fundus images," *Electronics* **11**(9), 1295 (2022).
34. Y. Weng et al., "NAS-Unet: neural architecture search for medical image segmentation," *IEEE Access* **7**, 44247–44257 (2019).
35. H. Huang et al., "UNet 3+: a full-scale connected UNet for medical image segmentation," in *IEEE Int. Conf. on Acoust., Speech and Signal Process. (ICASSP)*, pp. 1055–1059 (2020).

36. H. Cao et al., "Swin-UNet: UNet-like pure transformer for medical image segmentation," arXiv:2105.05537 (2021).
37. D. Drašković and M. Stanojević, "Experimental results for retinal disease classification based on optical coherence tomography (OCT) images," 2021, http://home.etf.bg.ac.rs/~draskovic/codes/OCT-classification/ (accessed July 2022).

Maša Stanojević, a researcher, received her MSc degree in electrical and computer engineering from the University of Belgrade, School of Electrical Engineering, Belgrade, Serbia. Her current research interests include machine learning and computer vision.

Dražen Drašković received his PhD in electrical and computer engineering from the University of Belgrade, School of Electrical Engineering, Belgrade, Serbia. He is working as an assistant professor at the University of Belgrade, School of Electrical Engineering, Department for Computer Science and Information Technology, Belgrade, Serbia. His research interests include application of AI algorithms in software systems, machine learning, and big data analysis.

Boško Nikolić received his PhD in electrical and computer engineering from the University of Belgrade, School of Electrical Engineering, Belgrade, Serbia. He is working as a professor at the University of Belgrade, School of Electrical Engineering, Department for Computer Science and Information Technology, Belgrade, Serbia. His research interests include information systems, artificial intelligence, natural language processing, and the development of educational systems.