CONTENTS

ABSTRACT

Diabetic Retinopathy (DR) is a situation coined for malfunctioning of human eye which make human vision blurry or even worst cases eye blindness. Usually properly trained ophthalmologists have to spend hours just to inspect each and every subtle features present and evaluate its significance and then arrive on conclusion. This whole process is very much time consuming and cumbersome from a human point of view. To reduce the human effort and window of error a lot of people tried to automate this whole process using state of the art techniques like Deep learning and computer vision. However there are many challenges like availability of balanced datasets, computationally efficient networks, proper fundus images to train a deep learning model. SO the primary objective in this research boils down to develop a DR method which is computation effective and reliable. On Kaggle EyePACS dataset our model achieves an accuracy of 92.1 % which is really good as compared to many state of the art available models.

LIST OF FIGURES

Chapter 1.

Introduction – Diabetic retinopathy

1.1 What is Diabetic Retinopathy?

Diabetes is a disease when blood sugar level in a human body imbalances. When the blood sugar level of human body increases a lot of side effects affects the proper functioning of human body, one of which is the most vital organ of human body that is *eye.* The blood sugar level, after increasing by a certain amount start damaging the human eye blood vessels in retina. Due to the sugar getting deposited there, if affects the flow of blood and the blood vessels can overflow or leak. Sometimes they also can get closed in which case eye will not receive sufficient blood supply. Sometimes due to this abnormality, new vessels can come up in eye, and all the above situations can steal your vision. Diabetes retinopathy starts to affect eye slowly, if diagnosed early it can be completely cured. In the next section we discuss about the stages in diabetic retinopathy.

1.1.1 Phases in Diabetic Retinopathy (DR)

Diabetic retinopathy situation of human eye can be broken down into two main stages:

1. NPDR (non – proliferative diabetic retinopathy)

   This is the primary stage where few of the blood vessels swell. Sometime they may leak as well or get closed off, both the situation leads to the loss of vision of a patient in later stages.

2. PDR (proliferative diabetic retinopathy)

   This is the more advanced stage. In this stage abnormal growth is observed in retina. The newly gown blood vessels are quite fragile and often bleed, which affects the vision and sometimes blocks it as well. The newly grown fragile tissues looks like a scar in the eye.

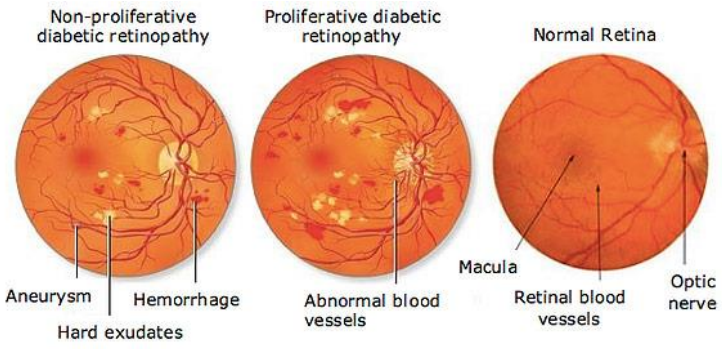


Figure 1 shows different stages of Diabetic Retinopathy.

1.1.2 Motivation behind Deep Learning based Classification of Disease

It is estimated by US Center for Disease Control and Prevention that 29.1 million people in the US are affected by diabetes and WHO estimated that 347 million have the same disease. Diabetic Retinopathy is a eye disease but is mainly caused by longstanding diabetes. Around 40-45 % of people that has diabetes are in some stage of DR which can e averted or slowed down if detected in time.

Detecting DR used to be a manual process some years back, which is a very complex and time taking process and involved human error as well. By the time Subject Matter Experts used to submit their review, often after one or two days, the delayed identification would lead to lost follow up, miscommunication and sometimes treatment as well.

DR can be identified using various clues like lesions, scars etc. The instruments needed to capture this information is actually a very high resolution camera and very costly thus making its multiple availability in poor area almost impossible. Thus a comprehensive and automated process to identify DR is very much needed, that too with good accuracy. This automated system can be developed with the help of image processing, computer vision and deep learning applications knowledge. A color fundus photograph of both eye will be needed for the purpose of algorithm development. Color fundus images has to be annotated for various stages of the disease as follows:

1. Normal
2. Mild
3. Moderate
4. Severe
5. Very Severe

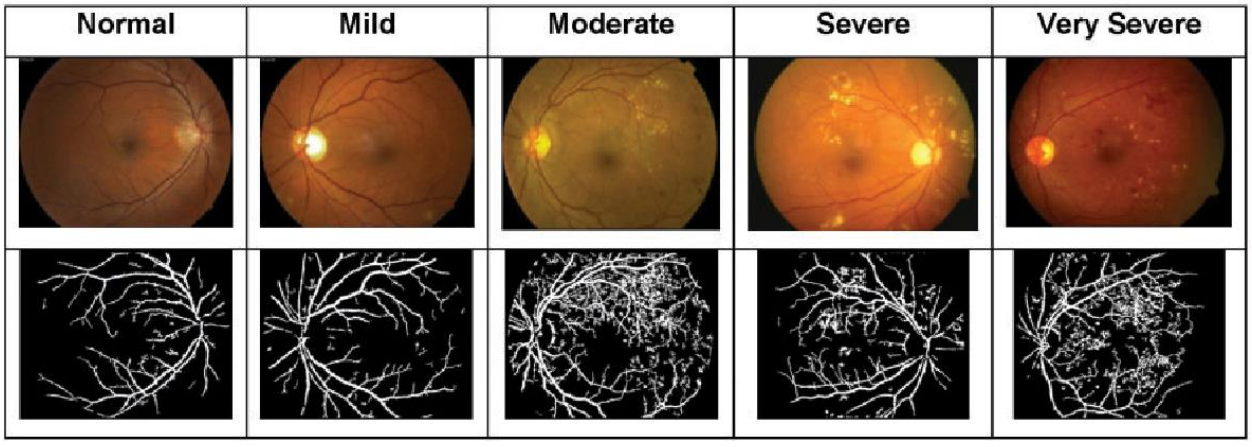


Figure 2 represents various stages of DR eye along with the edges map representing vessels

Chapter 2.

Literature Research and Review

2.1 Related Datasets

There exists several datsets of fundus images, developed by people and institutions all over the world. They are following:-

1. “Kaggle Diabetic Retinopathy Dataset” [14]
2. “E-Optha Dataset” [15]
3. “REVIEW - Retinal Vessel Image set for Estimation of Widths dataset” [12]
4. “STARE - Structured Analysis of the Retina dataset” [11]
5. “DRIVE - Digital Retinal Images for Vessel Extraction dataset” [14]
6. “MESSIDOR - Methods to Evaluate Segmentation and Indexing techniques in the field of Retinal Ophthalmology dataset” [14]

2.2 Related Works in Machine Learning

Before the development of deep learning algorithms, especially deep neural networks, a feature extraction step was needed for general computer vision tasks. Features, in this case, are distinguishing and significative small image patches. Many feature extraction algorithms were proposed in the 1990s, such as SIFT and SURF[1], which have been widely applied for object recognition or medical image retrieval. Following the feature extraction step, traditional machine learning classification alogorithms, such as sopport vector machines, logistic regression or decision trees, are trained using extracted features to classify the image. This pipeline was applied to almost all traditional computer vision analytics tasks including image & videos based classification, image & videos based segmentation, etc.

Automatic DR detection based on traditional computer vision techniques follows the same pipeline. First, specific features are extracted from the fundus images using single or combinations of manually designed feature extraction algorithms. Classifiers are then trained using the extracted feature to classify the DR stages. Several such works are briefly reviewed below. A multilayer neural network model was trained using features extracted by recursive region growing segmentation algorithms to perform binary DR classification task . [2] The sensitivity and specificity of the proposed system is 80.2% and 70.7% respectively. Recursive region growing segmentation algorithms was also adopted in to extract DR visual features such as exudates, hemorrhages, and microaneurysms. The features were then used for Healthy / Diabetic Retinopathy binary classification.[3] The sensitivity and specificity of the proposed system is 74.8% and 82.7% respectively.

Lee et al. [4] introduced methods for automatically grading the severity of three early DR lesions, namely hemorrhages, hard exudates, and cotton-wool spots. An algorithm is trained to classify NPDR based on the absence of these three types of lesions. The overall accuracy of the proposed system is 81.7%. In [27], a decision support system was proposed for early detection of Diabetic Retinopathy (presence of macroaneurysms) using Bayes optimality criteria. The method was able to spot and pinpoint the early

stage of Diabetic Retinopathy (DR) with a sensitivity of 100% and specificity of 67%. The automatic DR classification methods discussed in this section use manually designed features. The performiance of such methods is highly dependent on the feature extraction process. Machine learning algorithms are only used for numerical optimization based on the features designed by humans. Thus, domain knowledge of the DR disease plays a crucial role in building effective DR classification models. The limitation of applying conventional computer vision algorithms for Diabetic Retinopathy (DR) lesion detection or DR classification is that the manually designed features are often overspecified, incomplete or require a long time and know-how and understanding to design and validate.

2.3 Related Works in Deep Learning

Instead of manually designing features and feeding them to the classifier for DR screening, many researchers are now building end-to-end deep learning models which learn all the needed features automatically. Several works which use deep neural network models to detect DR automatically are briefly reviewed below. A 13-layer deep convolutional neural network (CNN) for screening DR was discussed in [29]. Several data augmentation techniques (i.e., image rotation, image flipping and image shifting) were applied to overcome the data imbalanced problem. Five thousand images from the EyePACS DR training set [5] were retained as the test set and the rest of the images from the EyePACS DR training set were used for training the proposed model. The trained network achieves 95% specificity and 30% sensitivity on the test set, which indicates that the proposed model is highly biased to the negative (healthy) case. In [6], a residual neural network [2] was used to classify the retinal image. The quadratic weighted kappa (QWK) [31] was adopted as the evaluation metric. The proposed system achieves a QWK score of 0.51 on the EyePACS DR test set. A deeply supervised ResNet was proposed in [7] to automatically classify the grade of DR. In the proposed architecture, three sets of additional side-output layers were added to the in-between layers of the ResNet to provide additional regularization during the training process. 20% images from the EyePACS DR training dataset were retained as the test set. The results show that by bargaining chip the predictions of in between supervised layers, ResNet can focus on multi-scale learning which leads to improved performance compared to standard ResNet without side-output layers. The QWK score, specificity, sensitivity and accuracy of the proposed system on the test set is 0.73, 94%, 67% and 81% respectively.

In [8], a Non-Local means denoising method was applied to the fundus image to remove noise from the EyePACS DR dataset. AlexNet [29] and GoogleNet[34] were utilized to classify the pre-processed retinal images. An AUC score of 0.78 is achieved by the GoogleNet model compared to an AUC score of 0.68 achieved by AlexNet. In [35], a computationally efficient CNN model, namely the MobileNet model [38], was used for binary DR screening (DR V.S. Healthy image) using the EyePACS DR dataset. An accuracy of 0.73 was achieved by the proposed method. Considering the great data imbalance nature of this dataset (73.4% of the image are healthy and 26.6% of the image contains DIABETIC RETINOPATHY), only using accuracy score may not appropriate for demonstrating the effectiveness of the proposed model. For an extreme example, a simple classifier which always predicts images as healthy images will also achieve an accuracy of 0.73 using the EyePACS DIABETIC RETINOPATHY dataset. In [36], GoogleNet and VGGNet were modified to two corresponding networks CKML (Combined Kernels with Multiple Losses Network) and VNXK (VGGNet with Extra Kernel). The retinal images were converted to a hybrid LGI color space. Models trained using the LGI retinal images were then compared with models trained using RGB retinal images. The results show that both CKML and VNXK achieve higher accuracy with LGI images. However, the proposed model is also highly biased to the healthy samples. The corresponding accuracy from class 0(healthy) to class 4(proliferative DR) are 97.6%, 11.9%, 57.9%, 33.2% and 36.8% respectively. In [9], a

novel Zoom-in-Net model was proposed. It impersonates the zoom-in process of an ophthalmologist who examines the retinal images. Zoom-in-Net consists of three parts, Inception-ResNet was adopted as the main network (M-Net) for DIABETIC RETINOPATHY classification. Two tiny CNNs, namely the Attention Network (A-Net) and the Crop Network (C-Net), were used for Attention Localization of the for DIABETIC RETINOPATHY images. The single Zoom-in-Net achieves a QWK score of 0.849 on the entire EyePACS DIABETIC RETINOPATHY test set. With an ensemble of three Zoom-in-Net models, the QWK score was improved to 0.854.

In the 80s the most used activation function was the sigmoid, a smooth non-linear function, that is continously differentiable. Despite its interesting properties, it has a very important concern, called gradient vanishing problem [11]. Sigmoid first derivative becomes flat not far from the origin, affecting network loss optimization due to near-to-zero gradients. In [12] ReLUs were used for improving Restricted Boltzmann Machines (RBMs), approximating stepped sigmoid units with ReLUs. In (Glorot, Bordes, and Bengio, 2011) the authors compared the performance of Sigmoid, Tanh and ReLU arriving to the conclusion that despite Sigmoid being more plausible biologically, Tanh and ReLU were more suitable to be used as activation function for training multi-layer perceptrons. ReLU networks have better performance in general, despite its non-differentiability at zero and its hard non-linearity. Furthermore, ReLU networks lead to sparse representations, being beneficial, both because information is represented in a more robust manner and because it leads to significant computational efficiency. Moreover, the simplicity of the function and its derivative reduces calculation time, being of significant importance when working with big networks. The constant value of the gradient, helps avoiding the gradient vanishing problem, allowing the design of deeper networks. For then on, ReLU has become the default activation function for deep learning. Many other activation functions have been published, like LeakyReLU, but not introducing significant performance gains.

Traditional models of pattern recognition in images since the late 50s have been based on extracting hand-crafted fixed engineered features or fixed kernels from the image and, then, using a trainable classifier on top of those features to get the final classification. Using this model, the problem of the DR detection has been tackled by feature extraction using on hand models targeted to the detection of microaneurisms, haemorrhages and exudates in retinal images (e.g. (Sudha and Thirupurasundari, 2014), (Torrents-Barrena et al., 2015), etc). This type of approach requires a good understanding of disease mechanisms to be able to find the important features present in the image. This knowledge is specific of the problem to be solved, requires a lot of labor.

A deeply supervised ResNet was proposed in [32] to automatically classify the grade of DIABETIC RETINOPATHY. In the proposed architecture, three sets of auxiliary side-output layers were added to the in between layers of the ResNet to provide auxiliary regularization during the training process. 20% images from the EyePACS DR training dataset were retained as the test set. The results show that by ba the predictions of in between supervised layers, ResNet can focus on multi-scale learning which leads to improved performance compared to standard ResNet without side-output layers. The QWK score, specificity, sensitivity and accuracy of the proposed system on the test set is 0.73, 94%, 67% and 81% respectively. The results show that by bargaining chip the predictions of in between supervised layers, ResNet can focus on multi-scale learning which leads to improved performance compared to standard ResNet without side-output layers. The QWK score, specificity, sensitivity and accuracy of the proposed system on the test set is 0.73, 94%, 67% and 81% respectively.

Deep learning techniques are not only very effective for solving general classification tasks, but also for prediction in medical imaging. For our case study of diabetic retinopathy disease grading, having enough data our method is able to perform near human level expertise.
Lee et al. [26] introduced methods for automatically bracketing the seriousness of three early DIABETIC RETINOPATHY lesions, namely hemorrhages, hard exudates, and cotton-wool spots. An algorithm is trained to classify NPDR based on the absence of these three types of lesions.

Medical applications including image classification, segmentation and localization are not excluded from the rapid emergent of deep learning . A summary of over 300 recent publications, a lots of image analytics based summary especially on medical image analytics using deep learning has been reported by [12]. Most of the recent deep learning algorithms proposed for retinal image analysis have used CNNs for color fundus images [12]. A lots and lots of varieties of applications are addressed: image-segmentation of anatomical structures , image segmentation analytics and image detection analytics of human eye retinal abnormalities , Correct and speedy figuring out of human eye diseases, and any kind of image quality assessment

In the last decade, several attempts to automatize the DR diagnosis through images of the eye fundus have been tested. They are basically focused on applying well-known pattern recognition models. In this work, we want to apply a Deep Convolutional Neural Network (DCNN) model, as it has been proven to be a very effective algorithm to solve general image classification problems. DCNN is a supervised learning model for automatic classification that does not require any pretreatment of the images, nor any feature analysis. Deep learning base expertises are totally paying attention on properly learning and understanding many levels of abstraction and representation which that helps to make inference of the informational intelligence that is hidden in any kind of data such as colored images of dogs cats. So In this manner having a totally complete full set of properly and correctly classified images with annotations and without any having any presumptive perceptive of the available features required to do the classification activity, the system can be abled to look and learn the properties of the image and its underlying features which usually minimizes a defined cost or maybe a loss function that is indirectly or directly associated with the classification cost and score index that has to be optimized. In this chapter we show that a DCNN is able to understand from data the most important features to make the classification of retinal images into the five DR categories, without the need of a hand-crafted feature extraction process. The thesis is organized as follows: we present the related work, next we explain the characteristics of the available data and why deep learning techniques can be applied over them, next we explain the methodology used for solving the problem, we show the obtained results and finally we expose the conclusions and further steps for improving the results.

In this chapter is shown that deep learning techniques are not only very effective for solving general classification tasks, but also for prediction in medical imaging. For our case study of diabetic retinopathy disease grading, having enough data our method is able to perform near human level expertise. Work of next chapters will be centered on testing the newer schemes, the use of alternative cost functions that encode the prior information of the ordering of the classes and other more elaborated methods for combining the information coming from both eyes.

Chapter 3.

Methodology

3.1 Data Preprocessing

3.1.1 Data Imbalance

Data Imbalance problem is one of the biggest challenge in image classification of medical images, or rare occurrence classes[16]. CNN are built to minimize errors through backpropagation. On sampling from training set in a imbalanced data set, probability of object belonging to particular class will be directly proportional to its imbalanced numbers, so in the quest of minimize errors model will be misled to believe that mostly observation belongs to set with high numbers than other. To negate the effects of data imbalance, different augmentation, and oversampling from minority class is tried out.

3.1.2 Data Augmentation

When number of data is limited but the CNN cannot learn properly with the amount of data available we cannot do anything but have to look for techniques which will help us increase our data and there we go for data augmentation as shown in *figure 3*.

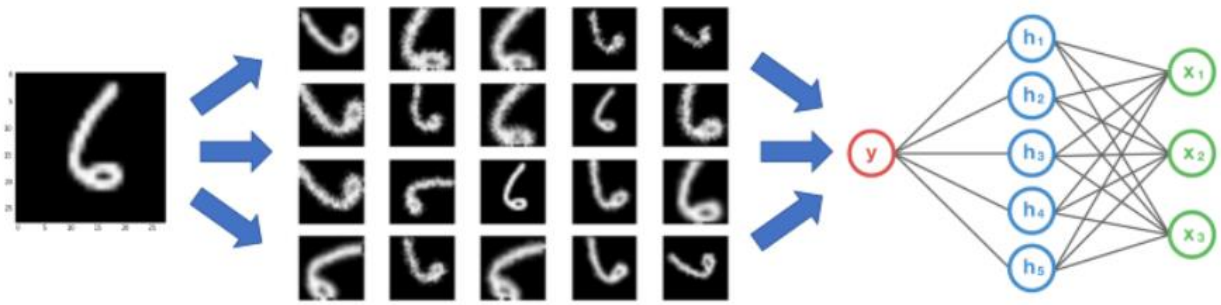


Figure 3 Data Augmentation in play

A CNN is said to be following the property called Invariance in *figure 3* as we can see that the above model will be invariant to rotation[17]. Similarly other properties like invariant to translation, cropping, illuminance also exists. So augmentation can be of different types, like offline augmentation and online augmentation. **Offline Augmentation**[18] is usually preferred for smaller datasets where we do the data augmentation a priori and store all the images for training. Other variety is **Online Augmentation**[18]**,** this variant is space efficient as it does augmentation at the time of training using various in house image processing functions and do not create new images to store them and does this process in batches otherwise it will lead to explosively increase in the size.

The following are the popular augmentation techniques with example:

3.1.2.1 Flip and Rotation

Images can be flipped horizontally as well as vertically as shown in *figure 4.* Horizontal flip can be seen as creating mirror image of an image. In some libraries or frameworks vertical flips option is not available, it can be taken as rotating image by 180 degree and performing horizontal flips[18].

Figure 4 Original Image, Horizontal flipped, Vertical flipped

3.1.2.2 Adding Noise

When neural network somehow learns very high frequency features or patterns that occurs a lot, it leads to overfitting of model. To cater this problem Gaussian noise[19] can be used as it has data points in all frequencies, and zero mean, which can effectively distort high frequency features. Another good suggestion is to use salt and pepper noise[20], it is random black and white pixels added to image as shown in *figure 5*

Figure 5 Original Image, Added Gaussian Noise, Added alt and Pepper Noise

3.1.2.3 Scaling and Cropping

Image can be scaled inward or outward, which also produces similar effect to that of cropping. All the process in scaling and cropping should be random with some preset values that should be taken. In the *figure 6* below following technique example is demonstrated

Figure 6 Original Image, Scaled 10% image, Cropped Image

3.2 DenseNet

CNN are playing a dominant roles in today's world's imaging technology [21]. From object detection, recognition to semantic and instance segmentation, CNN can be used to solve wide variety of problem statements [22]. Features from images are extracted in CNN by taking repeated convolutions using kernels, which are usually called standard ConvNets as shown [23] in *figure 7*.

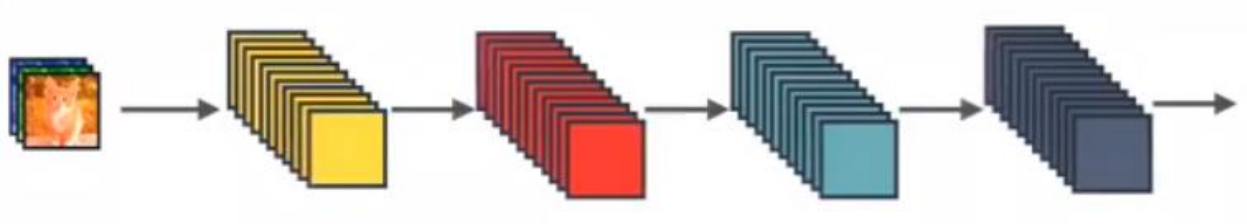


Figure 7 Standard ConvNet Concept

Gradient propagation was introduced in ResNet where identity is used as displayed in *figure 8*. Intitutively understood as a algorithm where a state is passed from one ResNet module to another one for better utilization of gradient values.

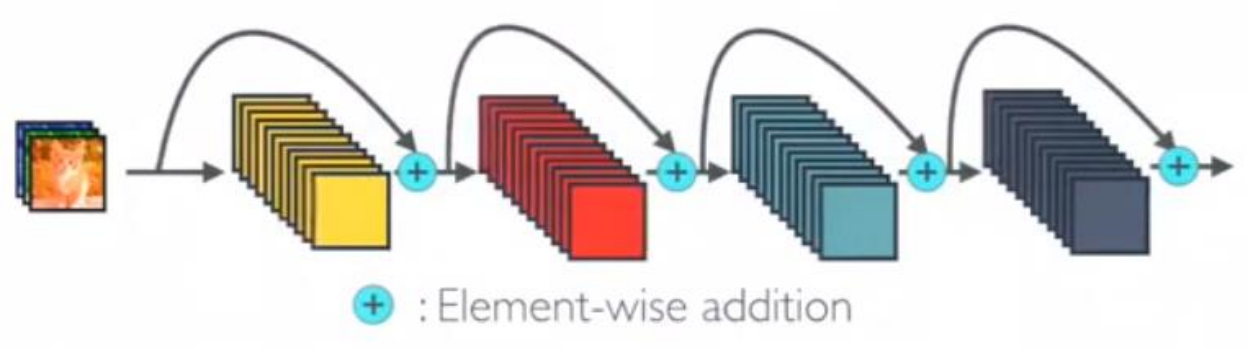


Figure 8 ResNET Architecture Example

In DenseNET all the layers pass its own and its preceding layer information to its succeeding layer. Thus each layers will have the information of the gradients which its previous layers have as shown in ***figure9***.

Consider a Convolutional Neural Network of L layers and a $image\ \mathrm{x}_0$. Let the non linear transforms of an image is represented by $\mathrm{H}_\mathrm{l}(\cdot)$ where l represents l^{th} layer, which can be anything among Batch Normalization (BN), Rectified Linear Unit (ReLU), Pooling or Convolution. Output of l^{th} layer is represented by x_l.

Usually in CNN output of l^{th} layer serves as a input for $(l+1)^{th}$ layer which gives rise to the transformations [27]:

$$x_l = H_l(x_{l-1}) \tag{1}$$

In ResNETs a skip connection [28] is added which bypasses the non-linear transformation with a identity function as follows:

$$x_l = H_l(x_{l-1}) + x_{l-1} \tag{2}$$

To further improvise the flow of information a different connectivity pattern in DenseNET[28] was introduced in which any subsequent layers are connected to all its previous layers. Lets assume that x_l is output of DenseNET l^{th} layer and it receives $x_0, x_1 \dots\dots x_{l-1}$ as inputs, so x_l can be written as :

$$x_l = H_l([x_0, x_1 \dots\dots x_{l-1}]) \tag{3}$$

$$Where\ [x_0, x_1 \dots\dots x_{l-1}]\ \in Concatenation\ of\ features\ maps\ from\ layers\ 0,1,2 \dots\dots l-1$$

$$Where\ \mathrm{H}_\mathrm{l}(\cdot)\ \in Composite\ function\ of\ BN, ReLU\ and\ Convolution\ (3x3)$$

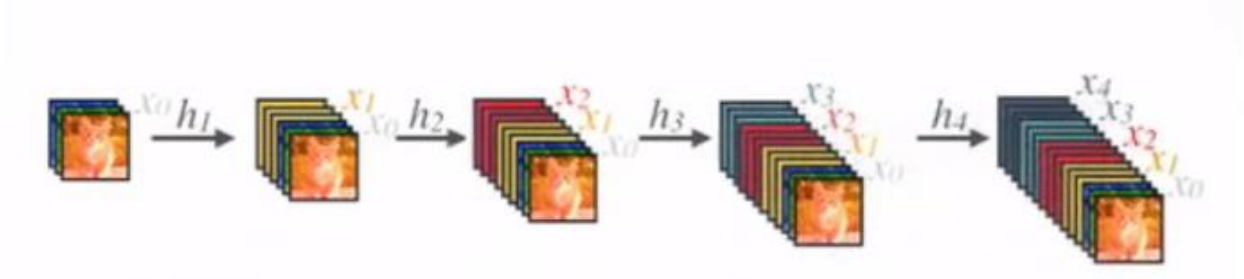


Figure 9 Represents Concatenation

Chapter 4.

Implementation and Discussion

This chapter deals with the models designed and implemented during working on this thesis. Aim of this exercise was to explore the problem statement in depth, compare performance between different models and to better understand the hyper parameter optimization process in various different state of the art models.

During past several years, most appropriately decade, a lot of people, companies, students and researchers are working towards the automatic diabetic retinopathy detection using image processing and computer vision. Most of the people have worked towards implementation of some state of the art classification models to get desired result.

4.1 Data Set

To obtain the fundus image datasets, we explored multiple websites and avenues but we found Kaggle[29] dataset as the best. The size of the total data set was 40 Gigabytes.

4.2 Methodology for Implementation

Convolution Neural Networks (CNN) gives us a lot of options for supervised deep learning for images example AlexNet[29] (Sutskever, Hinton and Krizhevsky, 2012), GoogleNet[30] (2015), VGGNet[31] (Zisserman & Simonyan 2015), ResNET[24] (Microsoft), all these architectures are based on special concept of neural network that is Convolution Neural Networks (CNNs).

CNNs are made up to decompose or extract information from images and hold those information in different layers. To extract features which are nothing but statistical information from data, pooling layers are used which are also called dimensional down-size blocks. Information collected by one layer is passed to another succeeding layer which can also be called as features of features helps us in classification. To perform the final classification it is evident that a fully connected layer is required which collects information from all the layers on its behind and make mathematical prediction as shown in *figure 10*.

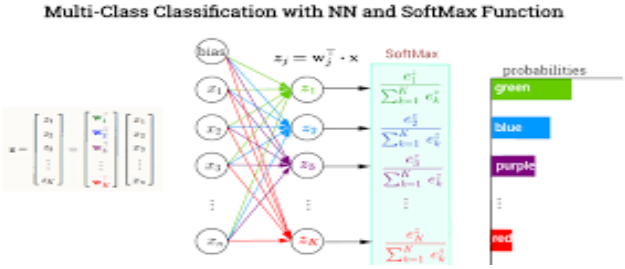


Figure 10 Softmax classifier doing final prediction

Usually different from our traditional algorithms, these CNN based algorithms provide us end-to-end learning scheme which are totally unaffected by human intervention or possibility of human error gets minimized here to the minimum if data sets is loaded properly.

Next we explain different aspects of the model used in this thesis that is DenseNET based Diabetic Retinopathy classification based on the available data from EyePACS, evaluation function, training procedure, model optimization and probabilistic combination of prediction from both the eyes.

4.3 Workflow of Deep Learning Model Implementation

In this section, we summarize the typical workflow of applying neural network models in a real-world application. The typical workflow is summarized as follows:

1. Data Acquisition: First, prepare a dataset for the application. The dataset is then split into three parts, namely the training set, the validation set and the test set. The training set is used for training the model with backpropagation algorithms. The validation set is used for hyperparameter optimization and the test set is used for performance evaluation.

2. Exploratory Data Analysis: After we obtained the dataset, exploratory data analysis (EDA) is performed in order to discover latent patterns such as imbalanced data distribution, features at different scales, identifying anomalies or outliers using statistical methods and illustrate data properties using graphical representations.

3. Data Preprocessing and Data Augmentation: After EDA, data preprocessing techniques and data augmentation techniques are applied on the dataset. Preprocessing techniques are widely used for improving the convergence of the neural network model . For image dataset, standardization (subtracting the mean and dividing by the standard deviation individually for RGB channel) is commonly used. Data augmentation techniques are used to generate artificial training samples and expand the training dataset. It has been widely used for improving the model performance and reducing overfitting, especially for training with a small dataset or imbalanced dataset . Commonly used image augmentation techniques include image cropping, image flipping, image rotating, etc.

4. Architecture Design: Next, we need to find a set of architectures which are suitable for the applications. If we have limited computational resources or a real-time prediction is needed, we may use computationally efficient CNN models for the task. On the other hand, we may use very deep CNN models to obtain good performance if adequate computational resources are provided.

5. Hyperparameter Optimization and Training: Next, we need to choose a set of hyperparameters to train the model. Hyperparameters are set of nonlearnable parameters which need to be predefined before the training process begins. The hyperparameter optimization process consists of sampling hyperparameters from the manually specified subset of the hyperparameter space, training the models with different hyperparameter settings and evaluating those models on the validation set. The combination of the hyperparameters which yields an optimal model is selected as the optimal hyperparameter set. The final model is the model trained with the optimal hyperparameter set

6. Performance Evaluation: After the model is trained using optimal hyperparameters, the performance of the final model is evaluated using the unseen test set. 7. Ensemble Learning (optional): Consistent performance improvement can be obtained by applying ensemble learning (i.e., ensemble learning refers to use multiple models to obtain better performance than any single model). Commonly used ensemble learning techniques include Bagging (i.e., voting with equal weight), Stacking (i.e., training a meta-classifier to combine the predictions of several base-level models), etc.

4.4 Evaluation Of Model

The problem to be modeled is essentially a classification problem and we define certain performance assessment criteria to evaluate the performance of each applied algorithm and derive inferences from it. The evaluation metrics for our models are listed as follows:

We tackle the classification problem using supervised learning methods due to the availability of true class labels of the available data. Hence it is possible to compute the correct and incorrect class predictions for the samples of data by comparing prediction results with the true labels and hence determine the misclassification error as the proportion of incorrectly classified samples. Conversely, the prediction accuracy is a measure of the proportion of samples that have been correctly classified by the models.

We determine the model performance using two types of validation methods. One being the hold-out accuracy and the other being cross-validation accuracy. The hold-out procedure consists of splitting the data into two sets namely the training and the testing set. The two sets of data are disjoint or mutually exclusive of each other and commonly the training set is the larger of the two sets. Hence, in this method the classifier is learned or trained on the training set without any exposure to the test set. The trained model is then applied to the test set and outputs predictions for each of the test sample classes which is compared with the true labels. Cross validation on the other hand involves splitting data into folds and training the classifier on data except a particular fold which is left for validation. This procedure is repeated across all folds and the final accuracy is representative of the average performance accuracy over all folds. We are interested to see how hold-out and cross validation accuracies of our chosen models compare against each other

In cases when data is imbalance, using accuracy as sole metric to evaluate the model is misleading. For example lets consider a example:

Let the number of images for 5 classes as following:

A – 5000

B – 500

C – 150

D – 53

E – 5

So here lets consider that our model is giving accuracy of 98 %.

2 % error – 2/100 * 5708 ~ 114 erros

So even if out model completely misclassifies class D and E still accuracy will look quite good which is not good and misleading.

So like in our cases where data is imbalanced we use metrics like confusion matrix as shown in *figure 11*

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Figure 15 Displays a confusion matrix

In confusion matrix for every class we have information of True Positives, True Negatives, False Positives and False Negatives. As you can see in the *figure 11* that the rows and columns are named as actual and predicted. And the values in each cell is the intersection of its row name and column name. Several metrics has been made out of this confusion matrix which are described as follows[33]:

Table 1 Different metrics from Confusion Matrix

Metric Name	Formula
Accuracy	(TP+TN)/total
Misclassification Rate	(FP+FN)/total
True Positive Rate, Sensitivity, Recall	TP/actual yes
False Positive Rate:	FP/actual no
Precision	TP/predicted yes
Prevalence	actual yes/total
True Negative Rate, Specificity	TN/actual no

Precision and recall are metrics which give us a picture about model performance and can be evaluated from the confusion matrix. Precision is a measure of how many of the predicted values in a class are correctly classified or actually part of the true labels of the class. Hence it is a measure of positive prediction by the model. Recall on the other hand is a measure of the amount of information correctly retrieved or in other words the number of samples correctly predicted. Models can be optimized based on a measure which combines or balances both precision and recall. This is called F-measure which is a weighted average of the precision and recall of the model. Precision and recall can be computed from the confusion matrix to determine model performance. In a binary sense, the confusion matrix portrays the number of true positives, true negatives, false positives and false negatives.

4.5 Data Pre-Processing and Augmentation

Exploratory Data Analysis (EDA) [64] is one of the most important process for any generalized machine and deep learning tasks. It refers to the primary inspection process of the data in order to discover underlying patterns such as imbalanced data distribution, identify anomalies or outliers using statistical methods and illustrates data properties using graphical representations. As the Kaggle Diabetic Retinopathy dataset [5] contains fundus retinal images captured from a fundus camera and the corresponding DR severity labels, we perform basic Exploratory Data Analysis on both the retinal images and the labels

The Kaggle dataset [5] is a large Diabetic Retinopathy image based dataset with very high-resolution images taken under different lighting and imaging conditions. There are in total 17653/5453 overall both eye pairs of colorful images for the totally training/test set and the corresponding Diabetic Retinopathy severity levels are provided. The DR levels are rated from 0 to 4 based on the severity of the DR disease assessed by an ophthalmologist. It can be seen that the distribution is very non-uniform, with 73% of the images labeled as healthy , 6% as mild DR, 15% as moderate DR, 2% as severe DR and 2% as proliferative DR. The EyePACS Diabetic Retinopathy training set also shows that the DR labels for a given pair of eyes are highly correlated. To be more specific, it can be seen that 87% of pairs of eyes have the same DR severity label, 95% of pairs of eyes have the DR label values differ by at most 1.

Random sample retinal images with different DR levels from the EyePACS DR training set are shown in Figure 14. Each row contains five retinal images with the same DR level. Retinal images in this dataset were taken in different illumination conditions or by different types of cameras . Therefore the size of the image and the light conditions vary (e.g., the 4th image with moderate DR, the 2nd and the 3rd set. image with proliferate DR in Figure 4.4 are relatively dark compared with the other images). The sizes of the training images varies from 400 × 315 to 5184 × 3456. For illustration purposes, all images shown in Figure 14 are resized to 256 × 256 using bilinear interpolation. The dataset contains some underexposed images (i.e., images which are too dark) and overexposed images (i.e., images which are too bright) [5]. It is useful to check whether there are many images which are poorly exposed since existing deep learning models are shown to be sensitive to image quality. [65]. We converted the RGB images to 8-bit grayscale images and computed the histogram of the grayscale images for screening the poorly exposed images. For an underexposed image, a large portion of the pixel values are close to 0 (which corresponds to black), whereas for an overexposed image, a large portion of the pixel values are close to 255 (which corresponds to white). We considered an if set. be underexposed if 90% of the pixel values are between 0 and 30. We consider an image to be overexposed if 80% of the pixel values of the image are between 225 and 255. The threshold is set differently since the retinal images contain some black borders. Note that there could be other ways to screen the poorly exposed image. For example, another equivalent method would be using the average pixel values of the grayscale image to screen the poorly exposed image. Our classification rule yields 145 underexposed images and 14 overexposed images in the training dataset. In the test set, our classification rule yields 190 underexposed images and 11 overexposed images. As the poorly exposed images comprised only 0.5% of the training set or test set, we did not remove any images from the training or test sets.

DenseNETs usually because of its skip connection requires a larget datasets of images to avoid overfitting. But a large dataset can be imbalanced as well so we require to have a balanced large datasets in general to avoid overfitting. But usually a datasets mainly of diseases can be hugely imabalanced and needs to be corrected usually through a technique discusses above called augmentation. Here since the dataset is not considerably large we have used offline segmentation. In order to perform image augmentation using image processing defined function following are the methods used :

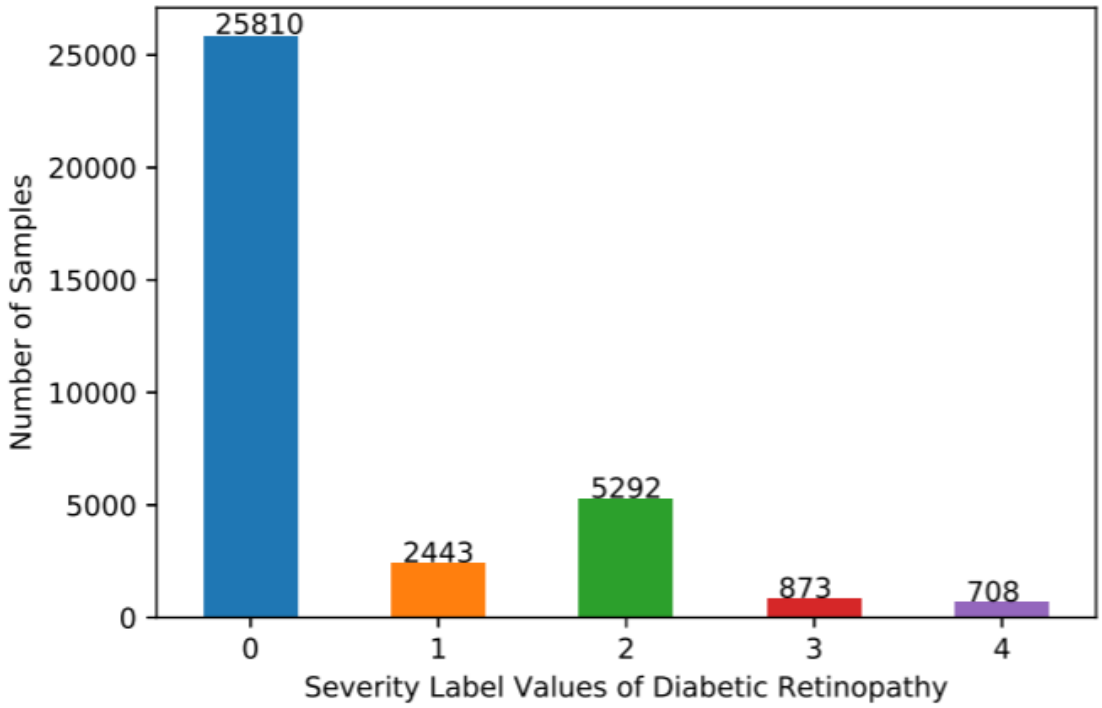


Figure 13: Distribution of Diabetic Retinopathy label values in Kaggle train

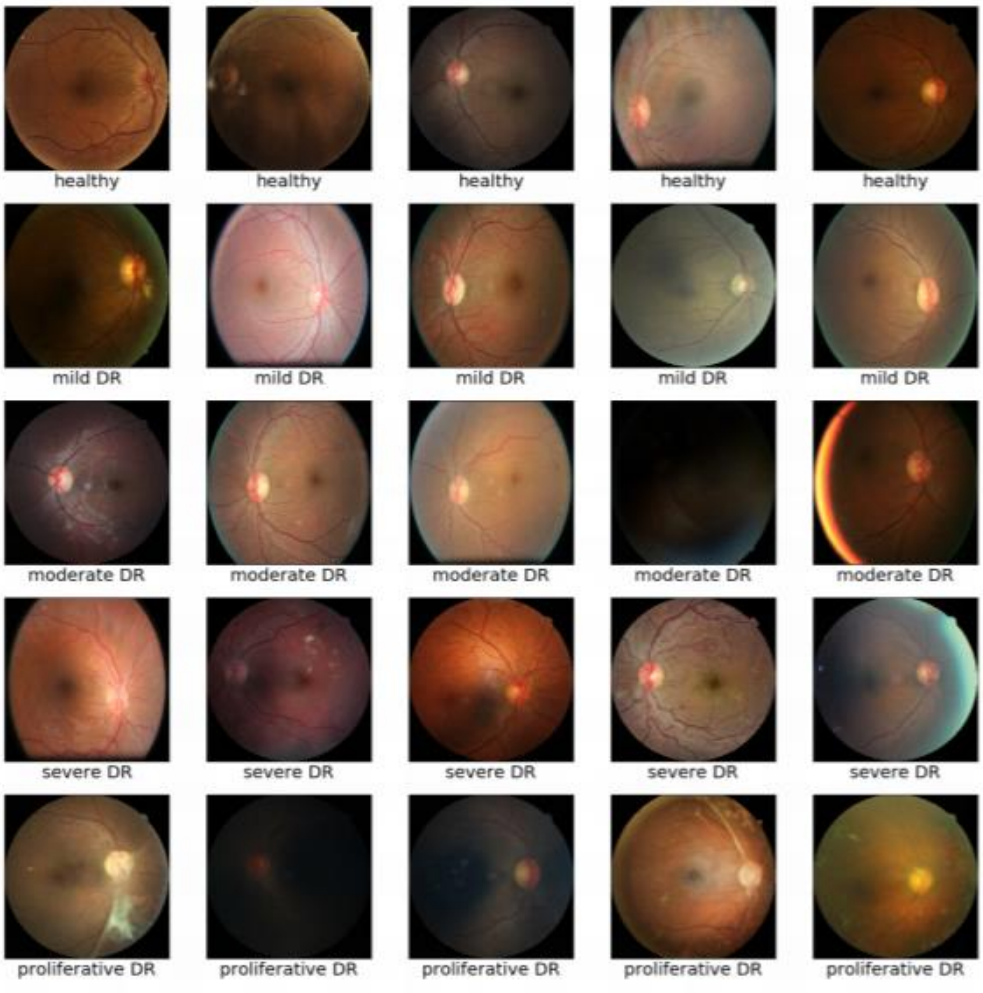


Figure 16 Random sample images from the Kaggle dataset

1. Cropping, random uniform
2. Rotation (0_ to 360_), random uniform
3. Mirroring (X and Y axes), random uniform
4. Brightness correction, random gaussian (s = 0.1)
5. Contrast correction, random Gaussian[34] (s = 0.1)

Offline segementation was performed in two stages, in first stage images were oversampled to create a kind of a balance between the numbers of images per class. Then a series of transforms described above were applied on randomly selected images from the minority classes based on the defined random numbers. The transforms make sure that after oversampling the images are not repetitive but are different from each other even maybe their sources were same making the final prediction invariant to rotation, translation, intensity and contrast over the training set. Also to make the convergence faster for DenseNET a mean normalization for pixel values was performed per image.

4.6 Model

A very popular upgradation of Resnet is DenseNET 201 [35], which is considered for classification of diabetic retinopathy. It consists of series of convolutional layers, activation blocks and max-pooling dimensional reduction blocks. Its end is attached to a fully connected layer on which a softmax function is applied which gives probability estimation of every class as a output. Architecture has been fully described in Appendix. Some parameters of DenseNET are activation function, optimization and regularization methods to be used, number of layers, number and size of classification layers, number of filters per convolution layer, size of the convolution, input size, number and size of classification layers. There donot exists any unique solution to this problem as there are very high number of parameters are involved. There can be different condition co-existing to give the same outputs.

4.7 Training and Testing Procedure

Since it's a multi-class classification problem, for optimization in the learning stage a log-loss function is used. Original training set was split into 2 randomly selected subsets with 80% data and 20% data respectively. The later smaller set was used to cross validate sets of obtained hyper parameter. Leaky ReLU[37] was selected as the activation function. To compensate the vanishing gradient problem, which occurs very frequently in deep neural networks, batch normalization was applied in all the layers, which reduced the internal covariance shift and regularization. As an additional regularizer dropout[36] with probability = 0.5 was introduced. There was no effect on using L2-regularization. To randomly initialize the weights Kaiming and Hee approach was used. Biases were initialized to zero.

4.5.1 Effective Learning Rate Identification Results

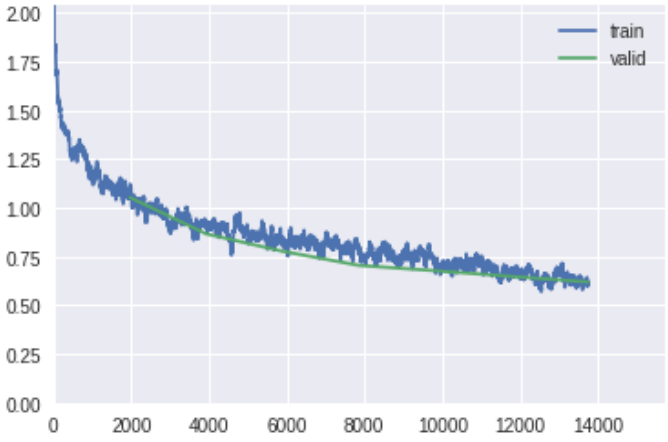
To identify the correct learning rate different values were tried out, results of which are as follows in form of *figure 12*:

Figure 11 Image describing performance for different learning rate

As you can clearly see that corresponding to learning rate = 1e-02 the Loss is minimized so it is a good starting point while training the model.

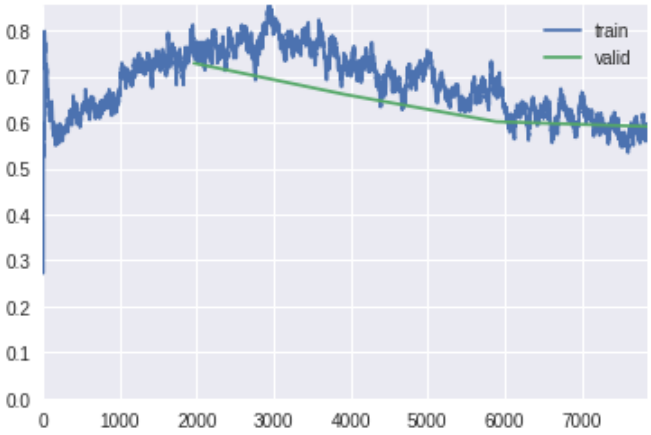
Training with Learning Rate = 1e-03 and weight decay 1e-06

epoch	train_loss	valid_loss	accuracy	time
0	1.104533	1.052842	0.621869	45:09
1	0.920231	0.866202	0.673997	44:55
2	0.803812	0.777084	0.705559	45:03
3	0.764604	0.704630	0.733455	44:49
4	0.691905	0.677392	0.743637	44:47
5	0.680551	0.648187	0.755447	44:52
6	0.599412	0.619531	0.768275	44:48



Training with Learning Rate = max learning rate, /3, /9 and weight decay 1e-06

epoch	train_loss	valid_loss	accuracy	time
0	0.757510	0.728719	0.730605	57:43
1	0.741443	0.660608	0.751578	57:34
2	0.683744	0.601534	0.767461	57:44
3	0.596244	0.591047	0.772551	57:55



Training with Learning Rate = max learning rate/3, /27, /9 and weight decay 1e-06

epoch	train_loss	valid_loss	accuracy	time
0	0.563727	0.533711	0.794339	56:48
1	0.590286	0.514370	0.804724	56:09
2	0.536467	0.500096	0.811647	56:23
3	0.490264	0.494299	0.813480	56:29

4.6 Test Results - Confusion Matrix

There are several machine learning algorithms but over the years of development general characteristics of learning and performance evaluation criteria have been established. The general idea is to divide the currently present data into training data, validation data and test data parts. The training and validation parts are used in the learning of the algorithm and construction of the model while the test dataset is utilized to evaluate the model performance on novel data that it has never observed beforehand in its learning. There are various performance measures of the model depending on whether it is a classification or regression problem. These measures include mean square error, absolute error and cross validation techniques for regression problems and prediction error, precision, recall, F-measure (weighted average of precision and recall of the model), confusion matrix, ROC (sensitivity vs specificity curves) and AUC curves for classification problems. The initial work by C Shang in 1996 involved developing an absolute error based algorithm for evaluating communication channel equalisers adapting single-layer perceptron models. Research by Paolo Sonego in 2008 shows the application of ROC curve estimates to evaluate performance in biomedical application models and classification of biological sequences and 3D structures. Recently, precision, recall and F-measures have found their application in evaluating models in text classification for sentiment analysis and social media analytics.

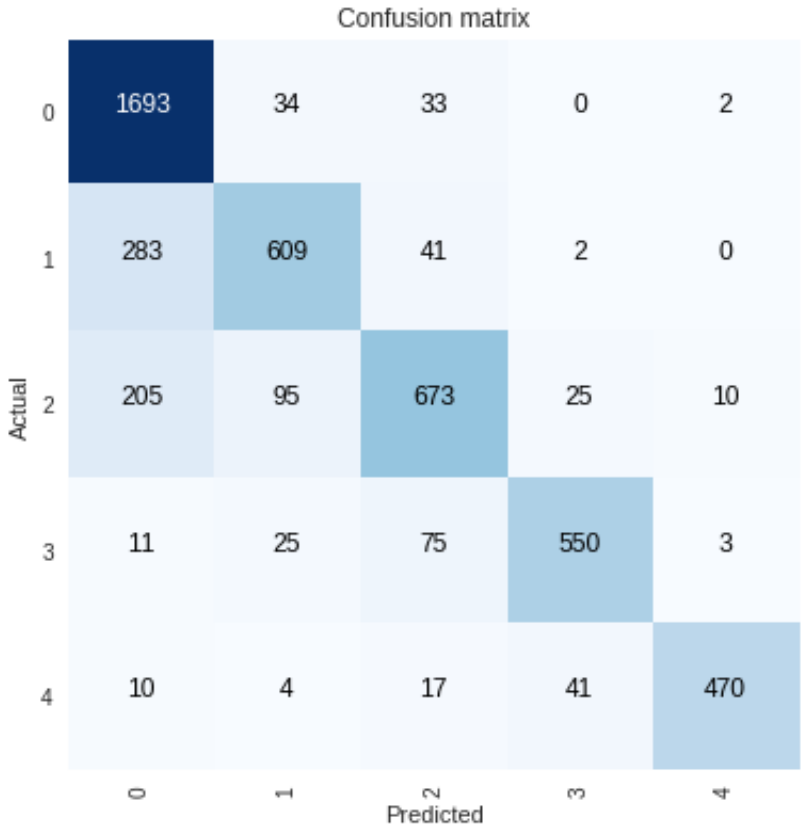


Figure 17 Test Results confusion Matrix

Table 2 Demonstrating Final Result of DenseNET Classification

Class/Metric (%)	Precision	Sensitivity	F1-Score	Accuracy
Normal	96.08	76.89	85.00	88.23
Mild	65.13	79.40	72.00	90.24
Moderate	66.77	80.22	73.00	90.00
Severe	82.83	89.00	86.00	96.29
Very Severe	86.72	96.90	92.00	98.23
Total	80.00	85.00	81.6	**92.6**

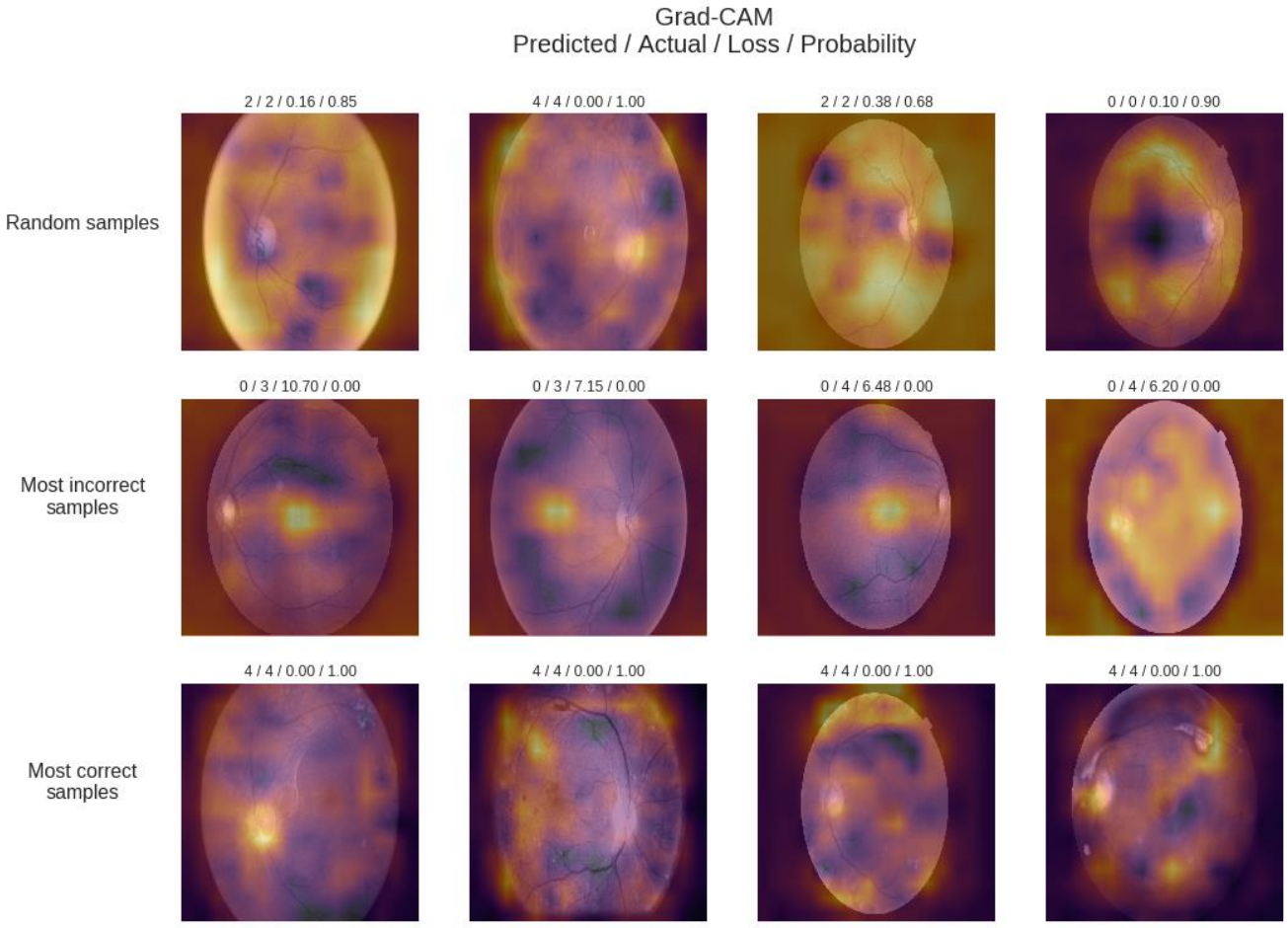


Figure 12 Demonstration of results for Random, Incorrect and Correctly Predicted

Chapter 5.

Summary and Prospectives

Medical diagnosis is heavily supported by Medical Imaging. The analysis of medical images requires highly specialized expertise that is provided by specialized doctors. The advent of advanced machine learning techniques, like deep learning, is facilitating the design of high performance automatic classifiers in a broad range of applications, also for medical diagnosis. The purpose of this thesis is the exploration of new automatic diagnostic methods for medical diagnosis, concretely for diabetic retinopathy disease grading. As stated before in this work, Diabetic Retinopathy is one of the main causes of blindness in the world. Early detection can reduce disease progression and consequently also the incidence of blindness. The diagnostic of DR is done primarily by retina fundus image analysis, that is done by ophthalmologists specifically trained for this purpose.

Automatic diagnostic systems for DIABETIC RETINOPATHY can reduce dramatically not only the costs associated to diagnostic, but also the probability of developing blindness in the general population. For this purpose we use supervised deep learning techniques. Given a class differentiation based on objective properties present in data, these parameterized models are able to learn the statistical regularities that have to be taken into account to separate the images. In this thesis, convolutional neural networks are used, which are efficient neural networks designed for exploiting local high correlations present in images.

Machine learning methods in general and deep learning in particular are models that learn from data. Thus, a key factor for the success is having a statistically representative dataset of the population from which we want to predict a concrete property, in this case, a disease class. For this purpose it is required to have a labeled dataset with enough samples (that are of the order of magnitude of thousands elements per class) in order to create models with good generalization. For this purpose, we use a public dataset of EyePACS.

After the work done in this thesis, we can conclude that designed models can be used successfully as a high confidence diagnostic tool that can help to reduce costs in diagnostic and also to reduce the incidence of the DIABETIC RETINOPATHY disease in the general population.

Future research lines can be focused on the next different directions, that we explain below:

1. *Reinforcement Learning*: Adding to the models the possibility of enhancing its performance, designing online learning methods that allow continuous learning of networks from the corrections done by ophthalmologists on inference time. Hospital Universitari Sant Joan de Reus wants to use the classification model developed in this thesis in its daily work. This needs a previous process of integration of computer systems as well as the permission of Catalan Health Care authorities. If it is finally done, learning from online work could be very interesting.

2. *Increase the number of classes to predict*: One of the lines to explore is increasing the number of properties to predict of the original model. Instead of only predicting diabetic retinopathy class,

other classes can also be included.

3. Use of Interpretation Model in other applications: The designed interpretation model is domain independent, therefore, it would be interesting the study of its application in other domains, ie. for interpretation of other medical imaging classification tasks like for example, radiology, ultrasound, or other related imaging applications.

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). Comput. Vis. Image Underst. 110, 3 (June 2008), 346-359. DOI=http://dx.doi.org/10.1016/j.cviu.2007.09.014

[2] Shah AR, Gardner TW. Diabetic retinopathy: research to clinical practice. Clin Diabetes Endocrinol. 2017;3:9. Published 2017 Oct 19. doi:10.1186/s40842-017-0047-y

[3] Gardner TW, Chew EY. Future opportunities in diabetic retinopathy research. Curr Opin Endocrinol Diabetes Obes. 2016;23(2):91–96. doi:10.1097/MED.0000000000000238

[4] Lee R, Wong TY, Sabanayagam C. Epidemiology of diabetic retinopathy, diabetic macular edema and related vision loss. Eye Vis (Lond). 2015;2:17. Published 2015 Sep 30. doi:10.1186/s40662-015-0026-2

[5] Ankita Gupta, Rita Chhikara, Diabetic Retinopathy: Present and Past, Procedia Computer Science, Volume 132, 2018, Pages 1432-1440, ISSN 1877-0509

[6] S. izza Rufaida and M. I. Fanany, "Residual convolutional neural network for diabetic retinopathy," in Proceedings of IEEE International Conference on Advanced Computer Science and Information Systems (ICACSIS). IEEE, 2017, pp. 367–374.

[7] D. Zhang et al., "Diabetic retinopathy classification using deeply supervised ResNet," in Proceedings of IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation (SmartWorld /SCALCOM/UIC /ATC/ CBDCom/IOP/SCI). IEEE, 2017.

[8] M. Alban and T. Gilligan, "Automated detection of diabetic retinopathy using fluorescein angiography photographs," Report of Standford Education, 2016.

[9] Z. Wang et al., "Zoom-in-net: Deep mining lesions for diabetic retinopathy detection," in Proceeding of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI). Springer, 2017, pp. 267–275

[10] Xavier Glorot, Yoshua Bengio ; Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

[11] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10), Johannes Fürnkranz and Thorsten Joachims (Eds.). Omnipress, USA, 807-814.

[12] Xu, Bing, Naiyan Wang, Tianqi Chen and Mu Li. "Empirical Evaluation of Rectified Activations in Convolutional Network." ArXiv abs/1505.00853 (2015): n. pag.

[13] Websites for data : http://www.ces.clemson.edu/~ahoover/stare/ , https://www.researchgate.net/deref/http%3A%2F%2Fwww5.cs.fau.de%2Fresearch%2Fdata%2Ffundus-images%2F

[14] Websites for data : https://www.kaggle.com/c/diabetic-retinopathy-detection

[15] Websites for data : http://www.adcis.net/en/third-party/e-ophtha/

[16] Johnson, J.M., Khoshgoftaar, T.M. Survey on deep learning with class imbalance. *J Big Data* **6,** 27 (2019) doi:10.1186/s40537-019-0192-5

[17] Alberto Bietti and Julien Mairal. 2017. Invariance and stability of deep convolutional representations. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (NIPS'17), Ulrike von Luxburg, Isabelle Guyon, Samy Bengio, Hanna Wallach, and Rob Fergus (Eds.). Curran Associates Inc., USA, 6211-6221.

[18] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J Big Data* **6,** 60 (2019) doi:10.1186/s40537-019-0197-0

[19] Hussain Z, Gimenez F, Yi D, Rubin D. Differential Data Augmentation Techniques for Medical Imaging Classification Tasks. *AMIA Annu Symp Proc*. 2018;2017:979–984. Published 2018 Apr 16.

[20] Ma H, Nie Y (2018) A two-stage filter for removing salt-and-pepper noise using noise detector based on characteristic difference parameter and adaptive directional mean filter. PLoS ONE 13(10): e0205736

[21] Yamashita, R., Nishio, M., Do, R.K.G. et al. Insights Imaging (2018) 9: 611. https://doi.org/10.1007/s13244-018-0639-9

[22] Chen H, Zhang Y, Zhang W et al (2017) Low-dose CT via convolutional neural network. Biomed Opt Express 8:679–694

[23] Martin Gjoreski, Stefan Kalabakov, Mitja Luštrek, Matjaž Gams, and Hristijan Gjoreski. 2019. Cross-dataset deep transfer learning for activity recognition. In Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers

[24] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778.

[25] Y. Zhu and S. Newsam, "DenseNet for dense flow," 2017 IEEE International Conference on Image Processing (ICIP), Beijing, 2017, pp. 790-794.

[26] Yu, F.; Koltun, V. Multi-scale context aggregation by dilated convolutions. arXiv, 2015; arXiv:1511.07122

[27] Li Y, Xie X, Shen L, Liu S. Reverse active learning based atrous DenseNet for pathological image classification. BMC Bioinformatics. 2019;20(1):445. Published 2019 Aug 28.

[28] EyePacs Dataset https://www.medicmind.tech/resources-2/

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.

[30] Pedro Ballester and Ricardo Matsumura Araujo. 2016. On the performance of GoogLeNet and AlexNet applied to sketches. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (AAAI'16). AAAI Press 1124-1128.

[31] arXiv:1409.1556 [cs.CV]

[32] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and *F*-score, with implication for evaluation. In *Proceedings of the 27th European conference on Advances in Information Retrieval Research* (ECIR'05), David E. Losada and Juan M. Fernández-Luna (Eds.). Springer-Verlag, Berlin, Heidelberg, 345-359. DOI=http://dx.doi.org/10.1007/978-3-540-31865-1_25

[33] Yong Xu, Jie Wen, Lunke Fei, Zheng Zhang, "Review of Video and Image Defogging Algorithms and Related Studies on Image Restoration and Enhancement", Access IEEE, vol. 4, pp. 165-188, 2016.

[34] S. Pei and T. Lee, "Nighttime haze removal using color transfer pre-processing and Dark Channel Prior," 2012 19th IEEE International Conference on Image Processing, Orlando, FL, 2012, pp. 957-960.

[35] Guan, Ziqiang, Ritesh Kumar, Yi Ren Fung, Yeahuay Wu and Madalina Fiterau. “A Comprehensive Study of Alzheimer's Disease Classification Using Convolutional Neural Networks.” ArXiv abs/1904.07950 (2019): n. pag.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1 (January 2014), 1929-1958.

[37] Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-rectified-linear-unit-ReLU-the-leaky-ReLU-LReLU-a-01-the-shifted-ReLUs_fig1_284579051 [accessed 21 Dec, 2019]

[38] A. G. Howard et al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.

Appendix

Model Summary

DenseNET :

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 256]           9,408
       BatchNorm2d-2         [-1, 64, 256, 256]             128
              ReLU-3         [-1, 64, 256, 256]               0
         MaxPool2d-4         [-1, 64, 128, 128]               0
       BatchNorm2d-5         [-1, 64, 128, 128]             128
              ReLU-6         [-1, 64, 128, 128]               0
            Conv2d-7        [-1, 128, 128, 128]           8,192
       BatchNorm2d-8        [-1, 128, 128, 128]             256
              ReLU-9        [-1, 128, 128, 128]               0
           Conv2d-10         [-1, 32, 128, 128]          36,864
      BatchNorm2d-11         [-1, 96, 128, 128]             192
             ReLU-12         [-1, 96, 128, 128]               0
           Conv2d-13        [-1, 128, 128, 128]          12,288
      BatchNorm2d-14        [-1, 128, 128, 128]             256
             ReLU-15        [-1, 128, 128, 128]               0
           Conv2d-16         [-1, 32, 128, 128]          36,864
      BatchNorm2d-17        [-1, 128, 128, 128]             256
             ReLU-18        [-1, 128, 128, 128]               0
           Conv2d-19        [-1, 128, 128, 128]          16,384
      BatchNorm2d-20        [-1, 128, 128, 128]             256
             ReLU-21        [-1, 128, 128, 128]               0
           Conv2d-22         [-1, 32, 128, 128]          36,864
      BatchNorm2d-23        [-1, 160, 128, 128]             320
             ReLU-24        [-1, 160, 128, 128]               0
           Conv2d-25        [-1, 128, 128, 128]          20,480
      BatchNorm2d-26        [-1, 128, 128, 128]             256
             ReLU-27        [-1, 128, 128, 128]               0
           Conv2d-28         [-1, 32, 128, 128]          36,864
      BatchNorm2d-29        [-1, 192, 128, 128]             384
             ReLU-30        [-1, 192, 128, 128]               0
           Conv2d-31        [-1, 128, 128, 128]          24,576
      BatchNorm2d-32        [-1, 128, 128, 128]             256
             ReLU-33        [-1, 128, 128, 128]               0
           Conv2d-34         [-1, 32, 128, 128]          36,864
      BatchNorm2d-35        [-1, 224, 128, 128]             448
             ReLU-36        [-1, 224, 128, 128]               0
           Conv2d-37        [-1, 128, 128, 128]          28,672
      BatchNorm2d-38        [-1, 128, 128, 128]             256
             ReLU-39        [-1, 128, 128, 128]               0
           Conv2d-40         [-1, 32, 128, 128]          36,864
      BatchNorm2d-41        [-1, 256, 128, 128]             512
             ReLU-42        [-1, 256, 128, 128]               0
           Conv2d-43        [-1, 128, 128, 128]          32,768
```

```
  AvgPool2d-44          [-1, 128, 64, 64]               0
BatchNorm2d-45          [-1, 128, 64, 64]             256
       ReLU-46          [-1, 128, 64, 64]               0
     Conv2d-47          [-1, 128, 64, 64]          16,384
BatchNorm2d-48          [-1, 128, 64, 64]             256
       ReLU-49          [-1, 128, 64, 64]               0
     Conv2d-50           [-1, 32, 64, 64]          36,864
BatchNorm2d-51          [-1, 160, 64, 64]             320
       ReLU-52          [-1, 160, 64, 64]               0
     Conv2d-53          [-1, 128, 64, 64]          20,480
BatchNorm2d-54          [-1, 128, 64, 64]             256
       ReLU-55          [-1, 128, 64, 64]               0
     Conv2d-56           [-1, 32, 64, 64]          36,864
BatchNorm2d-57          [-1, 192, 64, 64]             384
       ReLU-58          [-1, 192, 64, 64]               0
     Conv2d-59          [-1, 128, 64, 64]          24,576
BatchNorm2d-60          [-1, 128, 64, 64]             256
       ReLU-61          [-1, 128, 64, 64]               0
     Conv2d-62           [-1, 32, 64, 64]          36,864
BatchNorm2d-63          [-1, 224, 64, 64]             448
       ReLU-64          [-1, 224, 64, 64]               0
     Conv2d-65          [-1, 128, 64, 64]          28,672
BatchNorm2d-66          [-1, 128, 64, 64]             256
       ReLU-67          [-1, 128, 64, 64]               0
     Conv2d-68           [-1, 32, 64, 64]          36,864
BatchNorm2d-69          [-1, 256, 64, 64]             512
       ReLU-70          [-1, 256, 64, 64]               0
     Conv2d-71          [-1, 128, 64, 64]          32,768
BatchNorm2d-72          [-1, 128, 64, 64]             256
       ReLU-73          [-1, 128, 64, 64]               0
     Conv2d-74           [-1, 32, 64, 64]          36,864
BatchNorm2d-75          [-1, 288, 64, 64]             576
       ReLU-76          [-1, 288, 64, 64]               0
     Conv2d-77          [-1, 128, 64, 64]          36,864
BatchNorm2d-78          [-1, 128, 64, 64]             256
       ReLU-79          [-1, 128, 64, 64]               0
     Conv2d-80           [-1, 32, 64, 64]          36,864
BatchNorm2d-81          [-1, 320, 64, 64]             640
       ReLU-82          [-1, 320, 64, 64]               0
     Conv2d-83          [-1, 128, 64, 64]          40,960
BatchNorm2d-84          [-1, 128, 64, 64]             256
       ReLU-85          [-1, 128, 64, 64]               0
     Conv2d-86           [-1, 32, 64, 64]          36,864
BatchNorm2d-87          [-1, 352, 64, 64]             704
       ReLU-88          [-1, 352, 64, 64]               0
     Conv2d-89          [-1, 128, 64, 64]          45,056
BatchNorm2d-90          [-1, 128, 64, 64]             256
       ReLU-91          [-1, 128, 64, 64]               0
     Conv2d-92           [-1, 32, 64, 64]          36,864
BatchNorm2d-93          [-1, 384, 64, 64]             768
       ReLU-94          [-1, 384, 64, 64]               0
     Conv2d-95          [-1, 128, 64, 64]          49,152
BatchNorm2d-96          [-1, 128, 64, 64]             256
       ReLU-97          [-1, 128, 64, 64]               0
     Conv2d-98           [-1, 32, 64, 64]          36,864
BatchNorm2d-99          [-1, 416, 64, 64]             832
      ReLU-100          [-1, 416, 64, 64]               0
```

```
     Conv2d-101         [-1, 128, 64, 64]          53,248
BatchNorm2d-102         [-1, 128, 64, 64]             256
       ReLU-103         [-1, 128, 64, 64]               0
     Conv2d-104          [-1, 32, 64, 64]          36,864
BatchNorm2d-105         [-1, 448, 64, 64]             896
       ReLU-106         [-1, 448, 64, 64]               0
     Conv2d-107         [-1, 128, 64, 64]          57,344
BatchNorm2d-108         [-1, 128, 64, 64]             256
       ReLU-109         [-1, 128, 64, 64]               0
     Conv2d-110          [-1, 32, 64, 64]          36,864
BatchNorm2d-111         [-1, 480, 64, 64]             960
       ReLU-112         [-1, 480, 64, 64]               0
     Conv2d-113         [-1, 128, 64, 64]          61,440
BatchNorm2d-114         [-1, 128, 64, 64]             256
       ReLU-115         [-1, 128, 64, 64]               0
     Conv2d-116          [-1, 32, 64, 64]          36,864
BatchNorm2d-117         [-1, 512, 64, 64]           1,024
       ReLU-118         [-1, 512, 64, 64]               0
     Conv2d-119         [-1, 256, 64, 64]         131,072
  AvgPool2d-120         [-1, 256, 32, 32]               0
BatchNorm2d-121         [-1, 256, 32, 32]             512
       ReLU-122         [-1, 256, 32, 32]               0
     Conv2d-123         [-1, 128, 32, 32]          32,768
BatchNorm2d-124         [-1, 128, 32, 32]             256
       ReLU-125         [-1, 128, 32, 32]               0
     Conv2d-126          [-1, 32, 32, 32]          36,864
BatchNorm2d-127         [-1, 288, 32, 32]             576
       ReLU-128         [-1, 288, 32, 32]               0
     Conv2d-129         [-1, 128, 32, 32]          36,864
BatchNorm2d-130         [-1, 128, 32, 32]             256
       ReLU-131         [-1, 128, 32, 32]               0
     Conv2d-132          [-1, 32, 32, 32]          36,864
BatchNorm2d-133         [-1, 320, 32, 32]             640
       ReLU-134         [-1, 320, 32, 32]               0
     Conv2d-135         [-1, 128, 32, 32]          40,960
BatchNorm2d-136         [-1, 128, 32, 32]             256
       ReLU-137         [-1, 128, 32, 32]               0
     Conv2d-138          [-1, 32, 32, 32]          36,864
BatchNorm2d-139         [-1, 352, 32, 32]             704
       ReLU-140         [-1, 352, 32, 32]               0
     Conv2d-141         [-1, 128, 32, 32]          45,056
BatchNorm2d-142         [-1, 128, 32, 32]             256
       ReLU-143         [-1, 128, 32, 32]               0
     Conv2d-144          [-1, 32, 32, 32]          36,864
BatchNorm2d-145         [-1, 384, 32, 32]             768
       ReLU-146         [-1, 384, 32, 32]               0
     Conv2d-147         [-1, 128, 32, 32]          49,152
BatchNorm2d-148         [-1, 128, 32, 32]             256
       ReLU-149         [-1, 128, 32, 32]               0
     Conv2d-150          [-1, 32, 32, 32]          36,864
BatchNorm2d-151         [-1, 416, 32, 32]             832
       ReLU-152         [-1, 416, 32, 32]               0
     Conv2d-153         [-1, 128, 32, 32]          53,248
BatchNorm2d-154         [-1, 128, 32, 32]             256
       ReLU-155         [-1, 128, 32, 32]               0
     Conv2d-156          [-1, 32, 32, 32]          36,864
BatchNorm2d-157         [-1, 448, 32, 32]             896
```

```
       ReLU-158          [-1, 448, 32, 32]               0
     Conv2d-159          [-1, 128, 32, 32]          57,344
BatchNorm2d-160          [-1, 128, 32, 32]             256
       ReLU-161          [-1, 128, 32, 32]               0
     Conv2d-162           [-1, 32, 32, 32]          36,864
BatchNorm2d-163          [-1, 480, 32, 32]             960
       ReLU-164          [-1, 480, 32, 32]               0
     Conv2d-165          [-1, 128, 32, 32]          61,440
BatchNorm2d-166          [-1, 128, 32, 32]             256
       ReLU-167          [-1, 128, 32, 32]               0
     Conv2d-168           [-1, 32, 32, 32]          36,864
BatchNorm2d-169          [-1, 512, 32, 32]           1,024
       ReLU-170          [-1, 512, 32, 32]               0
     Conv2d-171          [-1, 128, 32, 32]          65,536
BatchNorm2d-172          [-1, 128, 32, 32]             256
       ReLU-173          [-1, 128, 32, 32]               0
     Conv2d-174           [-1, 32, 32, 32]          36,864
BatchNorm2d-175          [-1, 544, 32, 32]           1,088
       ReLU-176          [-1, 544, 32, 32]               0
     Conv2d-177          [-1, 128, 32, 32]          69,632
BatchNorm2d-178          [-1, 128, 32, 32]             256
       ReLU-179          [-1, 128, 32, 32]               0
     Conv2d-180           [-1, 32, 32, 32]          36,864
BatchNorm2d-181          [-1, 576, 32, 32]           1,152
       ReLU-182          [-1, 576, 32, 32]               0
     Conv2d-183          [-1, 128, 32, 32]          73,728
BatchNorm2d-184          [-1, 128, 32, 32]             256
       ReLU-185          [-1, 128, 32, 32]               0
     Conv2d-186           [-1, 32, 32, 32]          36,864
BatchNorm2d-187          [-1, 608, 32, 32]           1,216
       ReLU-188          [-1, 608, 32, 32]               0
     Conv2d-189          [-1, 128, 32, 32]          77,824
BatchNorm2d-190          [-1, 128, 32, 32]             256
       ReLU-191          [-1, 128, 32, 32]               0
     Conv2d-192           [-1, 32, 32, 32]          36,864
BatchNorm2d-193          [-1, 640, 32, 32]           1,280
       ReLU-194          [-1, 640, 32, 32]               0
     Conv2d-195          [-1, 128, 32, 32]          81,920
BatchNorm2d-196          [-1, 128, 32, 32]             256
       ReLU-197          [-1, 128, 32, 32]               0
     Conv2d-198           [-1, 32, 32, 32]          36,864
BatchNorm2d-199          [-1, 672, 32, 32]           1,344
       ReLU-200          [-1, 672, 32, 32]               0
     Conv2d-201          [-1, 128, 32, 32]          86,016
BatchNorm2d-202          [-1, 128, 32, 32]             256
       ReLU-203          [-1, 128, 32, 32]               0
     Conv2d-204           [-1, 32, 32, 32]          36,864
BatchNorm2d-205          [-1, 704, 32, 32]           1,408
       ReLU-206          [-1, 704, 32, 32]               0
     Conv2d-207          [-1, 128, 32, 32]          90,112
BatchNorm2d-208          [-1, 128, 32, 32]             256
       ReLU-209          [-1, 128, 32, 32]               0
     Conv2d-210           [-1, 32, 32, 32]          36,864
BatchNorm2d-211          [-1, 736, 32, 32]           1,472
       ReLU-212          [-1, 736, 32, 32]               0
     Conv2d-213          [-1, 128, 32, 32]          94,208
BatchNorm2d-214          [-1, 128, 32, 32]             256
```

```
       ReLU-215          [-1, 128, 32, 32]               0
     Conv2d-216           [-1, 32, 32, 32]          36,864
BatchNorm2d-217          [-1, 768, 32, 32]           1,536
       ReLU-218          [-1, 768, 32, 32]               0
     Conv2d-219          [-1, 128, 32, 32]          98,304
BatchNorm2d-220          [-1, 128, 32, 32]             256
       ReLU-221          [-1, 128, 32, 32]               0
     Conv2d-222           [-1, 32, 32, 32]          36,864
BatchNorm2d-223          [-1, 800, 32, 32]           1,600
       ReLU-224          [-1, 800, 32, 32]               0
     Conv2d-225          [-1, 128, 32, 32]         102,400
BatchNorm2d-226          [-1, 128, 32, 32]             256
       ReLU-227          [-1, 128, 32, 32]               0
     Conv2d-228           [-1, 32, 32, 32]          36,864
BatchNorm2d-229          [-1, 832, 32, 32]           1,664
       ReLU-230          [-1, 832, 32, 32]               0
     Conv2d-231          [-1, 128, 32, 32]         106,496
BatchNorm2d-232          [-1, 128, 32, 32]             256
       ReLU-233          [-1, 128, 32, 32]               0
     Conv2d-234           [-1, 32, 32, 32]          36,864
BatchNorm2d-235          [-1, 864, 32, 32]           1,728
       ReLU-236          [-1, 864, 32, 32]               0
     Conv2d-237          [-1, 128, 32, 32]         110,592
BatchNorm2d-238          [-1, 128, 32, 32]             256
       ReLU-239          [-1, 128, 32, 32]               0
     Conv2d-240           [-1, 32, 32, 32]          36,864
BatchNorm2d-241          [-1, 896, 32, 32]           1,792
       ReLU-242          [-1, 896, 32, 32]               0
     Conv2d-243          [-1, 128, 32, 32]         114,688
BatchNorm2d-244          [-1, 128, 32, 32]             256
       ReLU-245          [-1, 128, 32, 32]               0
     Conv2d-246           [-1, 32, 32, 32]          36,864
BatchNorm2d-247          [-1, 928, 32, 32]           1,856
       ReLU-248          [-1, 928, 32, 32]               0
     Conv2d-249          [-1, 128, 32, 32]         118,784
BatchNorm2d-250          [-1, 128, 32, 32]             256
       ReLU-251          [-1, 128, 32, 32]               0
     Conv2d-252           [-1, 32, 32, 32]          36,864
BatchNorm2d-253          [-1, 960, 32, 32]           1,920
       ReLU-254          [-1, 960, 32, 32]               0
     Conv2d-255          [-1, 128, 32, 32]         122,880
BatchNorm2d-256          [-1, 128, 32, 32]             256
       ReLU-257          [-1, 128, 32, 32]               0
     Conv2d-258           [-1, 32, 32, 32]          36,864
BatchNorm2d-259          [-1, 992, 32, 32]           1,984
       ReLU-260          [-1, 992, 32, 32]               0
     Conv2d-261          [-1, 128, 32, 32]         126,976
BatchNorm2d-262          [-1, 128, 32, 32]             256
       ReLU-263          [-1, 128, 32, 32]               0
     Conv2d-264           [-1, 32, 32, 32]          36,864
BatchNorm2d-265         [-1, 1024, 32, 32]           2,048
       ReLU-266         [-1, 1024, 32, 32]               0
     Conv2d-267          [-1, 128, 32, 32]         131,072
BatchNorm2d-268          [-1, 128, 32, 32]             256
       ReLU-269          [-1, 128, 32, 32]               0
     Conv2d-270           [-1, 32, 32, 32]          36,864
BatchNorm2d-271         [-1, 1056, 32, 32]           2,112
```

```
       ReLU-272         [-1, 1056, 32, 32]               0
     Conv2d-273          [-1, 128, 32, 32]         135,168
BatchNorm2d-274          [-1, 128, 32, 32]             256
       ReLU-275          [-1, 128, 32, 32]               0
     Conv2d-276           [-1, 32, 32, 32]          36,864
BatchNorm2d-277         [-1, 1088, 32, 32]           2,176
       ReLU-278         [-1, 1088, 32, 32]               0
     Conv2d-279          [-1, 128, 32, 32]         139,264
BatchNorm2d-280          [-1, 128, 32, 32]             256
       ReLU-281          [-1, 128, 32, 32]               0
     Conv2d-282           [-1, 32, 32, 32]          36,864
BatchNorm2d-283         [-1, 1120, 32, 32]           2,240
       ReLU-284         [-1, 1120, 32, 32]               0
     Conv2d-285          [-1, 128, 32, 32]         143,360
BatchNorm2d-286          [-1, 128, 32, 32]             256
       ReLU-287          [-1, 128, 32, 32]               0
     Conv2d-288           [-1, 32, 32, 32]          36,864
BatchNorm2d-289         [-1, 1152, 32, 32]           2,304
       ReLU-290         [-1, 1152, 32, 32]               0
     Conv2d-291          [-1, 128, 32, 32]         147,456
BatchNorm2d-292          [-1, 128, 32, 32]             256
       ReLU-293          [-1, 128, 32, 32]               0
     Conv2d-294           [-1, 32, 32, 32]          36,864
BatchNorm2d-295         [-1, 1184, 32, 32]           2,368
       ReLU-296         [-1, 1184, 32, 32]               0
     Conv2d-297          [-1, 128, 32, 32]         151,552
BatchNorm2d-298          [-1, 128, 32, 32]             256
       ReLU-299          [-1, 128, 32, 32]               0
     Conv2d-300           [-1, 32, 32, 32]          36,864
BatchNorm2d-301         [-1, 1216, 32, 32]           2,432
       ReLU-302         [-1, 1216, 32, 32]               0
     Conv2d-303          [-1, 128, 32, 32]         155,648
BatchNorm2d-304          [-1, 128, 32, 32]             256
       ReLU-305          [-1, 128, 32, 32]               0
     Conv2d-306           [-1, 32, 32, 32]          36,864
BatchNorm2d-307         [-1, 1248, 32, 32]           2,496
       ReLU-308         [-1, 1248, 32, 32]               0
     Conv2d-309          [-1, 128, 32, 32]         159,744
BatchNorm2d-310          [-1, 128, 32, 32]             256
       ReLU-311          [-1, 128, 32, 32]               0
     Conv2d-312           [-1, 32, 32, 32]          36,864
BatchNorm2d-313         [-1, 1280, 32, 32]           2,560
       ReLU-314         [-1, 1280, 32, 32]               0
     Conv2d-315          [-1, 128, 32, 32]         163,840
BatchNorm2d-316          [-1, 128, 32, 32]             256
       ReLU-317          [-1, 128, 32, 32]               0
     Conv2d-318           [-1, 32, 32, 32]          36,864
BatchNorm2d-319         [-1, 1312, 32, 32]           2,624
       ReLU-320         [-1, 1312, 32, 32]               0
     Conv2d-321          [-1, 128, 32, 32]         167,936
BatchNorm2d-322          [-1, 128, 32, 32]             256
       ReLU-323          [-1, 128, 32, 32]               0
     Conv2d-324           [-1, 32, 32, 32]          36,864
BatchNorm2d-325         [-1, 1344, 32, 32]           2,688
       ReLU-326         [-1, 1344, 32, 32]               0
     Conv2d-327          [-1, 128, 32, 32]         172,032
BatchNorm2d-328          [-1, 128, 32, 32]             256
```

```
       ReLU-329          [-1, 128, 32, 32]               0
     Conv2d-330           [-1, 32, 32, 32]          36,864
BatchNorm2d-331         [-1, 1376, 32, 32]           2,752
       ReLU-332         [-1, 1376, 32, 32]               0
     Conv2d-333          [-1, 128, 32, 32]         176,128
BatchNorm2d-334          [-1, 128, 32, 32]             256
       ReLU-335          [-1, 128, 32, 32]               0
     Conv2d-336           [-1, 32, 32, 32]          36,864
BatchNorm2d-337         [-1, 1408, 32, 32]           2,816
       ReLU-338         [-1, 1408, 32, 32]               0
     Conv2d-339          [-1, 128, 32, 32]         180,224
BatchNorm2d-340          [-1, 128, 32, 32]             256
       ReLU-341          [-1, 128, 32, 32]               0
     Conv2d-342           [-1, 32, 32, 32]          36,864
BatchNorm2d-343         [-1, 1440, 32, 32]           2,880
       ReLU-344         [-1, 1440, 32, 32]               0
     Conv2d-345          [-1, 128, 32, 32]         184,320
BatchNorm2d-346          [-1, 128, 32, 32]             256
       ReLU-347          [-1, 128, 32, 32]               0
     Conv2d-348           [-1, 32, 32, 32]          36,864
BatchNorm2d-349         [-1, 1472, 32, 32]           2,944
       ReLU-350         [-1, 1472, 32, 32]               0
     Conv2d-351          [-1, 128, 32, 32]         188,416
BatchNorm2d-352          [-1, 128, 32, 32]             256
       ReLU-353          [-1, 128, 32, 32]               0
     Conv2d-354           [-1, 32, 32, 32]          36,864
BatchNorm2d-355         [-1, 1504, 32, 32]           3,008
       ReLU-356         [-1, 1504, 32, 32]               0
     Conv2d-357          [-1, 128, 32, 32]         192,512
BatchNorm2d-358          [-1, 128, 32, 32]             256
       ReLU-359          [-1, 128, 32, 32]               0
     Conv2d-360           [-1, 32, 32, 32]          36,864
BatchNorm2d-361         [-1, 1536, 32, 32]           3,072
       ReLU-362         [-1, 1536, 32, 32]               0
     Conv2d-363          [-1, 128, 32, 32]         196,608
BatchNorm2d-364          [-1, 128, 32, 32]             256
       ReLU-365          [-1, 128, 32, 32]               0
     Conv2d-366           [-1, 32, 32, 32]          36,864
BatchNorm2d-367         [-1, 1568, 32, 32]           3,136
       ReLU-368         [-1, 1568, 32, 32]               0
     Conv2d-369          [-1, 128, 32, 32]         200,704
BatchNorm2d-370          [-1, 128, 32, 32]             256
       ReLU-371          [-1, 128, 32, 32]               0
     Conv2d-372           [-1, 32, 32, 32]          36,864
BatchNorm2d-373         [-1, 1600, 32, 32]           3,200
       ReLU-374         [-1, 1600, 32, 32]               0
     Conv2d-375          [-1, 128, 32, 32]         204,800
BatchNorm2d-376          [-1, 128, 32, 32]             256
       ReLU-377          [-1, 128, 32, 32]               0
     Conv2d-378           [-1, 32, 32, 32]          36,864
BatchNorm2d-379         [-1, 1632, 32, 32]           3,264
       ReLU-380         [-1, 1632, 32, 32]               0
     Conv2d-381          [-1, 128, 32, 32]         208,896
BatchNorm2d-382          [-1, 128, 32, 32]             256
       ReLU-383          [-1, 128, 32, 32]               0
     Conv2d-384           [-1, 32, 32, 32]          36,864
BatchNorm2d-385         [-1, 1664, 32, 32]           3,328
```

```
       ReLU-386         [-1, 1664, 32, 32]               0
     Conv2d-387          [-1, 128, 32, 32]         212,992
BatchNorm2d-388          [-1, 128, 32, 32]             256
       ReLU-389          [-1, 128, 32, 32]               0
     Conv2d-390           [-1, 32, 32, 32]          36,864
BatchNorm2d-391         [-1, 1696, 32, 32]           3,392
       ReLU-392         [-1, 1696, 32, 32]               0
     Conv2d-393          [-1, 128, 32, 32]         217,088
BatchNorm2d-394          [-1, 128, 32, 32]             256
       ReLU-395          [-1, 128, 32, 32]               0
     Conv2d-396           [-1, 32, 32, 32]          36,864
BatchNorm2d-397         [-1, 1728, 32, 32]           3,456
       ReLU-398         [-1, 1728, 32, 32]               0
     Conv2d-399          [-1, 128, 32, 32]         221,184
BatchNorm2d-400          [-1, 128, 32, 32]             256
       ReLU-401          [-1, 128, 32, 32]               0
     Conv2d-402           [-1, 32, 32, 32]          36,864
BatchNorm2d-403         [-1, 1760, 32, 32]           3,520
       ReLU-404         [-1, 1760, 32, 32]               0
     Conv2d-405          [-1, 128, 32, 32]         225,280
BatchNorm2d-406          [-1, 128, 32, 32]             256
       ReLU-407          [-1, 128, 32, 32]               0
     Conv2d-408           [-1, 32, 32, 32]          36,864
BatchNorm2d-409         [-1, 1792, 32, 32]           3,584
       ReLU-410         [-1, 1792, 32, 32]               0
     Conv2d-411          [-1, 896, 32, 32]       1,605,632
  AvgPool2d-412          [-1, 896, 16, 16]               0
BatchNorm2d-413          [-1, 896, 16, 16]           1,792
       ReLU-414          [-1, 896, 16, 16]               0
     Conv2d-415          [-1, 128, 16, 16]         114,688
BatchNorm2d-416          [-1, 128, 16, 16]             256
       ReLU-417          [-1, 128, 16, 16]               0
     Conv2d-418           [-1, 32, 16, 16]          36,864
BatchNorm2d-419          [-1, 928, 16, 16]           1,856
       ReLU-420          [-1, 928, 16, 16]               0
     Conv2d-421          [-1, 128, 16, 16]         118,784
BatchNorm2d-422          [-1, 128, 16, 16]             256
       ReLU-423          [-1, 128, 16, 16]               0
     Conv2d-424           [-1, 32, 16, 16]          36,864
BatchNorm2d-425          [-1, 960, 16, 16]           1,920
       ReLU-426          [-1, 960, 16, 16]               0
     Conv2d-427          [-1, 128, 16, 16]         122,880
BatchNorm2d-428          [-1, 128, 16, 16]             256
       ReLU-429          [-1, 128, 16, 16]               0
     Conv2d-430           [-1, 32, 16, 16]          36,864
BatchNorm2d-431          [-1, 992, 16, 16]           1,984
       ReLU-432          [-1, 992, 16, 16]               0
     Conv2d-433          [-1, 128, 16, 16]         126,976
BatchNorm2d-434          [-1, 128, 16, 16]             256
       ReLU-435          [-1, 128, 16, 16]               0
     Conv2d-436           [-1, 32, 16, 16]          36,864
BatchNorm2d-437         [-1, 1024, 16, 16]           2,048
       ReLU-438         [-1, 1024, 16, 16]               0
     Conv2d-439          [-1, 128, 16, 16]         131,072
BatchNorm2d-440          [-1, 128, 16, 16]             256
       ReLU-441          [-1, 128, 16, 16]               0
     Conv2d-442           [-1, 32, 16, 16]          36,864
```

```
BatchNorm2d-443         [-1, 1056, 16, 16]           2,112
       ReLU-444         [-1, 1056, 16, 16]               0
     Conv2d-445          [-1, 128, 16, 16]         135,168
BatchNorm2d-446          [-1, 128, 16, 16]             256
       ReLU-447          [-1, 128, 16, 16]               0
     Conv2d-448           [-1, 32, 16, 16]          36,864
BatchNorm2d-449         [-1, 1088, 16, 16]           2,176
       ReLU-450         [-1, 1088, 16, 16]               0
     Conv2d-451          [-1, 128, 16, 16]         139,264
BatchNorm2d-452          [-1, 128, 16, 16]             256
       ReLU-453          [-1, 128, 16, 16]               0
     Conv2d-454           [-1, 32, 16, 16]          36,864
BatchNorm2d-455         [-1, 1120, 16, 16]           2,240
       ReLU-456         [-1, 1120, 16, 16]               0
     Conv2d-457          [-1, 128, 16, 16]         143,360
BatchNorm2d-458          [-1, 128, 16, 16]             256
       ReLU-459          [-1, 128, 16, 16]               0
     Conv2d-460           [-1, 32, 16, 16]          36,864
BatchNorm2d-461         [-1, 1152, 16, 16]           2,304
       ReLU-462         [-1, 1152, 16, 16]               0
     Conv2d-463          [-1, 128, 16, 16]         147,456
BatchNorm2d-464          [-1, 128, 16, 16]             256
       ReLU-465          [-1, 128, 16, 16]               0
     Conv2d-466           [-1, 32, 16, 16]          36,864
BatchNorm2d-467         [-1, 1184, 16, 16]           2,368
       ReLU-468         [-1, 1184, 16, 16]               0
     Conv2d-469          [-1, 128, 16, 16]         151,552
BatchNorm2d-470          [-1, 128, 16, 16]             256
       ReLU-471          [-1, 128, 16, 16]               0
     Conv2d-472           [-1, 32, 16, 16]          36,864
BatchNorm2d-473         [-1, 1216, 16, 16]           2,432
       ReLU-474         [-1, 1216, 16, 16]               0
     Conv2d-475          [-1, 128, 16, 16]         155,648
BatchNorm2d-476          [-1, 128, 16, 16]             256
       ReLU-477          [-1, 128, 16, 16]               0
     Conv2d-478           [-1, 32, 16, 16]          36,864
BatchNorm2d-479         [-1, 1248, 16, 16]           2,496
       ReLU-480         [-1, 1248, 16, 16]               0
     Conv2d-481          [-1, 128, 16, 16]         159,744
BatchNorm2d-482          [-1, 128, 16, 16]             256
       ReLU-483          [-1, 128, 16, 16]               0
     Conv2d-484           [-1, 32, 16, 16]          36,864
BatchNorm2d-485         [-1, 1280, 16, 16]           2,560
       ReLU-486         [-1, 1280, 16, 16]               0
     Conv2d-487          [-1, 128, 16, 16]         163,840
BatchNorm2d-488          [-1, 128, 16, 16]             256
       ReLU-489          [-1, 128, 16, 16]               0
     Conv2d-490           [-1, 32, 16, 16]          36,864
BatchNorm2d-491         [-1, 1312, 16, 16]           2,624
       ReLU-492         [-1, 1312, 16, 16]               0
     Conv2d-493          [-1, 128, 16, 16]         167,936
BatchNorm2d-494          [-1, 128, 16, 16]             256
       ReLU-495          [-1, 128, 16, 16]               0
     Conv2d-496           [-1, 32, 16, 16]          36,864
BatchNorm2d-497         [-1, 1344, 16, 16]           2,688
       ReLU-498         [-1, 1344, 16, 16]               0
     Conv2d-499          [-1, 128, 16, 16]         172,032
```

```
BatchNorm2d-500         [-1, 128, 16, 16]             256
       ReLU-501         [-1, 128, 16, 16]               0
     Conv2d-502          [-1, 32, 16, 16]          36,864
BatchNorm2d-503        [-1, 1376, 16, 16]           2,752
       ReLU-504        [-1, 1376, 16, 16]               0
     Conv2d-505         [-1, 128, 16, 16]         176,128
BatchNorm2d-506         [-1, 128, 16, 16]             256
       ReLU-507         [-1, 128, 16, 16]               0
     Conv2d-508          [-1, 32, 16, 16]          36,864
BatchNorm2d-509        [-1, 1408, 16, 16]           2,816
       ReLU-510        [-1, 1408, 16, 16]               0
     Conv2d-511         [-1, 128, 16, 16]         180,224
BatchNorm2d-512         [-1, 128, 16, 16]             256
       ReLU-513         [-1, 128, 16, 16]               0
     Conv2d-514          [-1, 32, 16, 16]          36,864
BatchNorm2d-515        [-1, 1440, 16, 16]           2,880
       ReLU-516        [-1, 1440, 16, 16]               0
     Conv2d-517         [-1, 128, 16, 16]         184,320
BatchNorm2d-518         [-1, 128, 16, 16]             256
       ReLU-519         [-1, 128, 16, 16]               0
     Conv2d-520          [-1, 32, 16, 16]          36,864
BatchNorm2d-521        [-1, 1472, 16, 16]           2,944
       ReLU-522        [-1, 1472, 16, 16]               0
     Conv2d-523         [-1, 128, 16, 16]         188,416
BatchNorm2d-524         [-1, 128, 16, 16]             256
       ReLU-525         [-1, 128, 16, 16]               0
     Conv2d-526          [-1, 32, 16, 16]          36,864
BatchNorm2d-527        [-1, 1504, 16, 16]           3,008
       ReLU-528        [-1, 1504, 16, 16]               0
     Conv2d-529         [-1, 128, 16, 16]         192,512
BatchNorm2d-530         [-1, 128, 16, 16]             256
       ReLU-531         [-1, 128, 16, 16]               0
     Conv2d-532          [-1, 32, 16, 16]          36,864
BatchNorm2d-533        [-1, 1536, 16, 16]           3,072
       ReLU-534        [-1, 1536, 16, 16]               0
     Conv2d-535         [-1, 128, 16, 16]         196,608
BatchNorm2d-536         [-1, 128, 16, 16]             256
       ReLU-537         [-1, 128, 16, 16]               0
     Conv2d-538          [-1, 32, 16, 16]          36,864
BatchNorm2d-539        [-1, 1568, 16, 16]           3,136
       ReLU-540        [-1, 1568, 16, 16]               0
     Conv2d-541         [-1, 128, 16, 16]         200,704
BatchNorm2d-542         [-1, 128, 16, 16]             256
       ReLU-543         [-1, 128, 16, 16]               0
     Conv2d-544          [-1, 32, 16, 16]          36,864
BatchNorm2d-545        [-1, 1600, 16, 16]           3,200
       ReLU-546        [-1, 1600, 16, 16]               0
     Conv2d-547         [-1, 128, 16, 16]         204,800
BatchNorm2d-548         [-1, 128, 16, 16]             256
       ReLU-549         [-1, 128, 16, 16]               0
     Conv2d-550          [-1, 32, 16, 16]          36,864
BatchNorm2d-551        [-1, 1632, 16, 16]           3,264
       ReLU-552        [-1, 1632, 16, 16]               0
     Conv2d-553         [-1, 128, 16, 16]         208,896
BatchNorm2d-554         [-1, 128, 16, 16]             256
       ReLU-555         [-1, 128, 16, 16]               0
     Conv2d-556          [-1, 32, 16, 16]          36,864
```

```
      BatchNorm2d-557         [-1, 1664, 16, 16]           3,328
             ReLU-558         [-1, 1664, 16, 16]               0
          Conv2d-559          [-1, 128, 16, 16]         212,992
      BatchNorm2d-560          [-1, 128, 16, 16]             256
             ReLU-561          [-1, 128, 16, 16]               0
          Conv2d-562           [-1, 32, 16, 16]          36,864
      BatchNorm2d-563         [-1, 1696, 16, 16]           3,392
             ReLU-564         [-1, 1696, 16, 16]               0
          Conv2d-565          [-1, 128, 16, 16]         217,088
      BatchNorm2d-566          [-1, 128, 16, 16]             256
             ReLU-567          [-1, 128, 16, 16]               0
          Conv2d-568           [-1, 32, 16, 16]          36,864
      BatchNorm2d-569         [-1, 1728, 16, 16]           3,456
             ReLU-570         [-1, 1728, 16, 16]               0
          Conv2d-571          [-1, 128, 16, 16]         221,184
      BatchNorm2d-572          [-1, 128, 16, 16]             256
             ReLU-573          [-1, 128, 16, 16]               0
          Conv2d-574           [-1, 32, 16, 16]          36,864
      BatchNorm2d-575         [-1, 1760, 16, 16]           3,520
             ReLU-576         [-1, 1760, 16, 16]               0
          Conv2d-577          [-1, 128, 16, 16]         225,280
      BatchNorm2d-578          [-1, 128, 16, 16]             256
             ReLU-579          [-1, 128, 16, 16]               0
          Conv2d-580           [-1, 32, 16, 16]          36,864
      BatchNorm2d-581         [-1, 1792, 16, 16]           3,584
             ReLU-582         [-1, 1792, 16, 16]               0
          Conv2d-583          [-1, 128, 16, 16]         229,376
      BatchNorm2d-584          [-1, 128, 16, 16]             256
             ReLU-585          [-1, 128, 16, 16]               0
          Conv2d-586           [-1, 32, 16, 16]          36,864
      BatchNorm2d-587         [-1, 1824, 16, 16]           3,648
             ReLU-588         [-1, 1824, 16, 16]               0
          Conv2d-589          [-1, 128, 16, 16]         233,472
      BatchNorm2d-590          [-1, 128, 16, 16]             256
             ReLU-591          [-1, 128, 16, 16]               0
          Conv2d-592           [-1, 32, 16, 16]          36,864
      BatchNorm2d-593         [-1, 1856, 16, 16]           3,712
             ReLU-594         [-1, 1856, 16, 16]               0
          Conv2d-595          [-1, 128, 16, 16]         237,568
      BatchNorm2d-596          [-1, 128, 16, 16]             256
             ReLU-597          [-1, 128, 16, 16]               0
          Conv2d-598           [-1, 32, 16, 16]          36,864
      BatchNorm2d-599         [-1, 1888, 16, 16]           3,776
             ReLU-600         [-1, 1888, 16, 16]               0
          Conv2d-601          [-1, 128, 16, 16]         241,664
      BatchNorm2d-602          [-1, 128, 16, 16]             256
             ReLU-603          [-1, 128, 16, 16]               0
          Conv2d-604           [-1, 32, 16, 16]          36,864
      BatchNorm2d-605         [-1, 1920, 16, 16]           3,840
AdaptiveMaxPool2d-606            [-1, 1920, 1, 1]               0
AdaptiveAvgPool2d-607            [-1, 1920, 1, 1]               0
AdaptiveConcatPool2d-608            [-1, 3840, 1, 1]               0
          Flatten-609                 [-1, 3840]               0
      BatchNorm1d-610                 [-1, 3840]           7,680
          Dropout-611                 [-1, 3840]               0
           Linear-612                  [-1, 512]       1,966,592
             ReLU-613                  [-1, 512]               0
```

```
     BatchNorm1d-614                  [-1, 512]           1,024
         Dropout-615                  [-1, 512]               0
          Linear-616                    [-1, 5]           2,565
================================================================
Total params: 20,070,789
Trainable params: 2,206,917
Non-trainable params: 17,863,872
----------------------------------------------------------------
Input size (MB): 3.00
Forward/backward pass size (MB): 2294.66
Params size (MB): 76.56
Estimated Total Size (MB): 2374.23
```

“AI winters were not due to imagination traps, but due to lack of imaginations. Imaginations bring order out of chaos. Deep learning with deep imagination is the road map to AI springs and AI autumns.”

— Amit Ray, Compassionate Artificial Superintelligence AI 5.0 - AI with Blockchain, BMI, Drone, IOT, and Biometric Technologies