7/16/2020

CAPSTONE PROJECT

Blindness Detection in Diabetic patients-using Deep Learning

Priyanka Pavithran
Masters in Business Analytics
Carl H. Lindner School of Business

Contents

ABSTRACT

Millions of people suffer from diabetic retinopathy, the leading cause of blindness among working aged adults. It is a diabetes complication that affects the eyes. It is caused by damage to the blood vessels of the light-sensitive tissue at the back of the eye (retina).

Many organizations in India hope to detect and prevent this disease among people living in rural areas where medical screening is difficult to conduct. Diabetic retinopathy affects up to 80 percent of those who have had diabetes for 20 years or more. Diabetic retinopathy often has no early warning signs. Retinal (fundus) photography with manual interpretation is a widely accepted screening tool for diabetic retinopathy, with performance that can exceed that of in-person dilated eye examinations.

Currently, technicians travel to these rural areas to capture images and then rely on highly trained doctors to review the images and provide diagnosis. Their goal is to scale their efforts through technology; to gain the ability to automatically screen images for disease and provide information on how severe the condition may be.

PROBLEM STATEMENT

How do we know that a patient has diabetic retinopathy? Damaged blood vessels or abnormal growths in the retinal scan suggests if any detachment has occurred. These image features need to be identified.

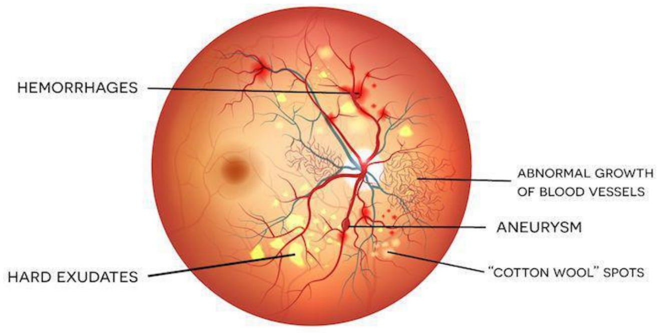


Fig 1. Simple picture to explain Diabetic Retinopathy

The sooner diabetic retinopathy is diagnosed, there is a greater chance of stabilizing the vision and preventing complete vision loss.

APPROACH

We have a large set of retina images under a variety of imaging conditions from **Aravind Eye Hospital, India.** The dataset consists of about 3662 retinal images. Each image has been rated for the severity of diabetic retinopathy on a scale of 0 to 4:

0 - No Diabetic retinopathy

1 - Mild

2 - Moderate

3 - Severe

4 - Proliferative Diabetic retinopathy

Like any real-world data set, we will encounter noise in both the images and labels. Images may contain artifacts, be out of focus, underexposed, or overexposed. The images were gathered from multiple clinics using a variety of cameras over an extended period of time, which will introduce further variation. They need to be pre-processed before training.

A machine learning model needs to be built using this labeled data to classify future images. Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries TensorFlow and allows us to define and train neural network models efficiently.

We first start looking at the distribution in train and test sets with respect to the five image classes.

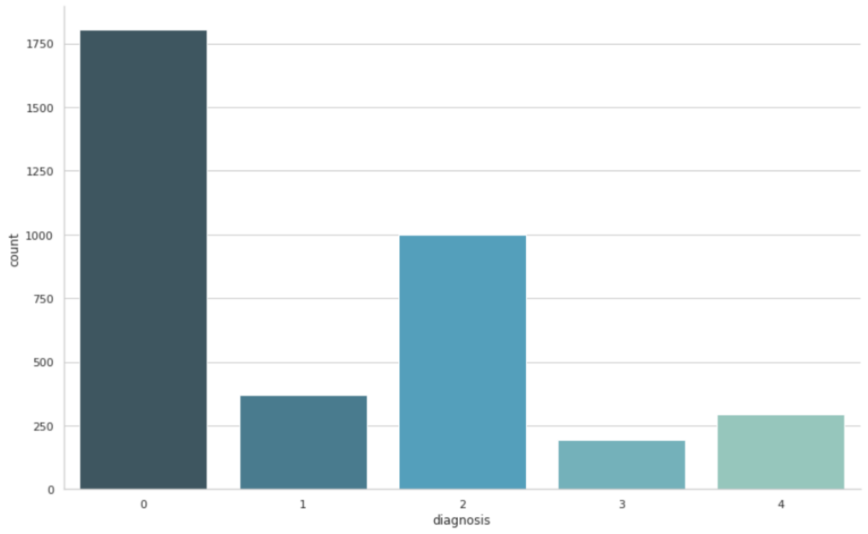


Fig 2. Distribution of classes in train set

The training data has 3662 images while the test data has 1928 images.

Why Deep learning?

The core of deep learning is that we now have fast enough computers and enough data to actually train large neural networks.

The biggest advantage is scalability i.e. as we construct larger neural networks and train them with more and more data, their performance continues to increase. This is generally different to other machine learning techniques that reach a plateau in performance. Hence, large amount of labeled data and substantial computing power are pre-requisites.

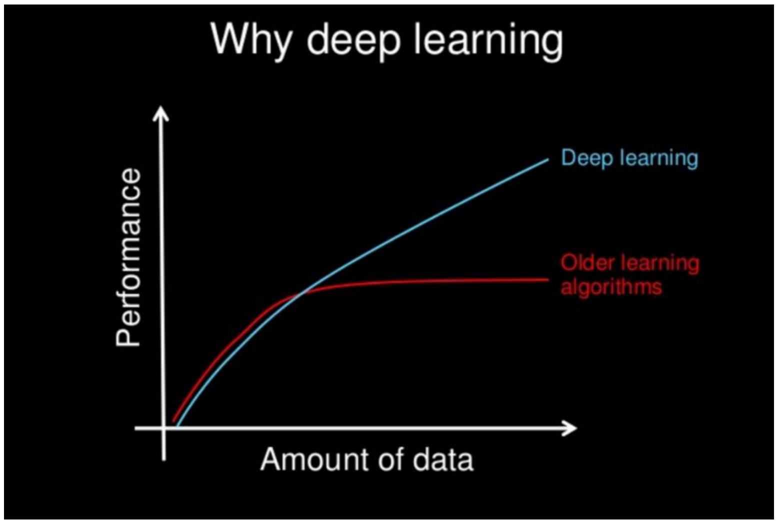


Fig 3. Scalability with deep learning

In addition to scalability, another often cited benefit of deep learning models is their ability to perform automatic feature extraction from raw data, also called feature learning.

Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features.

For example, in an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own.

Deep learning architectures can be constructed with a layer-by-layer method. Deep learning helps to disentangle these abstractions and pick out which features improve performance.

ResNets

A Convolutional Neural Network (CNN, or ConvNet) is multi-layer neural networks, designed to recognize visual patterns/features directly from pixel images with minimal preprocessing. Good ConvNets are beasts with millions of parameters and many hidden layers

According to the universal approximation theorem, given enough capacity, we know that a feedforward network with a single layer is sufficient to represent any function. However, the layer might be massive and the network is prone to overfitting the data. Therefore, there is a common trend in the research that the network architecture needs to go deeper.

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitively small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

AlexNet, VGG, Inception, ResNet are some of the popular variations of CNN to strike a tradeoff between computation and accuracy.

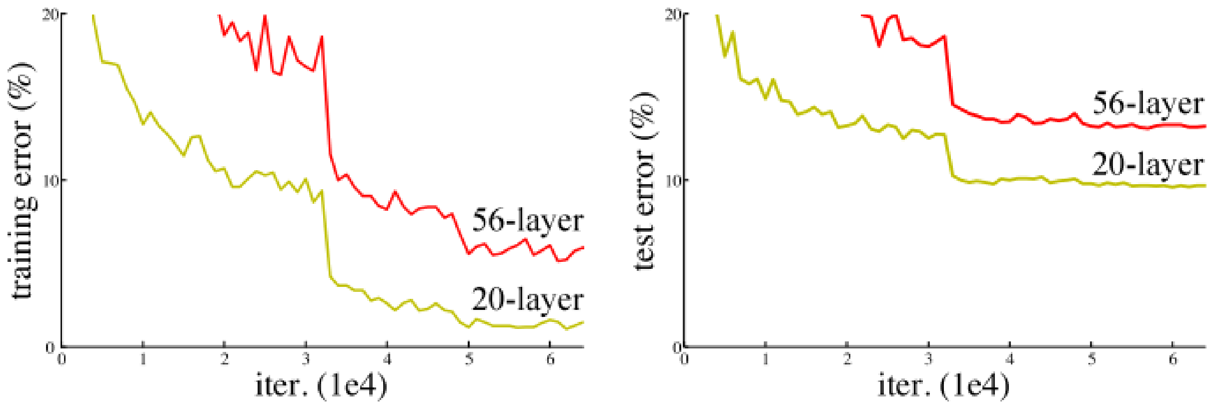


Fig 4. Example: increasing network depth leads to worse performance

So, The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:

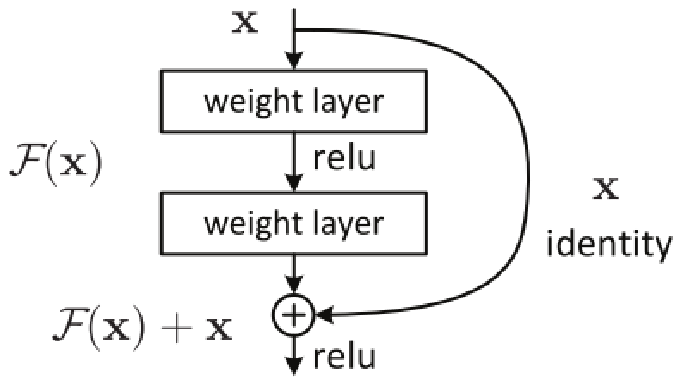

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig 5. A residual block

Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold and necessitates extra training data to recover.

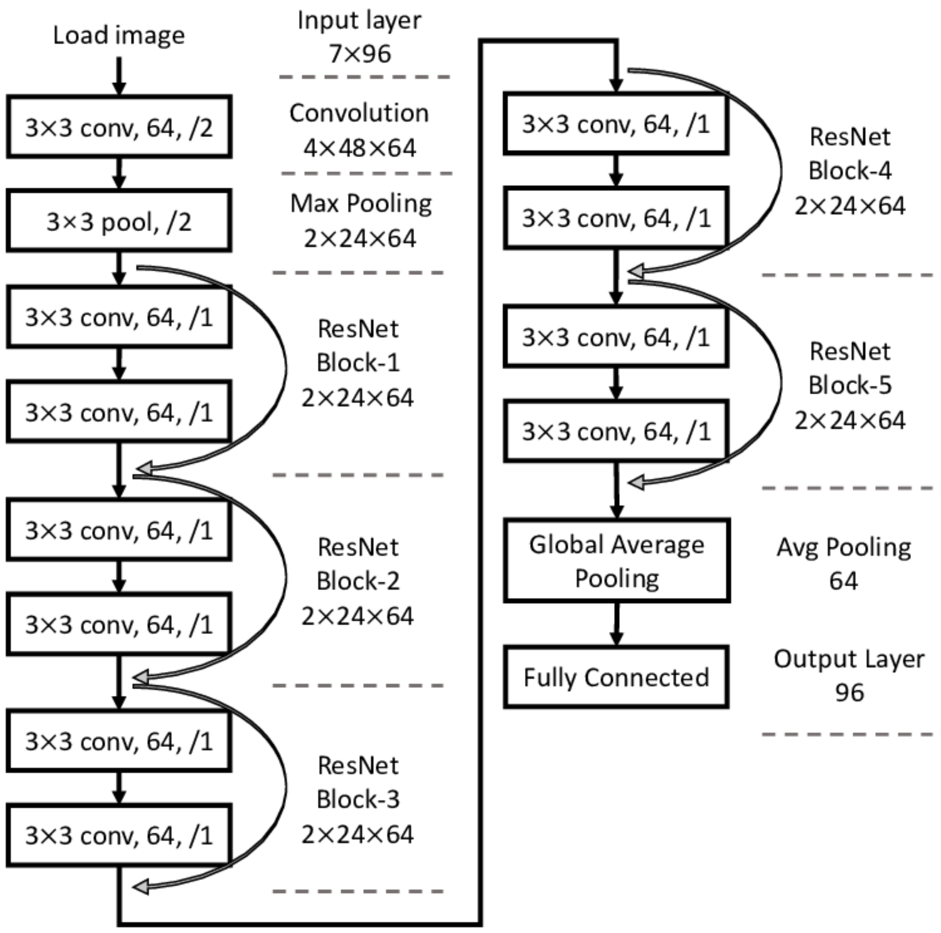


Fig 6. Architecture of Residual Neural Network

Image Input

Preprocessing

We first take at the original input data randomly picking five images from each class. The images below display retinal scans where each row depicts each severity level.

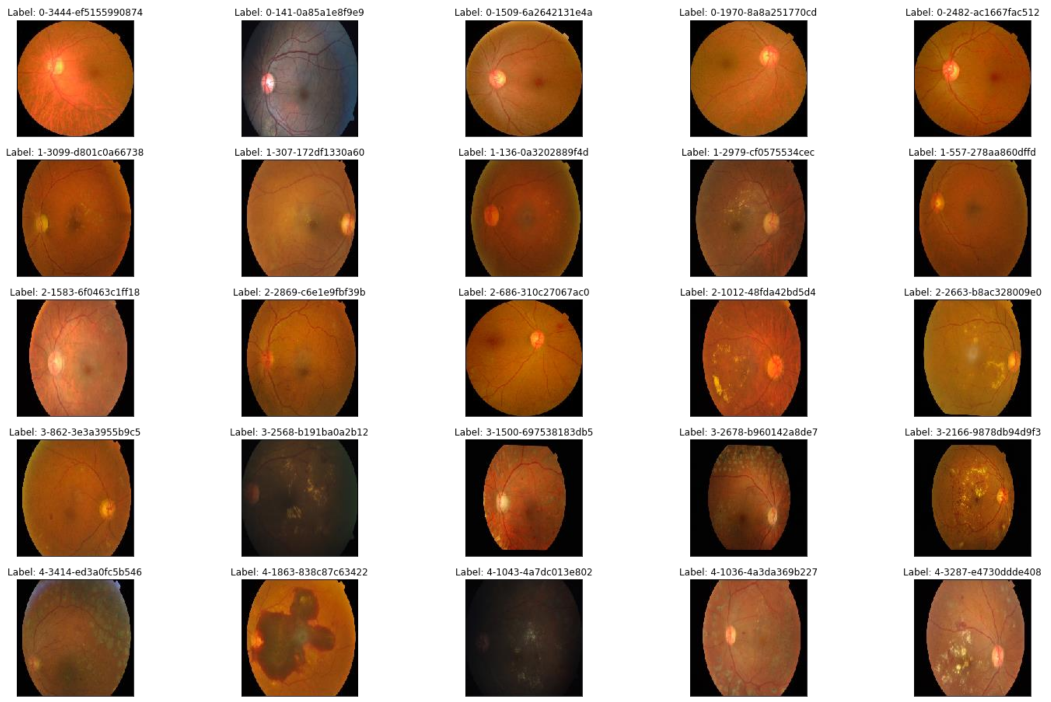

Label: 0-3444-ef5155990874
Label: 0-141-0a85a1e8f9e9
Label: 0-1509-6a2642131e4a
Label: 0-1970-8a8a251770cd
Label: 0-2482-ac1667fac512
Label: 1-3099-d801c0a66738
Label: 1-307-172df1330a60
Label: 1-136-0a3202889f4d
Label: 1-2979-cf0575534cec
Label: 1-557-278aa860dffd
Label: 2-1583-6f0463c1ff18
Label: 2-2869-c6e1e9fbf39b
Label: 2-686-310c27067ac0
Label: 2-1012-48fda42bd5d4
Label: 2-2663-b8ac328009e0
Label: 3-862-3e3a3955b9c5
Label: 3-2568-b191ba0a2b12
Label: 3-1500-697538183db5
Label: 3-2678-b960142a8de7
Label: 3-2166-9878db94d9f3
Label: 4-3414-ed3a0fc5b546
Label: 4-1863-838c87c63422
Label: 4-1043-4a7dc013e802
Label: 4-1036-4a3da369b227
Label: 4-3287-e4730ddde408

Fig 7. Display of retinal scans of each class row-wise

We can see two problems which make the severity category difficult to spot.

First, some images are very dark [example: second pic in the 4th row] and sometimes different color of illumination/background is confusing [example: second pic in the first row].

Second, we can get the uninformative dark areas for some pictures [example: third and fourth pic in the fourth row].

This is important when we reduce the picture size, as informative areas become too small. So, it is intuitive to crop the uninformative areas out in the second case.

<u>Gray-scaling</u>

We can try gray scale and feel understand better for some pictures, as color distraction is gone. Considering the first ten from above:

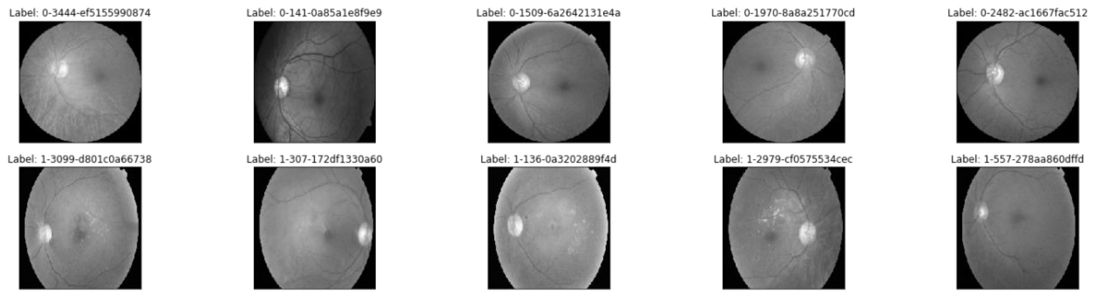

Label: 0-3444-ef5155990874
Label: 0-141-0a85a1e8f9e9
Label: 0-1509-6a2642131e4a
Label: 0-1970-8a8a251770cd
Label: 0-2482-ac1667fac512
Label: 1-3099-d801c0a66738
Label: 1-307-172df1330a60
Label: 1-136-0a3202889f4d
Label: 1-2979-cf0575534cec
Label: 1-557-278aa860dffd

Fig 8. Gray-scaled images

Gaussian Blur

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail.

Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales—see scale space representation and scale space implementation.

Using this technique on the images results in the following:

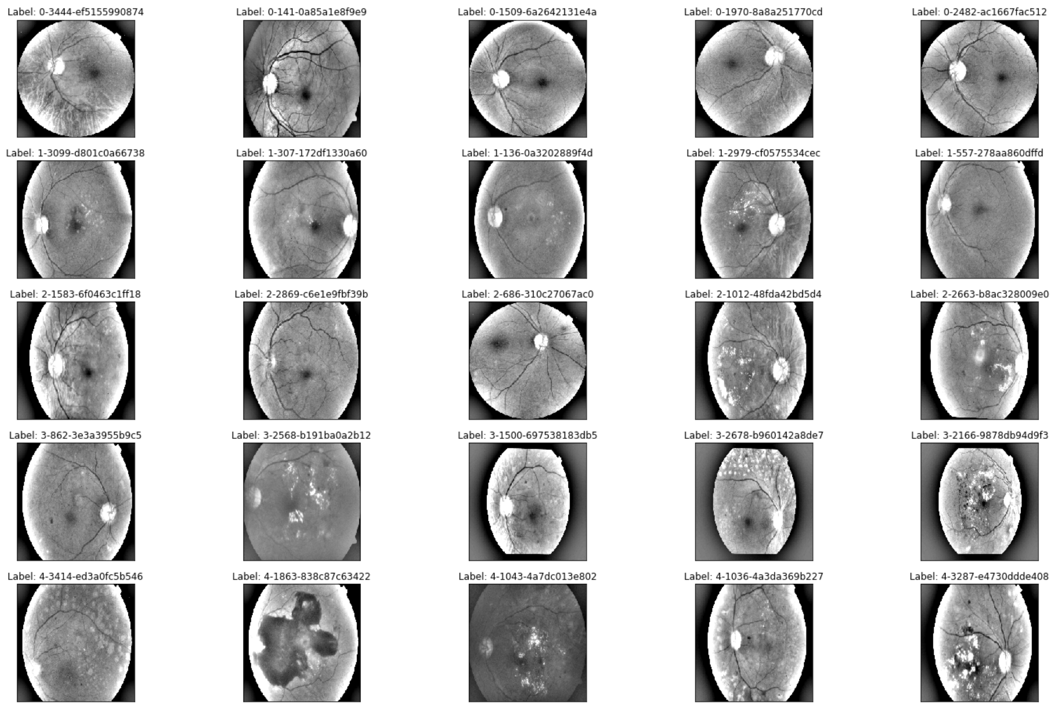

Label: 0-3444-ef5155990874
Label: 0-141-0a85a1e8f9e9
Label: 0-1509-6a2642131e4a
Label: 0-1970-8a8a251770cd
Label: 0-2482-ac1667fac512
Label: 1-3099-d801c0a66738
Label: 1-307-172df1330a60
Label: 1-136-0a3202889f4d
Label: 1-2979-cf0575534cec
Label: 1-557-278aa860dffd
Label: 2-1583-6f0463c1ff18
Label: 2-2869-c6e1e9fbf39b
Label: 2-686-310c27067ac0
Label: 2-1012-48fda42bd5d4
Label: 2-2663-b8ac328009e0
Label: 3-862-3e3a3955b9c5
Label: 3-2568-b191ba0a2b12
Label: 3-1500-697538183db5
Label: 3-2678-b960142a8de7
Label: 3-2166-9878db94d9f3
Label: 4-3414-ed3a0fc5b546
Label: 4-1863-838c87c63422
Label: 4-1043-4a7dc013e802
Label: 4-1036-4a3da369b227
Label: 4-3287-e4730ddde408

Fig 9. Gaussian Blurred Images

Auto-Cropping

To crop out the uninformative black areas, we can try auto cropping. We identify the grayscale values with respect to a given threshold for this purpose.

Input the preprocessed images (crop non-informative regions, Gaussian blur to enhance features) . We use the keras data generator to load the image data efficiently as input and validation set to the model.

The ImageDataGenerator is an easy way to load and augment images in batches for image classification tasks. The API provides features for pipelining of image data from directories as well as from paths mentioned in a dataframe.

The batch_size in data generator determines the number of samples in each mini batch. Its maximum is the number of all samples, which makes gradient descent accurate, the loss will decrease towards the minimum if the learning rate is small enough, but iterations are slower. We set the batch size as 8.

The steps_per_epoch the number of batch iterations before a training epoch is considered finished.

Stages of Model Building

1. **Base Model**: we use the keras resnet50 as the base model and build additional layers on this. This gives the advantage of speeding up the training and obtain higher accuracy in image classification.

2. Add additional dense layers so that the model can tune additional parameters for better classification based on training data. The additional layers are as follows-

   (i) **GlobalAveragePooling2D**:
   Creating ConvNets often goes hand in hand with pooling layers. Suppose that you're training a convolutional neural network. The inputs for this layer are images, of height H, width W and with three channels. Thus, they're likely RGB images. Using a 3x3x3 kernel, a convolution operation is performed over the input image, generating N so-called "feature maps" of size Hfm×Wfm. One feature map learns one particular feature present in the image. Through activating, these feature maps contribute to the outcome prediction during training, and for new data as well.

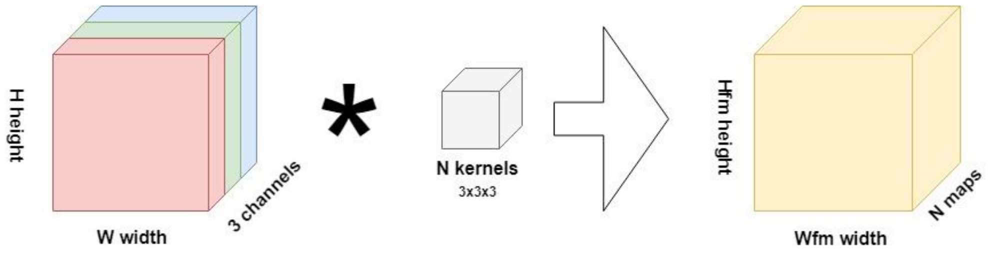


Fig 10. Feature maps in CNN

In the convolution operation of a ConvNet, a small block slides over the entire input image, taking element-wise multiplications with the part of the image it currently slides over. This is a relatively expensive operation which can be accommodated with Pooling. Pooling is basically "downscaling" the image obtained from the previous layers. It can be compared to shrinking an image to reduce its pixel density.

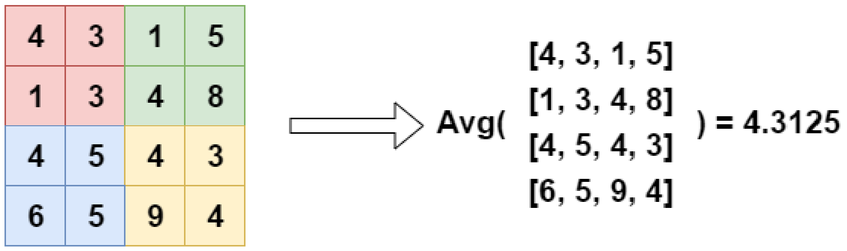


Fig 11. Global Average Pooling

Global Average Pooling is often used to replace the fully-connected or densely-connected layers in a classifier. Instead, the model ends with a convolutional layer that generates as many feature maps as the number of target classes, and applies global average pooling to each in order to convert each feature map into one value. As feature maps can recognize certain elements within the input data, the maps in the final layer effectively learn to “recognize” the presence of a particular class in this architecture. By feeding the values generated by global average pooling into a Softmax activation function, you once again obtain the multiclass probability distribution that you want.

(ii) Dropout Regularization

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

If neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

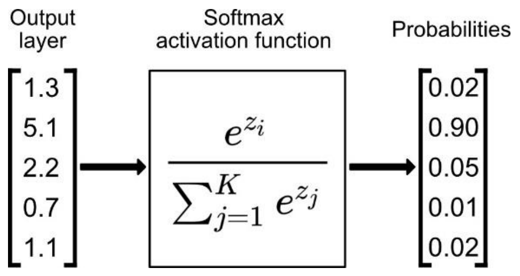
The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

(iii) Dense Layer

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

Convolutional layers act to detect features that help classification, by picking up edges and curves and then from there detecting shapes, but as they can only filter images a dense layer is required to look at the output of the final convolutional neurons/filters and output a number (or numbers if one hot encoding is being used) as the classification.

In the final layer we use the activation function **Softmax**, which results in a multiclass probability distribution over our target classes. The predicted class is, therefore, the item in the list where confidence score is the highest.


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02
Fig 12. The mathematical formula of SoftMax

Model Optimization

Adam Optimization: Adam is an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks.
The algorithms leverage the power of adaptive learning rates methods to find individual learning rates for each parameter.
Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum.

Loss Function: Categorical Cross-entropy/SoftMax Loss
Categorical cross-entropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model’s outputs are to that vector, the lower the loss. The mathematical equation for this is given as:

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

where ŷ is the predicted value.

Fine-Tuning the Model

We then train the model using fit_generator, with a train and validation data generators. We save the networks values at certain conditions on the metrics to monitor, such as loss or accuracy on the training or validation dataset.

Callback APIs used:

EarlyStopping: A problem with training neural networks is in the choice of the number of training epochs to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

The training loop will check at end of every epoch whether the validation loss is no longer decreasing, considering the min_delta and patience if applicable. Once it's found no longer decreasing, model.stop_training is marked True and the training terminates.

ReduceLROnPlateau: Keras also provides ReduceLROnPlateau callback that reduces the learning rate by some factor whenever the learning stagnates. It is believed that sometimes our model will benefit from lowering the learning rate when trapped in the plateau region.

This callback "monitors" a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced by the "factor" specified. Improvement is specified by the "min_delta" argument.

We run the model for a number of iterations/epochs for the best value of chosen metric, in this case the validation loss.

RESULTS

Validation Accuracy: **0.9378**

Model Loss Graph:

The following graphs shows the change of (i) Loss (ii) Accuracy in the training and validation sets with the number of epochs

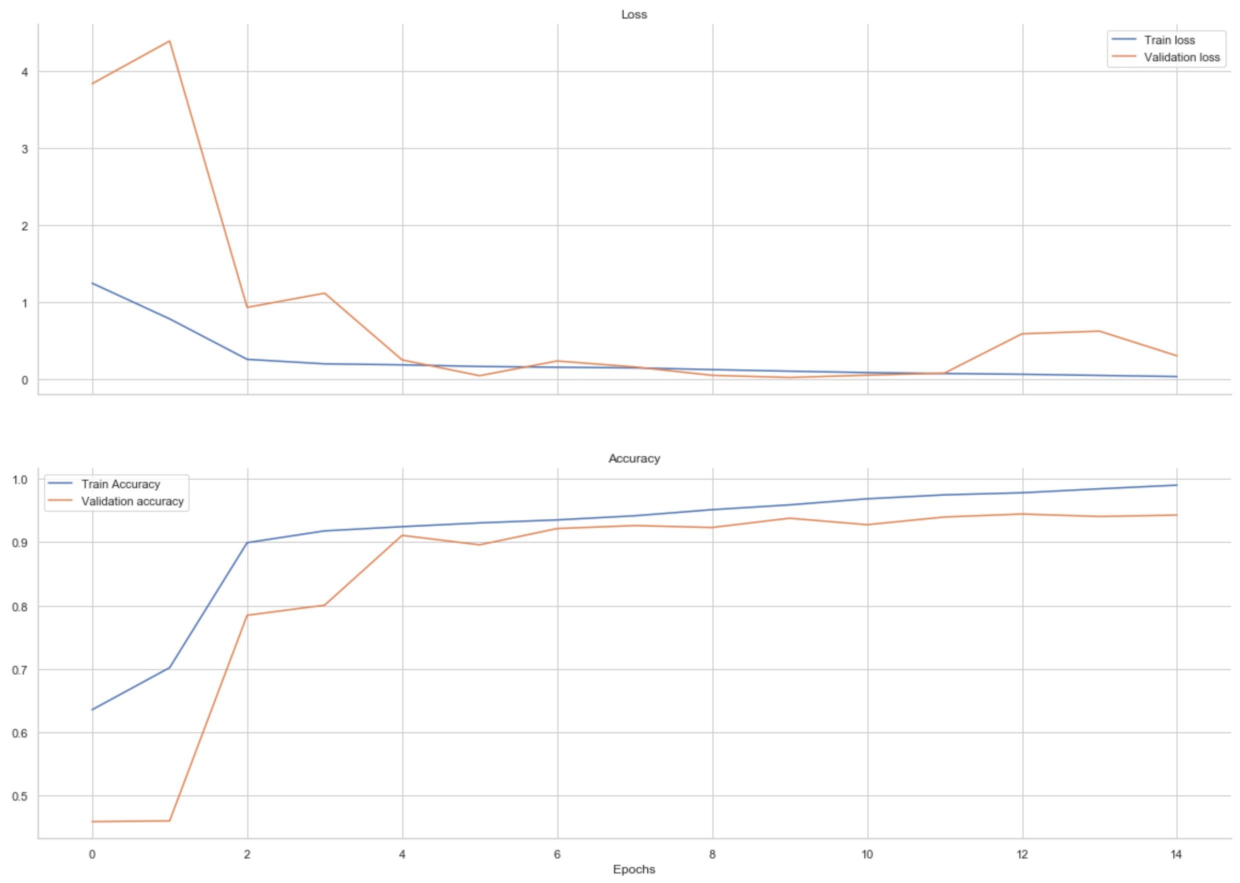


Fig 13. Loss and Accuracy v/s number of epochs

Model Evaluation:

Using the train and validation set we check for the prediction accuracy. The following is the confusion matrix heatmap of true vs predicted classes by the model.

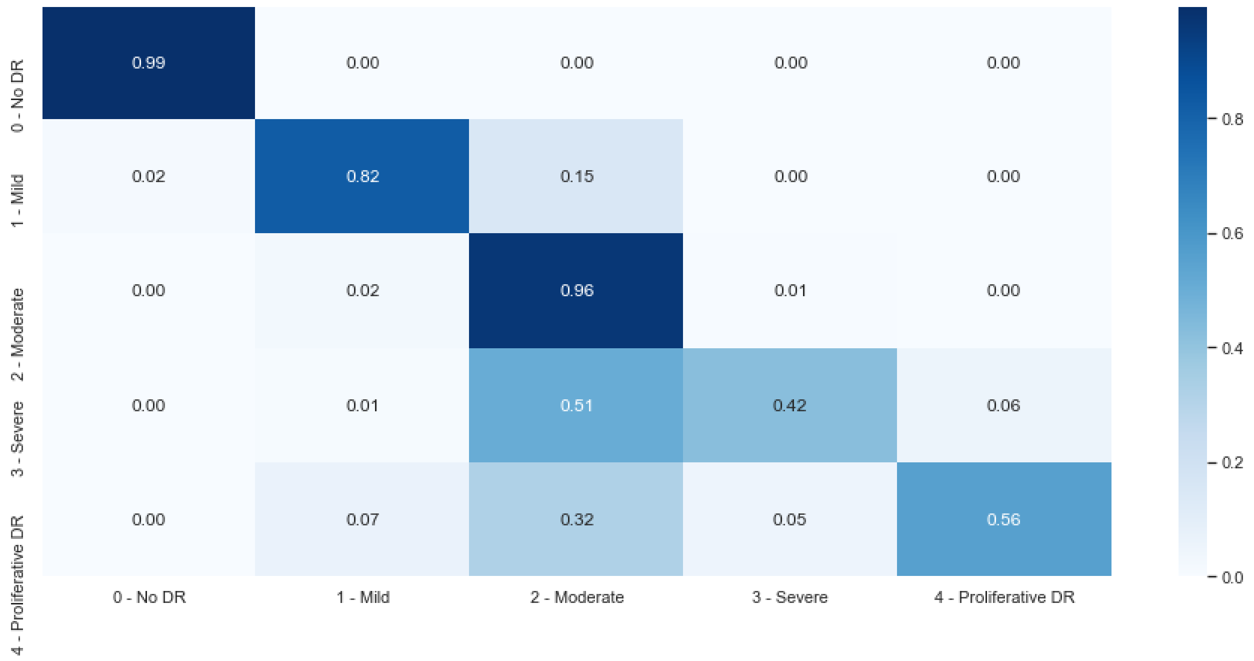


Fig 14. Confusion Matrix Heatmap

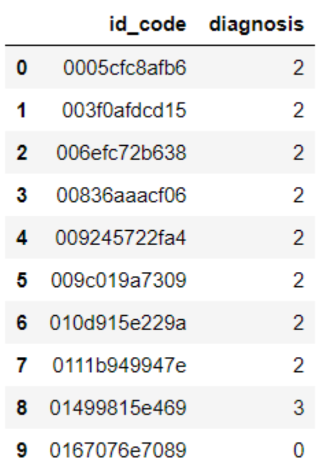
For example, out of all samples that are of '2-moderate' class, 96% has been predicted correctly, 2% has wrongly been classified as mild and 1% as severe.

Cohen's Kappa score

Cohen's Kappa statistic is a very useful metric in multiclass classification. In this case, measures such as the accuracy, or precision/recall do not provide the complete picture of the performance of our classifier. The Kappa statistic (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy (random chance).

The Cohen Kappa's score obtained for our model with the train and validation set is is **0.924**

The predicted classes for the first ten test images are as follows:



	id_code	diagnosis
0	0005cfc8afb6	2
1	003f0afdcd15	2
2	006efc72b638	2
3	00836aaacf06	2
4	009245722fa4	2
5	009c019a7309	2
6	010d915e229a	2
7	0111b949947e	2
8	01499815e469	3
9	0167076e7089	0

Fig 15. Snapshot of results

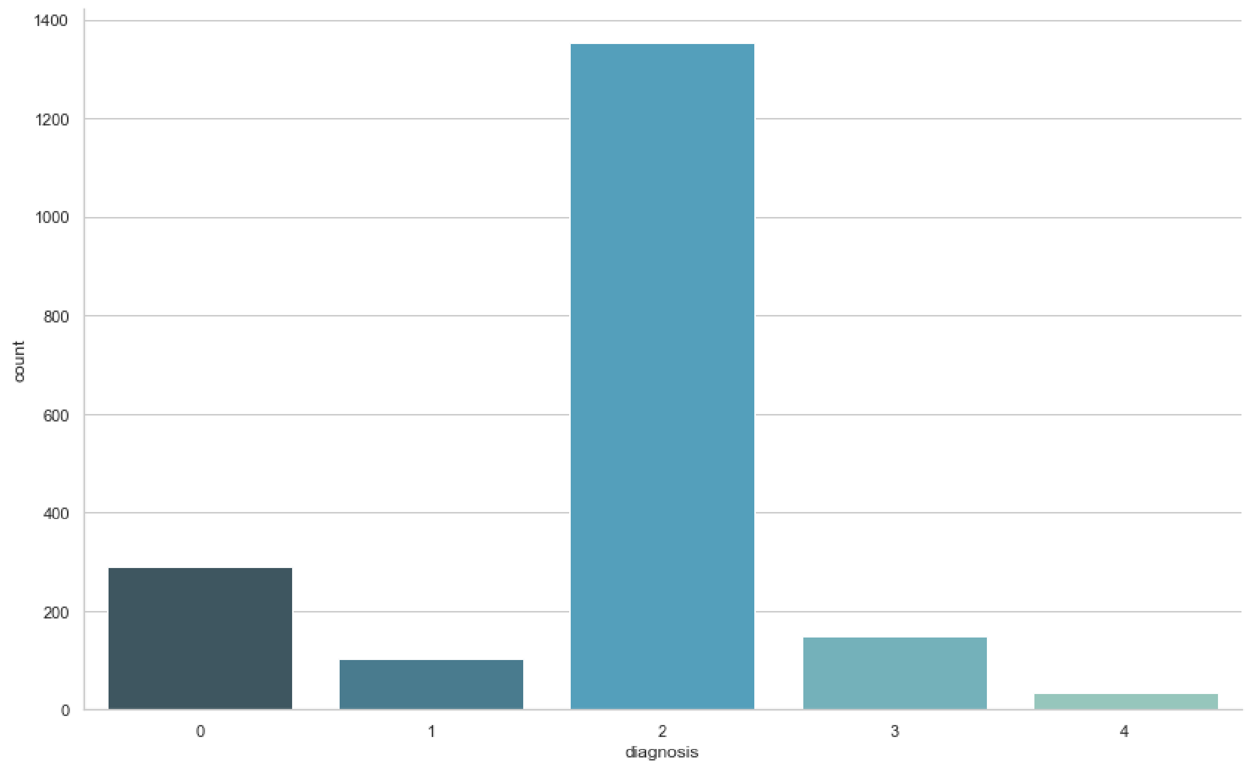


Fig 16. Distribution of predicted classes for test data

References

1. https://keras.io/api/applications/
2. https://www.tensorflow.org/api_docs/python/tf/keras/Model
3. https://keras.io/api/callbacks/
4. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33#:~:text=ResNet%20is%20a%20powerful%20backbone,mitigate%20the%20vanishing%20gradient%20problem
5. https://towardsdatascience.com/keras-data-generators-and-how-to-use-them-b69129ed779c
6. https://towardsdatascience.com/multi-class-metrics-made-simple-the-kappa-score-aka-cohens-kappa-coefficient-bdea137af09c
7. https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60