Saint Joseph's University
Project Diabetic Retinopathy
Sean, Sothey

Diabetic Retinopathy Detection With Deep Learning

Abstract

Diabetic retinopathy (DR) is an eye disease caused by the complication of diabetes and we should detect it early for effective treatment. As diabetes progresses, the vision of a patient may start to deteriorate and lead to diabetic retinopathy. As a result, two groups were identified, namely non-proliferative diabetic retinopathy (NPDR) and proliferative diabetic retinopathy (PDR). In this paper, to detect diabetic retinopathy, deep learning technique (Convolutional Neural Network) was used by experimenting in 4 different models, such as basic CNN, Xception (from Keras), InceptionV3 (from Keras) and TeachableMachine (from Google). CNN is very specialized in image classification tasks; therefore, extracting and preprocessing images techniques are not necessary. The classifier can classify the raw images rapidly based on the associated classes. 500 images from each class are selected, total to 2,500 images. Experimental results show that basic CNN has achieved prediction accuracy of 33%, Xception at 42%, InceptionV3 at 53%, and TeachableMachine at 38%. This infers that the InceptionV3 model outperforms all other models.

Keywords: CNN, Xception, InceptionV3, TeachableMachine, F1-Score, Accuracy

Overview

Diabetic Retinopathy (DR) is the leading eye disease to cause of blindness in the working-age population and is associated with long-standing diabetes.

The vision impairment can be slowed or averted if DR is detected in time. However, this disease often shows very few symptoms until it is too late to provide effective treatment. Currently, detecting DR is a time-consuming manual process. It requires a trained clinician to examine and evaluate digital color fundus photographs of the retina. It usually takes 1-2 days for the results which leads to delayed treatment.

Clinicians can identify DR by the presence of lesions associated with the vascular abnormalities caused by the disease. While this approach is effective, there're not enough clinicians and equipment to meet the high demand due to the growth of diabetes patients. As the number of individuals with diabetes continues to grow, the infrastructure needed to prevent blindness due to DR will become even more insufficient.

The need for a comprehensive and automated method of DR screening has been recognized for a long time, and previous efforts have made good progress using image classification, pattern recognition, and machine learning. With color fundus photography as input, the goal is to build an automated detection system to the limit of what is possible, ideally resulting in models with realistic clinical potential that can improve DR detection. This process requires many machine learning tasks and techniques to build a good classifier. To start, it is essential to obtain enough datasets, including useful information, such as number of attributes and classes.

Datasets Description

The dataset is in image format with a large set of high-resolution retina images taken under a variety of imaging conditions. A left and right field is provided for every subject. Images are labeled with a subject ID as well as either left or right (e.g. 1_left.jpeg is left eye of patient id 1).

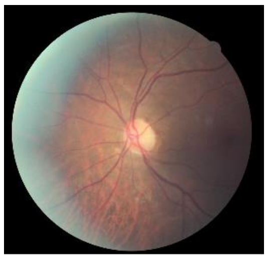
Each image has been rated by clinicians on a scale 0 to 4 according to the presence of diabetic retinopathy.

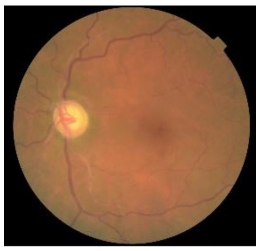
0 - No DR
1 - Mild
2 - Moderate
3 - Severe
4 - Proliferative DR

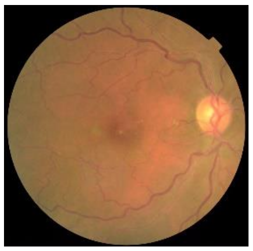
The acquired images are in RGB form with high resolution JPEG format. Due to the extremely large size of this dataset (82.23 GB), it was separated into archive folders. Each folder has approximately 8,400 images (8 GB).

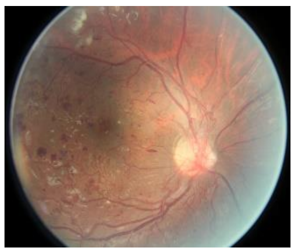
- train.zip - the training set (5 folders)
- test.zip - the test set (7 folders)
- trainLabels.csv - contains the scores (0 to 4) for the training set

Images Sample


10_left, score: 0

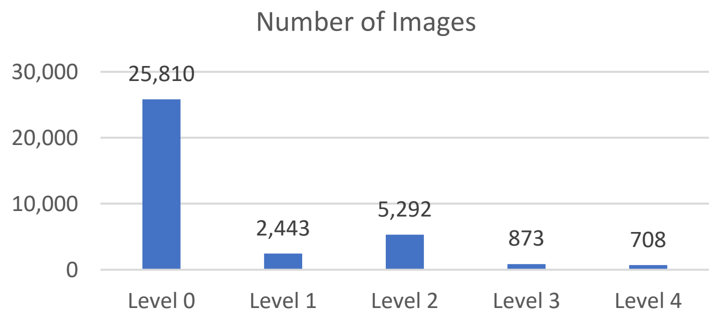

35_left, score: 1


25_right, score: 2


16_left, score: 4

Select and Balance Dataset

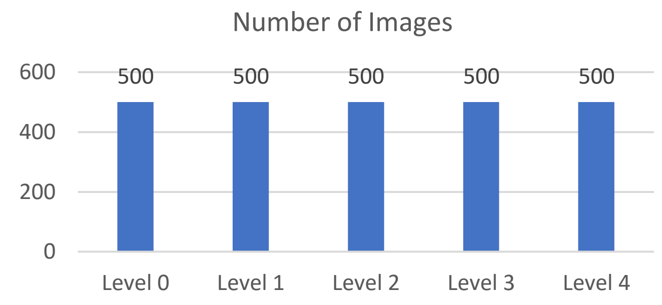
The entire dataset was not balanced as almost 74% of the images are represented for the class 0 which is "NO DR", followed by "Moderate" at 15%, "Mild" at 7%, and "Severe" and "Proliferative DR" together at 4%.



With this imbalanced dataset, the basic CNN model was created and tested in order to have some intuitive of where to go next. As a result, it has a roughly accuracy rate of 71% on testing set. However, by looking at the classification report and confusion matrix, the model has predicted only on the class 0 (“NO DR”) and missing out all other classes. Therefore, a new balanced dataset was manually created.

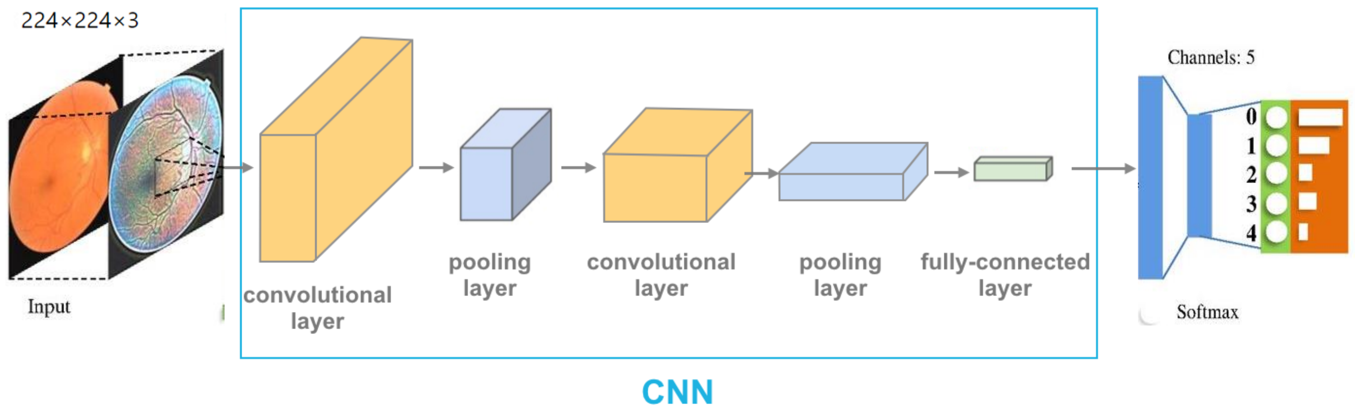
```
                  precision    recall  f1-score   support

           No DR       0.71      1.00      0.83       303
            Mild       0.00      0.00      0.00        39
        Moderate       0.00      0.00      0.00        61
          Severe       0.00      0.00      0.00        13
Proliferative DR       0.00      0.00      0.00        13

        accuracy                           0.71       429
       macro avg       0.14      0.20      0.17       429
    weighted avg       0.50      0.71      0.58       429
```

First, extract the train label on CSV file for 500 on each level (0, 1, 2, 3, 4) totaling to 2,500 images. Then, use the TEXTJOIN function in excel program to combine all 2,500 rows into one cell. Now that I have all the image filenames in one line, I can copy and paste the image filenames in search tab in dataset folder. Unfortunately, the search tab has limitation of character to search, so I was able to search for only 10 to 15 images at a time, which takes approximately 4-5 hours to complete. Finally, I have a well-balanced dataset.



Convolutional Neural Networks Architecture

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a ReLU layer and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.



When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (input channels). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by a width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameter).
- The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.

Pooling

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically, the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer. The subarea of the original input image in the receptive field is increasingly growing as getting deeper in the network architecture. This is due to applying over and over again a convolution which takes into account the value of a specific pixel, but also some surrounding pixels.

Weights

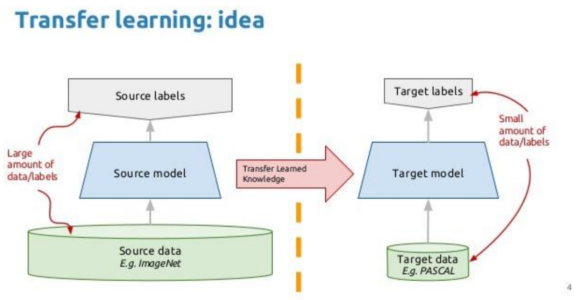
Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

Transfer Learning

The basic premise of transfer learning is simply by taking a model trained on a large dataset and transfer its knowledge to a smaller dataset. For object recognition CNN model, we freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images — such as edges, patterns, gradients — and the later layers identify specific features within an image such as eyes or wheels.

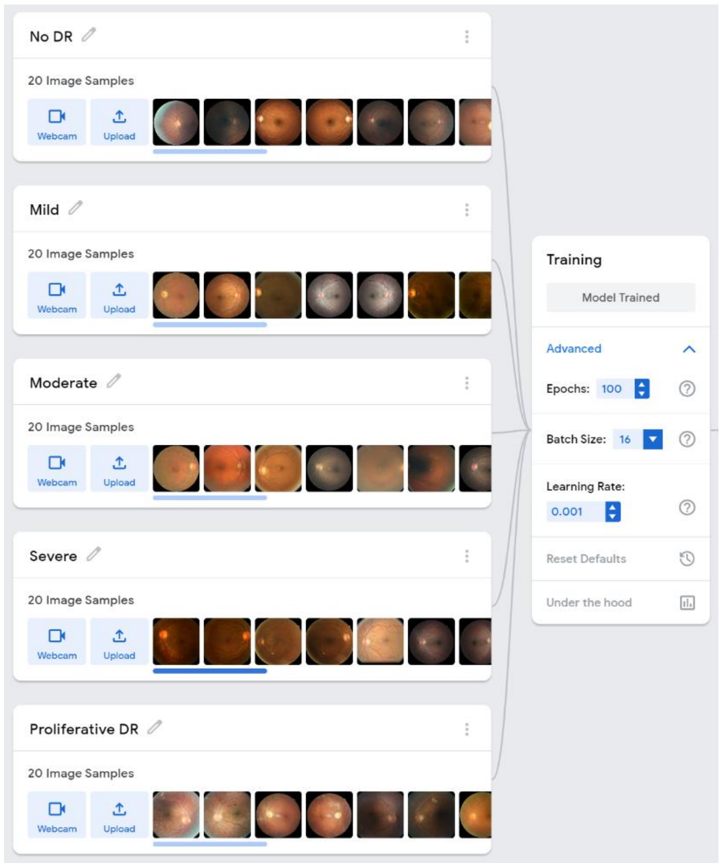
Thus, we can use a network trained on unrelated categories in a massive dataset (usually ImageNet) and apply it to our own problem because there are universal, low-level features shared between images.



Within this research project, Keras Application deep learning models that are made available alongside pre-trained weights will be used as the pretrained models.

Teachable Machine Reverse Engineering

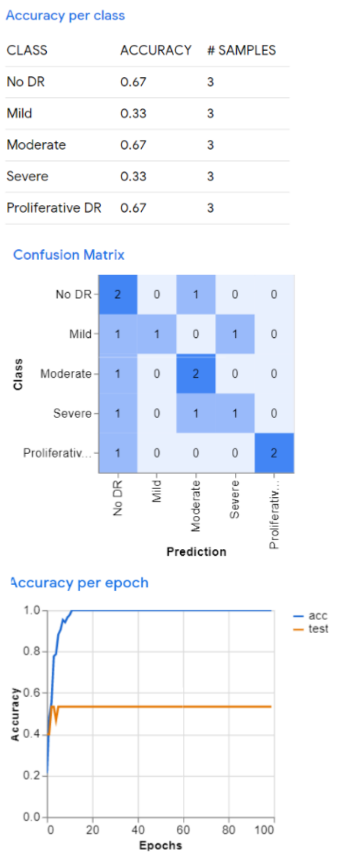
Teachable Machine is a web tool, developed by Google, that makes it fast and easy to create machine learning models for your projects, no coding required. It trains a computer to recognize your images, sounds, and poses, then export your model for your sites, apps, and more. In order to obtain the image classification model from this platform, I first built the model by passing 100 images (20 for each class) then trained the model using epoch = 100, batch size = 16, and learning rate = 0.001. The result is signified in the pictures below. After that, I exported the model into Keras Model (h5 file) and used Netron application to read its architectures. Then, I can mimic the Teachable Machine model by either building in manually (layer by layer) or loading instantly and retraining with the prepared balanced dataset.



Epochs: 100

Batch Size: 16

Learning Rate: 0.001



Accuracy per class

CLASS	ACCURACY	# SAMPLES
No DR	0.67	3
Mild	0.33	3
Moderate	0.67	3
Severe	0.33	3
Proliferative DR	0.67	3

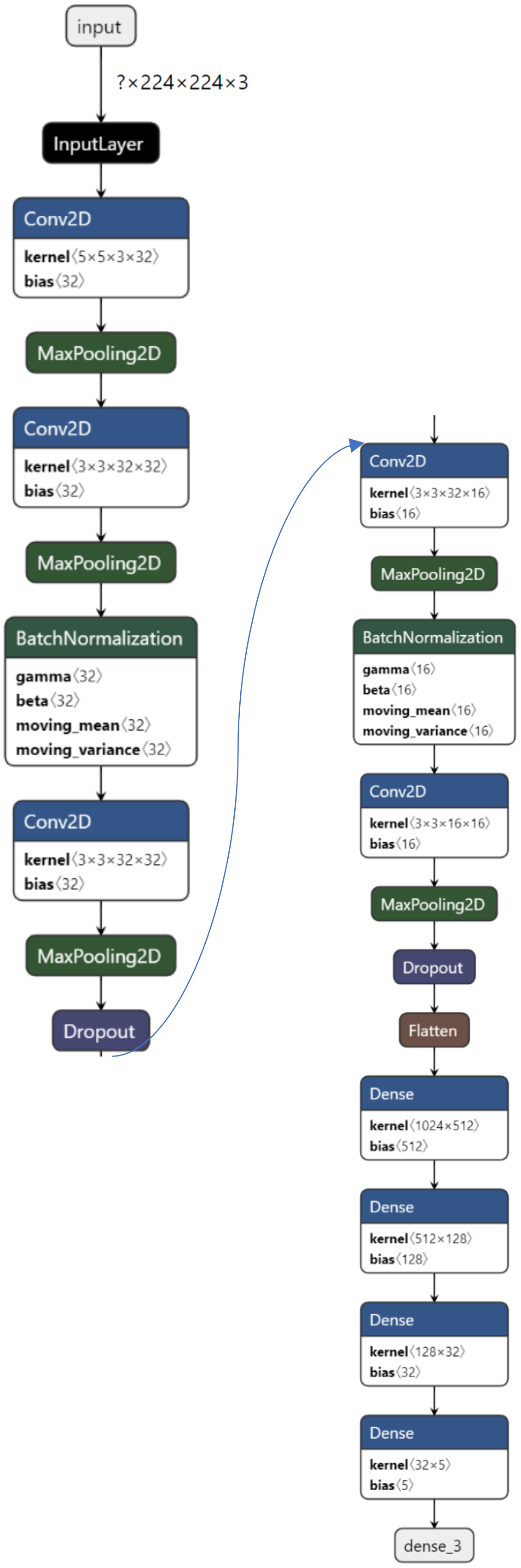
Confusion Matrix

Class \ Prediction	No DR	Mild	Moderate	Severe	Proliferativ...
No DR	2	0	1	0	0
Mild	1	1	0	1	0
Moderate	1	0	2	0	0
Severe	1	0	1	1	0
Proliferativ...	1	0	0	0	2

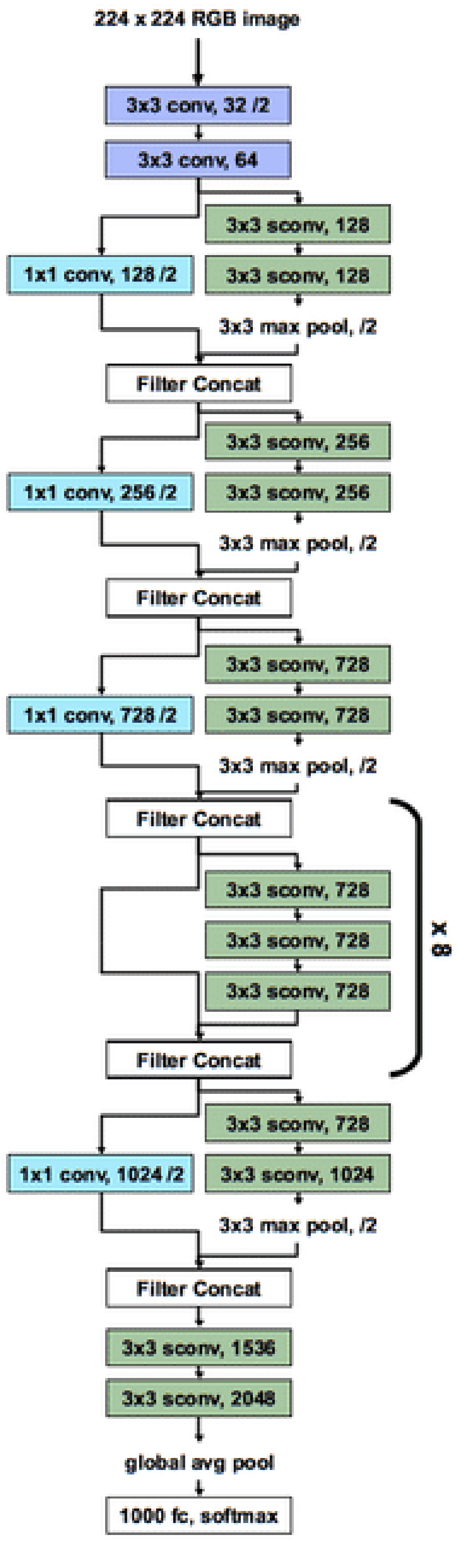
Models Information and Architecture

Model	Size	Parameter	Depth
Basic CNN 5 Layers	2.5 MB	15,245,541	5
Xception	84.2 MB	21,980,525	126
InceptionV3	88.2 MB	22,921,829	159
TeachableMachine	2.35 MB	538,808	2

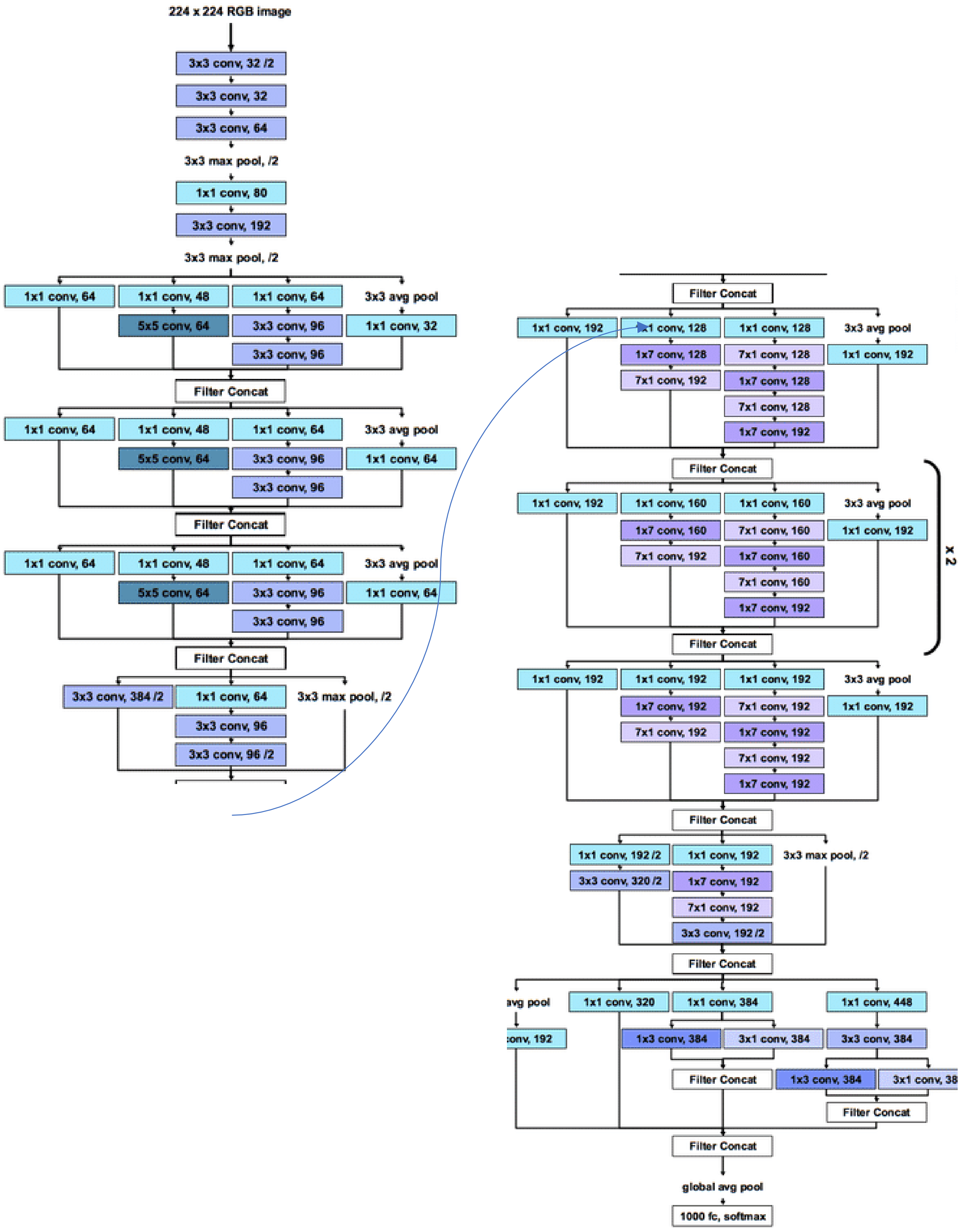
1- Basic CNN with 5 Layers



2- Xception


224 x 224 RGB image
3x3 conv, 32 /2
3x3 conv, 64
3x3 sconv, 128
1x1 conv, 128 /2
3x3 sconv, 128
3x3 max pool, /2
Filter Concat
3x3 sconv, 256
1x1 conv, 256 /2
3x3 sconv, 256
3x3 max pool, /2
Filter Concat
3x3 sconv, 728
1x1 conv, 728 /2
3x3 sconv, 728
3x3 max pool, /2
Filter Concat
3x3 sconv, 728
3x3 sconv, 728
3x3 sconv, 728
x 8
Filter Concat
3x3 sconv, 728
1x1 conv, 1024 /2
3x3 sconv, 1024
3x3 max pool, /2
Filter Concat
3x3 sconv, 1536
3x3 sconv, 2048
global avg pool
1000 fc, softmax

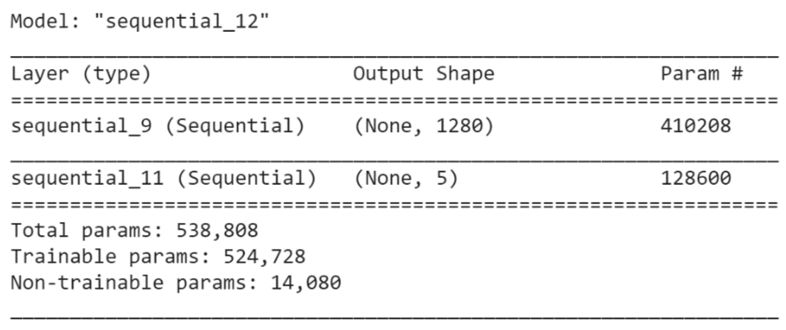
3- InceptionV3



4- TeachableMachine

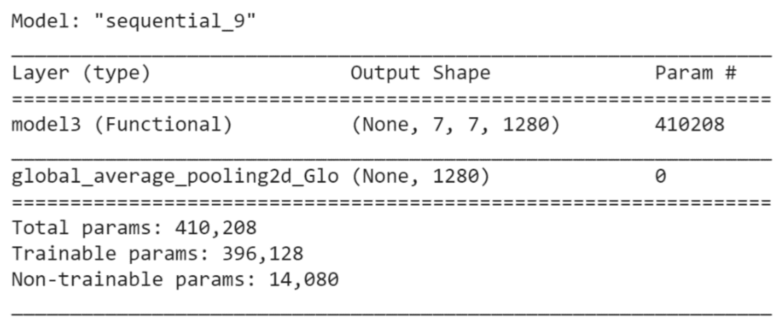
For this model, additional dense layer was not added as I wanted to use the model as is. The model contains only two initialed sequential layers; therefore, I had to extract each sequential layer to see more details, as well as the Functional layer, using Netron application.

Original Layers



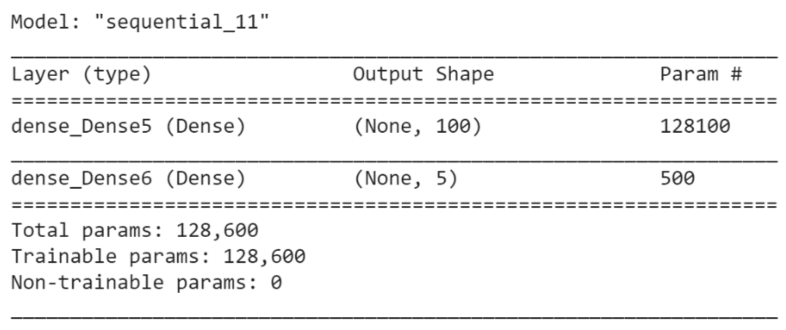
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_9 (Sequential)    (None, 1280)              410208
_________________________________________________________________
sequential_11 (Sequential)   (None, 5)                 128600
=================================================================
Total params: 538,808
Trainable params: 524,728
Non-trainable params: 14,080
_________________________________________________________________
```

Frist Sequential Layer

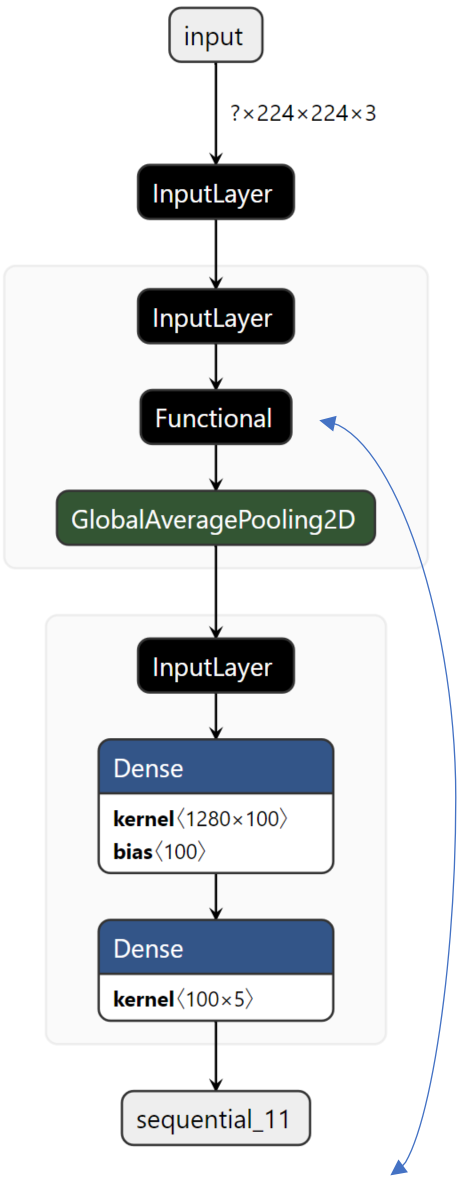


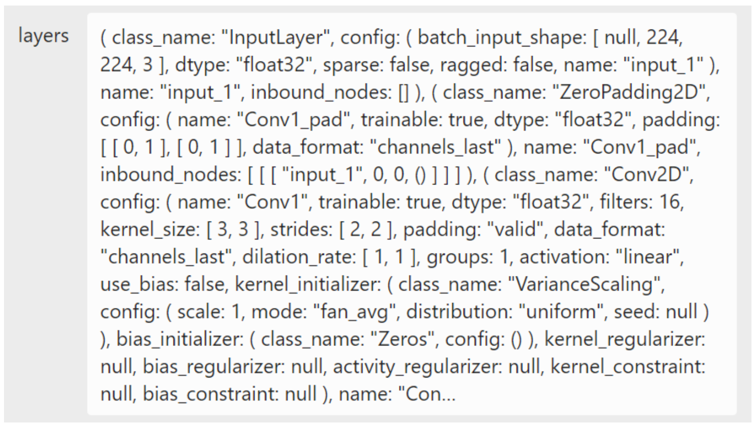
```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model3 (Functional)          (None, 7, 7, 1280)        410208
_________________________________________________________________
global_average_pooling2d_Glo (None, 1280)              0
=================================================================
Total params: 410,208
Trainable params: 396,128
Non-trainable params: 14,080
_________________________________________________________________
```

Second Sequential Layer



```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_Dense5 (Dense)         (None, 100)               128100
_________________________________________________________________
dense_Dense6 (Dense)         (None, 5)                 500
=================================================================
Total params: 128,600
Trainable params: 128,600
Non-trainable params: 0
_________________________________________________________________
```





layers (class_name: "InputLayer", config: (batch_input_shape: [null, 224, 224, 3], dtype: "float32", sparse: false, ragged: false, name: "input_1"), name: "input_1", inbound_nodes: []), (class_name: "ZeroPadding2D", config: (name: "Conv1_pad", trainable: true, dtype: "float32", padding: [[0, 1], [0, 1]], data_format: "channels_last"), name: "Conv1_pad", inbound_nodes: [[["input_1", 0, 0, ()]]]), (class_name: "Conv2D", config: (name: "Conv1", trainable: true, dtype: "float32", filters: 16, kernel_size: [3, 3], strides: [2, 2], padding: "valid", data_format: "channels_last", dilation_rate: [1, 1], groups: 1, activation: "linear", use_bias: false, kernel_initializer: (class_name: "VarianceScaling", config: (scale: 1, mode: "fan_avg", distribution: "uniform", seed: null)), bias_initializer: (class_name: "Zeros", config: ()), kernel_regularizer: null, bias_regularizer: null, activity_regularizer: null, kernel_constraint: null, bias_constraint: null), name: "Con...

Reduce Overfitting

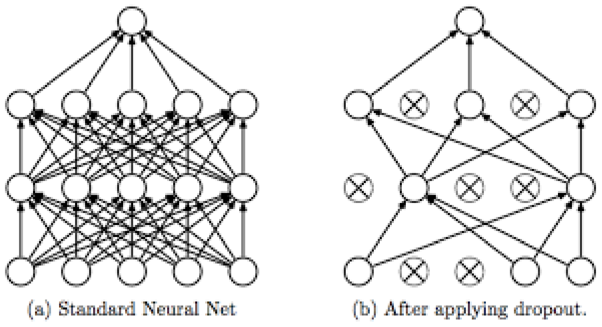
Overfitting is referred to a model that learns the training dataset too well, performing well on the training dataset but does not perform well on a holdout sample, same as the case above. Below is a list of five of the most common methods.

- Regularization: penalize the model during training base on magnitude of the activations.
- Weight: constrain the magnitude of weights to be within a range or below a limit.
- Dropout: probabilistically remove inputs during training.
- Noise: add statistical noise to inputs during training.
- Early Stopping: monitor model performance on a validation set and stop training when performance degrades.

Most of these methods have been demonstrated to approximate the effect of adding a penalty to the loss function. Each method approaches the problem differently, offering benefits in terms of a mixture of generalization performance, configurability, and/or computational complexity.

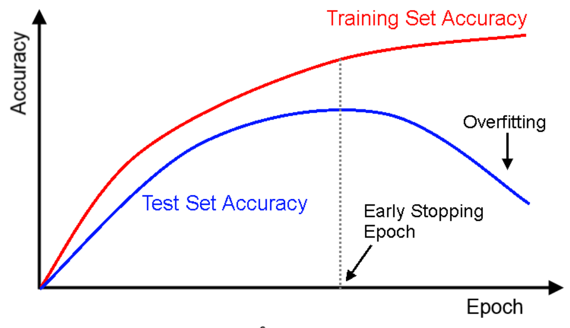
In this paper, Dropout and Early Stopping methods, the two most effective way, are applied to reduce overfitting.

Dropout



$$E_R = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_i w_i I_i\right)^2 + \sum_{i=1}^{n} p_i(1-p_i) w_i^2 I_i^2$$

Early Stopping



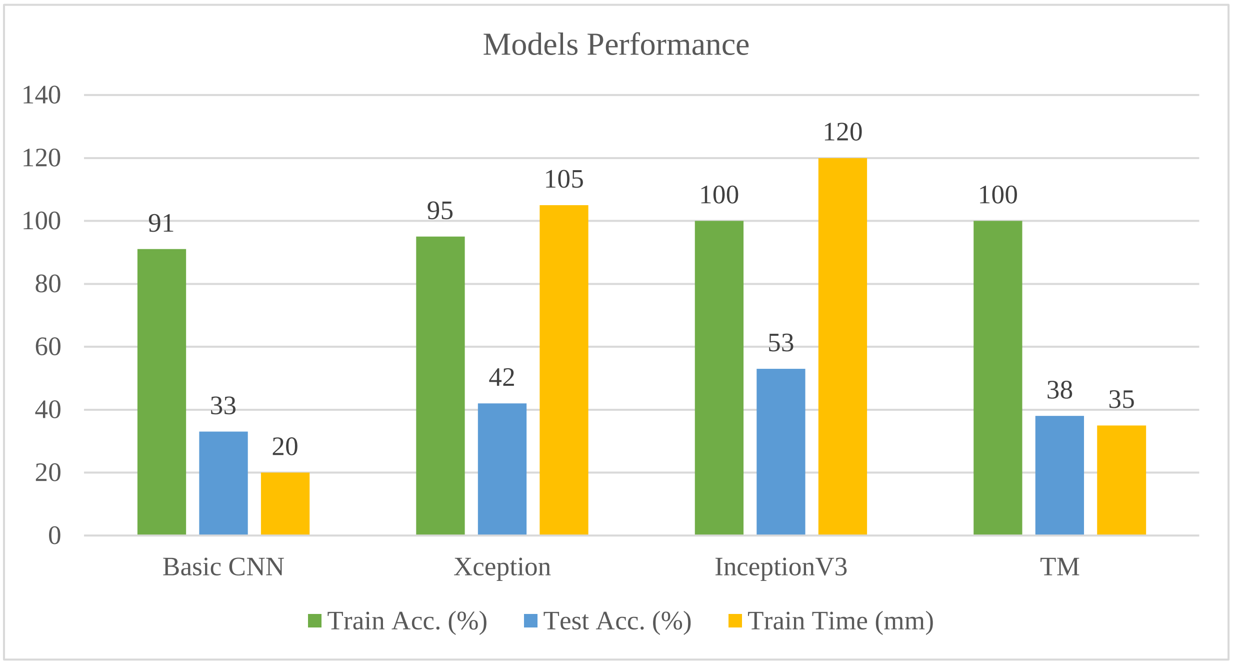
$$f_\rho(x) = \int_Y y d\rho(y|x), x \in X,$$

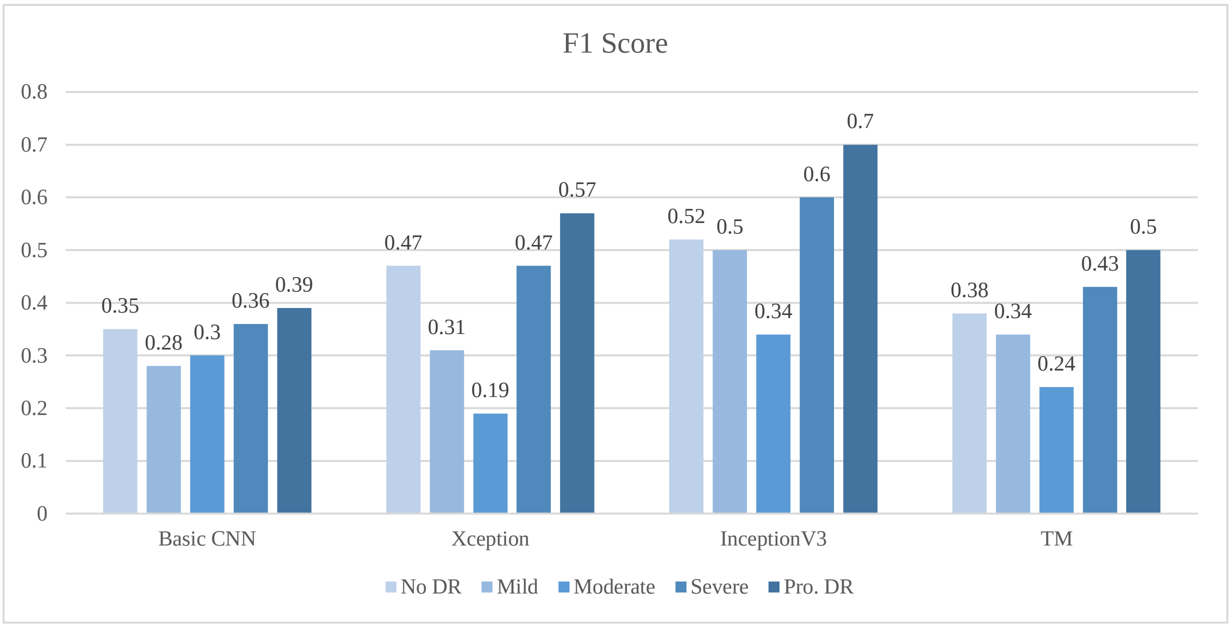
Result

After testing the models multiple times with the 2,500 images, the solid results of F1-score and accuracies were able to be collected within the table below. These results were finalized after reducing the overfitting.

F1-Score					
Model	No DR	Mild	Moderate	Severe	Pro. DR
Basic CNN	0.35	0.28	0.3	0.36	0.39
Xception	0.47	0.31	0.19	0.47	0.57
InceptionV3	0.52	0.5	0.34	0.6	0.7
TM	0.38	0.34	0.24	0.43	0.5

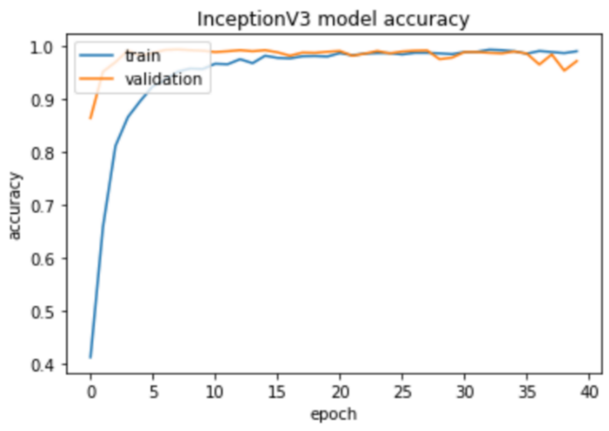
Model	Train Accuracy %	Test Accuracy %	Training Time (mn)
Basic CNN	91	33	20
Xception	95	42	105
InceptionV3	100	53	120
TeachableMachine	100	38	35



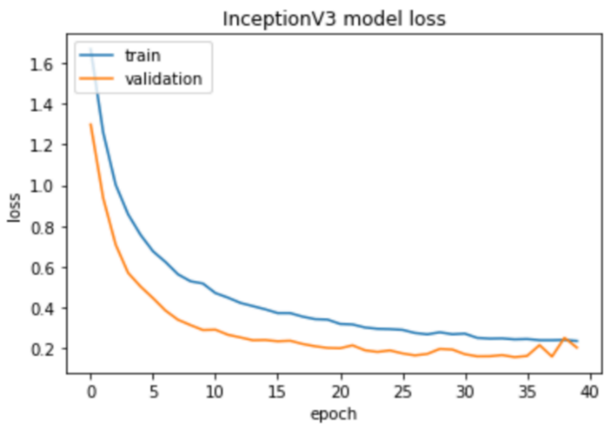


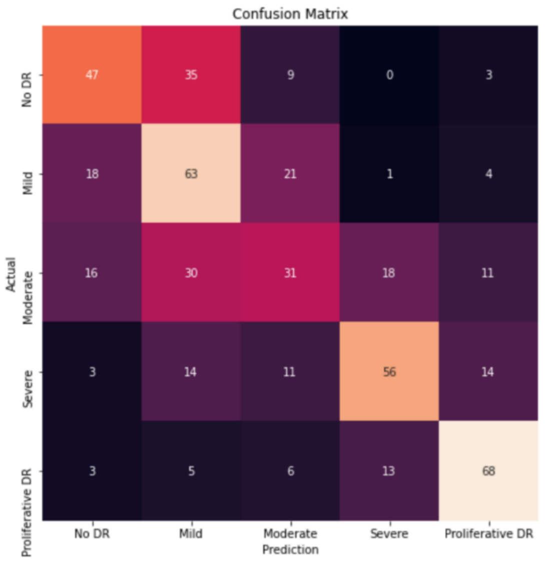
Conclusion

In order to get a good accuracy in prediction, balancing dataset was the top priority task. As a result, the classifier has improved significantly. It could predict images in all classes. However, to achieve image classification for this project, it required high Ram and GPU machine for training these models, despite only about 7% (2,500 images, 80% for training, 20% for testing) of the whole dataset were used, and augmented images were used as the validation data only. To maximize the models' performance and minimize the Ram and GPU usage, I had to keep adjusting the number of epochs and see how the learning curves behaved to finalize a pattern whether they are under or overfitting. The most common problems that I encountered while training deep neural networks is overfitting. Overfitting occurs when a model tries to predict a trend in data that is too noisy. This is causing the model to be overly complex with too many parameters. Therefore, overfitting method (Dropout and Early Stopping) was used to elevate the test accuracy, and it has also helped the model to work well when new data is added. Finally, among the 4 CNN models, InceptionV3 has the highest test accuracy up to 53% and highest F1 score for each class, as well as the learning curve and confusion matrix. In my opinion this accuracy rate is very acceptable if we compare to Keras and Google models. For image classification task, they generally use the most powerful computation machines with huge data (millions of images) in order to achieve a high accuracy model up to 90% or more.



```
                  precision    recall  f1-score   support

           No DR       0.54      0.50      0.52        94
            Mild       0.43      0.59      0.50       107
        Moderate       0.40      0.29      0.34       106
          Severe       0.64      0.57      0.60        98
Proliferative DR       0.68      0.72      0.70        95

        accuracy                           0.53       500
       macro avg       0.54      0.53      0.53       500
    weighted avg       0.53      0.53      0.53       500
```





****All Python notebooks for this project are available here.*

Source

https://www.kaggle.com/c/diabetic-retinopathy-detection/data

https://keras.io/api/applications/

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://teachablemachine.withgoogle.com/

https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html