Machine Learning Engineer Nanodegree

Capstone Project

Samuel A. Rodriguez L.

August 12th, 2019

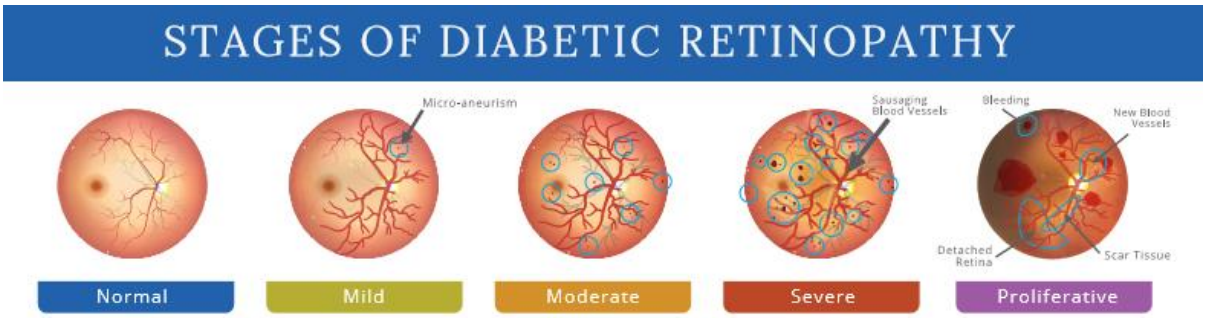
Diabetic Retinopathy Detection (Kaggle1 Competition)

Overview

Millions of people suffer from *diabetic retinopathy*[1], the leading cause of blindness among working aged adults. Generally, this condition is treatable, but it has to be caught early enough. In developing countries like India, the ophthalmologist-patient ratio is at a dismal *1:10,000*[2]. Therefore, an automated tool for grading the severity of diabetic retinopathy would be very useful for accelerating detection and treatment, especially in populations living in rural areas.

Recently, there have been a number of attempts to utilize deep learning to diagnose DR and automatically grade diabetic retinopathy. This includes a previous *competition*[3] and *work*[4] by Google. Even one deep-learning based system is FDA *approved*[5].

Clearly, this dataset and deep learning problem is quite important.



Problem Statement

This is a multi-class classification problem. The goal is to create an algorithm that is able to specify the severity of the disease given retina images taken using *fundus photography*[6] as input and producing as output 1 of 5 possible categories of severity going from 0 to 4 (0 means not having the condition).

Given the characteristics of the problem applying a convolutional neural network as image recognition technique seems to be the most likely approach to find a satisfactory solution. Two of the most successful architectures in recent years have been Microsoft's *ResNet*[7] and Google's *Inception*[8] both of which seem to be reasonable options to implement. Furthermore, to take advantage of pre-trained models I will use transfer learning and then fine-tune all the weights.

Metrics

The metric will be the *Quadratic Weighted Kappa*[9], which measures the agreement between two ratings. This metric typically varies from 0 (random agreement between raters) to 1 (complete agreement between raters). In the event that there is less agreement between the raters than expected by chance, this metric may go below 0. The quadratic weighted kappa is calculated between the scores assigned by the human rater and the predicted scores.

Images have five possible ratings, 0,1,2,3,4. Each image is characterized by a tuple (e,e), which corresponds to its scores by Rater A (human) and Rater B (predicted). The quadratic weighted kappa is calculated as follows. First, an N x N histogram matrix O is constructed, such that O corresponds to the number of images that received a rating i by A and a rating j by B. An N-by-N matrix of weights, w, is calculated based on the difference between raters' scores:

An N-by-N histogram matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater's histogram vector of ratings, normalized such that E and O have the same sum.

The main advantage of this metric is that it takes into account the possibility of agreement by chance and it also considers the distance between labels by penalizing more the predictions that are further apart from the ground truth. This kind of weighted

penalty is useful when dealing with classification of different stages of the same disease, where there is a hierarchical relation between the labels. If the truth is that the Diabetic Retinopathy is Severe, it is better to have a model that predicts the disease as Moderate than a model that classifies it as Mild, even though both models predict incorrectly, one prediction is better than the other.

II. Analysis

Data Exploration

The training data consists of two large sets of retina images taken using fundus photography under a variety of imaging conditions from multiple clinics using different cameras over an extended period. Images may contain artifacts, be out of focus, underexposed or overexposed. One set corresponds to images taken from APTOS 2015 Kaggle competition and consists of 35126 images *resized to 1024x1024*[10], and the other consists of *3662 images*[11] provided for Kaggle's APTOS 2019.

A clinician has rated each image for the severity of diabetic retinopathy:

Diagnosis	No DR	Mild	Moderate	Severe	Proliferative
Label	0	1	2	3	4

The test set consists of 1928 images provided by Kaggle for which we don't know the label and have to provide a diagnosis.

Files:

- trainLabels.csv – Table containing image filenames and diagnosis for 2015 training images set.
- train.csv – Table containing image filenames and diagnosis for 2019 images training images set.
- test.csv – Table containing image filenames for 2019 test images set.
- resized_train.zip – Folder with 35126 resized (1024x1024) images from APTOS 2015.
- train_images.zip – Folder containing 3662 images from APTOS 2019.

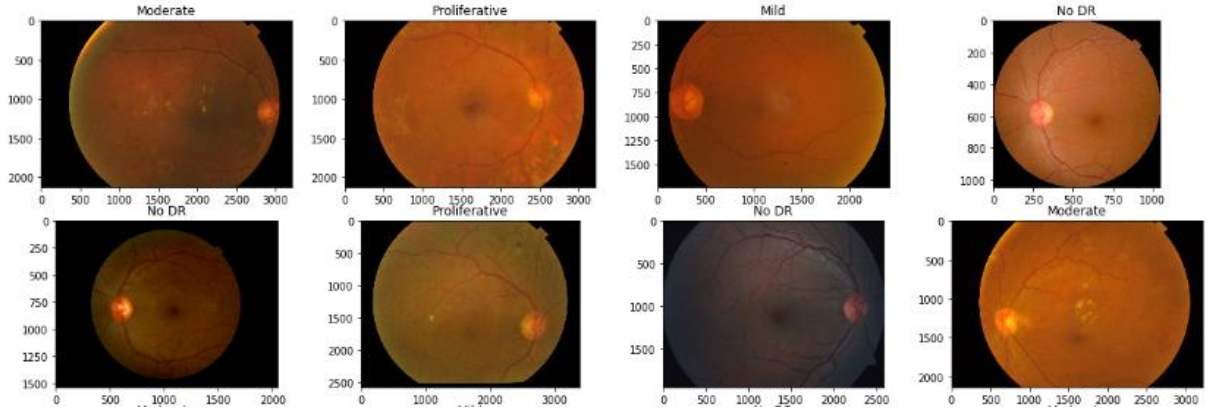
trainLabels.csv (2015)

	image	level
0	10_left	0
1	10_right	0
2	13_left	0
3	13_right	0
4	15_left	1

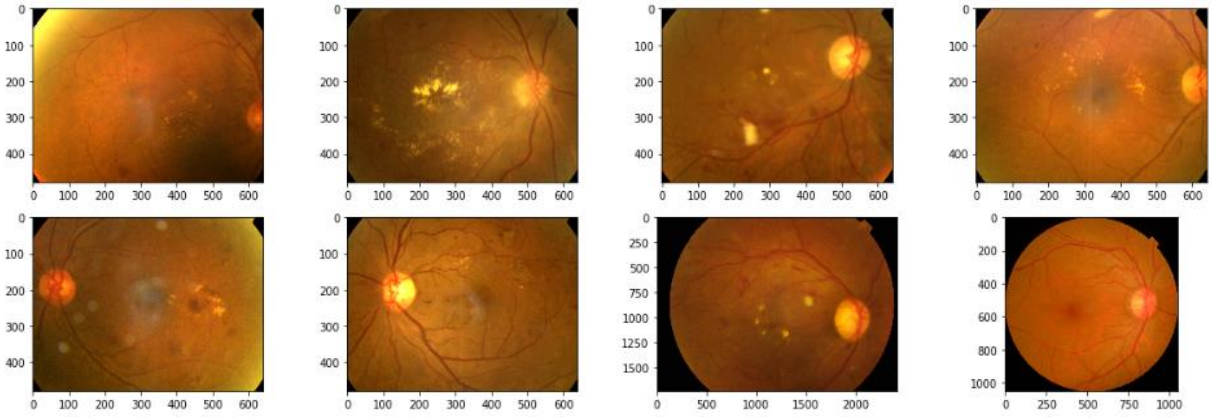
train.csv (2019)

	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4
2	0024cdab0c1e	1
3	002c21358ce6	0
4	005b95c28852	0

train_images.zip (2019)



test_images.zip (2019)



Label Distribution New vs Old

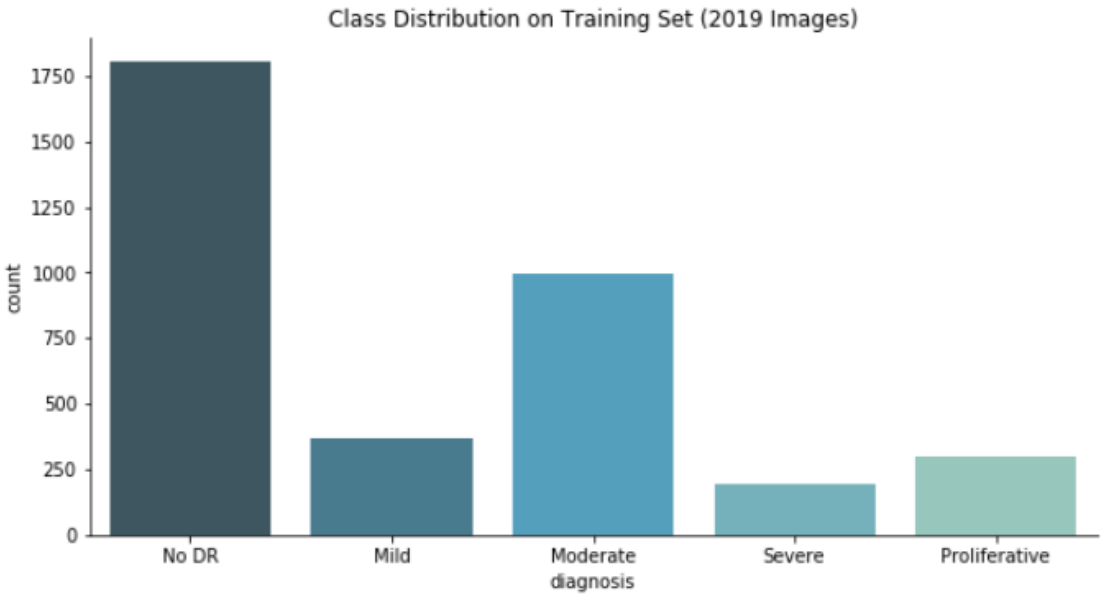
	No DR	Mild	Moderate	Severe	Proliferative
New Data	49.3%	10.1%	27.3%	5.3%	8.1%
Old Data	73.5%	7.0%	15.1%	2.5%	2.0%

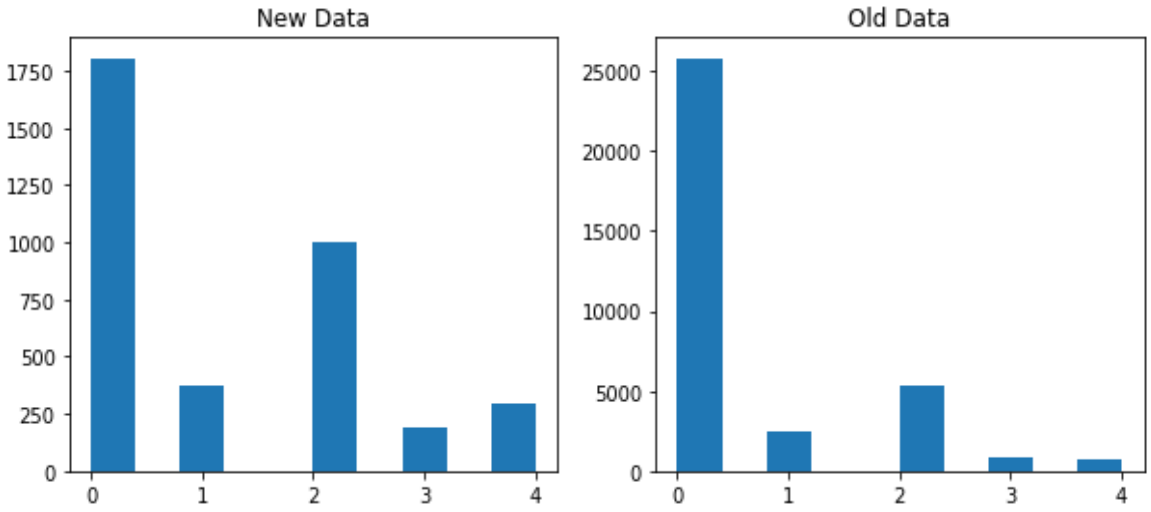
Observations:

- The .csv files from 2015 and 2019 have different column names. Renaming the columns of the old image set to match the column names of the new set will probably avoid indexing mistakes and allow a better workflow with both datasets.
- The training set from APTOS 2019 is not resized; there is a wide range of different image resolutions on the dataset. Preprocessing will be necessary to have homogenous dimensions in all images.
- There are several images on the training set with a lot of uninformative black space. These areas can be considered as noise and removing them might help the model learn better.
- The test set consistently appears to be 'zoomed-in' when compared to the training set. This suggests that it might be difficult for our model to generalize to the test data. Therefore applying data augmentations will probably be crucial for good results, particularly scaling the images.
- The Class Distribution is somewhat similar on both training sets; however, it is more unbalanced on the old dataset. Training on both datasets will help combat overfitting since the model is less prone to memorizing just one specific type of distribution.

Exploratory Visualization

When dealing with disease severity we will often encounter unbalanced distributions, where most of the classes correspond to the 'normal' or 'healthy' category. The plot below shows exactly that. Most of our images consist of retinas without Diabetic Retinopathy and generally the more severe the condition the less frequent it is. However, we can see that the new data is more balanced across labels.


Class Distribution on Training Set (2019 Images)
count
1750
1500
1250
1000
750
500
250
0
No DR
Mild
Moderate
Severe
Proliferative
diagnosis


New Data
1750
1500
1250
1000
750
500
250
0
0
1
2
3
4
Old Data
25000
20000
15000
10000
5000
0
0
1
2
3
4

Algorithms and Techniques

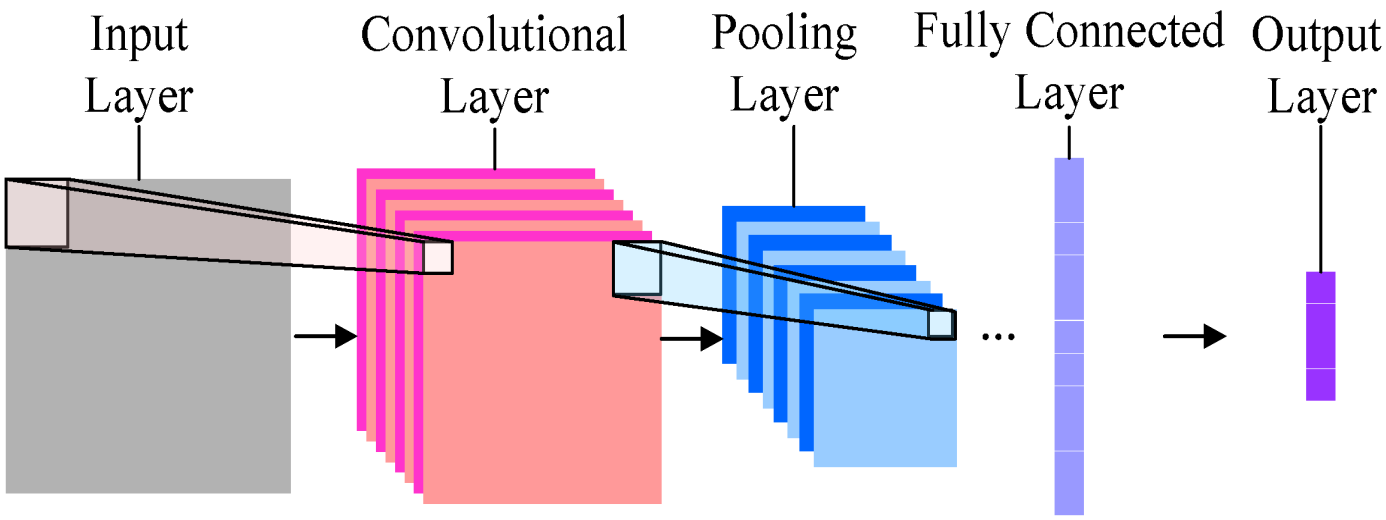
Given the characteristics of the problem, applying a convolutional neural network (CNN) as image recognition technique seems to be the most likely approach to find a satisfactory solution.

The main attribute of convolutional neural networks is their ability to learn and recognize different features or patterns present in an image. Contrary to traditional fully connected neural networks, CNN's architecture is optimized for image inputs. Instead of connecting every neuron to every other neuron of the previous layer, we are going to connect each neuron to only a local region of the input, unsurprisingly this characteristic of CNN's is called *Local Connectivity*. We do this by defining *Convolutional Layers,* which consist of a set of learnable parameters called filters. Each filter 'goes through' the width and height dimensions of the input applying dot products between its parameters and the input at every position, this will produce a two-dimensional activation map that represents the responses of that filter at every spatial position (see figure below). The goal of each filter is to capture a particular feature that the image might contain. For example, if we manually create a filter to detect vertical edges, then the final activation map of that filter will have higher values where edges are present. Luckily, since the network will automatically adjust its parameters during training, we do not have to specify every filter weight explicitly. In every layer, each filter will produce an activation map, which we will stack in the depth dimension and pass it as input to the next layer. In general, the first-layer filters of a CNN will learn to detect simple features like edges, curves or color while last-layer filters might detect more complex structures like ears, wheels, or eyes. The more layers the input goes through, the more sophisticated patterns the future ones can detect.

Although increasing the number of convolutional layers allows the model to learn more detailed characteristics of the image, it also increases the number of parameters to train, hence increasing training time and the risk of overfitting. To avoid this we usually intercalate convolutional layers with *Pooling Layers*, which reduce spatial dimensions (width and height) and thus progressively reduce the complexity of the network. Another advantage of pooling layers is that they make the network more robust to changes in the position of the feature in the image. The pooling layer operates upon each activation map separately to create a new set of the same number of pooled activation maps. Pooling involves selecting a pooling operation, much like a filter to be applied to activation maps. The size of the pooling operation or filter is smaller than the size of the activation map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels.

Finally, a fully connected layer is added at the top of the network to 'translate' all the features detected by previous layers into a category. For example, if the previous layer detected a black nose, fur, paws and a tale, then, the dense layer will learn to associate this features with the final expected class, namely, a Dog.

Convolutional Neural Network[12]



Furthermore, to take advantage of pre-trained models I will use *Transfer Learning* and *Fine-Tuning*. The idea is starting our network with the weights of a model that was already trained on a specific dataset. Pre-trained weights are useful because the model (particularly the ones trained on *ImageNet*[13]) already learned to detect features that are common to multiple different types of images; therefore, starting with this weights will save a lot of time normally required to transform the randomly-initialized filters into actual feature detectors.

Benchmark

After analyzing the discussions on Kaggle, I concluded that a Resnet50 model pre-trained on ImageNet was a decent starting point to improve upon.

Baseline Architecture:

```
def create_model(input_shape, n_out):
  input_tensor = Input(shape=input_shape)
  base_model = applications.ResNet50(weights=None,
                     include_top=False,
                     input_tensor=input_tensor)
  base_model.load_weights('../input/resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5')

  x = GlobalAveragePooling2D()(base_model.output)
  x = Dropout(0.5)(x)
  x = Dense(2048, activation='relu')(x)
  x = Dropout(0.5)(x)
  final_output = Dense(n_out, activation='softmax', name='final_output')(x)
  model = Model(input_tensor, final_output)

  return model
```

Baseline parameters:

```
BATCH_SIZE                = 8
EPOCHS                    = 20
WARMUP_EPOCHS             = 2
LEARNING_RATE             = 1e-4
WARMUP_LEARNING_RATE      = 1e-3
IMAGE_SIZE                = (512, 512)
```

Baseline Metric (Quadratic Weighted Kappa):

Kaggle's Public Test Set
0.65

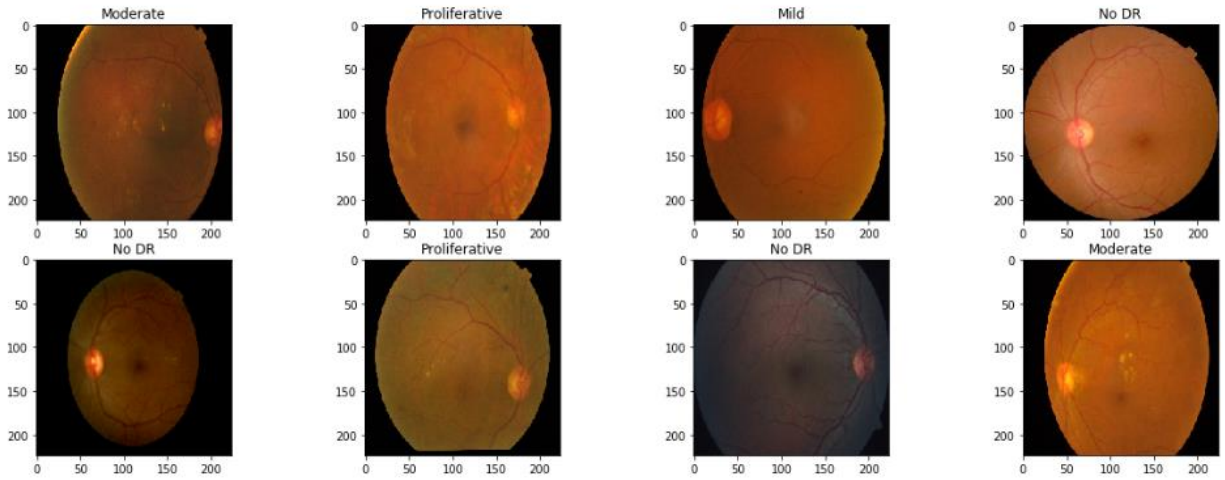
III. Methodology

Data Preprocessing

As suggested on the Data Exploration section the images were all resized before training. The selected size was (224 x 224) since it yielded the best results when considering both training time and kappa score. This good performance probably has to

do with the fact that the base model used was pre-trained on ImageNet, which has the same dimensions.

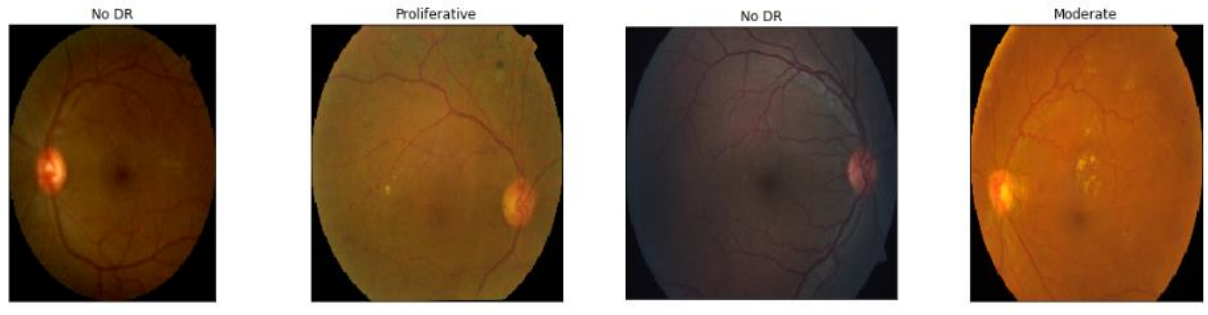
Resized images (224x224)



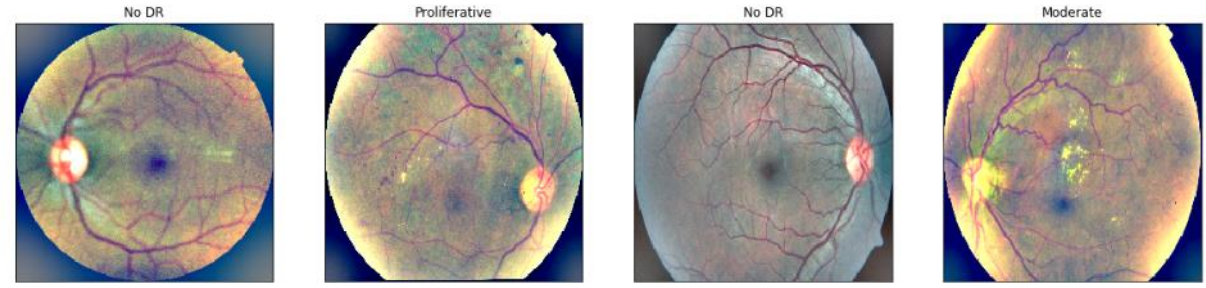
Considering approaches from the current and previous competition, I implemented different types of pre-processing; interestingly none of them resulted in a consistent improvement of the kappa score.

Below are some of the pre-processing methods tried.

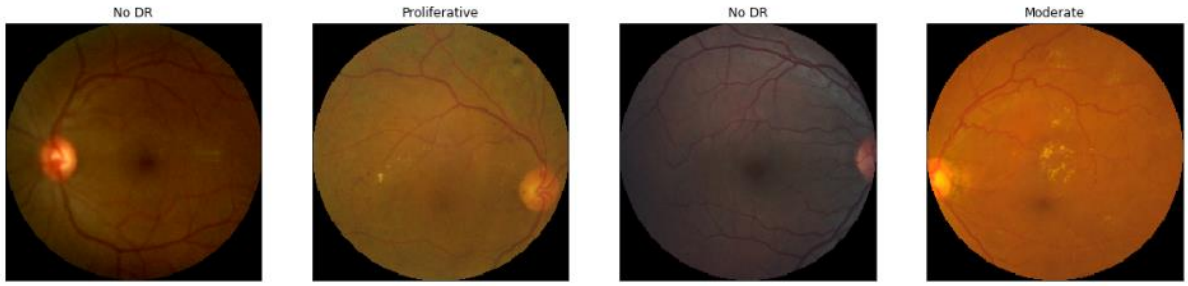
Crop black areas[12]



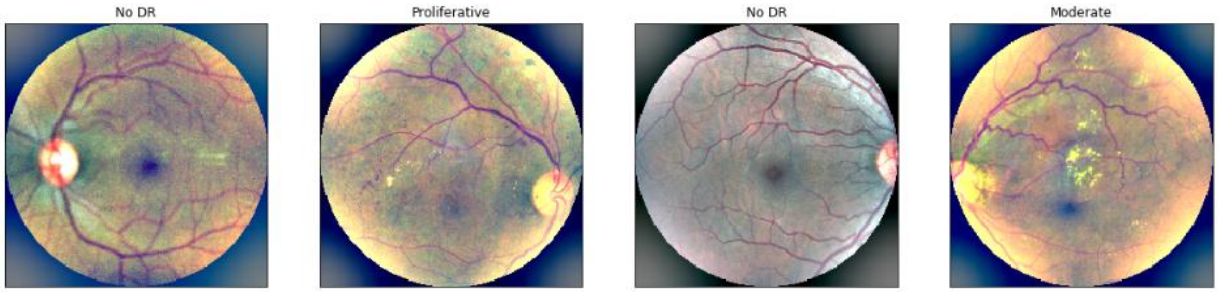
Crop Black areas + Ben Graham's Filter[12]



Make a perfect circle crop around the retina[13]



Circle crop + Ben Graham's Filter[12]



Implementation

After understanding the data at hand (image format, label distribution, etc.) and having a working baseline model, I started modifying different hyper-parameters of the benchmark to get a sense of what worked in this problem.

The first thing that became apparent was the training time of the benchmark. I wanted to try a lot of different configurations at first to know which direction to take, but every epoch took more than acceptable for rapid iteration. Therefore, the first modifications had the purpose of reducing training time:

- Changed model from ResNet50(~25M parameters) to *DenseNet121*[14](~8M parameters)
- Reduced Image size from 512 to 256
- Resized and preloaded all training images into an array and then train instead of loading and resizing each image batch at a time.
- Removed Dense Layer at the top of the model.

It turned out that DenseNet with considerable less parameters than ResNet not only trained faster but also had a better kappa score. Considering DenseNet's time advantages and good performance, I discarded the Idea of using InceptionV3, which has about the same amount of parameters as ResNet.

Modifying the problem to be a multi-label classification problem improved the score by a small amount. The method consists of converting each categorical label vector to contain all the previous labels (e.g. [0,0,1,0,0] ---> [1,1,1,0,0], [0,1,0,0,0] ---> [1,1,0,0,0]). The intuition behind it is that there is a hierarchical structure in the classes, in which each class possesses features of the previous classes; this is usually the case when dealing with severity levels of a disease.

I also applied *Test Time Augmentation*[15] (TTA), which did improve the cross validation score but did not improve the score on Kaggle's test set. The idea is that when making predictions we show multiple augmented examples of each image to the model and then take the average of the predictions as the final prediction for that sample. When submitting the model + TTA to the competition I came across a buffer overflow error. I was using Keras function *genetator.flow()* which loads all the images in memory. This ran fine on the 1928 test images available, but Kaggle then reruns it against the complete test set, which consist of 13,000 images approximately, this caused the memory error. I solved the issue by replacing *generator.flow()* with *generator.flow_from_dataframe(),* which loads the images in batches.

Implementing a custom *Kappa Callback*[16] to save the model with best Kappa validation score helped improve the leader board score. A custom *Image Generator*[16] using *imgaug* library[17] to apply multiple specific augmentations also proved to be beneficial in terms of generalization.

With this and other small modifications the Kappa Score got to **0.725.**

After having a tuned, working model, the next biggest improvement came when I used the data form 2015's APTOS competition and divided the problem into two:

- First phase: Build another model with the same architecture to train on old data and validate on new.

- Second phase: Load pre-trained weights from previous model and then retrain/validate on new data.

Some small modifications in the first model were necessary to obtain good results, the most important ones were:

- Adding a dense layer and applying warm-up steps (freeze base-model weights while training top layer)
- Reversing multi-label modification to a normal classification problem.

This approach improved the score to **0.754**.

Refinement

After trying different optimizers (Adam, Adamax, Nadam), finetunning & warmup learning rates, image sizes (224, 256, 320, 512) and batch sizes (8, 32, 64, 128), the most successful setup was the following:

First model (Train on old validate on new)

```
TOP_LAYERS                = 1024
OPTIMIZER                 = Adam
LEARNING_RATE             = 1-e5
EPOCHS                    = 15
BATCH_SIZE                = 32
WARMUP_LEARNING_RATE      = 1e-3
WARMUP_EPOCHS             = 2
WARMUP_BATCH_SIZE         = 128
IMAGE_SIZE                = (224, 224)
```

Second Model (Train and validate on new)

```
TOP_LAYERS                = 1024
OPTIMIZER                 = Adam
LEARNING_RATE             = 1-e5
BATCH_SIZE                = 64
EPOCHS                    = 20
LEARNING_RATE             = 1e-4
IMAGE_SIZE                = (224, 224)
```

With this configuration, I obtained the highest kappa score so far: **0.768.**

Summary	
Model	**Kappa Score**
Baseline	**0.65**
DenseNet	**0.68**
DenseNet (Tuned)	**0.725**
DenseNet + Old Data	**0.754**
DenseNet + Old Data (Tuned)	**0.768**

IV. Results

Model Evaluation and Validation

The improvement of the model was monitored using mainly the official leaderboard score provided by Kaggle. This score is highly reliable since it's obtained after running the model against an completely unseen test set with around 13,000 images (bigger than 2019's training set). Moreover, while exploring the data we could notice that there is a considerable difference between training and test sets. The training set often contains big uninformative black areas and the test set consistently appears to be 'zoomed-in' in comparison to training data. Although these differences make obtaining a good score more difficult, they also guarantee to some extent the robustness of a good scoring model.

The final architecture, techniques and hyper-parameters were chosen because they yielded the best Kappa Score when tested against the unseen test set.

Justification

The last iteration of the model obtained a Quadratic Weighted Kappa of **0.768**, which is a considerable improvement over the Resnet50 benchmark, which obtained an official Kappa Score of **0.65**.

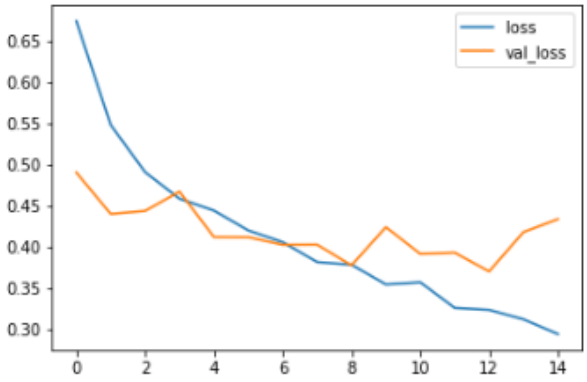
When I submitted the benchmark model and received an official position on the leaderboard, I was between position 1300 and 1400 from 2100+ participants. At the

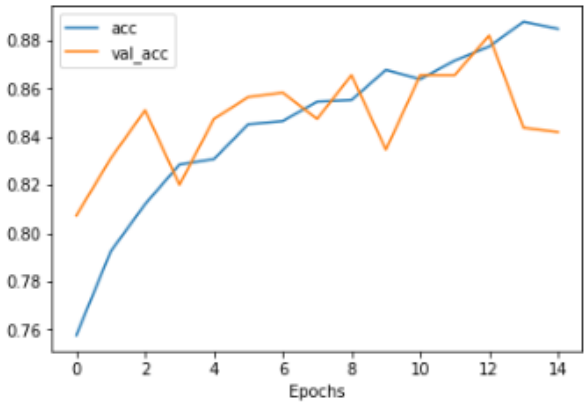
moment of submission, the model that received **0.768** corresponded to position 678 from 2300+ participants. This represents a jump from the top 65% to the top 30%, which is a considerable improvement that exceeded my expectations.

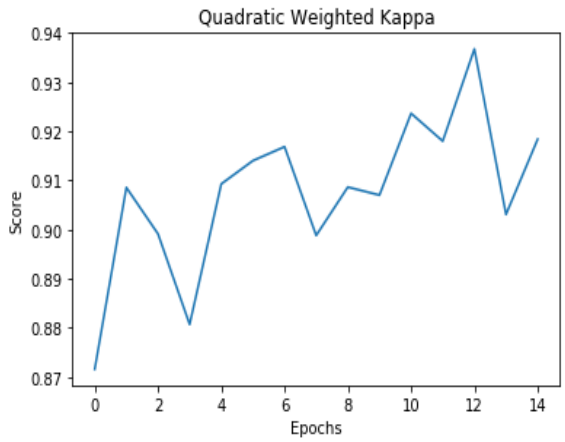
V. Conclusion

In the plots bellow we can appreciate the training and validation loss change across epochs. For the most part of the training phase, the model generalized well to the validation set. It is around epochs 8-12 that we start to see signs of overfitting, where the validation loss starts to increase and the validation accuracy dips. The validation Kappa score drops after epoch 12, this is where the custom kappa callback to save the best model based only on the Kappa score proved useful.

Accuracy & Loss







Reflection

This was my first experience with Kaggle competitions. I got the chance to learn how to approach complicated computer vision problems in practice. The process was not as smooth as I thought it would be; almost every little configuration involved some extra steps with which I needed to familiarize first. I also found interesting that, at least in this competition, the majority of the innovation and work required to improve the model did not focus on the architecture as I previously expected but rather on data preprocessing, augmentation techniques and different training strategies.

In addition to finding a fast model to be able to iterate quickly over different configurations, I learned to work on different ideas in parallel while other models are training. Kaggle makes this process intuitive and organized. This is a crucial skill in order to have an efficient workflow, since waiting for each idea to yield a result means a lot of idle time. The fact that this was a synchronous kernels-only competition and the unseen test set was 7x bigger than the one available made it a requirement to be clever when using memory resources, this took a couple of days to fully understand and find a workaround.

Another main difficulty was the big differences between training and testing images, which resulted into high cross-validation scores but much lower official kappa scores. This demanded many different augmentations and other techniques to build a robust model that could generalize well to the testing set. The availability of the training set from 2015 APTOS competition was also crucial to tackle this problem.

Although a Quadratic Weighted Kappa Score of **0.768** can be considered a good score, and it is a significant improvement over the benchmark, this is not enough to deploy in the general setting. Extra rigorous quality standards need to be in place when working with healthcare problems that directly involve patients. Even if the kappa score was higher, the model will only be as good as the physicians that labeled the images. That rigorousness need also be applied to the labeling process in order to build a reliable model.

Improvement

There are a number of techniques that could be implemented to further improve the current model. Ensemble methods is one of the strategies that proved to be most successful according to the discussion forums of the competition. EfficientNets would be a good candidate to use as a voter for the final prediction. K-fold cross-validation can also be implemented to make better use of the available data and better measure the robustness of the model. Since none of these added implementations change the current architecture but are rather added on top of it, they would likely result in an improvement on the final score.

References

1. https://nei.nih.gov/health/diabetic/retinopathy
2. https://www.sankaranethralaya.org/a-step-towards-combating-blindness-in-rural-areas.html
3. https://kaggle.com/c/diabetic-retinopathy-detection
4. https://ai.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html
5. https://www.fda.gov/NewsEvents/Newsroom/PressAnnouncements/ucm604357.html
6. https://en.wikipedia.org/wiki/Fundus_photography
7. https://arxiv.org/abs/1512.03385
8. https://arxiv.org/abs/1409.4842
9. https://en.wikipedia.org/wiki/Cohen%27s_kappa
10. https://www.kaggle.com/tanlikesmath/diabetic-retinopathy-resized
11. https://www.kaggle.com/c/aptos2019-blindness-detection/data
12. https://www.mdpi.com/2078-2489/7/4/61
13. http://www.image-net.org
14. https://www.kaggle.com/ratthachat/aptos-updatedv14-preprocessing-ben-s-cropping
15. https://www.kaggle.com/taindow/pre-processing-train-and-test-images
16. https://www.kaggle.com/xhlulu/aptos-2019-densenet-keras-starter
17. https://towardsdatascience.com/test-time-augmentation-tta-and-how-to-perform-it-with-keras-4ac19b67fb4d
18. Custom Image Generator & Custom Quadratic Weighted Kappa Callback
19. https://imgaug.readthedocs.io/en/latest/index.html