Machine Learning Engineer Nanodegree

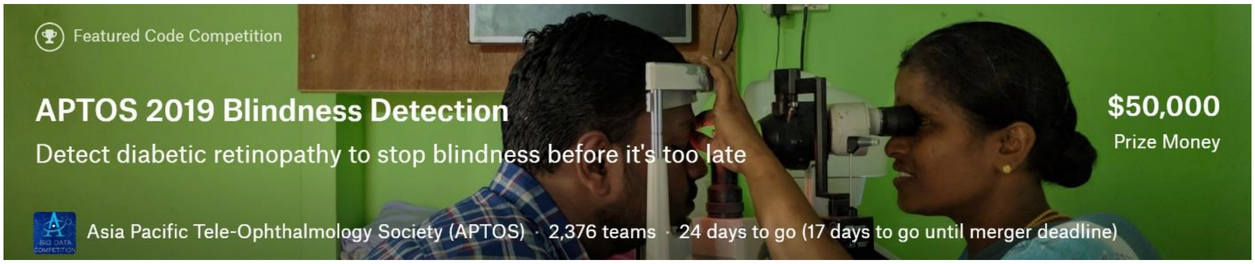
Capstone Project

Aaron Balson Caroltin J

August 2019

I. Definition

Project Overview



Featured Code Competition

APTOS 2019 Blindness Detection

Detect diabetic retinopathy to stop blindness before it's too late

$50,000

Prize Money

Asia Pacific Tele-Ophthalmology Society (APTOS) · 2,376 teams · 24 days to go (17 days to go until merger deadline)

Imagine being able to detect blindness before it happened.

Millions of people suffer from diabetic retinopathy, the leading cause of blindness among working aged adults. Aravind Eye Hospital in India hopes to detect and prevent this disease among people living in rural areas where medical screening is difficult to conduct. Successful entries in this Kaggle competition will improve the hospital's ability to identify potential patients. Further, the solutions will be spread to other Ophthalmologists through the 4th Asia Pacific Tele-Ophthalmology Society (APTOS) Symposium. The dataset for the project can be found here.

Problem Statement

Currently, Aravind technicians travel to rural areas to capture images and then rely on highly trained doctors to manually review the images and provide diagnosis. One study found a 49% error rate for diagnosing Retinopathy among internists, diabetologists, and medical residents **[1.1]**. Another study shows the Sensitivity (95% CI) of direct ophthalmoscopy (looking into patients) for AR (Any Retinopathy), NSTDR (Non-Sight

Threatening Diabetes Retinopathy) and STDR (Sight Threatening Diabetes Retinopathy) was found to be 55.67% (50.58–60.78), 37.63% (32.67–42.59) and 68.25% (63.48–73.02) respectively **[1.2]**. When diagnosing from fundus photography (without patients), this number can only get worse.

The goal is to scale and assist their efforts through technology; to gain the ability to automatically screen images for disease and provide information on how severe the condition may be, as accurately and soon as possible.

Hence the problem statement is to classify the image to the correct stage of Retinopathy from one of five given classes. Following are the outline of tasks used to achieve desired solution.

1. Download data and dependencies to setup the workspace.
2. Preprocess and resample the data which was highly imbalanced.
3. Identify a Convoluted Neural Network Classifier which is adequate (for the purpose of academia) using transfer learning with better computational and memory efficiency.
4. Adjust the parameters and train the model.
5. Measure the performance metrics to evaluate the model.
6. Predict the test dataset and submit back the results to acquire the final ranking of the model based on kappa score metric.

Metrics

The final iteration of the model will be evaluated based on

1. Accuracy
2. Recall (sensitivity)
3. Quadratic Weighted Kappa (QWK)

After rebalancing the dataset using resampling, accuracy has become a useful metrics. Recall (sensitivity) and F1 score are also keenly tracked for the medical model. However, the competition measures the model using Cohen’s Kappa statistics hence it is our primary metrics to evaluate before submission.

We can construct the following table:

True	Pred	Agreement
1	1	True Positive (TP)

0	0	True Negative (TN)
0	1	False Positive (FP)
1	0	False Negative (FN)

Total = TP + TN + FP + FN

Accuracy = (TP+TN) / Total

Accuracy is the ratio of correct predictions to total entries.

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1 Score = 2*(Precision * Recall) / (Precision + Recall)

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It takes value between 0,1.

Quadratic Weighted Kappa (QWK, the Greek letter κ), also known as Cohen's Kappa, is the *official evaluation metric for the competition*. Cohen’s kappa statistic is a very good measure that can handle very well both multi-class and imbalanced class problems. For our kernel, we will use a custom callback to monitor the score, and plot it at the end. According to the Wikipedia article **[1.3]**, we have Cohen's kappa measures the agreement between two raters who each classify N items into C mutually exclusive categories. The definition of κ is

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e},$$

where p_o is the relative observed agreement among raters (identical to accuracy), and pe is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly seeing each category.

Simply put, if two scores disagree, then the penalty will depend on how far they are apart. That means that our score will be higher if (a) the real value is 4 but the model predicts a 3, and the score will be lower if (b) the model instead predicts a 0. This metric makes sense for this competition, since the labels 0-4 indicates how severe the illness is. Intuitively, a model that predicts a severe retinopathy (3) when it is in reality a proliferative retinopathy (4) is probably better than a model that predicts a mild retinopathy (1).

II. Analysis

Data Exploration

We are provided with a large set of retina images taken using fundus photography under a variety of imaging conditions. A clinician has rated each image for the severity of diabetic retinopathy on a scale of 0 to 4:

0 - No DR

1 - Mild

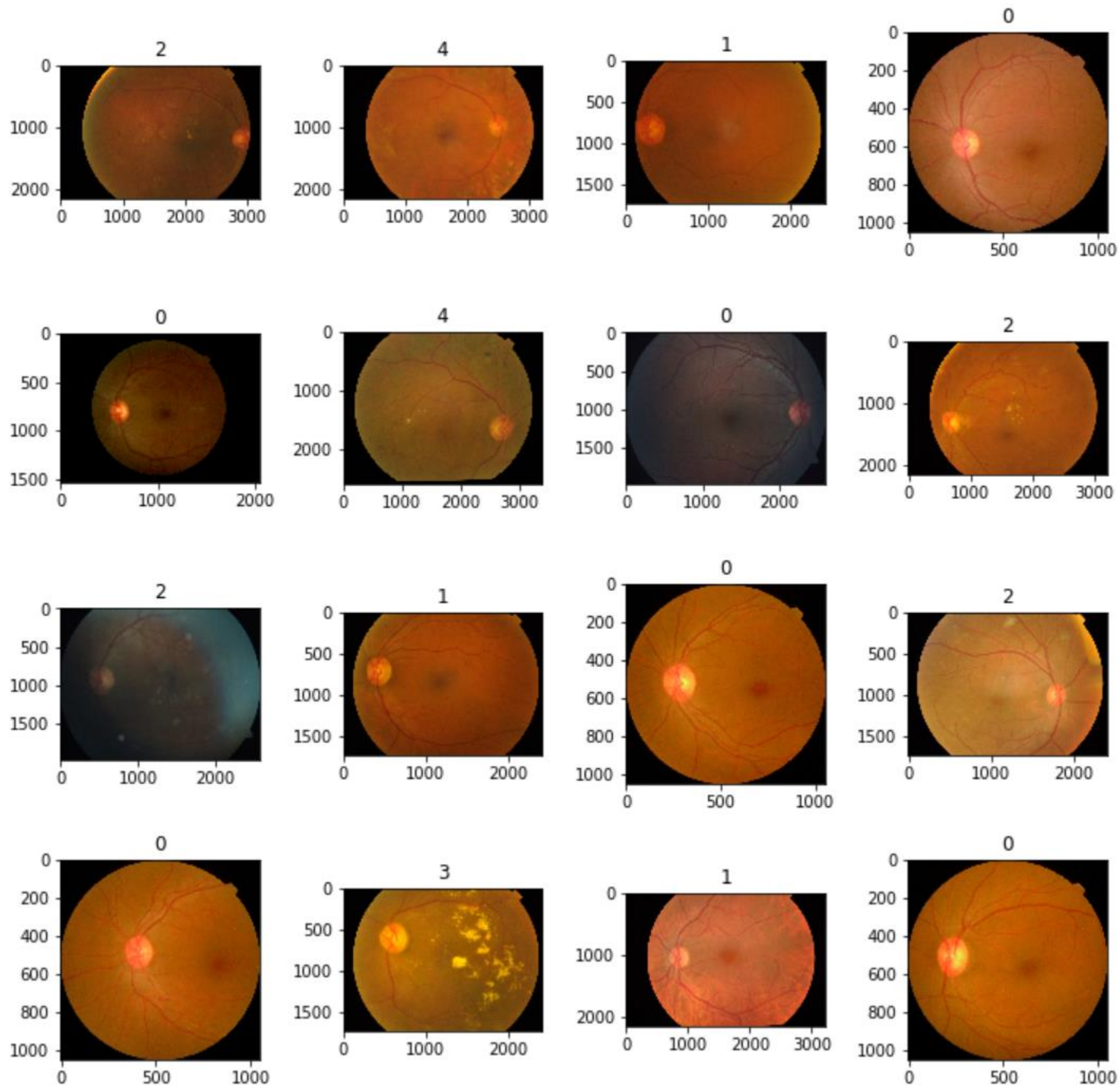
2 - Moderate

3 - Severe

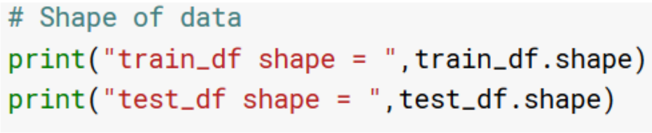
4 - Proliferative DR

Images 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. We were given train and test datasets separately.

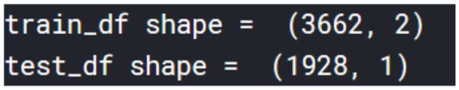
Fig-1: Few sample fundus images from training set (with classes on top and pixel sizes)



Initial data exploration revealed the following details about the data. We have 3662 samples for training and 1928 for testing (only) compared to 2016 competition having over 30K.



```
# Shape of data
print("train_df shape = ",train_df.shape)
print("test_df shape = ",test_df.shape)
```



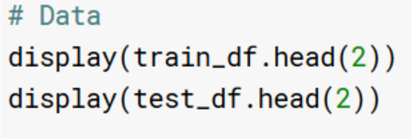
```
train_df shape =  (3662, 2)
test_df shape =  (1928, 1)
```

Train dataset has one feature column id_code and one label diagnosis.

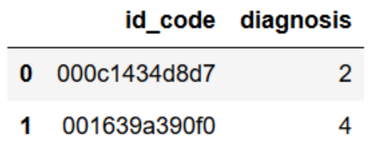
Test dataset has only id_code.

id_code: corresponding image name of that sample

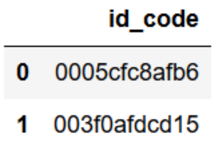
diagnosis: contains the severity of DR.



```
# Data
display(train_df.head(2))
display(test_df.head(2))
```

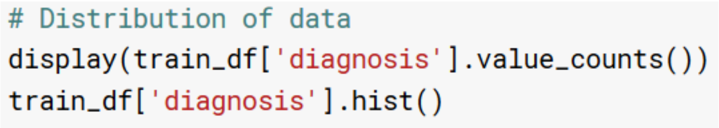


	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4

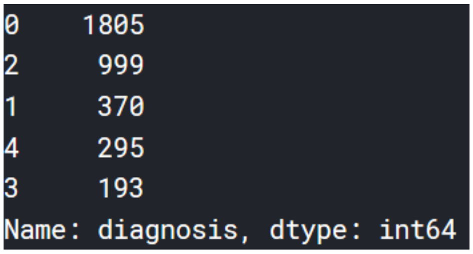


	id_code
0	0005cfc8afb6
1	003f0afdcd15

In medical data sets, data are predominately composed of "normal" samples with only a small percentage of "abnormal" ones, leading to the so-called class imbalance problems. **[2.1].** We can see the normal (0 – No DR) has huge samples (1805) while other severe classes are relatively small.



```
# Distribution of data
display(train_df['diagnosis'].value_counts())
train_df['diagnosis'].hist()
```



```
0    1805
2     999
1     370
4     295
3     193
Name: diagnosis, dtype: int64
```

Observations: Relatively smaller dataset is given for training and testing, compared to 2016 competition. Class imbalance is also noticed.

Exploratory Visualization

As shown below, the class frequency distribution of the train dataset shows high imbalance and traditional classification algorithms doesn’t fare well **[2.2]** without addressing them. We decided to down sample class 0 and up sample (synthesizing) other classes to make them balanced (1000 each).

Fig-2: Frequency distribution of diagnosis classes

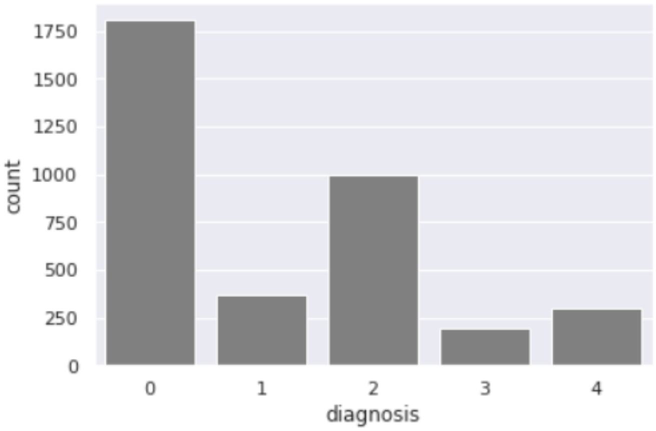


Fig-3: Resolution, scale, size and brightness variances in dataset.

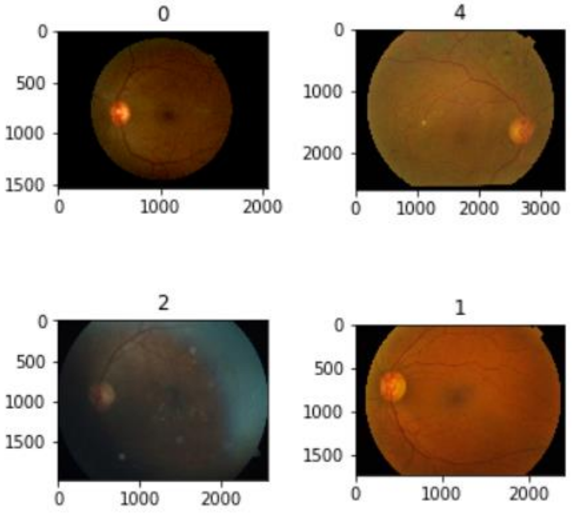


Image sizes vary from 1000 to 3000 pixels (width and height) as shown above which need to be resized. However, we are aware of dangers some unclear images possess to learning. Given time and effort, it is possible to isolate and correct (using custom filters or cropping) or remove them from training set. This will be a scope for improvement but for now, hopefully such images are non- detrimental to training results.

Algorithms and Techniques

The test dataset comes with no label; hence it is required to submit(commit) the model to Kaggle workspace and wait (for hours!) to get back the test score/results for the iteration.

Considering this workflow, we went looking after a light weight network with good Transfer Learning capability. DenseNet121 architecture is chosen because of its efficiency in computation and memory utilization over ResNet50 and VGG **[2.3] [2.4]** and also quicker to iterate within Kaggle. The final network parameters/settings are discussed under implementation section. Images are resized to 224 x 224 x 3 pixels to match that of ImageNet's format on which the DenseNet model was originally trained.

ImageDataGenerator is used to generate augmented images.

Few parameters used are

- zoom_range=0.10   # set range for random zoom
- fill_mode='constant'# set mode for filling points outside the input boundaries
- cval=0.   # value used for fill_mode = "constant"
- horizontal_flip=True # randomly flip images
- vertical_flip=True   # randomly flip images
- rotation_range=20   # Degree range for random rotations

SMOTE - Synthetic Minority Oversampling Technique. This is a statistical technique for increasing the number of cases in your dataset in a balanced way. The module works by generating new instances from existing minority cases that you supply as input. This implementation of SMOTE does **not** change the number of majority cases. **[2.5]**

The default (few) parameters used are

- Sampling_strategy='auto'
- K_neighbors=5
- N_jobs=1
- Ratio=None

Benchmark

During proposal reviews 1 & 2, it was decided to set the benchmark based on average human medical practitioner's highest sensitivity [CI=.95] for direct ophthalmoscopy (with patients) as suggested by **[1.1], [1.2]** never tops 68.25%. It is safe to assume then, the sensitivity of doctors doing fundus image diagnosis (with no patients) will be even lesser. By beating 68.25%, the model can be used in real world to assist diagnosis, at least on par with trained doctors. We also decided not to benchmark on a fellow competitor since competition is in progress and models are evolving over months. Hence we skipped the initial/preliminary/benchmark model creation and went straight with transfer learning. However, the challenge is to run the iterations several times with personal goal to beat the public score resulted by previous submissions. The real-world metric is sensitivity (recall) and public scoring uses a stricter kappa score. We can also keep track of F1 score, recall from training vs validation runs.

III. Methodology

Data Preprocessing

These are the steps taken for cleaning and preprocessing data.

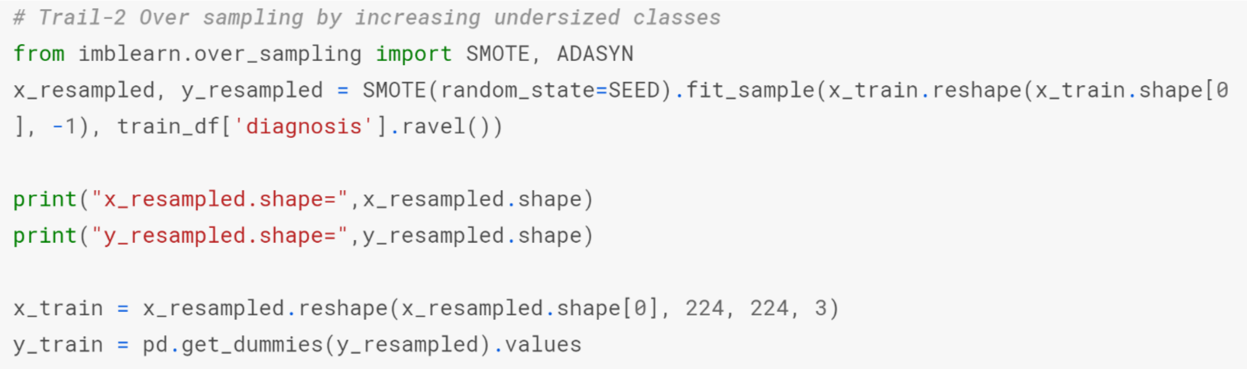
1. To address the imbalances in dataset, it is decided to down-sample the class-0 from training set and do up-sampling on remaining classes.
2. We randomly removed 805 from class-0's 1805 entries by keeping other classes intact.
3. To address the scale and size variances in each image, we resized each image in memory as pixel array 224 x 224 x 3 (w x h x d) where d = {R, G, B} channels. We created two arrays one each for training and testing set.
4. The reason to choose 224 x 224 x 3 was the underlying network (DenseNet121) was also trained on ImageNet using same format and to match their bottle neck features.
5. We applied SMOTE technique to training set to up-sample remaining classes. Now the training set is balanced with 5000 samples (1000 for each class).
6. The diagnosis target column in train dataset was initially one-hot encoded for multi-class classification. Since kappa score is the official evaluation metrics, encoding a class by encompassing all the classes before it seems to work out well for kappa scores. Instead of predicting a single label, we will change our target to be a multilabel problem; i.e., encoding a class 4 retinopathy would usually be [0, 0, 0, 1, 0], but in our case we will predict [1, 1, 1, 1, 0]. For more details, please check out this paper on Ordinal Regression, also implemented by Lex's kernel.
7. To address the scarcity in dataset, we used image augmentation (with above mentioned transformation) to better generalize the training.

Implementation

Totally we did 7 incremental improvements (model submissions) of which we will discuss mostly the last version in detail Kaggle V4 (commit-7).

In earlier iterations when there was class-0 imbalance, the accuracy and precision were superficially high but model performance was poor. In first trail, down-sample all classes to match the class with lowest samples (each class was reduced to 193 samples). However, this introduced data scarcity and doesn't generalize well on submission.

In second trail, resample was to perform both down-sampling class-0 to 1000 and up-sampling other classes to 1000 so each class gets 1000 images. The complicated coding part was to up-sample the dataset using SMOTE technique. I was getting array size mismatch errors till I was able to reshape 4D to 2D array as input to SMOTE and output from 2D back to 4D without losing information and adding back to training set. SMOTE intelligently creates new synthetic samples for underrepresented classes based on clustering and k-nearest neighbors of sparsely distributed classes.

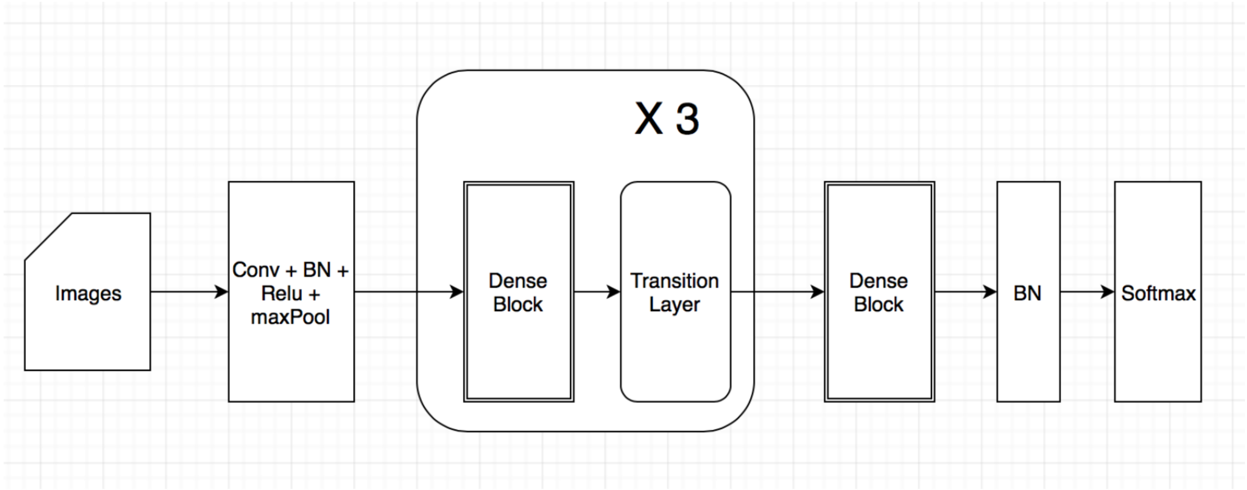


```
# Trail-2 Over sampling by increasing undersized classes
from imblearn.over_sampling import SMOTE, ADASYN
x_resampled, y_resampled = SMOTE(random_state=SEED).fit_sample(x_train.reshape(x_train.shape[0
], -1), train_df['diagnosis'].ravel())

print("x_resampled.shape=",x_resampled.shape)
print("y_resampled.shape=",y_resampled.shape)

x_train = x_resampled.reshape(x_resampled.shape[0], 224, 224, 3)
y_train = pd.get_dummies(y_resampled).values
```

We used image data augmentation for variety and used train-test split for creating new train, validation sets.

We used DenseNet121 for transfer learning in all iterations. DenseNet contains a feature layer (convolution) capturing low-level features from images, several dense blocks, and transition layers between adjacent dense blocks **[3.1]**

Fig-4: DenseNet121 Architecture



The architecture of Densenet

Using Keras and a Tensorflow backend, we start with a sequential model and add more layers. We used DenseNet121 model (no tops) with ImageNet pretrained weights DenseNet-BC-121-32-no-top.h5 as our input layer (of 224 x 224 x 3) and followed it by Global Average Pooling (GAP) layer, a Dropout layer and Dense layer with Sigmoid activation as output layer.

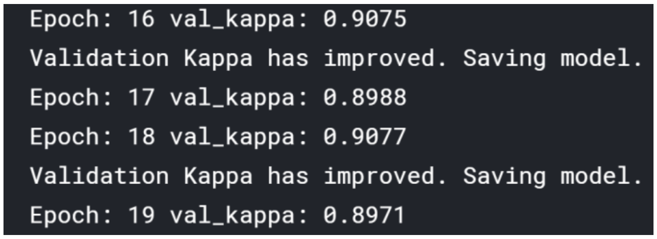
GAP layer reduces the dimensions from previous layer and render network to train faster. Dropout layer helps to randomly switch off 20% neuron activations from previous layer, thus empowering weaker nodes to participate in training and to avoid overfit. The output layer consists of 5 nodes, one each for our classes and uses Sigmoid activation to predict the labels (which are in multilabel format [1,1,1,0,0] for class 2).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 7, 7, 1024)        7037504
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 5125
=================================================================
Total params: 7,042,629
Trainable params: 6,958,981
Non-trainable params: 83,648
_________________________________________________________________
```

Following properties were used on model to compile, fit & predict

- Epoch = 50
- Train-validation split = 90-10
- Learning rate = 0.00005
- Dropout = 0.20
- Batch_Size = 16
- Steps_per_epoch = train_set / 16
- Metrics = 'accuracy', mean_pred, precision, recall, f1_score, fbeta_score, fmeasure
- Optimizer = Adam
- Loss = binary_crossentropy

Since kappa score is used to evaluate final model by Kaggle, apart from tracking these metrics, a new callback method *KappaMetrics* was introduced to model.fit which outputs kappa score to console on each epoch and saves the model at each highest kappa score. The final model will be the one with highest kappa score.



```
Epoch: 16 val_kappa: 0.9075
Validation Kappa has improved. Saving model.
Epoch: 17 val_kappa: 0.8988
Epoch: 18 val_kappa: 0.9077
Validation Kappa has improved. Saving model.
Epoch: 19 val_kappa: 0.8971
```

Since the test data lacks label, the result from model.predict is submitted each time to Kaggle server to return us test score metrics (kappa score): sometimes after hours! This disconnected workflow introduced difficulty in quick evaluations or model tuning or using cross-validations. Every design / hyperparameter change is submitted to get test score back.

Refinement

The initial solutions (V1, V2) started as huge promise in terms of accuracy (imbalanced) and the design pretty much stayed the same through future iterations except for below tuning. But initial versions were not submitted so public score was not available. However, the network architecture is same for all 7 iterations except these parameters.

Public score is ranked based on private test data held by Kaggle and uses kappa score. Hence the aim was to incrementally increase kappa score (and hence the public score) without dropping other metrics (Accuracy, F1 score, Precision, Recall all recorded at high nineties in all iterations).

From below table, intermediate and final versions show clear increase in model performance backed by public score. APTOS final commit-7 can be found here.

Table-1: Iteration Movement with configuration changes

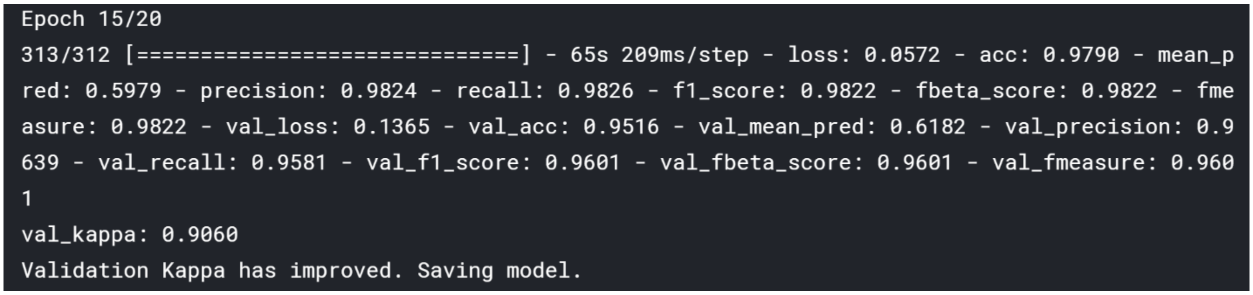
Iterations =>	V1, V2	V3	V4	V5	V6	V7 (latest)
train-valid split	85-15	85-15	85-15	85-15	90-10	90-10
resampling	No	No	Down	Down, Up	Down, Up	Down, Up
data-aug-rotation	40, 30	20	0	0	0	0
data-aug-zoom	0.15	0.15	0.15	0.15	0.10	0.10
learning-rate	0.00001	0.00001	0.00001	0.00005	0.00005	0.00005
epoch	20	20	25	30	20	50
batch-size	32	32	32	32	16	16
dropout	0.5	0.5	0.5	0.5	0.2	0.2
kappa-metrics	??	0.8900	0.9017	0.8900	0.9180	0.9270
public-score	N/A	0.667	0.674	0.686	0.696	0.709

IV. Results

Model Evaluation and Validation

Kaggle Competition withholds test data labels (given features only) and model has to submit predictions to server and wait hours to get result. This workflow also limits one submission at a time. For these reasons, model evaluation (final test score) and validation (hyper parameter tuning) are performed over multiple submissions (v1 – v7) as shown in above table.

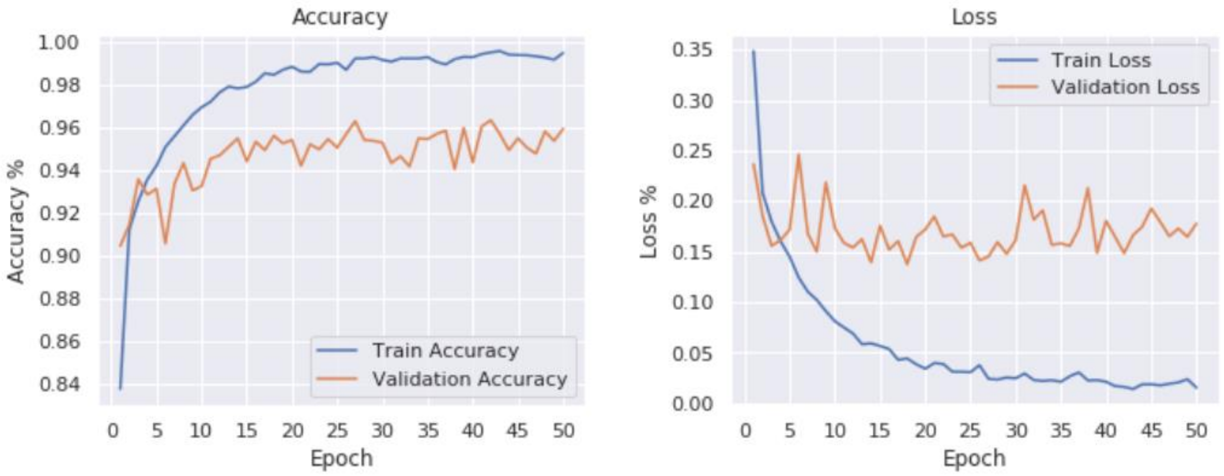
Metrics like accuracy, F1 score, kappa are tracked during the model training phase using training & validation splits, which are the only indicators available during local runs.



```
Epoch 15/20
313/312 [==============================] - 65s 209ms/step - loss: 0.0572 - acc: 0.9790 - mean_p
red: 0.5979 - precision: 0.9824 - recall: 0.9826 - f1_score: 0.9822 - fbeta_score: 0.9822 - fme
asure: 0.9822 - val_loss: 0.1365 - val_acc: 0.9516 - val_mean_pred: 0.6182 - val_precision: 0.9
639 - val_recall: 0.9581 - val_f1_score: 0.9601 - val_fbeta_score: 0.9601 - val_fmeasure: 0.960
1
val_kappa: 0.9060
Validation Kappa has improved. Saving model.
```

The training vs validation accuracy stays within 4% and loss stays within 15% (starting to slightly overfit).

Fig-5: CNN Performance 1



The public score seems to be a factor of kappa score (correlated). We performed sensitivity analysis (varying inputs) across iterations and attain similar results to verify that the model is immune to sensitivity.

kappa-metrics	??	0.8900	0.9017	0.8900	0.9180	0.9270
public-score	N/A	0.667	0.674	0.686	0.696	0.709

Final model characteristics

- The model (V7) is finalized with public score reaching (0.709) and seems reasonable and aligning with solution expectations.
- Final parameters of V7 seem appropriate for now.
- Final model seems to generalize well to unseen data from server.

- The model may be robust but as there are other models with 0.850 public score (but trained for a longer time). Small perturbations in training data doesn't affect the results.
- It is early to completely trust the model, but has potential to eventually become one - to augment or replace human decision making; when it uses better preprocessing stages with custom filters and learn from more genuine samples.

Justification

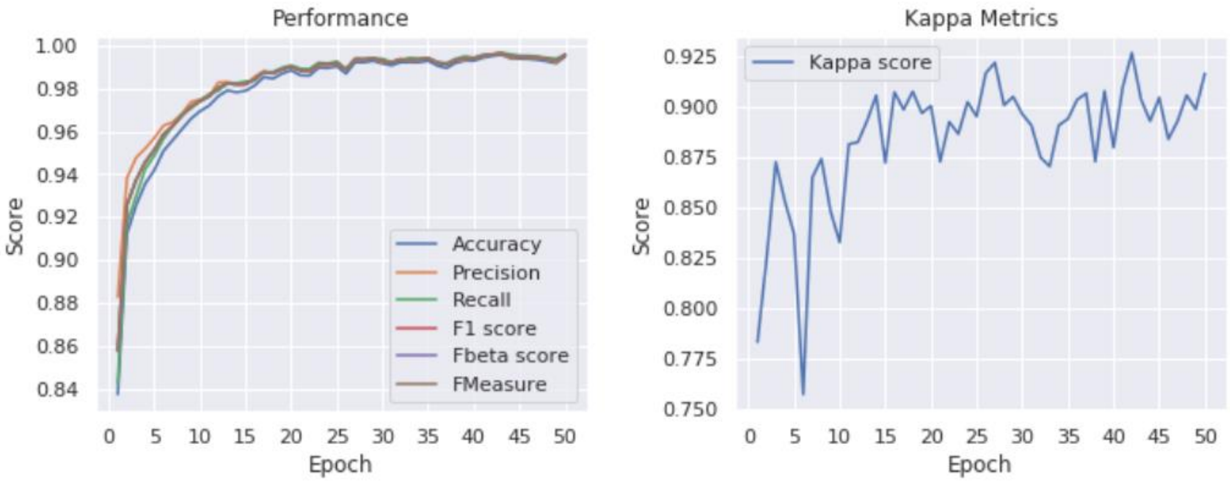
From Kaggle's perspective, the final model (3 days old) exhibited a public score of 0.709 and ranked 1487 out of 2444 teams which is reasonable considering how top models with 0.840 have been evolving last 3 months. The benchmark is defined as sensitivity (recall) [CI=0.95] of 68.25% and public score of 0.709 is defined in kappa metric. Though there is no direct means to compare them head on head,

- 0.01 – 0.20 slight agreement
- 0.21 – 0.40 fair agreement
- 0.41 – 0.60 moderate agreement
- 0.61 – 0.80 substantial agreement
- 0.81 – 1.00 almost perfect or perfect agreement

our score of 0.709 is in substantial agreement with test dataset when it comes to predicting disease class. Also, our validation phase agrees with high recall and kappa score. I believe our final solution is significant / substantial enough to solve the problem.

As part of benchmark, I had also set myself to challenge my previous public score on every submission and was able to do (Table-1)

Fig-6: CNN Performance 2



V. Conclusion

Free-Form Visualization

Following are the pre-processing efforts done on input sample by the winner of Diabetic Retinopathy competition on 2016. He scaled the images to a given radius, then subtracted local average color to reduce differences in lighting. Similarly, other people have come up with clever filters to bring uniformity to samples.

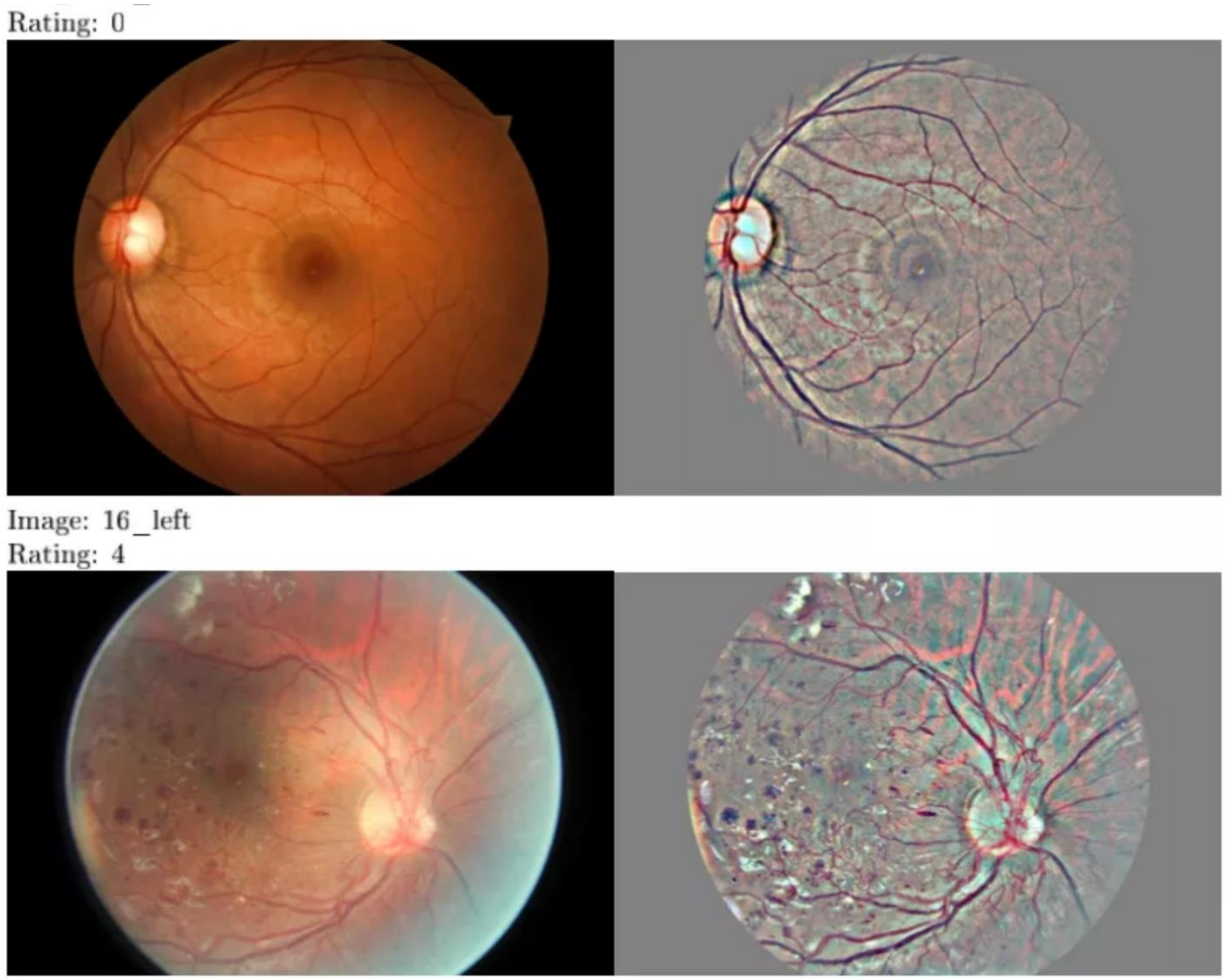


Figure 4: Two images from the training set. Original images on the left and preprocessed images on the right.

Reflection

For a while I was unsure about the type of capstone project to pursue and contemplated on many until I found this one. I had little time and CNN experience with me but this hospital is at my home city which made me pick. After getting proposal approved, I am glad to see many people compete on same problem at different levels which gave insight into how real-world projects work. It was interesting to see people compete on same problem and come with so much diverse solutions.

Kaggle platform was explored and problem statement was defined with related data and workspace identified. Many CNN kernels were referred for inspiration and many techniques came to light. EDA was done along with resampling and pre-processing. It was the most challenging part of the project from code perspective. Setting up the network was easy since I had completed Dog Breed project just a week back and was fresh on me. Another challenge was to work within Kaggle's workflow and submission system. Test score feedback was slow and hard to wait and plan things between iterations.

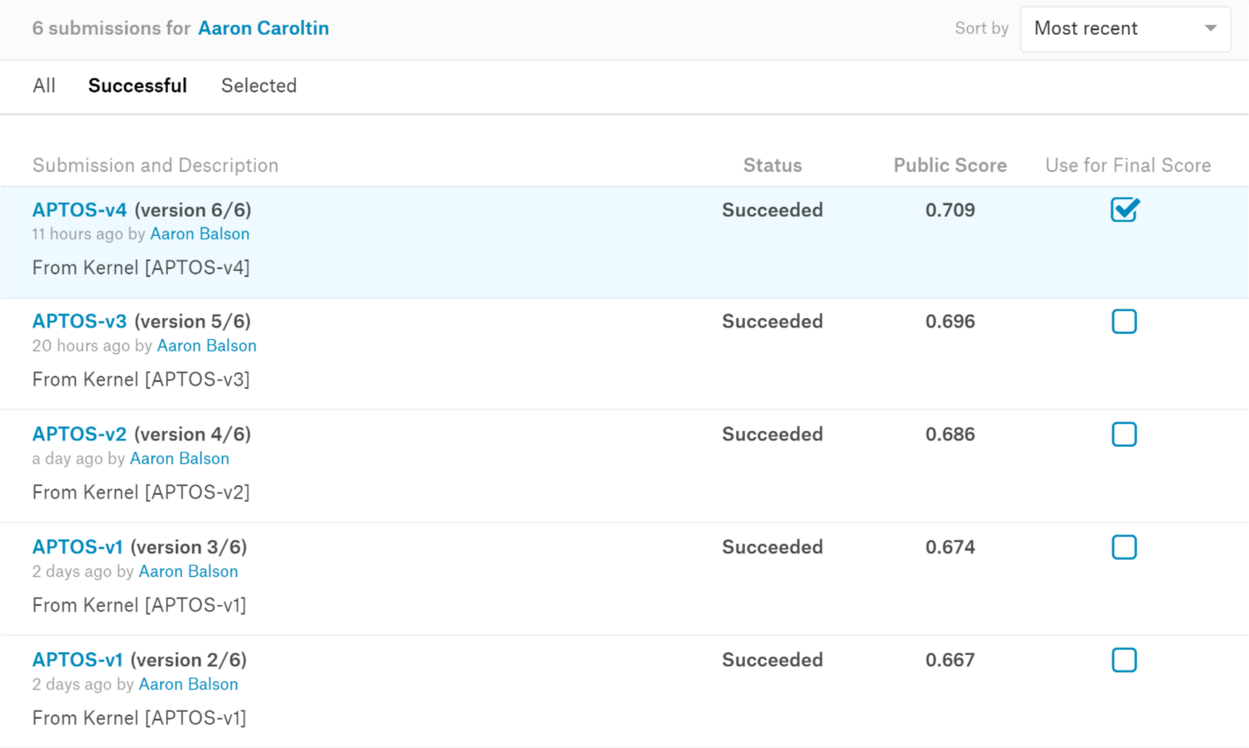
Even hyper parameter tuning or running k-fold cross validation was hard to work in this kernel because of workflow and test data format (without label, only features) so model.predict results are to be sent to server to get back the test score/metics. Many people might have solved it differently or changed their workflow, I will check around the platform. Benchmark was already set to beat 68.25% and iteration by iteration I had to do it. At times kernel would just wipe and had to restart all over. Big lesson learnt. Choose a workflow based on the dataset!

Writing project report was very intensive and I had to write up in middle of iterations. Certainly, the final model and solution fit my expectation for the problem and CNN projects are all about re-use I am sure I can apply 50% of my learnings here to any other CNN project.

Improvement

1. Image classification problems are 50% spend on cleaning and pre-processing the input sample with adequate filters to address common problems like angle of shots, lighting effect, blur so on. I wish to take up Ben Graham preprocessing (Reducing lighting-condition effects - above) and few other filters to include in preprocessing tool kit.
2. Most of the samples have black flimsy area around edges which is unproductive. Except the central eye area, these edges can be clipped away so I will look into "cropping uninformative area"
3. There can be improvement done using other transfer learning algorithms when time is not a constraint (we selected densenet121 mostly for its computational efficiency running in public notebooks)

4. We balanced the training set but we can even go ahead and balance the validation set as well.
5. Certainly, the next iteration can beat our current solution simply by doing (5) above.
6. Following is my Kaggle submission status. I wish to improve on that, beyond this submission while time permits.



6 submissions for Aaron Caroltin

Submission and Description	Status	Public Score	Use for Final Score
APTOS-v4 (version 6/6) 11 hours ago by Aaron Balson From Kernel [APTOS-v4]	Succeeded	0.709	☑
APTOS-v3 (version 5/6) 20 hours ago by Aaron Balson From Kernel [APTOS-v3]	Succeeded	0.696	□
APTOS-v2 (version 4/6) a day ago by Aaron Balson From Kernel [APTOS-v2]	Succeeded	0.686	□
APTOS-v1 (version 3/6) 2 days ago by Aaron Balson From Kernel [APTOS-v1]	Succeeded	0.674	□
APTOS-v1 (version 2/6) 2 days ago by Aaron Balson From Kernel [APTOS-v1]	Succeeded	0.667	□

[1.1]https://www.sciencedaily.com/releases/2019/03/190318141135.htm

[1.2]https://www.sciencedirect.com/science/article/abs/pii/S1871402114000277

[1.3]https://en.wikipedia.org/wiki/Cohen%27s_kappa#targetText=Cohen's%20kappa,for%20qualitative%20(categorical)%20items.

[2.1]https://www.researchgate.net/publication/42609795_A_learning_method_for_the_class_imbalance_problem_with_medical_data_sets

https://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6

[2.2]https://pdfs.semanticscholar.org/54b9/694c967105b824f4ccb40e991e22a8785fbe.pdf

[2.3]https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

[2.4]https://www.reddit.com/r/MachineLearning/comments/67fds7/d_how_does_densenet_compare_to_resnet_and/

[2.5]https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/smote#targetText=SMOTE%20stands%20for%20Synthetic%20Minority,that%20you%20supply%20as%20input.

[3.1] https://medium.com/@smallfishbigsea/densenet-2b0889854a92#targetText=Densenet%20contains%20a%20feature%20layer,layers%20between%20adjacent%20dense%20blocks.&targetText=The%20depth%20of%20a%20dense%20layer's%20output%20is%20'growth_rate'.