Final Project Report - ENME808E

iWildCam 2019 - FGVC6
Categorize Animals in the wild

Author Name
Koyal Bhartia

Implementation of Multi-class Classification on Image data

Date : 16/05/2019

Contents

List of Figures

1 Introduction

Wild Cams or Camera traps automatically collects large quantities of image data. This data is used to monitor biodiversity and population density of animal species. One challenge that arises here is the classification of the image into it's different species. Another interesting problem here is to classify species in a "new" region that **may not have seen** in previous training data..!!

2 Problem Statement

In this challenge the training data and test data are from different regions, namely The American Southwest and the American Northwest. The species seen in each region overlap, but are not identical, and the challenge is to classify the test species correctly. This is an FGVCx competition as part of the FGVC6 workshop at CVPR 2019, and is sponsored by Microsoft AI for Earth. Challenge link:
https://www.kaggle.com/c/iwildcam-2019-fgvc6/overview

Following are the given data in the project:

- **Train.csv** - Image ID, CategoryID, Datecaptured, Location etc.
- **Test.csv** - Image ID, Datecaptured, Location etc.
- **Trainimages.zip** - 1,96,157 training images from 138 different locations in Southern California
- **Testimages.zip** - 1,53,730 testing images from 100 locations in Idaho.

Submission: Corresponding CategoryID for the testing images **Evaluation:** F1 Score

3 The given Training and Testing Data

The training set contains 196,157 images from 138 different locations in Southern California; whereas The test set contains 153,730 images from 100 locations in Idaho. The competition task is to label each image with one of the given label ids: such as: deer, empty, rabbit, bison etc.

Each training image has at least one associated annotation, containing a category id that maps the annotation to it's corresponding category label. These annotations are stored in the JSON format. Data challenges include: Illumination, Motion blur, Small ROI, Perspective, Weather etc.

3.1 A peak into the animal classes in the data

Following is a bried representation of the classes in the traning data that is provided.

```
dictionary = {0:'empty',
              1:'deer',
              2:'moose',
              3:'squirrel',
              4:'rodent',
              5:'small_mammal',
              6:'elk',
              7:'pronghorn_antelope',
              8:'rabbit',
              9:'bighorn_sheep',
              10:'fox',
              11:'coyote',
              12:'black_bear',
              13:'raccoon',
              14:'skunk',
              15:'wolf',
              16:'bobcat',
              17:'cat',
              18:'dog',
              19:'opossum',
              20:'bison',
              21:'mountrain_goat',
              22:'mountain_lion',
             }
dictionary_reverse = dict((i,j) for j,i in dictionary.items())

train_ids = set(train_csv.category_id.values)
print("There are the following classes: {} in train".format(train_ids))
```

```
There are the following classes: {0, 1, 3, 4, 8, 10, 11, 13, 14, 16, 17, 18, 19, 22} in train
```

```
train_df['category_id'].value_counts()
```

```
0     131457
19     14106
13      8623
11      7209
8       6938
1       6102
16      5975
17      4759
3       3398
18      3035
4       2210
14      1361
10      1093
22        33
Name: category_id, dtype: int64
```

Figure 1: Representation of classes of animals in the data

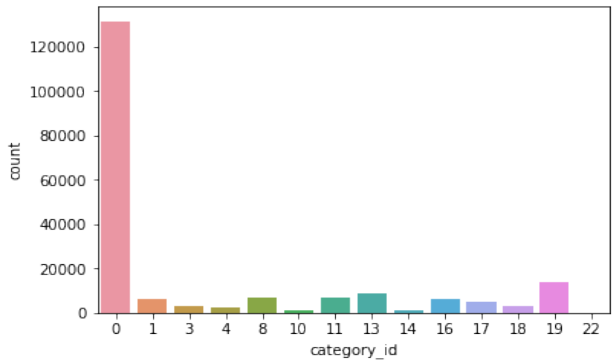


Figure 2: Spread of the classes of animals in traning set

3.2 Sample data

Following is a sample of the training images for the calssses bobcat, cat coyote and deer.

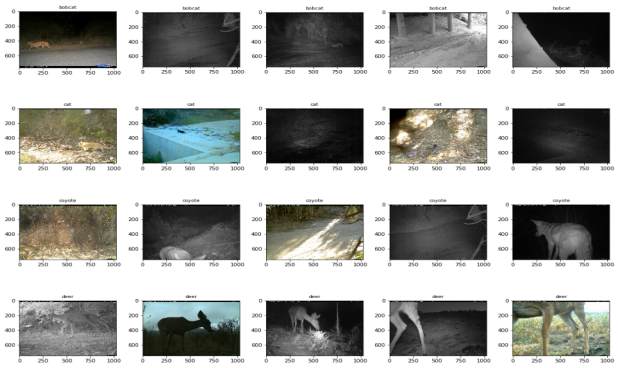


Figure 3: Sample set of traning image

On similar lines is the sample of unlabeled testing data:

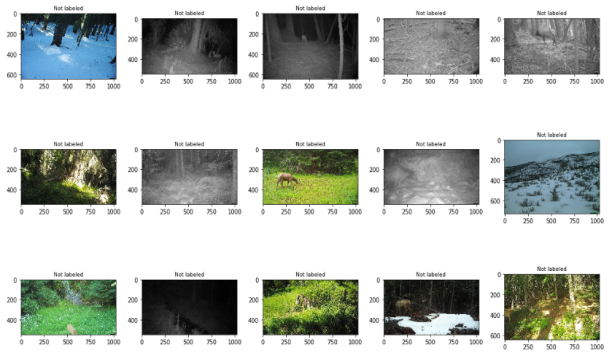


Figure 4: Sample set of testing image

3.3 Resizing image data

As seen the images are quiet big i.e 1024*747 in size. For faster computation, training and prediction of the model ,the images have been resized to the size 32*32, using the **cv2 function:** cv2.resize(image, (32,)*2).astype('uint8').

3.4 Normalizing of the data

Once the images are resized for both the traing and the testing set of images, it is normlaized in the range 0-¿1 with division by 255, as RGB values range betwee 0 and 255.

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255.
x_test /= 255.
```

Figure 5: Normalization of resized data

The category ID of the training images is then coverted into a form which can be used by the model for training purpose. This is done as follows:

```
target_dummies = pd.get_dummies(test_new['category_id'])
train_label = target_dummies.columns.values
y_test= target_dummies.values
```

Figure 6: Fetching Category ID

This gives ytrain in the form of 0's and 1's for aall the classses for all the images i.e the value of ytrain is 1 for the class of the image, and the rest of the 13 columns is 0.

Once the data is ready, it is stored in a separate folder, and loaded for reuse. Thus this is a one time effort and process.

Note: For the sake of submission I did the above data manipulation for the entire set of images. However for faster analysis of the data, I took 50% of the data for training and 10% for testing, and carried the pre-processing only on this dataset.

4 Model 1: Covolutional Neural Network Model

4.1 Why CNN?

In a CNN model, the image is passed through a series of convolutional, nonlinear, pooling and fully connected layers(dense) which then generates the output.

- **Conv2D:** This 2D convolution layer is typically used for spatial convolution over images. Among the several arguments that keras.layers.Conv2D() inputs, the arguments that I took care of are:
  - **Filters:**Integer that monitors the dimensionality of the output space.
  - **Kernelsize:**Tuple of 2 integers, specifying the height and width of the 2D convolution window. The filter multiplies its values by the original pixel values which are then summed up. This is analogous to identifying boundaries and simple colours on the image.
  - **padding:** Convolution results in a image of smaller size which needs padding. Padding "same" has been chosen in my model as it results in the input and output having the same size.
- **Activation:** ReLU - (activation function max(0,x).This function brings nonlinear property.
- **Pooling Layer:** MaxPool2D - This acts as a downsampling filter. It looks at 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. More the pooling dimension is high, more is the downsampling. With a combination of convolutional and pooling layers, CNN combines local features and learn more global features of the image.
- **Dropout:** This is a regularization method, where a proportion of nodes in the layer are randomly ignored for each training sample. This forces the network to learn features in a distributed way. This improves generalization and reduces the overfitting.
- **Flatten:** This layer is used to convert the final feature maps into a one single 1D vector. It combines all the found local features of the previous convolutional layers.
- **Dense:**In the last layer,(Dense(10,activation="softmax")), outputs distribution of probability of each class of animal.

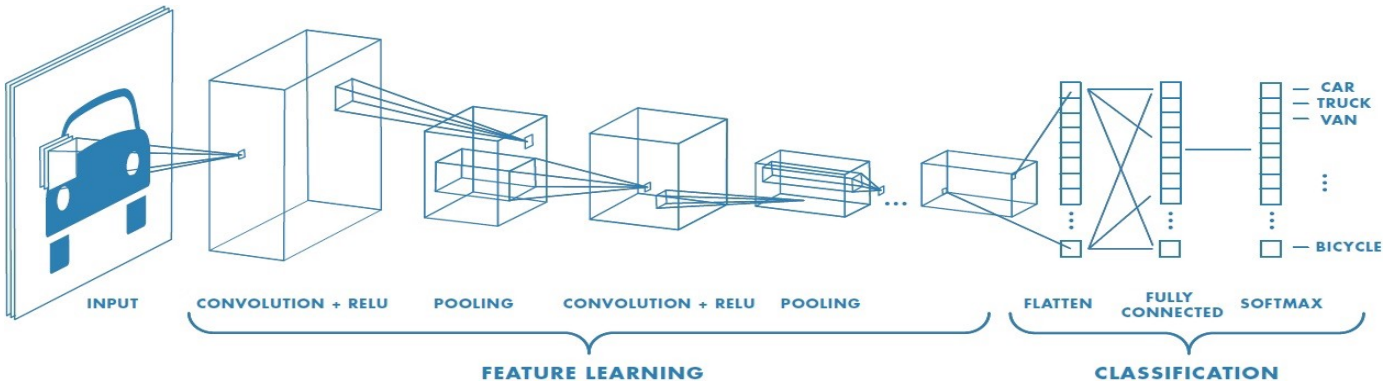


Figure 7: Overview of CNN used for Image Classification

4.2 Implementation of the Model

Following is the summary of the model that I implemented after several tweaks on the parameters as shown later:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               590080
_________________________________________________________________
activation_4 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 14)                3598
_________________________________________________________________
activation_5 (Activation)    (None, 14)                0
=================================================================
Total params: 659,246
Trainable params: 659,246
Non-trainable params: 0
```

Figure 8: Summary of implemented model

Following are the parameters that I finally chose to give me the best results:

Sl.No	Layer/Activity	Argument
1	Two Conv2D	32 filters of kernel size=(3,3)
-	-	padding='same'
-	-	inputshape=xtrain.shape[1:]
-	-	activation='ReLU'
2	MaxPooling2D	poolsize=(2,2)
3	Dropout	0.25
4	Two Conv2D	64 filters of kernel size=(3,3)
-	-	padding='same'
-	-	inputshape=xtrain.shape[1:]
-	-	activation='ReLU'
5	MaxPooling2D	poolsize=(2,2)
6	Dropout	0.25
7	Flatten	-
8	Dense	256
9	Activation	ReLU
10	Dropout	0.5
11	Dense	14(No. of classes)
12	Activation	Softmax
13	Compile	loss='categoricalcrossentropy'
-	-	optimizer='adam'
-	-	metrics=['accuracy']

Here the loss function measure how poorly the model performs on images with known labels. It is the error rate between the observed labels and the predicted ones. "categoricalcrossentropy" has been because there are more than 2 classes. I trained the entire input training set with the model above, and then used model.predict() to generate the testing Y predicted data or the Category ID for each case of the testing images. This was done using softmax, hence it gave the value in the form of probabilties. The class with the max probability was taken as the class or the Y data of the test.

This model was run for a **batchsize of 64, epochs =7, and validation split of** 10%. The F1 metrics, accuracy and loss is calculated after each epoch using the Callback method.

```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=7,
    callbacks=[checkpoint, f1_metrics],
    validation_split=0.1
)
```

Figure 9: Model.fit parameters

```
Train on 176669 samples, validate on 19630 samples
Epoch 1/7
176669/176669 [==============================] - 601s 3ms/step - loss: 0.7027 - acc: 0.7813 - v
al_loss: 0.4972 - val_acc: 0.8365

Epoch 00001: val_acc improved from -inf to 0.83653, saving model to model.h5


/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
arning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
arning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)


val_f1: 0.4982 — val_precision: 0.6123 — val_recall: 0.4771
Epoch 2/7
176669/176669 [==============================] - 599s 3ms/step - loss: 0.5096 - acc: 0.8348 - v
al_loss: 0.4090 - val_acc: 0.8646

Epoch 00002: val_acc improved from 0.83653 to 0.86460, saving model to model.h5
val_f1: 0.6389 — val_precision: 0.7239 — val_recall: 0.6056
Epoch 3/7
 28160/176669 [===>..........................] - ETA: 8:12 - loss: 0.4603 - acc: 0.8503
```

Figure 10: CNN Model getting trained for 7 epochs and 0.1 validation

This gave me the following results for Loss, Accuracy and F1 Metrics:

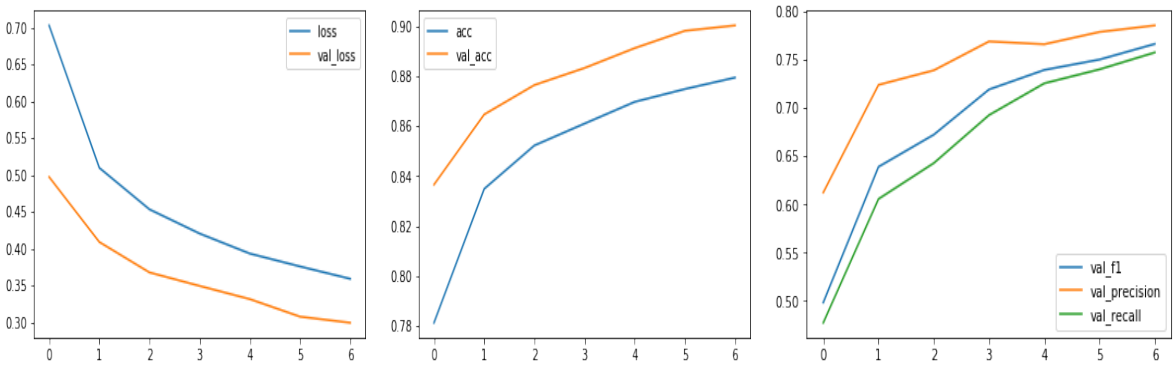


Figure 11: CNN Model Learning Curves

The metric function "accuracy" is used is to evaluate the performance the model. This metric function is similar to the loss function, except that the results from the metric evaluation are not used when training the model (only for evaluation)

Following are the values of the accuracy and loss obtained for the **7 epochs.**

```
root: {} 4 items
  val_loss: [] 7 items
      0: 0.4972127668054749
      1: 0.4089790015395539
      2: 0.36768504905567084
      3: 0.34938281766322576
      4: 0.331480304249318
      5: 0.3076944368169797
      6: 0.2994698095810517
  val_acc: [] 7 items
      0: 0.8365257258628985
      1: 0.8645950076899478
      2: 0.8763627101739815
      3: 0.8831889964704664
      4: 0.8911360163380161
      5: 0.8980641875045978
      6: 0.900203769788776
  loss: [] 7 items
      0: 0.702733633048842
      1: 0.509595532493687
      2: 0.4532328912467548
      3: 0.42040614971693163
      4: 0.3932595863173406
      5: 0.37580373094266617
      6: 0.35904860331238375
  acc: [] 7 items
      0: 0.7812802472403306
      1: 0.834826709839619
      2: 0.8521925182096619
      3: 0.8609490063352281
      4: 0.8696432311278108
      5: 0.8747658049782473
      6: 0.8793619706862298
```


	Id	Predicted
0	b005e5b2-2c0b-11e9-bcad-06f10d5896c4	0
1	f2347cfe-2c11-11e9-bcad-06f10d5896c4	0
2	27cf8d26-2c0e-11e9-bcad-06f10d5896c4	0
3	f82f52c7-2c1d-11e9-bcad-06f10d5896c4	0
4	e133f50d-2c1c-11e9-bcad-06f10d5896c4	0
5	8784ac8a-2c1f-11e9-bcad-06f10d5896c4	0
6	4808497d-2c1a-11e9-bcad-06f10d5896c4	0
7	1f21b58a-2c09-11e9-bcad-06f10d5896c4	6
8	69fea4d2-2c04-11e9-bcad-06f10d5896c4	0
9	f8ff7f72-2c04-11e9-bcad-06f10d5896c4	6
10	4e95d1fe-2c08-11e9-bcad-06f10d5896c4	0
11	cdd236f2-2c0b-11e9-bcad-06f10d5896c4	0
12	be5015e9-2c03-11e9-bcad-06f10d5896c4	0
13	2350ed7d-2bf9-11e9-bcad-06f10d5896c4	0
14	83ea235a-2c0b-11e9-bcad-06f10d5896c4	0

Figure 12: Sample of Kaggle Submission - CNN Model

I submitted the above model on Kaggle. The F1 score that I obtained on the given test images as calcualted by Kaggle was **0.3.** This gave me a rank of 105 among around 200 teams.

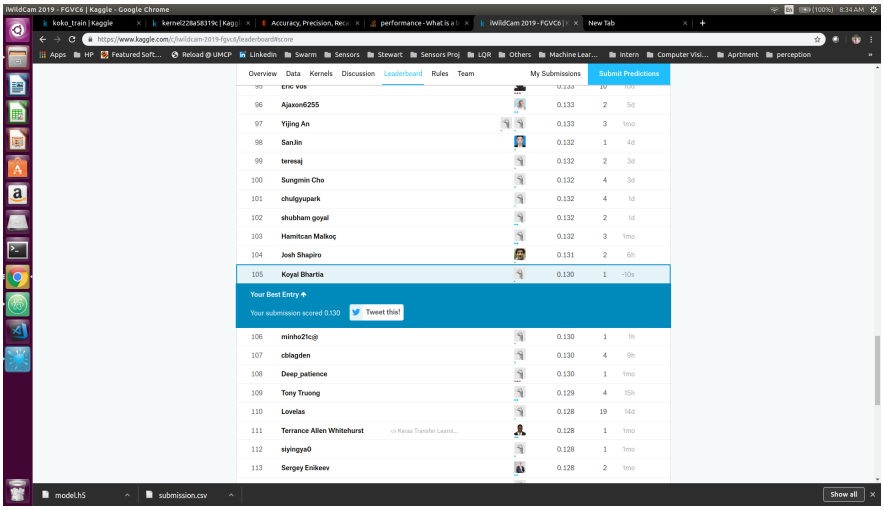


Figure 13: Rank with the CNN model submission

4.3 How I reached the above model

The above model was the final optimzed model. This involved a lot of trials and errors by varying several parameters, optimizers, batchsize, no of layers and validation split.

The above model took a very long time to run on the complete dataset. Hence for the purpose of calculation of in sample and out of sample error, I took a traning set of 50% of the images, and test split of 10%. Following are the different cases that followed on the reduced dataset:

4.4 Different test cases to arrive at the model

4.5 Case 1: CNN Model, BS=32, Epoch=3, VS=0.05

Following are the results obtained.

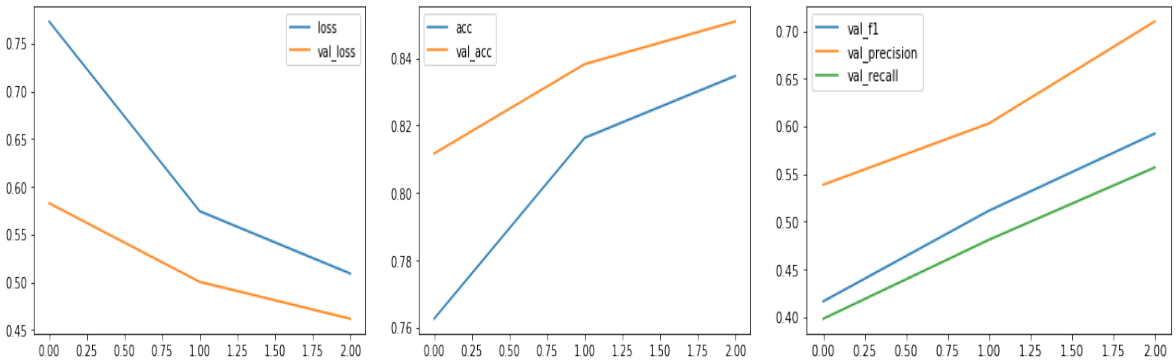


Figure 14: Case 1 Learning Curves

4.6 Case 2: Case 1 with additional Conv layer(128) and Dense-512

This was an extension of the Case 1, here I added an additional **2 Conv2D layers of 128 filters, along with the ReLU activation layer**. Basically the addition of this:

```
classifier.add(Conv2D(128, (3, 3), padding='same'))
classifier.add(Activation('relu'))
classifier.add(Conv2D(128, (3, 3)))
classifier.add(Activation('relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(0.25))
```

Figure 15: Update in Case 2

This gave the following results:

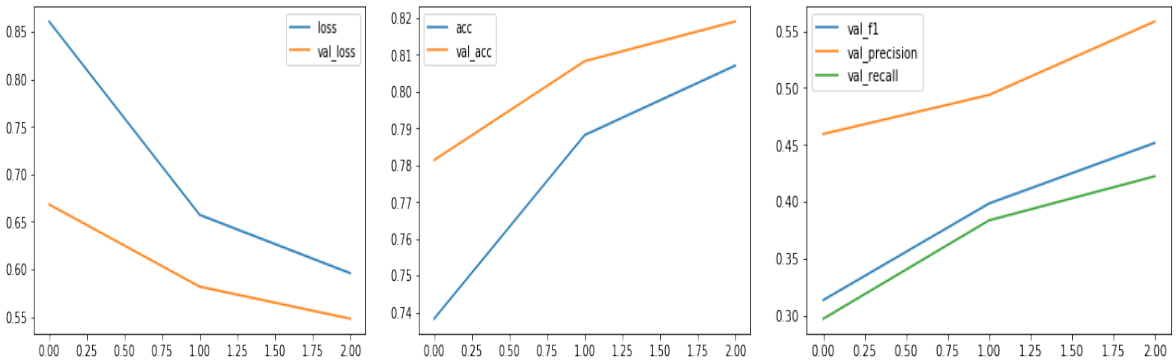


Figure 16: Case 2 Learning Curves

4.7 Case 3: Case 1 with SGD optimizer and VS=0.1

Here I tested the effect of the SGD optimizer on the model. Following was the optimizer used: **sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)** This gave the following results:

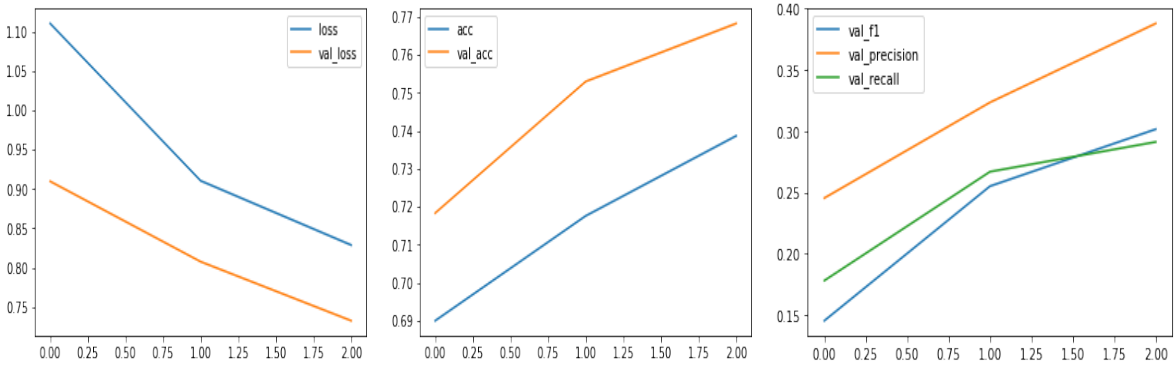


Figure 17: Case 3 Learning Curves

Hoeffding/Genralization bound was calculated for all the cases above using the equation: This has been calculated for a confidence of 95%. This gives δ=1-confidence=0.05

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}} \tag{1}$$

Here N is the no of sample points. In my case I have taken 50% of training data, hence giving N = 98150.

Case	In Sample	Out Sample
1	**Accuracy**: 0.6779317371370351	0.6772287315333673
Hoeffding Bound:	**F1 score:** 0.6728471667618514	0.6720626736258773
0.6685121869234605	**Recall:** 0.6779317371370351	0.6772287315333673
-	**Precision:** 0.6693551815655341	0.6726930146435672
2	Accuracy: 0.6707080998471727	0.6726439123790117
Hoeffding Bound:	**F1 score:** 0.6615995064791624	0.6640389543640219
0.6572645266407715	**Recall:** 0.6707080998471727	0.6726439123790117
-	**Precision:** 0.6640357797234191	0.6673461334350839
3	Accuracy: 0.6515944982170148	0.6539989811512991
Hoeffding Bound:	**F1 score:** 0.6311046249991121	0.6322650330210252
0.6267696451607212	**Recall:**0.6515944982170148	0.6539989811512991
-	**Precision:** 0.619951165467038	0.6209099305709768

Additionl Case: RMSProp Optimizer Though the above analysis had more or less similar results for the F1 Score, the result was very different when I used the RMSProp optimizer, as follows: **keras.optimizers.rmsprop(lr=0.0004, decay=1e-6)**

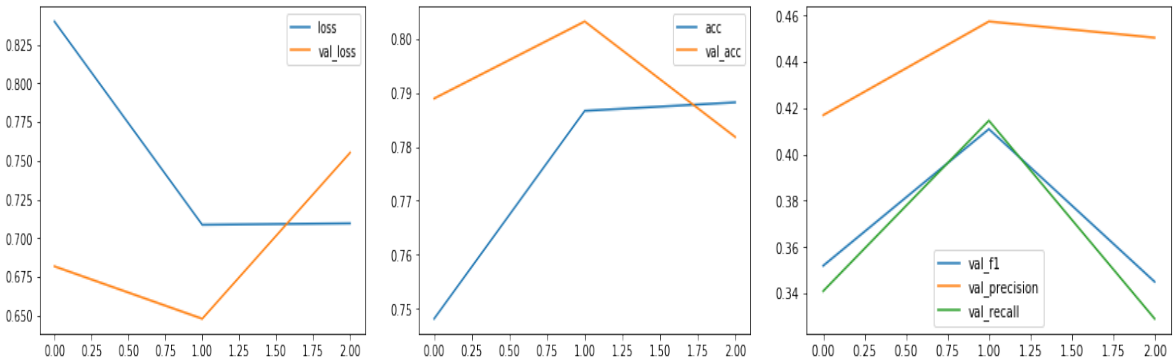


Figure 18: Additional Case Learning Curves

Optimizers are one of the most important function is the optimizer which iteratively improves parameters like the filters kernel values, weights and bias of neurons, in order to minimise the loss. As can be seen different optimizers have given very varied results.

5 Model 2: DenseNet121 Model

The next model I used is the densenet model. DenseNet has several advantages over the general CNN models. This is useful when we want to keep increasing the depth of deep convolutional networks. When CNN's go deep, the path for information from the input layer until the output layer (and for the gradient in the opposite direction) becomes so big, that they can get vanished before reaching the other side.

DenseNets simplify the connectivity pattern between layers. Here each layer has direct access to the gradients from the loss function and the original input image.

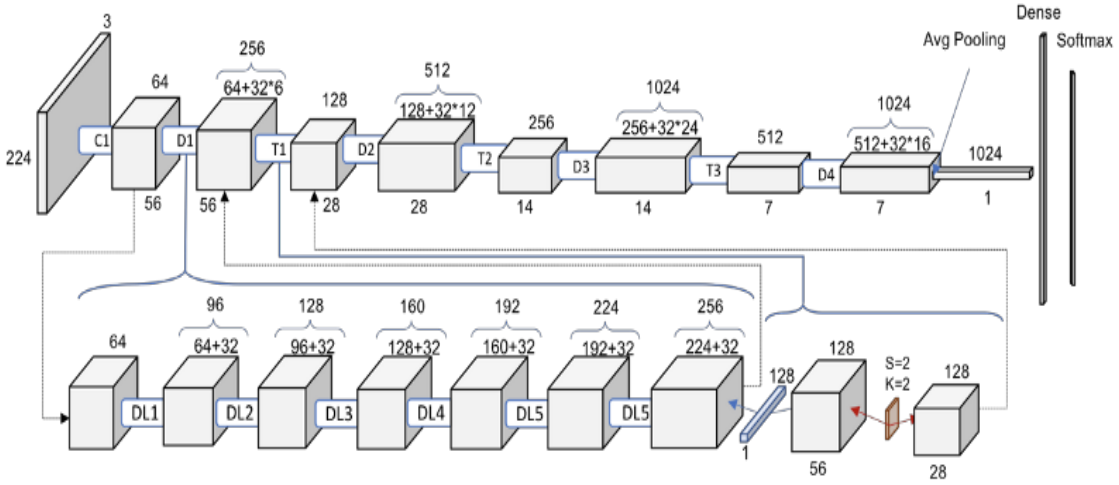
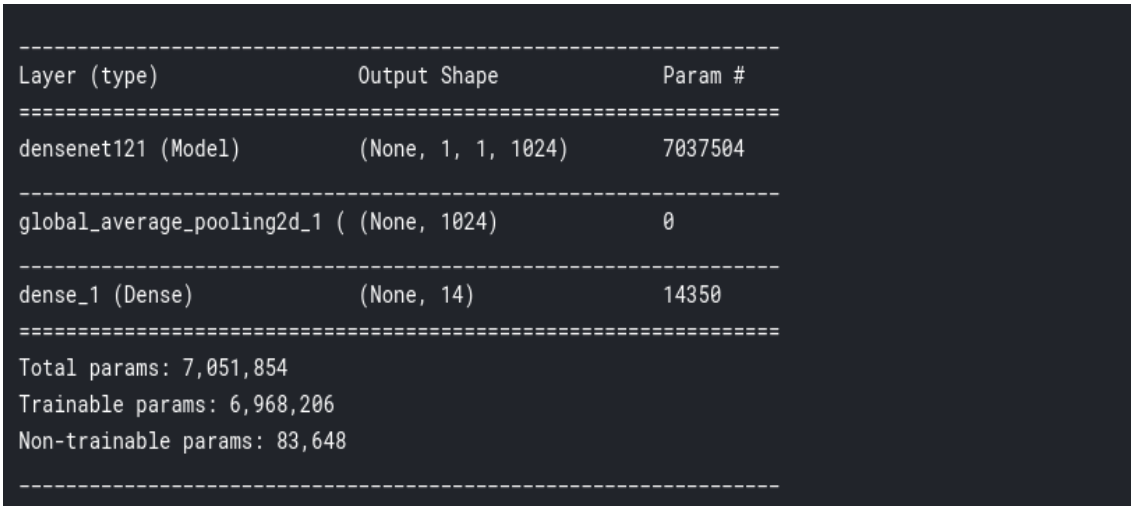


Figure 19: The DenseNet Architecture

I have used the BC model, which is an extension of the B model. It is useful for the cases when we want to reduce the number of output feature maps.

Following is the summary of the model used.

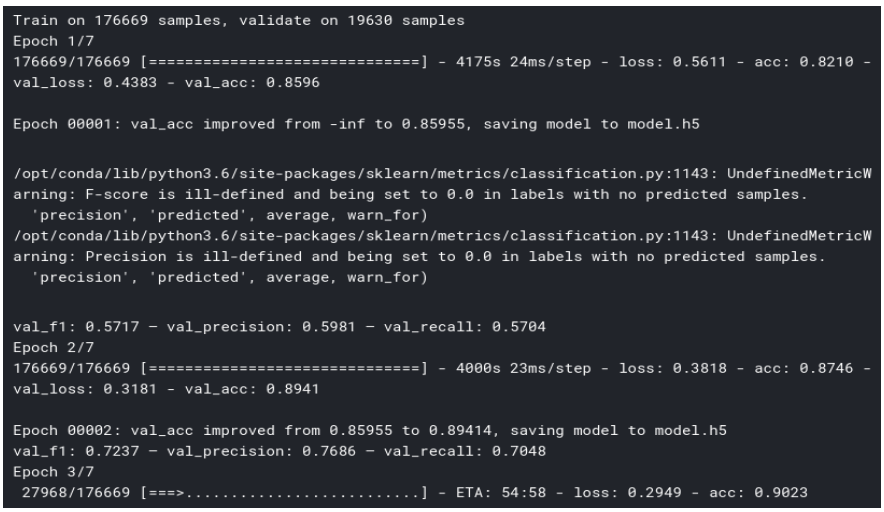


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 1, 1, 1024)        7037504
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                14350
=================================================================
Total params: 7,051,854
Trainable params: 6,968,206
Non-trainable params: 83,648
_________________________________________________________________
```

Figure 20: Summary of the DenseNet model

5.1 Case 1

I ran the model on the entire dataset.This was along with a **batchsize of 64, epochs as 7 and a validation split of 0.1.**



```
Train on 176669 samples, validate on 19630 samples
Epoch 1/7
176669/176669 [==============================] - 4175s 24ms/step - loss: 0.5611 - acc: 0.8210 -
val_loss: 0.4383 - val_acc: 0.8596

Epoch 00001: val_acc improved from -inf to 0.85955, saving model to model.h5


/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
arning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
arning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)


val_f1: 0.5717 — val_precision: 0.5981 — val_recall: 0.5704
Epoch 2/7
176669/176669 [==============================] - 4000s 23ms/step - loss: 0.3818 - acc: 0.8746 -
val_loss: 0.3181 - val_acc: 0.8941

Epoch 00002: val_acc improved from 0.85955 to 0.89414, saving model to model.h5
val_f1: 0.7237 — val_precision: 0.7686 — val_recall: 0.7048
Epoch 3/7
 27968/176669 [===>..........................] - ETA: 54:58 - loss: 0.2949 - acc: 0.9023
```

Figure 21: DenseNet epoch run

Following is the result obatined with the callback on the F1 metric after every epoch.

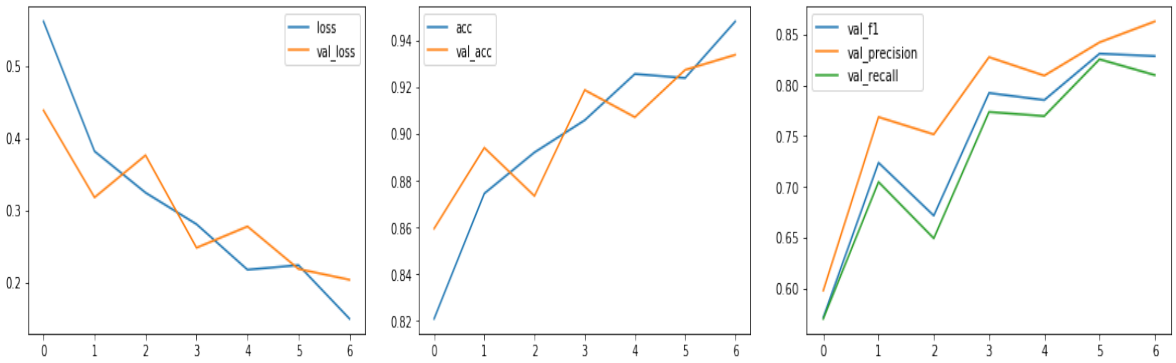


Figure 22: DenseNet121 Learning Curves

I submitted the above model on Kaggle. This gave me a submission F1 score of **0.139**. This improved my rank from 105 in the previous model to 78 which reduced to 79 as shown in the scrrenshot.

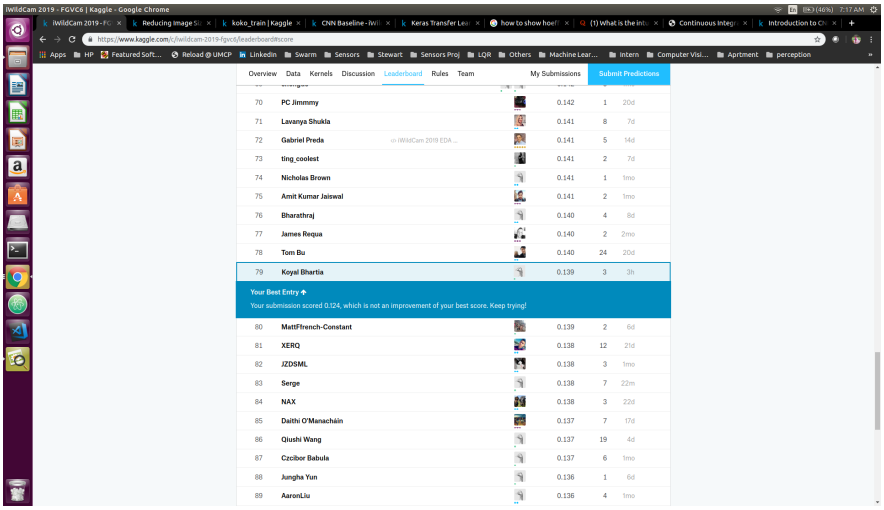


Figure 23: Kaggle Rank with DenseNet Model

As this model takes several hours(more than 9) to run, I tested the same on 50% of the training data as done for the above model.

Following are the results obtained: Accuracy: 0.6838003056546103 F1 score: 0.6832325553012545 Recall: 0.6838003056546103 Precision: 0.683871123261332

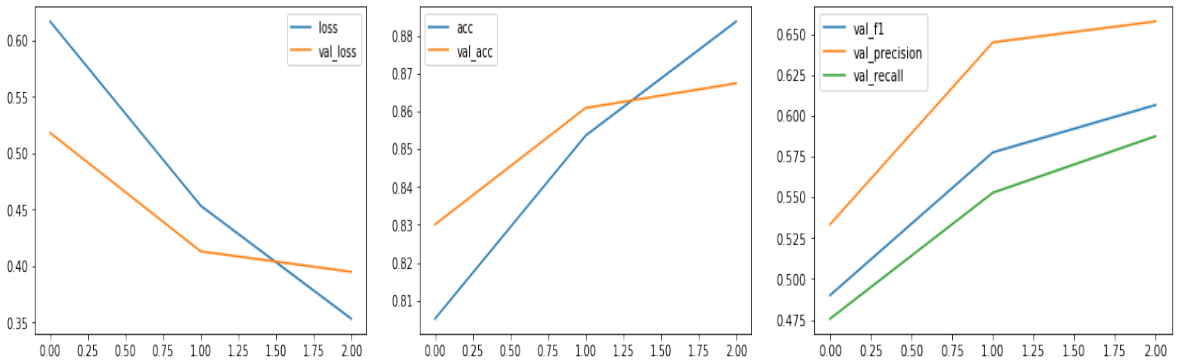


Figure 24: Case1: DenseNet121 Learning Curves

The above was implemented for a batchsize of 32, epochs=3 and validation split as 0.05.

5.2 Case 2

I then checked the above denseNet model for a **batchsize of 64, epochs=4 and a validation split of 0.75** to check the effect of these parameters on the output.

The results were surprising:

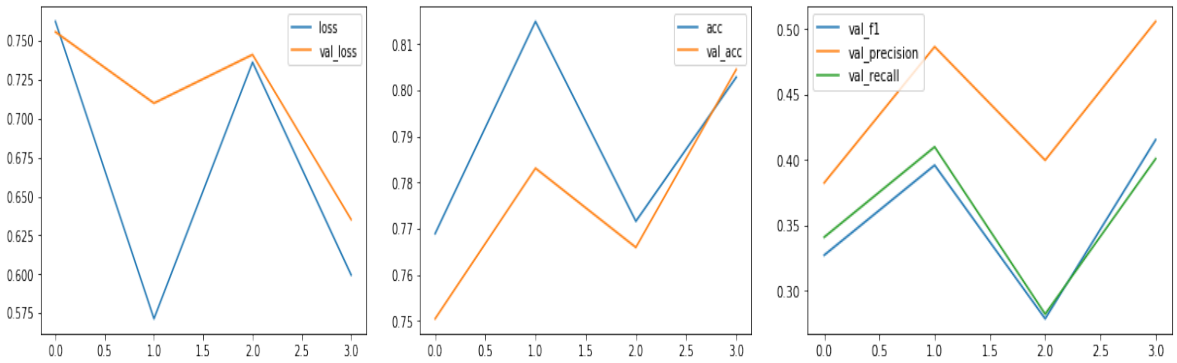


Figure 25: Case 2: DenseNet121 Learning Curves

Following are the accuracy, loss and F1 metrics discussed above. .

Case	In Sample	Out Sample
1	**Accuracy**: 0.6838003056546103	0.6812022414671421
Hoeffding Bound:	**F1 score:** 0.6832325553012545	0.6807824587940099
0.6788975754628636	**Recall**: 0.6838003056546103	0.6812022414671421
-	**Precision**: 0.683871123261332	0.6829454036877467
2	**Accuracy**: 0.6716658176260826	0.6728476821192053
Hoeffding Bound:	**F1 score:** 0.6586204203990005	0.6589630623588139
0.6542854405606096	**Recall**: 0.6716658176260826	0.6728476821192053
-	**Precision**: 0.6488682115809022	0.6484037458626973

6 Appendix: The F1 Score

F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

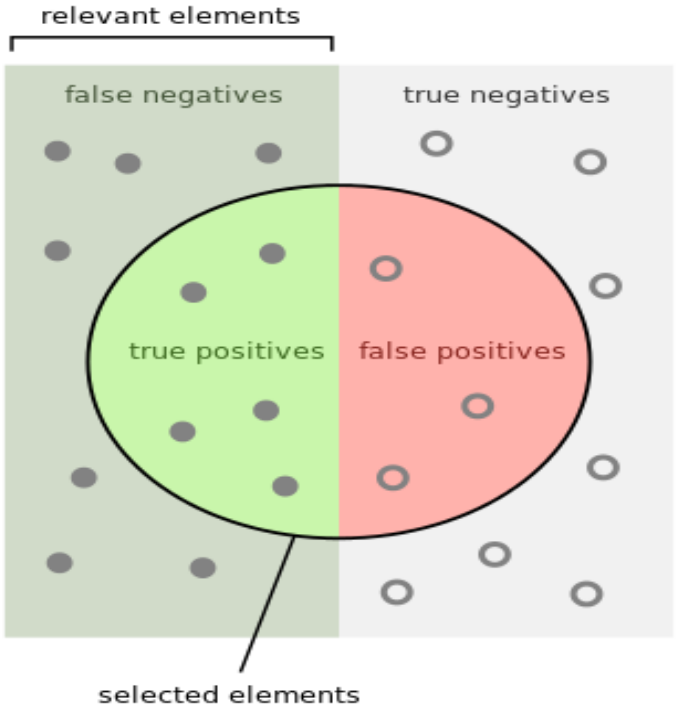


Figure 26: Representation of True+ve and True-ve

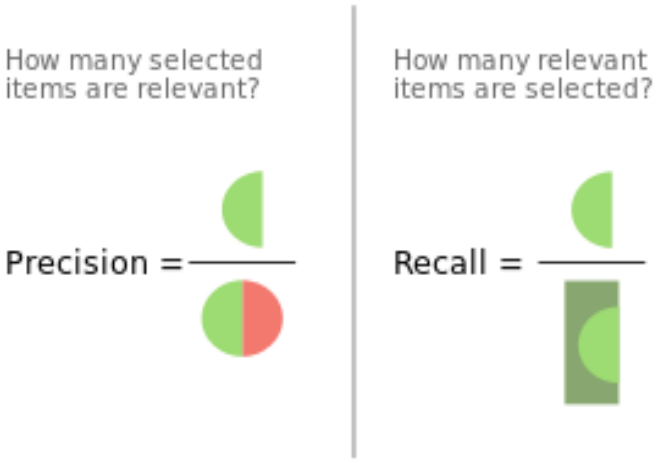


Figure 27: Representation of Precision and Recall

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Figure 28: F1 Score

7 Conclusion

This project on multiclass image classification gave me a very deep insight on the use of training of machine learning models using images. A beginner at Neural Networks, a dig into the understanding, working and the parameters involved in the training and evaluation of the CNN involved the learning of several new and interesting terms.

The use of "softmax" activation function in the output layer and the the efficient use of the Adam gradient descent optimization algorithm with the logarithmic loss function in Keras was very interesting and fun to play around with.

Following is my present rank in the Kaggle Competition.

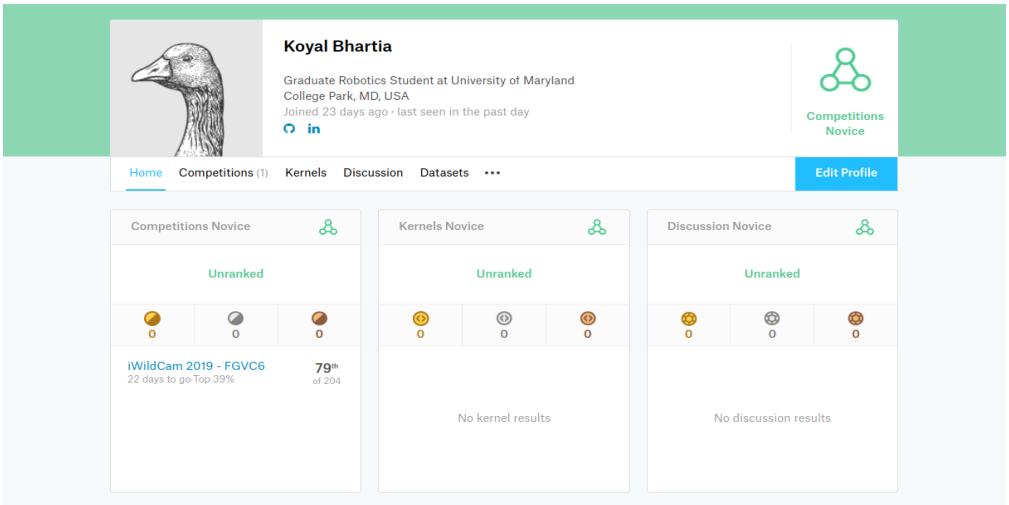


Figure 29: Kaggle Rank

8 Acknowledgement

I would like to thank Professor Nikhil Chopra for his constant support and guidance. He helped us in every step of the project from the selection to the implementation stage. I am grateful to have had his encouragement. I would also like to thank my batch mates as the incites gained on improving the accuracy and performance of the project was very helpful.

Note: The classification and Confusion matrix for each case can be found in the codes provided.