**SCS -3253 Machine Learning Final Project**

**Session – Winter 2019**

**Project team:** Abdallah Alhamdani

Sonali Gupta

**Project Description:** Detect diabetic retinopathy to stop blindness before it's too late

**Data Source:** APTOS 2019 Blindness Detection – Kaggle Competition

[**Introduction**](#_f765xj441ta5) **3**

[What is diabetic retinopathy?](#_ju6053hlkq7n) 3

[Real World Problem.](#_383xhov11v9j) 3

[Mapping Real World Problem to a Machine Learning/Deep Learning Problem.](#_7zoqcb1c46y) 3

[**Data loading and exploration**](#_cs2ysclfhqmg) **4**

[Here, we’re fixing the random seed for reproducibility of the same results and also defining the Image size as 224.](#_ft0q9bv1evdh) 4

[load images and csv files](#_idwebpq7posm) 4

[splitting training data into train and validation](#_hxnn61crfyfn) 4

[exploring different classes/ target variables](#_9yqwj1ficagx) 5

[**Data prep and image processing**](#_w8lpbkjfmcqu) **6**

[data augmentation using image preprocessing - RGB, resize and enhance](#_rucs9o1exftb) 6

[convert classes from categorical into ordinal variables](#_faxyqkgwnlr7) 8

[**Model building 2019**](#_3mbb39szuj2l) **9**

[Baseline Model – self built architecture (2019 data)](#_t6e3zd7q83wd) 10

[DenseNet 121 – with data augmentation (2019 data)](#_66250jw269r) 11

[ResNet50 pretrained on 2015](#_3lqtyfkqav0m) 11

[EfficientNetB4](#_n2dfh6jqu98c) 12

[**Performance Metrics**](#_pkjpk0dxh3z0) **12**

[Confusion Metrics/Precision Recall](#_3ufk3bf5dvv0) 12

[Cohen Kappa](#_h628jhzew6lu) 12

[Exploratory Data Analysis and Feature Engineering](#_pk7vmjwr6gu4) 14

[From the above distributions we can see that the classes 0 and 2 are dominant classes which means the data is imbalanced. Train and validation sets have similar percentage of images in each class.](#_37t90o3yjjpo) 14

[Image Preprocessing](#_du5cdklpwtzo) 14

[Image Features Enhancement](#_7te8h8begw5j) 14

[We’re plotting a sample of images with and without gaussian-blur and the label above the image is Class\_label – Index – Image\_id.](#_a56u88p2ht40) 15

[Cropping dark edges from images](#_px2ne4wbjh07) 16

[Here we’re saving the pre-processed (enhanced and cropped) images (training, test and validation) into the numpy arrays and storing on the disk](#_lugvvop5v7qp) 16

## Introduction

#### What is diabetic retinopathy?

"Diabetic retinopathy is an eye condition that can cause vision loss and blindness in people who have diabetes. It affects blood vessels in the retina (the light-sensitive layer of tissue in the back of your eye).

If you have diabetes, it’s important for you to get a comprehensive dilated eye exam at least once a year. Diabetic retinopathy may not have any symptoms at first — but finding it early can help you take steps to protect your vision." -<https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/diabetic-retinopathy>

#### Real World Problem.

Currently trained technicians from Aravind Eye Hospital in India travel to rural areas where medical screening is difficult to conduct and capture images of retina, they then rely on highly trained doctors to review the images and provide diagnosis. The goal is to scale their efforts through technology; to gain the ability to automatically screen images for disease and provide information on how severe the condition may be.

#### Mapping Real World Problem to a Machine Learning/Deep Learning Problem.

We are provided with data of thousands of images of Retina and we also have the corresponding diagnosis report for that image which tells us the severity of diabetic retinopathy on a scale of 0-4.

This can be mapped to categorical classification problem, where our inputs will be images of retina and output will be an integer between 0-4 where '0' means 'No DR' and '4' means 'Proliferative DR'.

## Data loading and exploration

#### Here, we’re fixing the random seed for reproducibility of the same results and also defining the Image size as 224.

np.random.seed(seed) -  every time we call the NumPy’s other random function, the result will be the same

tf.set\_random\_seed(seed) to set the seed to produce same results everytime we call the TensorFlow random function.

And printing the contents of the drive where our data is saved.

### load images and csv files

* We’re loading our train and test data using pandas read\_csv into df\_train and df\_test.
* we are loading a csv which has filename in column 'id\_code' and the corresponding diagnosis given by doctor in column 'diagnosis'

### splitting training data into train and validation

Before we proceed any further is it important for us to stratify, split the given train set into train and cross validate so that we will have a set of unseen data to evaluate our models on.

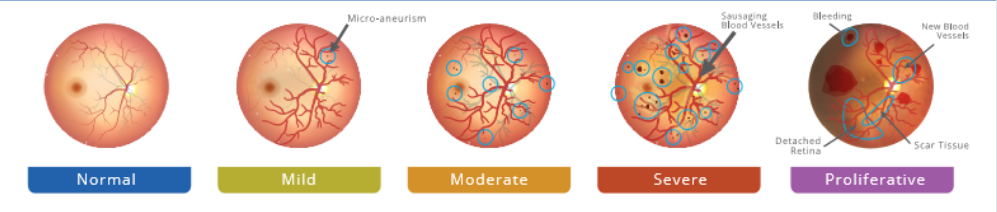
We’re creating two variables id\_code and diagnosis by assigning the columns id\_code and diagnosis from our training dataset, shuffling the data randomly. And then splitting the training dataset into train and validation using stratified sampling “test-size = 0.15” which means train: validation is 85:15 ratio on our labels “diagnosis” in order to split the data to ensure that the train and validation sets have approximately the same percentage of **samples** of each target class as the complete set.

print(train\_x.shape) shows the number of data points in train\_x dataset.

### exploring different classes/ target variables

* 1. **exploring different classes/ target variables**

#### 



We’re creating the plot\_distribution function to plot our data.

class\_distribution = labels.value\_counts().sort\_index() is counting number of datapoints in each label and sorting by labels in ascending by default.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

We’re plotting train and validation data distribution by label “diagnosis” which has 5 classes to show equal distribution of data in train and validation by labels and their proportion in percentage.

From the above distributions we can see that the classes 0 and 2 are dominant classes which means the data is imbalanced. Train and validation sets have similar percentages of images in each class.

## Data prep and image processing

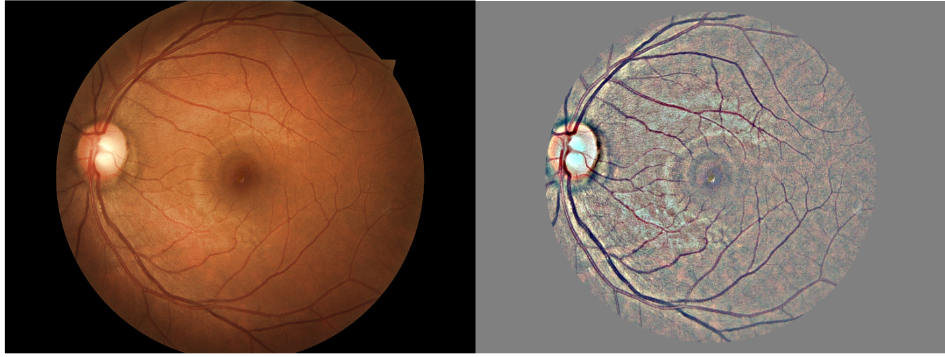
### data augmentation using image preprocessing - RGB, resize and enhance

The **Preprocessing\_images** function reads an image from hard disk converts it into RGB (by default it is read in BGR) resizes the image into 224X224 pixels and there is an option to enable Ben’s Preprocessing technique if enabled a Gaussian filter is applied which is a low pass filter as it helps removing noise the image and it is merged with the original image by giving different weights as it helps in enhancing the features we care about.

#### 

#### A close up of a logo Description automatically generated

This image shows cropped out the additions dark areas in the images as this information is not useful



The above image shows that the preprocessing has worked as the nerve,blood vessel features are visually more enhanced .

After processing of every image the corresponding numerical representation of the image is added to a numpy array, train, cv, and test sets have different arrays. These arrays are also stored in .npy format on disk so that they can just be loaded when required.

### convert classes from categorical into ordinal variables

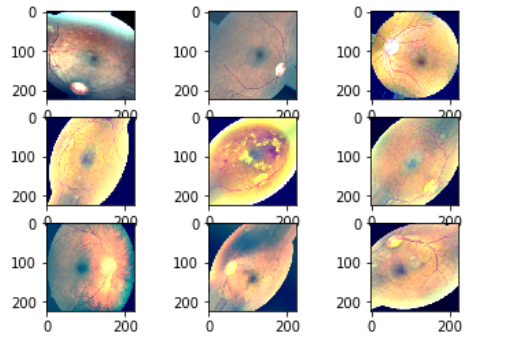
Instead of predicting a single label, we will change our target to be a multilabel problem; i.e., if the target is a certain class, then it encompasses all the classes before it. E.g. encoding a class 4 retinopathy would usually be [0, 0, 0, 1], but in our case we will predict [1, 1, 1, 1]. For more details, please check out [Lex's kernel](https://www.kaggle.com/lextoumbourou/blindness-detection-resnet34-ordinal-targets).

The given output variables are categorical and this means the problem is posed as a categorical classification problem.it is better to transform such problems into ordinal regression problems.

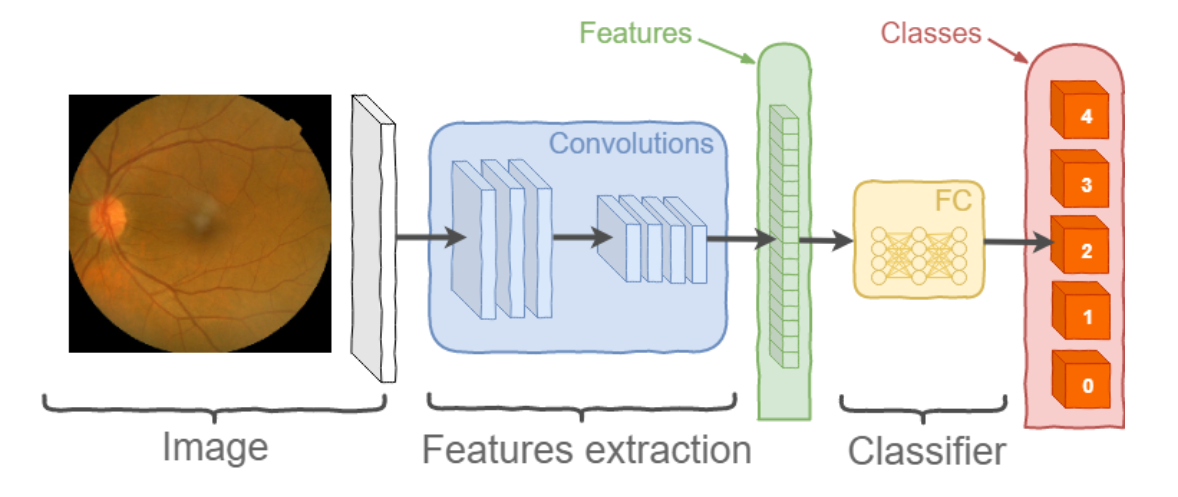
**ImageDataGenerator**

Keras provides the ImageDataGenerator class that defines the configuration for image data preparation and augmentation. This includes capabilities such as:

* Sample-wise standardization.
* Feature-wise standardization.
* ZCA whitening.
* Random rotation, shifts, shear and flips.
* Dimension reordering.
* Save augmented images to disk.

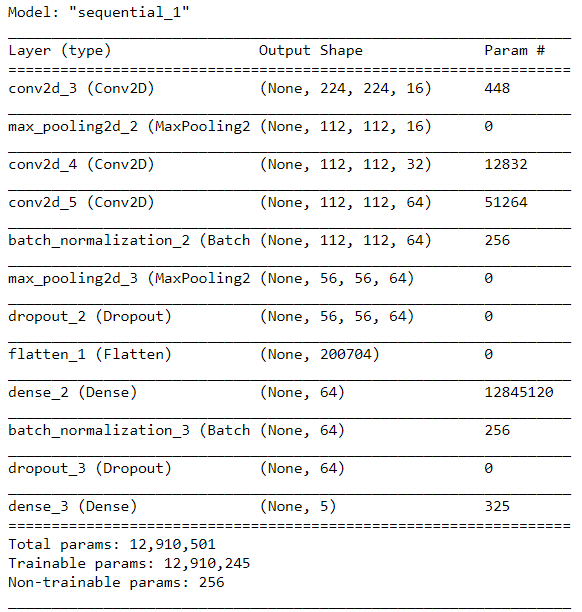


## Model building 2019



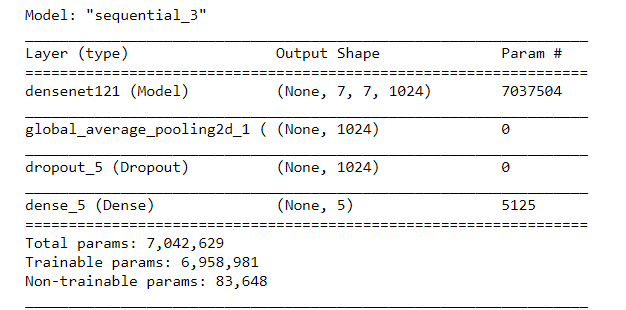
we’ve tried a couple of models, such as **DenseNet121, ResNet50** and **Efficientnet -B4** architecture with **7M 23M, 17 M** parameters, **82.2** *top-1* and **96.4** *top-5* errors weren’t able to surpass the **66.2%** accuracy limit on submission set, nonetheless, some of them have shown **>95.0%** acc on validation

### Baseline Model – self built architecture (2019 data)

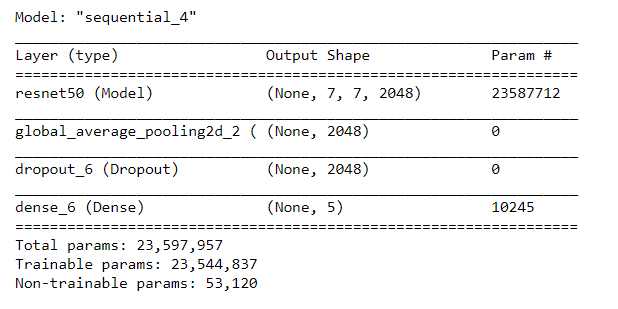


result loss: 0.4679

### DenseNet 121 – with data augmentation (2019 data)



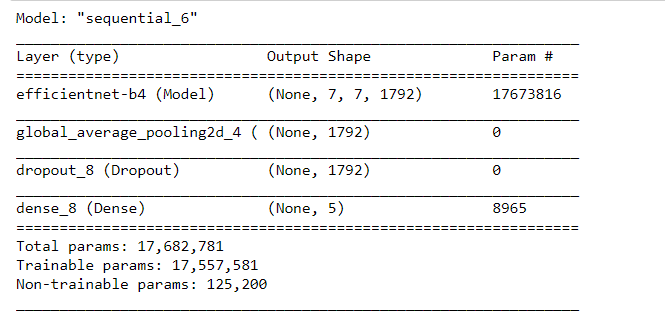
### ResNet50 pretrained on 2015



The number of parameters here have increased to around 23.5M as compared to the Densenet121 architecture with only 7M parameters.

Although the performance of this model is not significantly better that what is achieved by Densenet121 as this achieved a kappa score of **91.65** and also the recall matrix for this model is very similar.

### EfficientNetB4



**The number of parameters used by this architecture is more than Densenet121 and less than Resnet50 but the performance achieved here is the best among the three, as this model scores a kappa score of 92.01** and also the recall matrix has improved.

## Performance Metrics

### Confusion Metrics/Precision Recall

### Cohen Kappa

Quadratic Weighted Kappa(QWK) Kappa is a score which takes into account both accuracy of the model with respect to the doctor's diagnosis and also the agreement of the model and Doctor by chance it is represented by



'κ' and is defined as where po 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. If the raters are in complete agreement, then kappa =1. If there is no agreement among the raters other than what would be expected by chance (as given by pe), kappa =0.

Weighted Kappa is a small variation to this, here if two raters disagree with each other then the score is given according to the distance of the ratings given by both raters. That means that our score will be higher if (a) the real value is 4 but the model predicts a as 3, and the score will be lower if (b) the model instead predicts a as 0.

**Loss Function**

Binary cross entropy is calculated on each output class and is summed up to obtain the loss for each data point

1. **Model Evaluation**
   1. **plot validation kappa score – loss and accuracy**
   2. **combining and comparing all models**

#### Exploratory Data Analysis and Feature Engineering

#### From the above distributions we can see that the classes 0 and 2 are dominant classes which means the data is imbalanced. Train and validation sets have similar percentage of images in each class.

#### Image Preprocessing

#### Image Features Enhancement

#### A screenshot of a cell phone Description automatically generated

#### We’re plotting a sample of images with and without gaussian-blur and the label above the image is Class\_label – Index – Image\_id.

A picture containing table, photo, plate, filled

Description automatically generated

A picture containing photo, different, many, white

Description automatically generated

#### Cropping dark edges from images

#### Here we’re saving the pre-processed (enhanced and cropped) images (training, test and validation) into the numpy arrays and storing on the disk

* 1. Transforming Output variables

The given output variables are categorical, and this means the problem is posed as a categorical classification problem with reference to the following paper #https://arxiv.org/pdf/0704.1028.pdf it is better to transform such problems into ordinal regression problems.

Converting the labels into dummy variables and then creating the ordinal regression function to label the previous categories as 1

1. **Modelling**

**Loading ML dependencies**

We’ll be evaluating our models on Cohen Kappa Valuation matrix. Therefore, we’ll first write the functions for val\_kappa.

* 1. **Model Building**
     1. **Baseline Model – self built architecture**

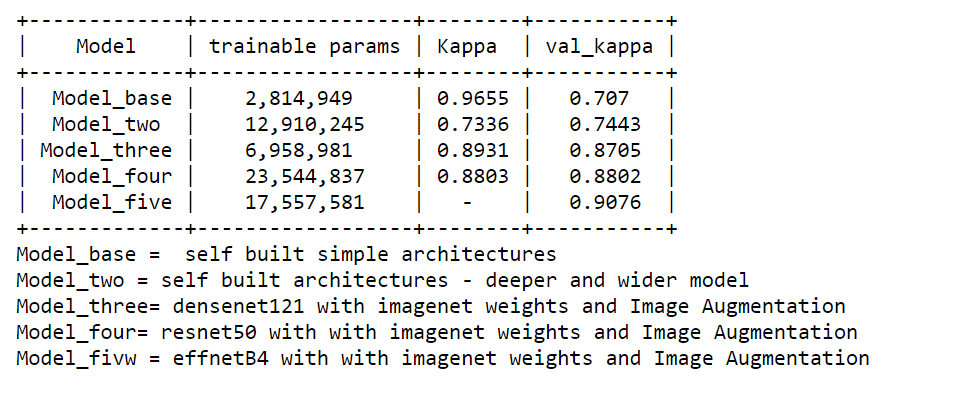
Here we have tried to build a deeper and wider neural network

**A screenshot of a cell phone

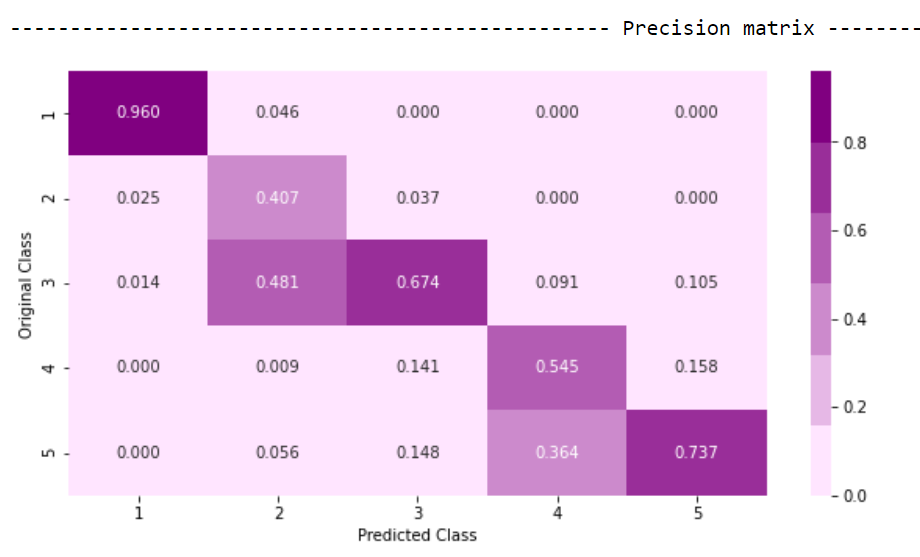
Description automatically generated**

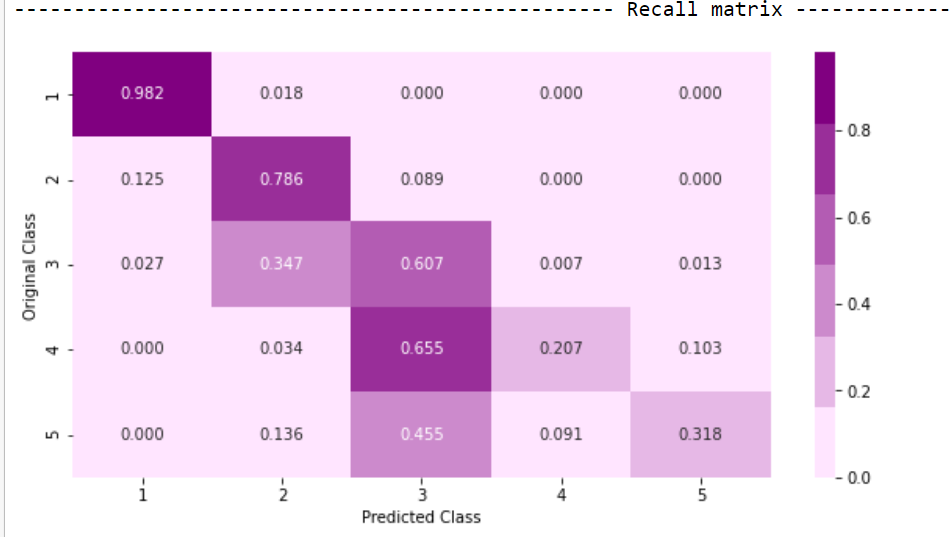
**Now, we’ll evaluate our model by plotting the Confusion, Precision and Recall matrix**

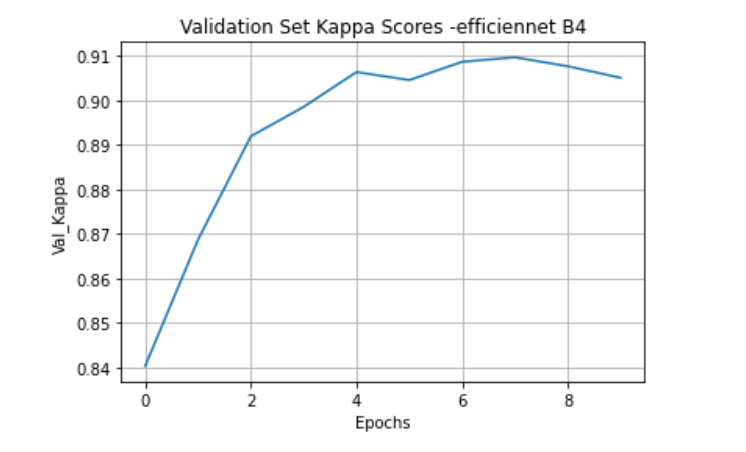
**Results:**

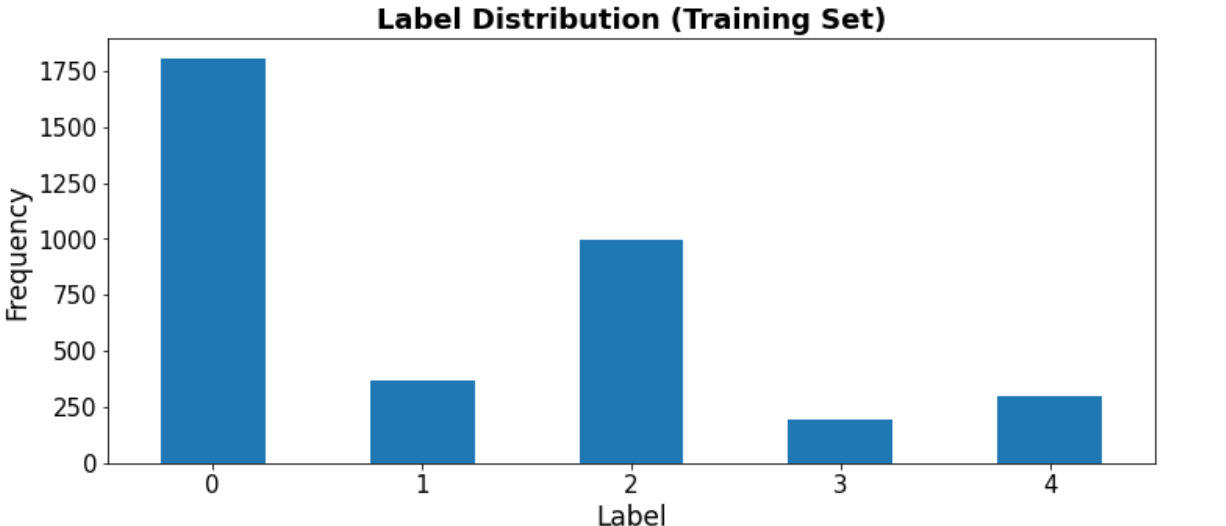
****

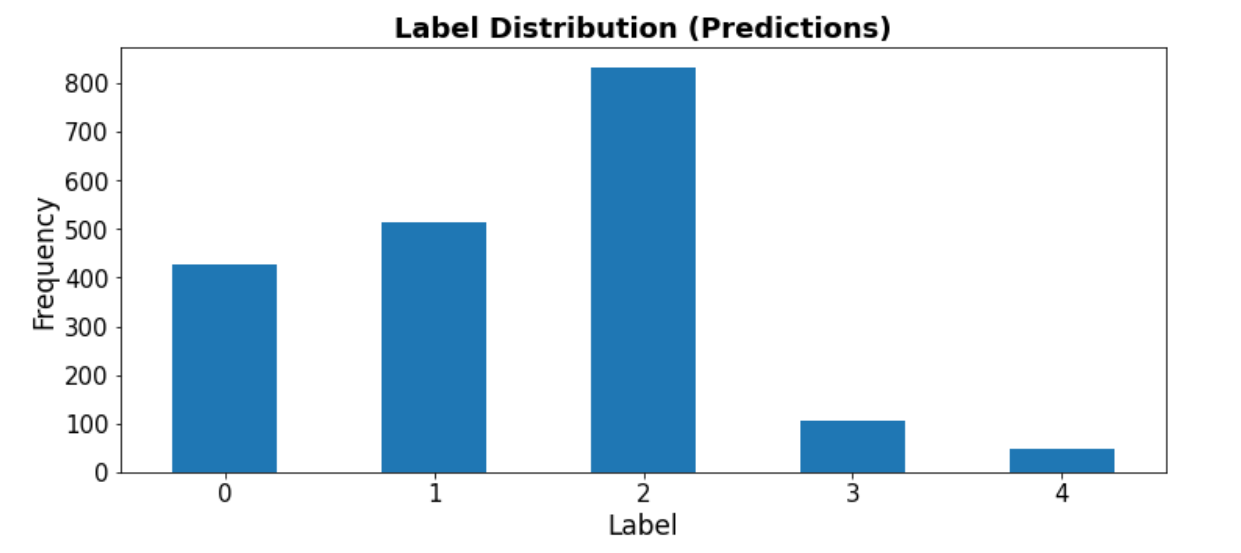
****

****

****

****

****

****