SOHEIL SHIRVANI

Retail Product Classification and detection

Snap Company Test Case Project

Introduction

In this project we are going to first develop a model to classify the retail products which are given in 18 classes. We are assumed to be given an image of a shelf consisting multiple retail products. We then want to detect each of the objects in the image and apply a classification model to determine its class.

We first are going to determine some techniques used in this project, then we will demonstrate some models to be fitted on the dataset. We separate our test and train dataset using a list given by the provider. 0.2 data are used as validation and 0.8 others are used as the training dataset.

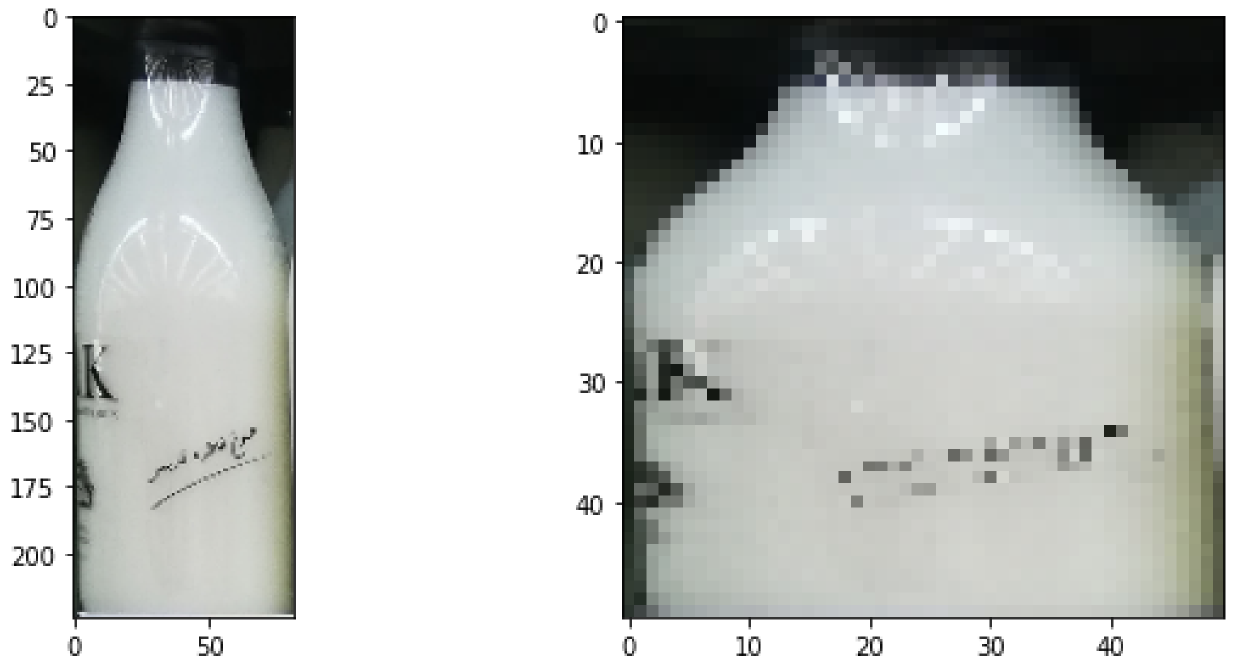
Transfer learning is used to develop a model to fit on the dataset. For preprocessing, the in-built preprocess of the model is used. Multiple models have been tried and the best result is going to be introduced.

Results are plotted and analyzed at the end of this report.

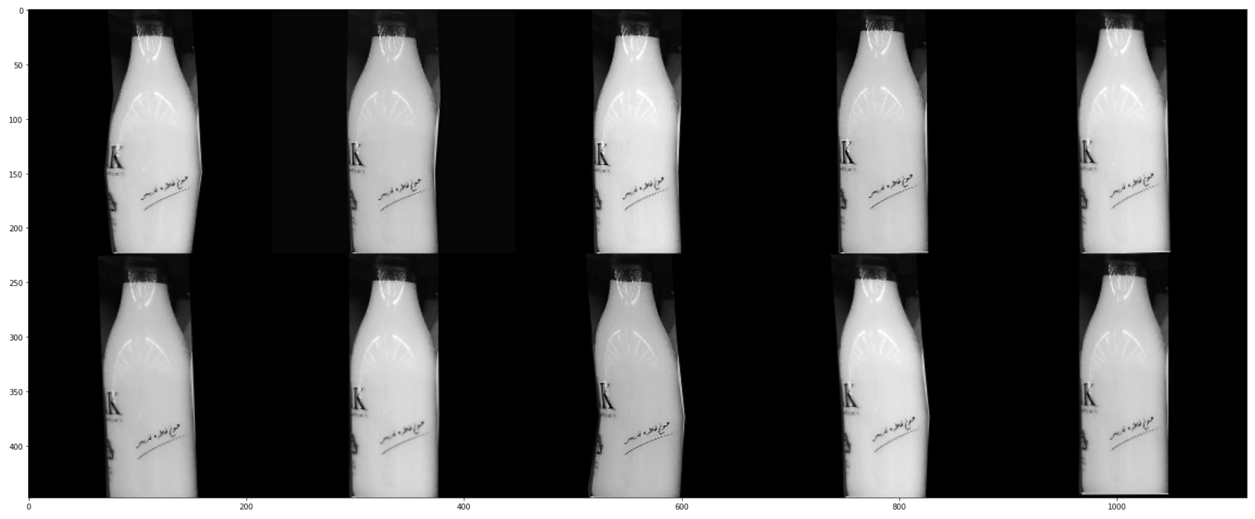
Techniques

There are 3445 product images in size of 224*224*3, a sample of this dataset is shown below:

The black part in these images may be a problem for our model as model can overfit to the black parts. Thus, it is possible to crop the black part and resize the image again. A sample to this technique is done below:



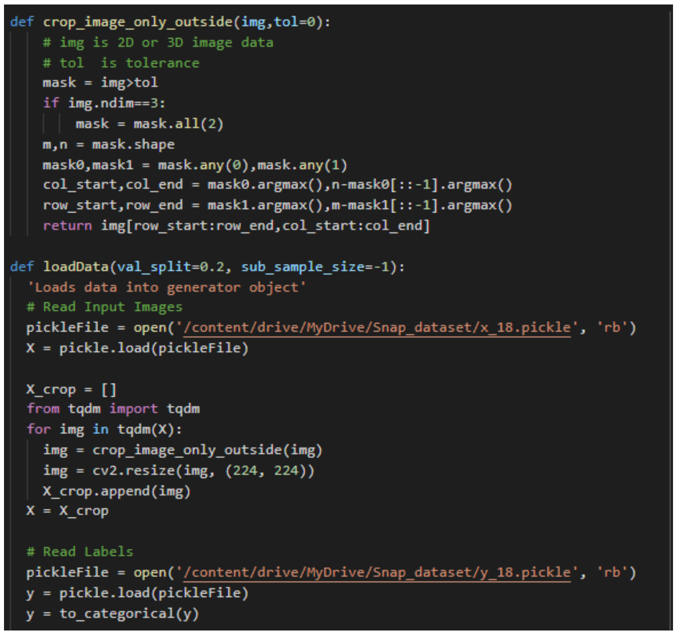
Since there are not much data in our dataset, we use augmentation technique to create a better dataset. This technique applies some deformations on the origin images and create an artificial dataset. The result dataset helps the model to face less overfitting. A sample of this technique is shown below:



Result of applying augmentation can be any of the above images.

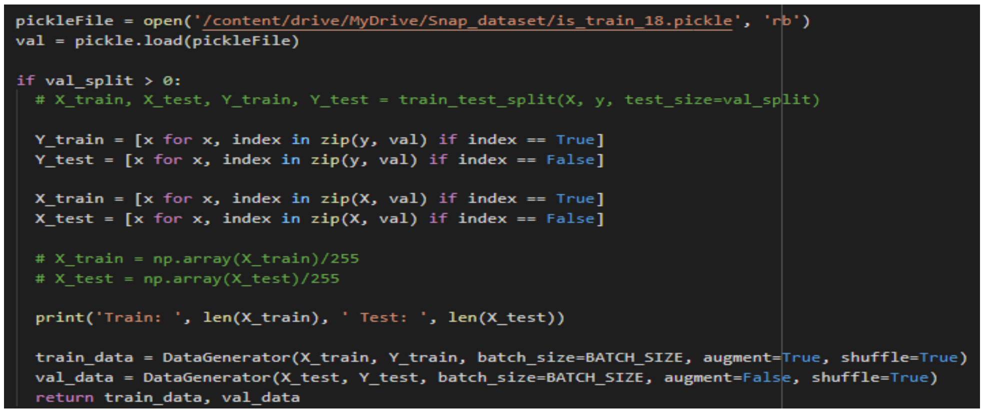
Loading Data

The data is first loaded using the pickle files provided by the company, some samples were given in the techniques section. Then, the back parts of each image are cropped and images are resized back to the origin size 224*224*3. The labels are loaded using the pickle file and converted to One Hot Format using keras categorical function.



```python
def crop_image_only_outside(img,tol=0):
    # img is 2D or 3D image data
    # tol  is tolerance
    mask = img>tol
    if img.ndim==3:
        mask = mask.all(2)
    m,n = mask.shape
    mask0,mask1 = mask.any(0),mask.any(1)
    col_start,col_end = mask0.argmax(),n-mask0[::-1].argmax()
    row_start,row_end = mask1.argmax(),m-mask1[::-1].argmax()
    return img[row_start:row_end,col_start:col_end]

def loadData(val_split=0.2, sub_sample_size=-1):
  'Loads data into generator object'
  # Read Input Images
  pickleFile = open('/content/drive/MyDrive/Snap_dataset/x_18.pickle', 'rb')
  X = pickle.load(pickleFile)

  X_crop = []
  from tqdm import tqdm
  for img in tqdm(X):
    img = crop_image_only_outside(img)
    img = cv2.resize(img, (224, 224))
    X_crop.append(img)
  X = X_crop

  # Read Labels
  pickleFile = open('/content/drive/MyDrive/Snap_dataset/y_18.pickle', 'rb')
  y = pickle.load(pickleFile)
  y = to_categorical(y)
```

A validation pickle file is loaded. This file contains indexes of the validation set from the origin input dataset. This file is provided by the company and in this project the same number of validation set is used.

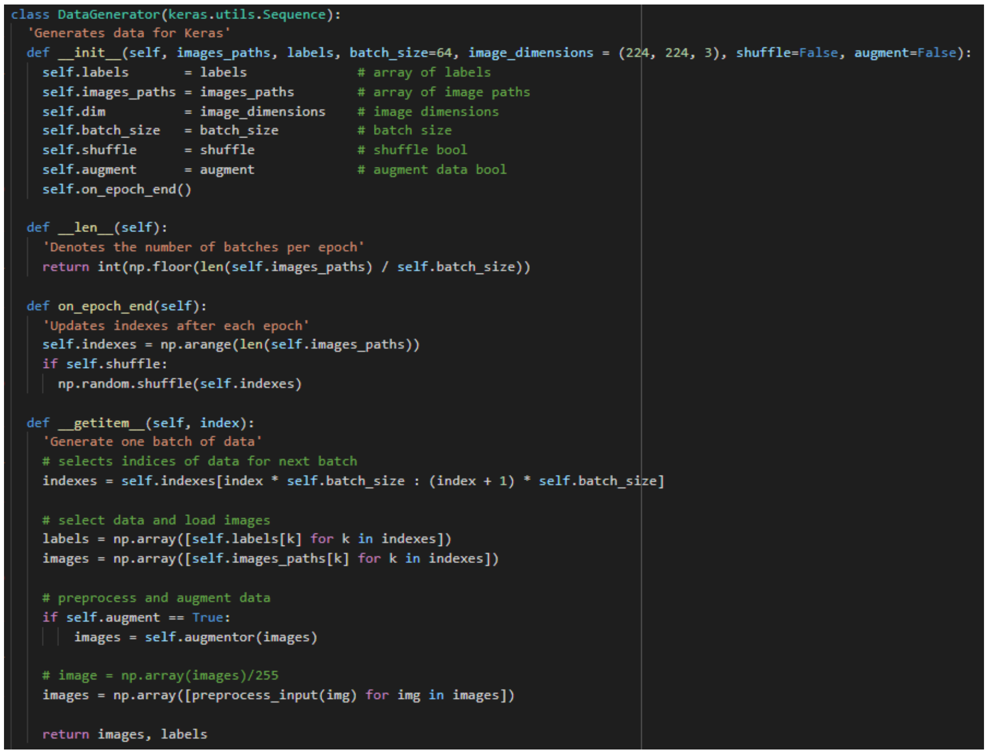


```python
pickleFile = open('/content/drive/MyDrive/Snap_dataset/is_train_18.pickle', 'rb')
val = pickle.load(pickleFile)

if val_split > 0:
  # X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=val_split)

  Y_train = [x for x, index in zip(y, val) if index == True]
  Y_test = [x for x, index in zip(y, val) if index == False]

  X_train = [x for x, index in zip(X, val) if index == True]
  X_test = [x for x, index in zip(X, val) if index == False]

  # X_train = np.array(X_train)/255
  # X_test = np.array(X_test)/255

  print('Train: ', len(X_train), ' Test: ', len(X_test))

  train_data = DataGenerator(X_train, Y_train, batch_size=BATCH_SIZE, augment=True, shuffle=True)
  val_data = DataGenerator(X_test, Y_test, batch_size=BATCH_SIZE, augment=False, shuffle=True)
  return train_data, val_data
```

2402 data are in train set and 1043 data are in validation set. The data the converted to a Data Generator introduced in the following section.

Data Generator

Dataset Generator which is inherited form keras dataset generator was customized for this project as below:



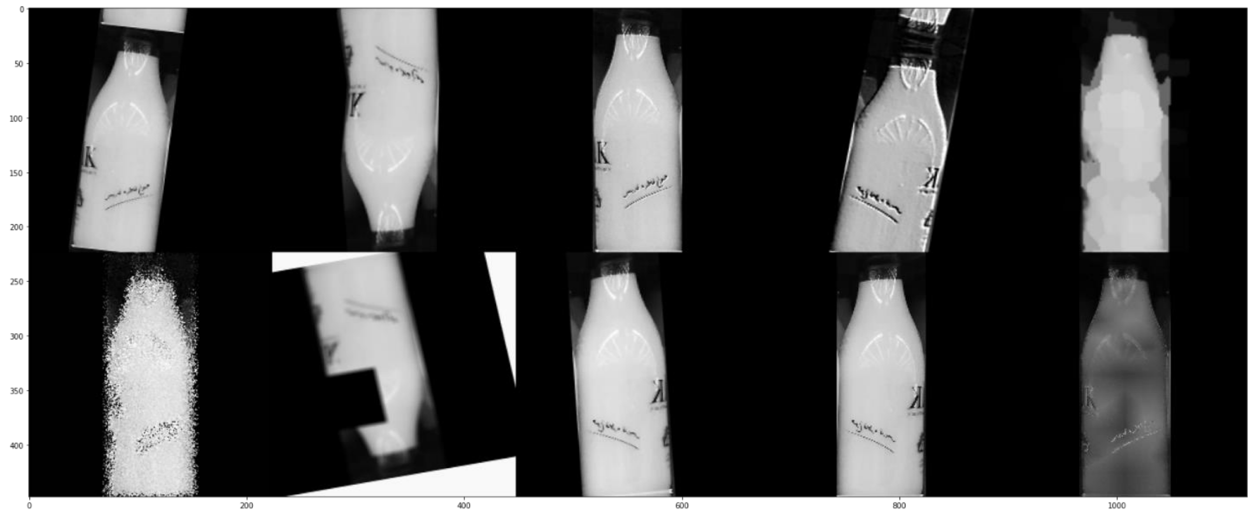
```python
class DataGenerator(keras.utils.Sequence):
  'Generates data for Keras'
  def __init__(self, images_paths, labels, batch_size=64, image_dimensions = (224, 224, 3), shuffle=False, augment=False):
    self.labels       = labels              # array of labels
    self.images_paths = images_paths        # array of image paths
    self.dim          = image_dimensions    # image dimensions
    self.batch_size   = batch_size          # batch size
    self.shuffle      = shuffle             # shuffle bool
    self.augment      = augment             # augment data bool
    self.on_epoch_end()

  def __len__(self):
    'Denotes the number of batches per epoch'
    return int(np.floor(len(self.images_paths) / self.batch_size))

  def on_epoch_end(self):
    'Updates indexes after each epoch'
    self.indexes = np.arange(len(self.images_paths))
    if self.shuffle:
      np.random.shuffle(self.indexes)

  def __getitem__(self, index):
    'Generate one batch of data'
    # selects indices of data for next batch
    indexes = self.indexes[index * self.batch_size : (index + 1) * self.batch_size]

    # select data and load images
    labels = np.array([self.labels[k] for k in indexes])
    images = np.array([self.images_paths[k] for k in indexes])

    # preprocess and augment data
    if self.augment == True:
        images = self.augmentor(images)

    # image = np.array(images)/255
    images = np.array([preprocess_input(img) for img in images])

    return images, labels
```

This data generator loads each batch of image, find its indexes (random indexes if shuffle is True) and then for each image in the batch, image and its relative label is loaded. For each image in the batch (if augmentation is True) an augmentation is applied and each image preprocessed using VGG16 in-built preprocess. The images and their relative labels are then loaded to the model and update the weights.

This generator is used to create batches of training and validation data after loading them. Batch size and image dimension are two important factors in dataset generator.

Augmentation

Augmentation used in this project is shown below:

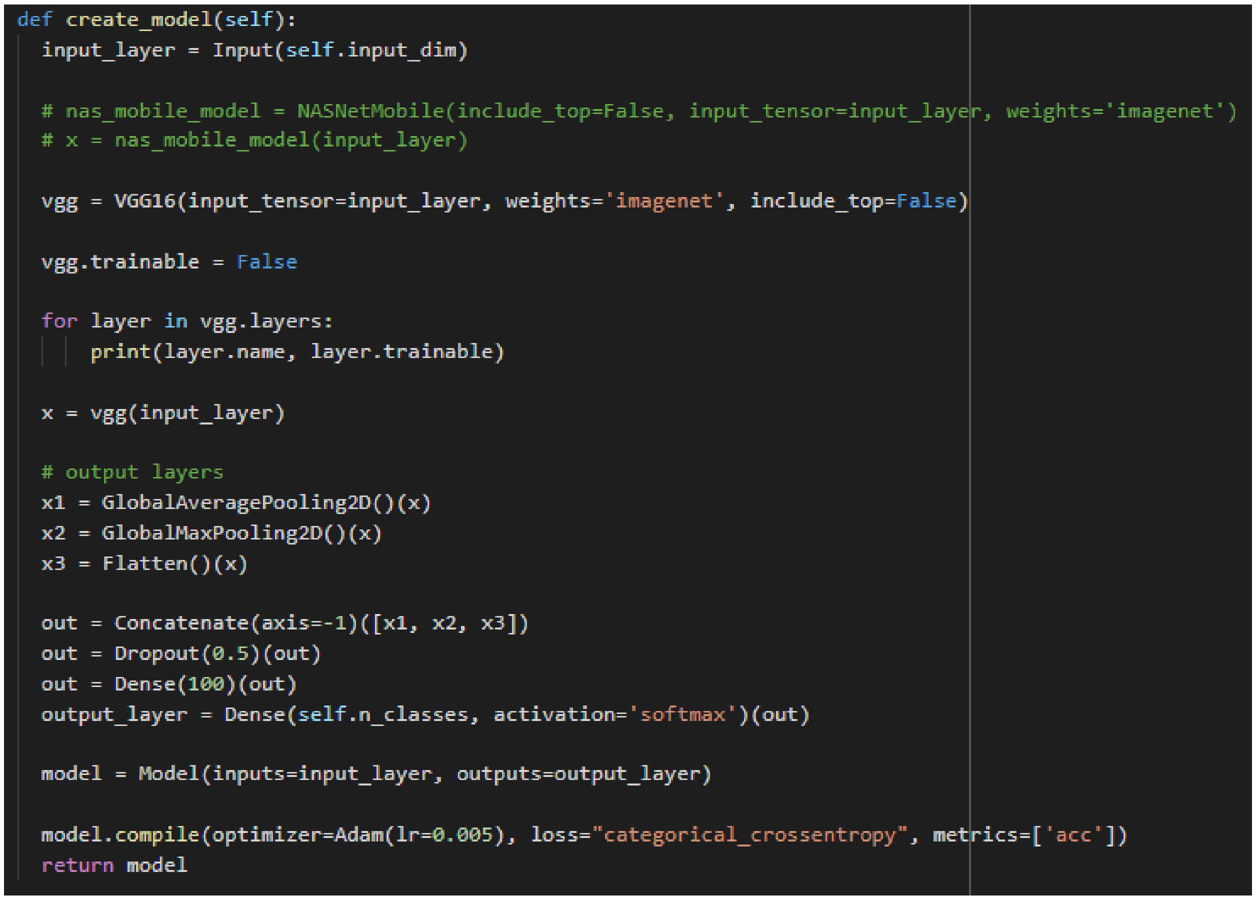


This augmentation seems a lot. This is the reason why did not get a good training accuracy but our validation accuracy was good.

This technique helps the model to touch a better accuracy. A model without augmentation was tested and performed poor.

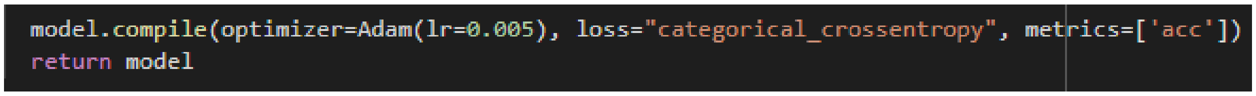
Model Selection

The base model used in this project is VGG16. Other models like InceptionV3, NasNetMobile, and VGG19 were tested but the best result was achieved by VGG16. Adding a GlobalMaxPooling, GlobalAveragePooling, and Flattening to the last Conv layer in VGG16 and Concatenating them, then Adding a 100-layer dense and 18-layer dense (18 is the number of classes) our final model was created.



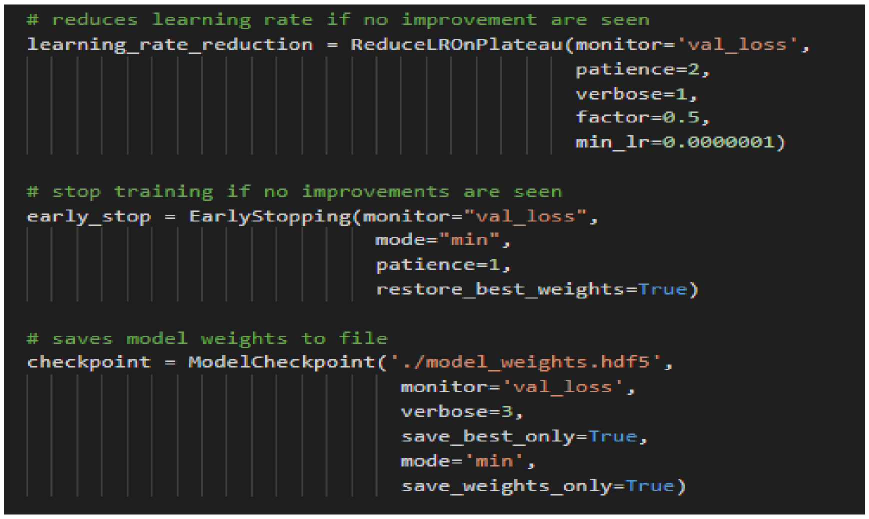
```python
def create_model(self):
  input_layer = Input(self.input_dim)

  # nas_mobile_model = NASNetMobile(include_top=False, input_tensor=input_layer, weights='imagenet')
  # x = nas_mobile_model(input_layer)

  vgg = VGG16(input_tensor=input_layer, weights='imagenet', include_top=False)

  vgg.trainable = False

  for layer in vgg.layers:
      print(layer.name, layer.trainable)

  x = vgg(input_layer)

  # output layers
  x1 = GlobalAveragePooling2D()(x)
  x2 = GlobalMaxPooling2D()(x)
  x3 = Flatten()(x)

  out = Concatenate(axis=-1)([x1, x2, x3])
  out = Dropout(0.5)(out)
  out = Dense(100)(out)
  output_layer = Dense(self.n_classes, activation='softmax')(out)

  model = Model(inputs=input_layer, outputs=output_layer)

  model.compile(optimizer=Adam(lr=0.005), loss="categorical_crossentropy", metrics=['acc'])
  return model
```

The model is then complied using Adam optimizer with initial learning rate of 0.005 and “categorical cross entropy” loss function.



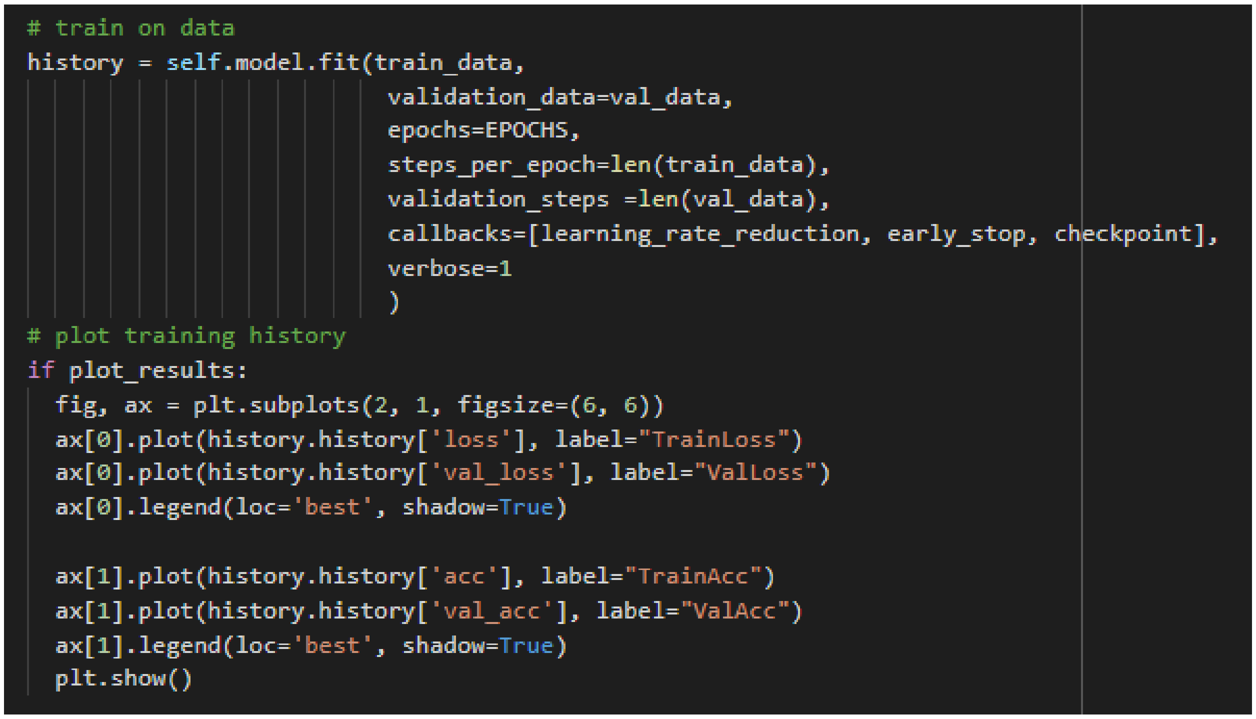
```python
model.compile(optimizer=Adam(lr=0.005), loss="categorical_crossentropy", metrics=['acc'])
return model
```

Three Callbacks including Learning rate reduction, Early Stopping, and Model Check Points were created and the model is fitted using these callbacks.



```python
# reduces learning rate if no improvement are seen
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0000001)

# stop training if no improvements are seen
early_stop = EarlyStopping(monitor="val_loss",
                           mode="min",
                           patience=1,
                           restore_best_weights=True)

# saves model weights to file
checkpoint = ModelCheckpoint('./model_weights.hdf5',
                             monitor='val_loss',
                             verbose=3,
                             save_best_only=True,
                             mode='min',
                             save_weights_only=True)
```

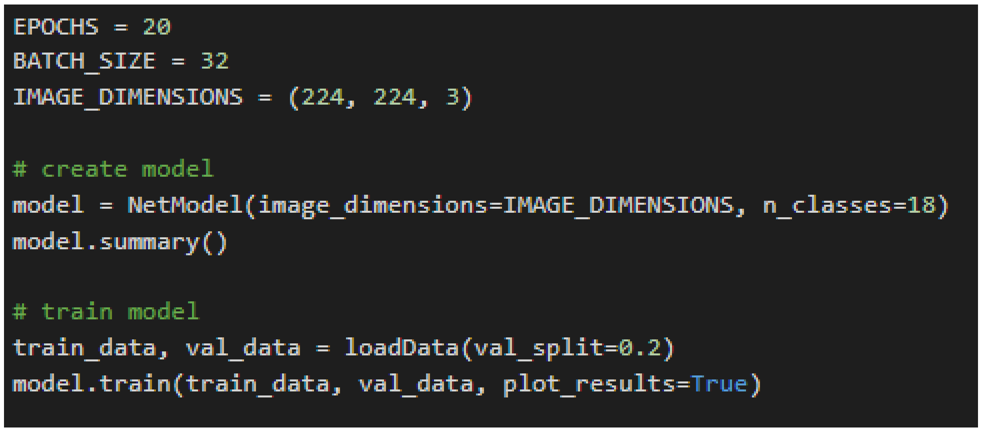
The model trained using the above techniques and after completion, the results are plotted.



```python
# train on data
history = self.model.fit(train_data,
                         validation_data=val_data,
                         epochs=EPOCHS,
                         steps_per_epoch=len(train_data),
                         validation_steps =len(val_data),
                         callbacks=[learning_rate_reduction, early_stop, checkpoint],
                         verbose=1
                         )
# plot training history
if plot_results:
    fig, ax = plt.subplots(2, 1, figsize=(6, 6))
    ax[0].plot(history.history['loss'], label="TrainLoss")
    ax[0].plot(history.history['val_loss'], label="ValLoss")
    ax[0].legend(loc='best', shadow=True)

    ax[1].plot(history.history['acc'], label="TrainAcc")
    ax[1].plot(history.history['val_acc'], label="ValAcc")
    ax[1].legend(loc='best', shadow=True)
    plt.show()
```

Results

We run our model with below conditions:

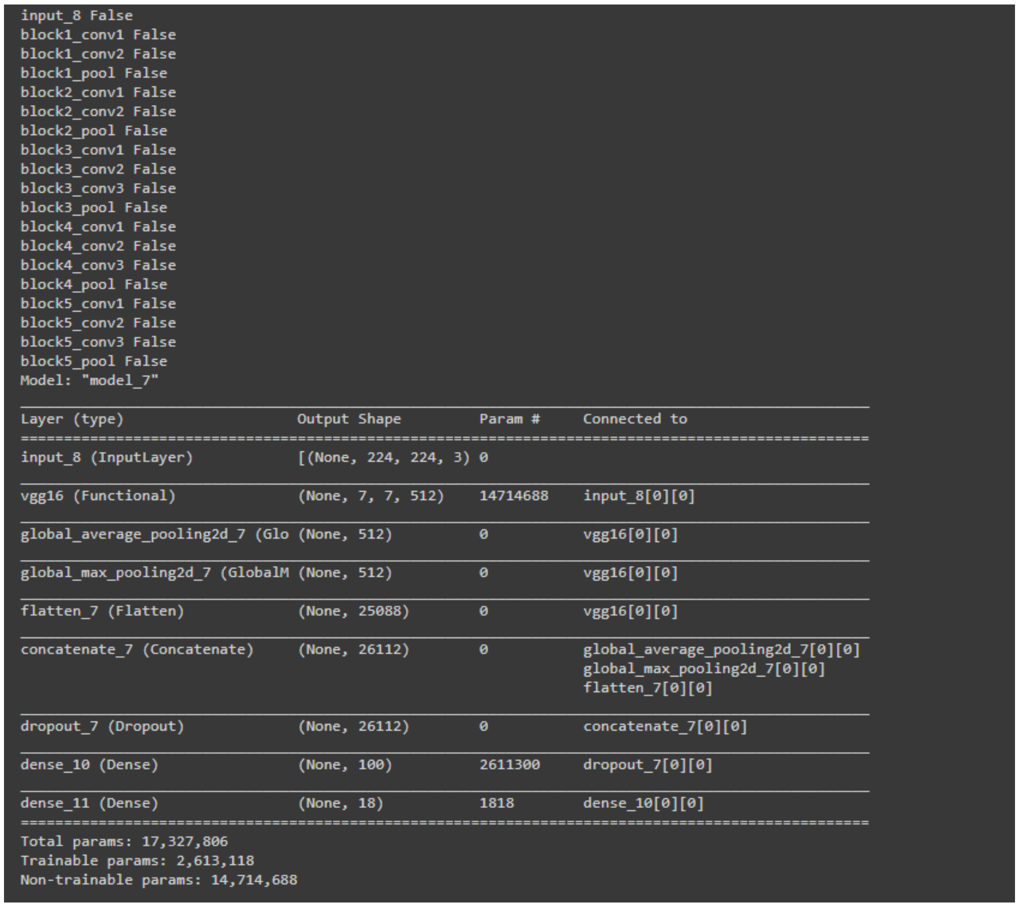

```
EPOCHS = 20
BATCH_SIZE = 32
IMAGE_DIMENSIONS = (224, 224, 3)

# create model
model = NetModel(image_dimensions=IMAGE_DIMENSIONS, n_classes=18)
model.summary()

# train model
train_data, val_data = loadData(val_split=0.2)
model.train(train_data, val_data, plot_results=True)
```

Here our batch size is 32, our input size is the origin size of the images. We fit our model for 20 epochs but because of early stopping callbacks our model converged earlier.

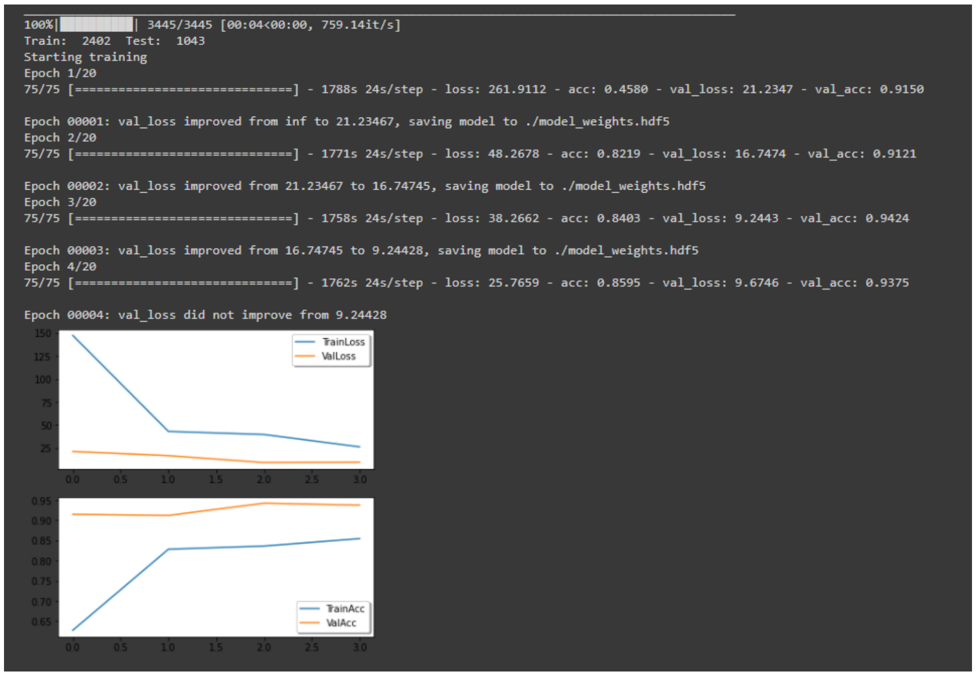
Turning of VGG16 layers trainable (meaning we did not train VGG model weights) we managed to reach the best accuracy. Augmentation and choosing VGG16 as our base model, also were the techniques made our accuracy higher.

After completion of training our model plot the accuracy and losses of each epoch. Training was done in 4 epochs before model stops and results were gathered. Model weights were saved and they are ready for further usage.

Best Results are shown below:



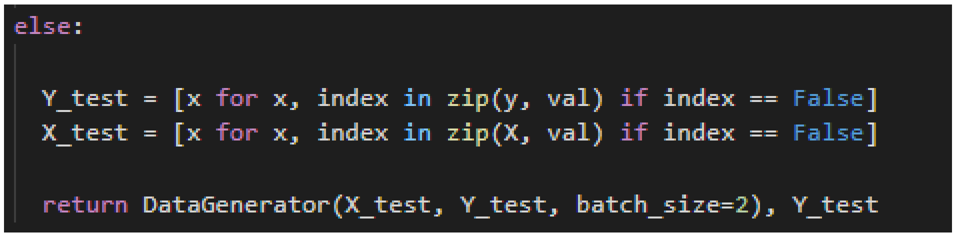
```
input_8 False
block1_conv1 False
block1_conv2 False
block1_pool False
block2_conv1 False
block2_conv2 False
block2_pool False
block3_conv1 False
block3_conv2 False
block3_conv3 False
block3_pool False
block4_conv1 False
block4_conv2 False
block4_conv3 False
block4_pool False
block5_conv1 False
block5_conv2 False
block5_conv3 False
block5_pool False
Model: "model_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_8 (InputLayer)            [(None, 224, 224, 3) 0

__________________________________________________________________________________________________
vgg16 (Functional)              (None, 7, 7, 512)    14714688    input_8[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_7 (Glo (None, 512)          0           vgg16[0][0]
__________________________________________________________________________________________________
global_max_pooling2d_7 (GlobalM (None, 512)          0           vgg16[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 25088)        0           vgg16[0][0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 26112)        0           global_average_pooling2d_7[0][0]
                                                                 global_max_pooling2d_7[0][0]
                                                                 flatten_7[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 26112)        0           concatenate_7[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 100)          2611300     dropout_7[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 18)           1818        dense_10[0][0]
==================================================================================================
Total params: 17,327,806
Trainable params: 2,613,118
Non-trainable params: 14,714,688
```



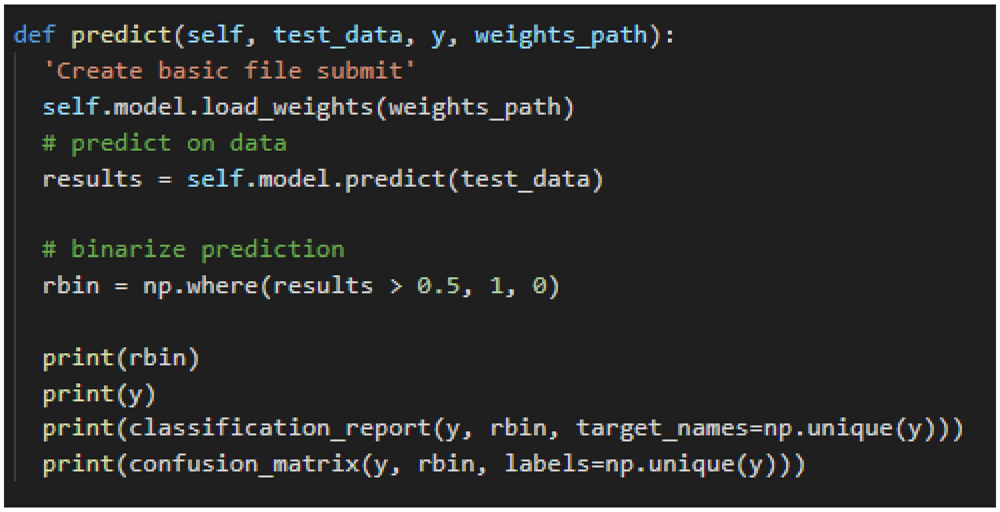
```
100%|██████████| 3445/3445 [00:04<00:00, 759.14it/s]
Train:  2402  Test:  1043
Starting training
Epoch 1/20
75/75 [==============================] - 1788s 24s/step - loss: 261.9112 - acc: 0.4580 - val_loss: 21.2347 - val_acc: 0.9150

Epoch 00001: val_loss improved from inf to 21.23467, saving model to ./model_weights.hdf5
Epoch 2/20
75/75 [==============================] - 1771s 24s/step - loss: 48.2678 - acc: 0.8219 - val_loss: 16.7474 - val_acc: 0.9121

Epoch 00002: val_loss improved from 21.23467 to 16.74745, saving model to ./model_weights.hdf5
Epoch 3/20
75/75 [==============================] - 1758s 24s/step - loss: 38.2662 - acc: 0.8403 - val_loss: 9.2443 - val_acc: 0.9424

Epoch 00003: val_loss improved from 16.74745 to 9.24428, saving model to ./model_weights.hdf5
Epoch 4/20
75/75 [==============================] - 1762s 24s/step - loss: 25.7659 - acc: 0.8595 - val_loss: 9.6746 - val_acc: 0.9375

Epoch 00004: val_loss did not improve from 9.24428
```

As it is shown above, our best accuracy on validation data reached 94% and for training data it was 85%. Our accuracy on training data was a little lower because of over use of augmentation. Due to deadline of this project, I could not tune the augmentation, so results may not be the best.

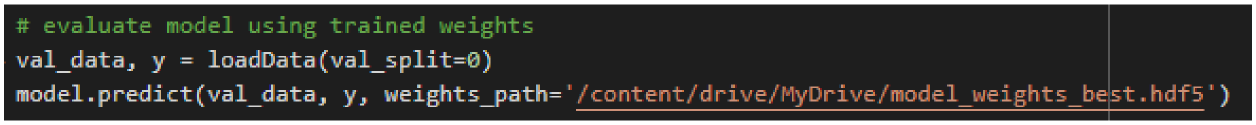
We can now evaluate model by just giving Validation Generator to the model and predict the results:



```
else:

  Y_test = [x for x, index in zip(y, val) if index == False]
  X_test = [x for x, index in zip(X, val) if index == False]

  return DataGenerator(X_test, Y_test, batch_size=2), Y_test
```

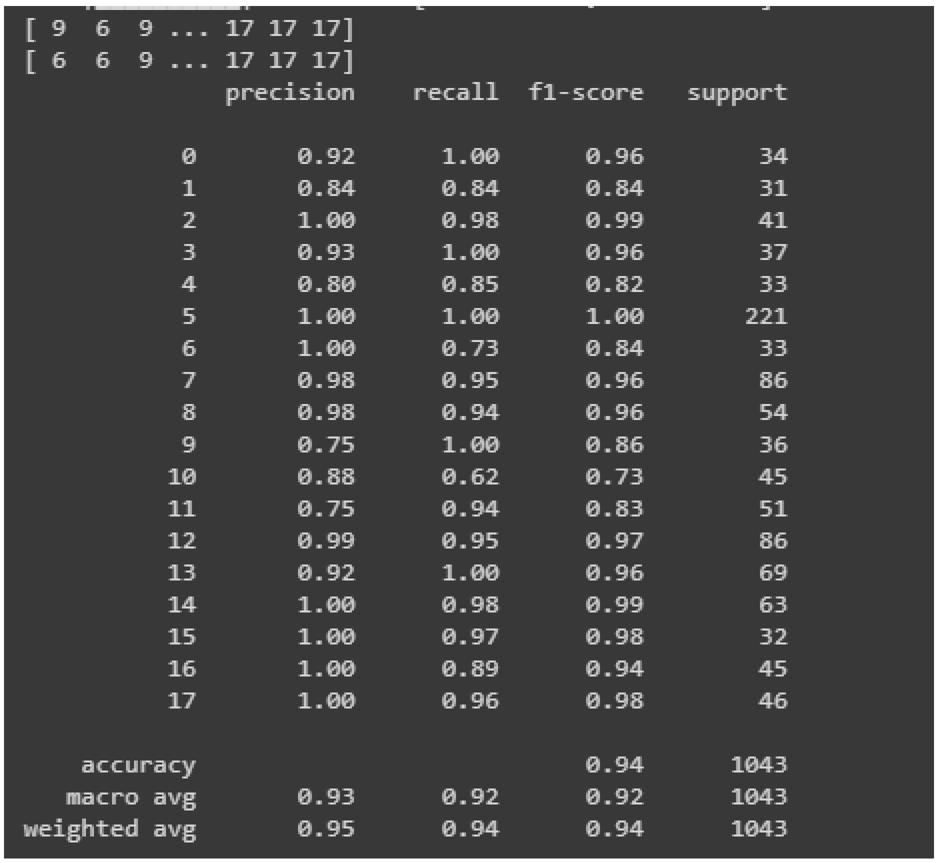


```
def predict(self, test_data, y, weights_path):
  'Create basic file submit'
  self.model.load_weights(weights_path)
  # predict on data
  results = self.model.predict(test_data)

  # binarize prediction
  rbin = np.where(results > 0.5, 1, 0)

  print(rbin)
  print(y)
  print(classification_report(y, rbin, target_names=np.unique(y)))
  print(confusion_matrix(y, rbin, labels=np.unique(y)))
```



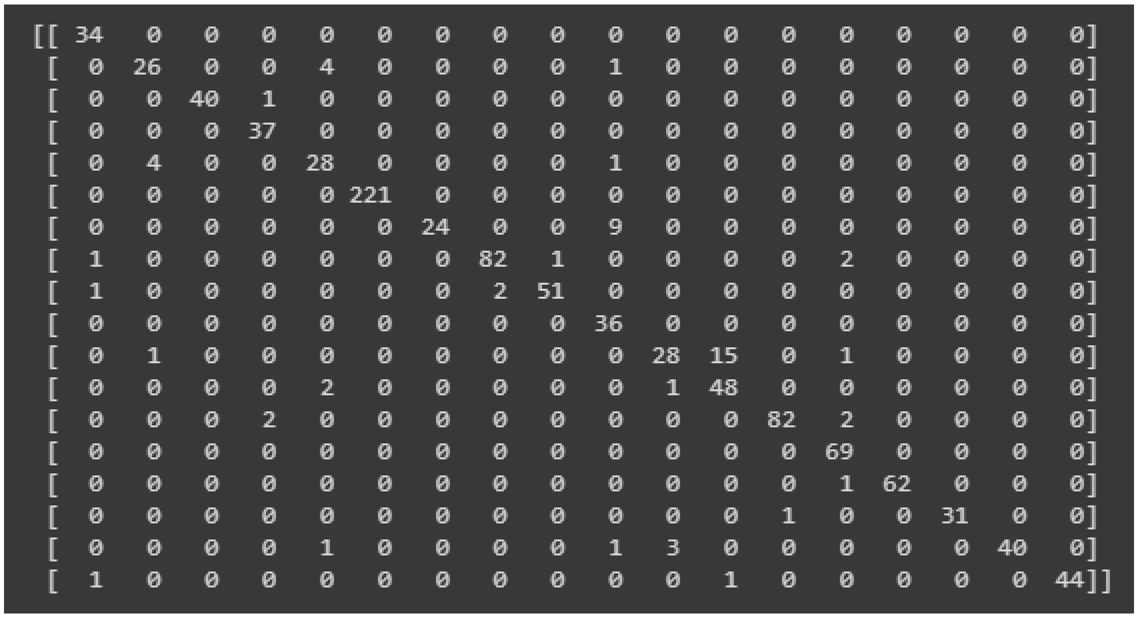
```
# evaluate model using trained weights
val_data, y = loadData(val_split=0)
model.predict(val_data, y, weights_path='/content/drive/MyDrive/model_weights_best.hdf5')
```

By predicting the validation data generator as shown above, results are like below:



```
[ 9  6  9 ... 17 17 17]
[ 6  6  9 ... 17 17 17]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        34
           1       0.84      0.84      0.84        31
           2       1.00      0.98      0.99        41
           3       0.93      1.00      0.96        37
           4       0.80      0.85      0.82        33
           5       1.00      1.00      1.00       221
           6       1.00      0.73      0.84        33
           7       0.98      0.95      0.96        86
           8       0.98      0.94      0.96        54
           9       0.75      1.00      0.86        36
          10       0.88      0.62      0.73        45
          11       0.75      0.94      0.83        51
          12       0.99      0.95      0.97        86
          13       0.92      1.00      0.96        69
          14       1.00      0.98      0.99        63
          15       1.00      0.97      0.98        32
          16       1.00      0.89      0.94        45
          17       1.00      0.96      0.98        46

    accuracy                           0.94      1043
   macro avg       0.93      0.92      0.92      1043
weighted avg       0.95      0.94      0.94      1043
```

And Confusion matrix is like below:



```
[[ 34   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0  26   0   0   4   0   0   0   0   1   0   0   0   0   0   0   0   0]
 [  0   0  40   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0  37   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   4   0   0  28   0   0   0   0   1   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0 221   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  24   0   0   9   0   0   0   0   0   0   0   0]
 [  1   0   0   0   0   0   0  82   1   0   0   0   0   2   0   0   0   0]
 [  1   0   0   0   0   0   0   2  51   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  36   0   0   0   0   0   0   0   0]
 [  0   1   0   0   0   0   0   0   0   0  28  15   0   1   0   0   0   0]
 [  0   0   0   0   2   0   0   0   0   0   1  48   0   0   0   0   0   0]
 [  0   0   0   2   0   0   0   0   0   0   0   0  82   2   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  69   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1  62   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  31   0   0]
 [  0   0   0   0   1   0   0   0   0   1   3   0   0   0   0   0  40   0]
 [  1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0  44]]
```

As it can be seen only small amount of data is misclassified. Most of them were in class 1.

Code, Report, and best weights are attached to the same file.

Good Luck,
Soheil Shirvani