National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Vivek Vindhyachal Rai
Student ID: X21142157

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vivek Vindhyachal Rai
Student ID:	X21142157
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	09/11/2023
Project Title:	Configuration Manual
Word Count:	967
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	9th November 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vivek Vindhyachal Rai
X21142157

1 Introduction

The manual has important information on system, hardware and software specifications. It also outlines the process for executing the study on the "AutoML library for transfer learning workflows". Section 2 covers system specifications like hardware and software setups. Section 3 explains how to set up the environment, import libraries, and do pre-processing. The fourth section discusses model development and assessment.

2 System Configuration

System Configuration discuss the hardware and software requirements for the code to run efficiently

Table 1: Hardware requirement

Operating System	Microsoft Windows 11
RAM	16GB
Processor	Minimum Intel® Core™ i5
Graphics	NVIDIA® GeForce RTX™ 3050 Laptop GPU
Speed in Hz	3.2GHz

2.1 Software Requirements

Table 2: Software requirement

Browser	Google chrome
Coding language	Programming Language
Development environment	Google Colaboratory
version control	Github

3 Setting Environment

Google Colaboratory is a really great tool for Python-based stuff like data analysis and machine learning. It's basically a cloud platform that comes with libraries like NumPy,

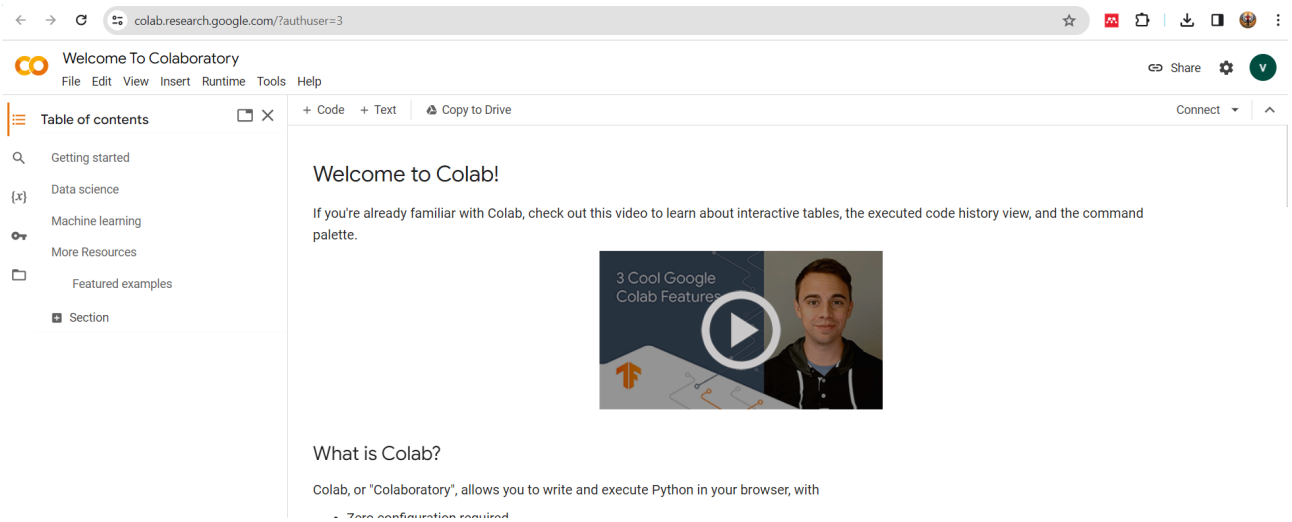

Figure 1: Google Colaboratory

pandas and TensorFlow already installed. This means you can get right to work without any hassle od installing packages/library. The best part is that it offers free CPU and GPU resources, so you can do all those computationally intensive tasks without worrying about your own computer slowing down. Also it's super easy to collaborate with others because it works through web browsers and there's no need to install anything locally. You can even store and share your data using Google Drive. And if you like customizing things or adding lots of text no problem Google Colab lets you do all that too. So yeah, it's a really versatile solution for all kinds of projects and research. Due to all this advantages We have used Google colaboratory for your coding and development of library.

4 Datasets

Kaggle is a popular website for people who are into data science and machine learning. It's a place where you can work together with others to analyze data, create models and even compete against each other. The site has lots of different datasets to choose from as well as tools that let you do complex calculations using the cloud. This makes it perfect for trying out new projects that are based on data. One of the best things about Kaggle is its community - everyone is encouraged to share what they know and help each other solve problems. You can take part in competitions, exchange ideas and learn from other people who are interested in machine learning. Whether you're just starting out or already have experience with data science Kaggle has plenty of tutorials and forums to help you out. It's a great resource that promotes innovation and expertise in this field.

Table 3: Source of our dataset

Emotion classification dataset	Facial Emotion [1]
Pneumonia Classification dataset	Pneumonia Classification[2]

5 Package development

5.1 Importing Important library

```
import math
import pandas as pd
import numpy as np
import cv2
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Conv2D, GlobalAveragePooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization, Activation
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
```

Figure 2: Library used for Package development

5.2 Functions development

5.2.1 build_model function

```
def build_model(bottom_model, classes):
    model = bottom_model.layers[-2].output
    model = GlobalAveragePooling2D()(model)
    model = Dense(classes, activation = 'softmax', name = 'out_layer')(model)

    return model
```

Figure 3: Build model function

5.2.2 modeling function

```
def modeling(img_features,img_labels, batch_size = 32, epochs = 25):
    """
    input: img_features,img_labels
    img_features: numpy array of all the images read. Hence the input will be the numpy images
    img_labels: numpy array of the target variables.
    Thus,
    img_features = X
    img_labels = y

    output: performane table sorted with the accuracies

    batch_size and epochs are hyper parameters. Change them accordingly to the requirement.
    """
```

Figure 4: modeling function

5.2.3 Split-train inside Modeling function

```
X_train, X_valid, y_train, y_valid = train_test_split(img_features,
                                                      img_labels,
                                                      shuffle = True,
                                                      stratify = img_labels,
                                                      test_size = 0.1,
                                                      random_state = 42)
X_train.shape, X_valid.shape, y_train.shape, y_valid.shape

num_classes = y_train.shape[1]

X_train = X_train / 255.
X_valid = X_valid / 255.
```

Figure 5: Splitting and training

5.2.4 VGG-19,Resnet 50,MobileNet,EfficientNetB1,DenseNet-121 inside Modeling function

```
vgg = tf.keras.applications.VGG19(weights = 'imagenet',
                                  include_top = False,
                                  input_shape = (48, 48, 3))

head = build_model(vgg, num_classes)
model = Model(inputs = vgg.input, outputs = head)
early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)
lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(X_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(X_train,
                                       y_train,
                                       batch_size = batch_size),
                    validation_data = (X_valid, y_valid),
                    steps_per_epoch = len(X_train) / batch_size,
                    epochs = epochs,
                    callbacks = callbacks,
                    use_multiprocessing = True)

yhat_valid_vgg = np.argmax(model.predict(X_valid), axis=1)
```

Figure 6: VGG-19

```
"""# ResNet 50"""

res = tf.keras.applications.ResNet50(weights = 'imagenet',
                                     include_top = False,
                                     input_shape = (48, 48, 3))

head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(X_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(X_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (X_valid, y_valid),
                                       steps_per_epoch = len(X_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_res = np.argmax(model.predict(X_valid), axis=1)
```

Figure 7: Resnet 50

```
"""# MobileNet"""

res = tf.keras.applications.MobileNet(weights = 'imagenet',
                                      include_top = False,
                                      input_shape = (48, 48, 3))

head = build_model(res, num_classes)
model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(X_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(X_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (X_valid, y_valid),
                                       steps_per_epoch = len(X_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_mob = np.argmax(model.predict(X_valid), axis=1)
```

Figure 8: MobileNet

```
"""# EfficientNetB1"""

res = tf.keras.applications.EfficientNetB7(weights = 'imagenet',
                                           include_top = False,
                                           input_shape = (48, 48, 3))

head = build_model(res, num_classes)
model = Model(inputs = res.input, outputs = head)
early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(X_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(X_train,
                                       y_train,
                                       batch_size = batch_size),
                                       validation_data = (X_valid, y_valid),
                                       steps_per_epoch = len(X_train) / batch_size,
                                       epochs = epochs,
                                       callbacks = callbacks,
                                       use_multiprocessing = True)

yhat_valid_eff = np.argmax(model.predict(X_valid), axis=1)
```

Figure 9: EfficientNetB1

```
"""# DenseNet"""

res = tf.keras.applications.DenseNet121(weights = 'imagenet',
                                        include_top = False,
                                        input_shape = (48, 48, 3))

head = build_model(res, num_classes)

model = Model(inputs = res.input, outputs = head)

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                               min_delta = 0.00005,
                               patience = 11,
                               verbose = 1,
                               restore_best_weights = True,)

lr_scheduler = ReduceLROnPlateau(monitor = 'val_accuracy',
                                 factor = 0.5,
                                 patience = 7,
                                 min_lr = 1e-7,
                                 verbose = 1,)

callbacks = [early_stopping,lr_scheduler,]

train_datagen = ImageDataGenerator(rotation_range = 15,
                                   width_shift_range = 0.15,
                                   height_shift_range = 0.15,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,)
train_datagen.fit(X_train)

optims = [optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999),]

model.compile(loss = 'categorical_crossentropy',
              optimizer = optims[0],
              metrics = ['accuracy'])

history = model.fit(train_datagen.flow(X_train,
                                       y_train,
                                       batch_size = batch_size),
                    validation_data = (X_valid, y_valid),
                    steps_per_epoch = len(X_train) / batch_size,
                    epochs = epochs,
                    callbacks = callbacks,
                    use_multiprocessing = True)

yhat_valid_den = np.argmax(model.predict(X_valid), axis=1)

y_true = np.argmax(y_valid, axis=1)
```

Figure 10: DenseNet-121

5.2.5 Panda dataframe for tabular representation of output

We use pandas dataframe for output generation

```
y_true = np.argmax(y_valid, axis=1)
accuracy_vgg = accuracy_score(y_true, yhat_valid_vgg)
precision_vgg = precision_score(y_true, yhat_valid_vgg, average='weighted')
recall_vgg = recall_score(y_true, yhat_valid_vgg, average='weighted')
f1_vgg = f1_score(y_true, yhat_valid_vgg, average='weighted')

accuracy_res = accuracy_score(y_true, yhat_valid_res)
precision_res = precision_score(y_true, yhat_valid_res, average='weighted')
recall_res = recall_score(y_true, yhat_valid_res, average='weighted')
f1_res = f1_score(y_true, yhat_valid_res, average='weighted')

accuracy_mob = accuracy_score(y_true, yhat_valid_mob)
precision_mob = precision_score(y_true, yhat_valid_mob, average='weighted')
recall_mob = recall_score(y_true, yhat_valid_mob, average='weighted')
f1_mob = f1_score(y_true, yhat_valid_mob, average='weighted')

accuracy_eff = accuracy_score(y_true, yhat_valid_eff)
precision_eff = precision_score(y_true, yhat_valid_eff, average='weighted')
recall_eff = recall_score(y_true, yhat_valid_eff, average='weighted')
f1_eff = f1_score(y_true, yhat_valid_eff, average='weighted')

accuracy_den = accuracy_score(y_true, yhat_valid_den)
precision_den = precision_score(y_true, yhat_valid_den, average='weighted')
recall_den = recall_score(y_true, yhat_valid_den, average='weighted')
f1_den = f1_score(y_true, yhat_valid_den, average='weighted')


data = {
'Model': ['VGGNet', 'ResNet', 'MobileNet', 'EfficientNet', 'DenseNet'],
'Accuracy': [accuracy_vgg, accuracy_res, accuracy_mob, accuracy_eff, accuracy_den],
'Precision': [precision_vgg, precision_res, precision_mob, precision_eff, precision_den],
'Recall': [recall_vgg, recall_res, recall_mob, recall_eff, recall_den],
'F1 Score': [f1_vgg, f1_res, f1_mob, f1_eff, f1_den]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Set 'Model' as the index
df.set_index('Model', inplace=True)

df = df.sort_values(by='Accuracy', ascending=False)
return df
```

Figure 11: Tabular output

6 Package upload in Google colab

We can use github but here we have upload package 1st in google colab.After loading package we import the package and cv2 similar like pandas or numpy

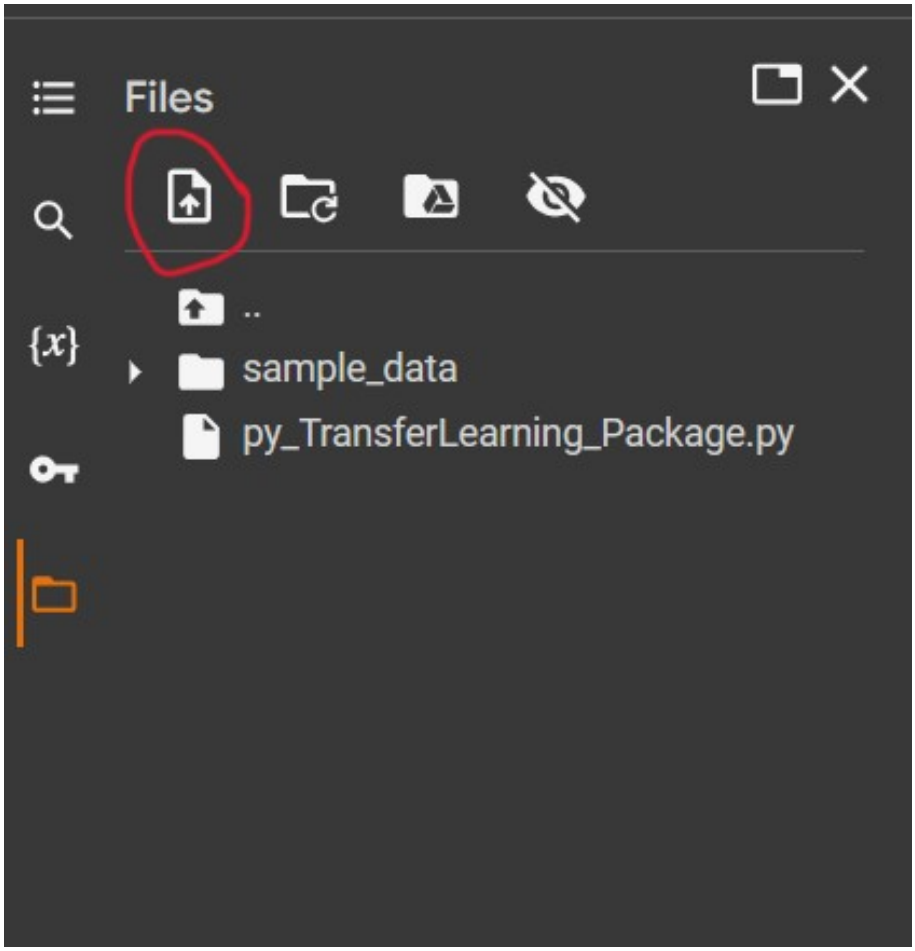

Figure 12: Package upload in google colab

```
[ ]  import py_TransferLearning_Package
     import cv2
```

Figure 13: import the automl package

7 Using Automl library for Facial emotion classification

In below code we have used the imported automl package and we got the output of inform of table like below

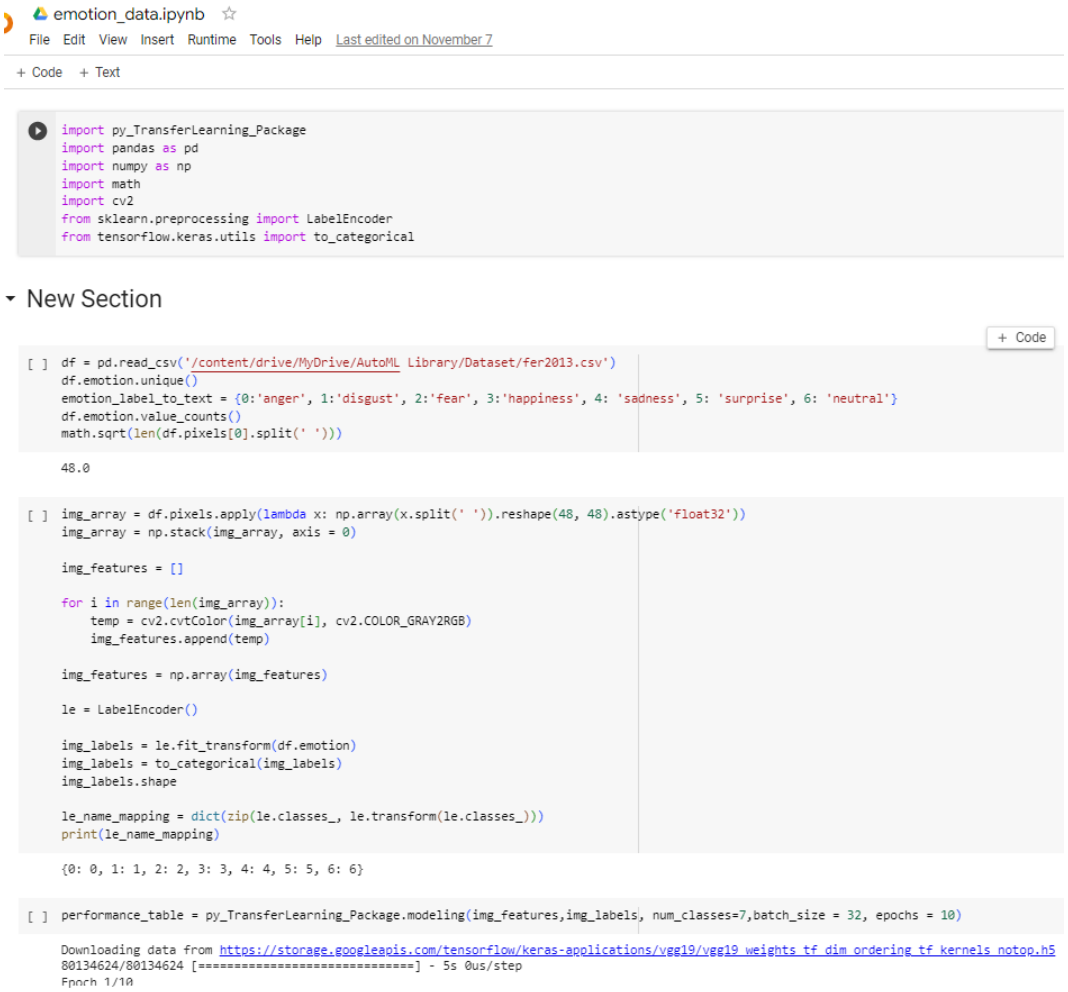

Figure 14: Facial emotion classification

Model	Accuracy	Precision	Recall	F1 Score
VGGNet19	0.641683	0.654075	0.641683	0.640460
DenseNet-121	0.610476	0.633642	0.610476	0.612249
EfficientNet-B7	0.572862	0.595214	0.572862	0.578197
ResNet50	0.564503	0.612886	0.564503	0.560699
MobileNetV2	0.460574	0.634810	0.460574	0.451941

Figure 15: Sample output from the emotion classification use case

8 Using AutoML library for Pneumonia classification

In below code we have used the imported automl package and we got the output of inform of table like below

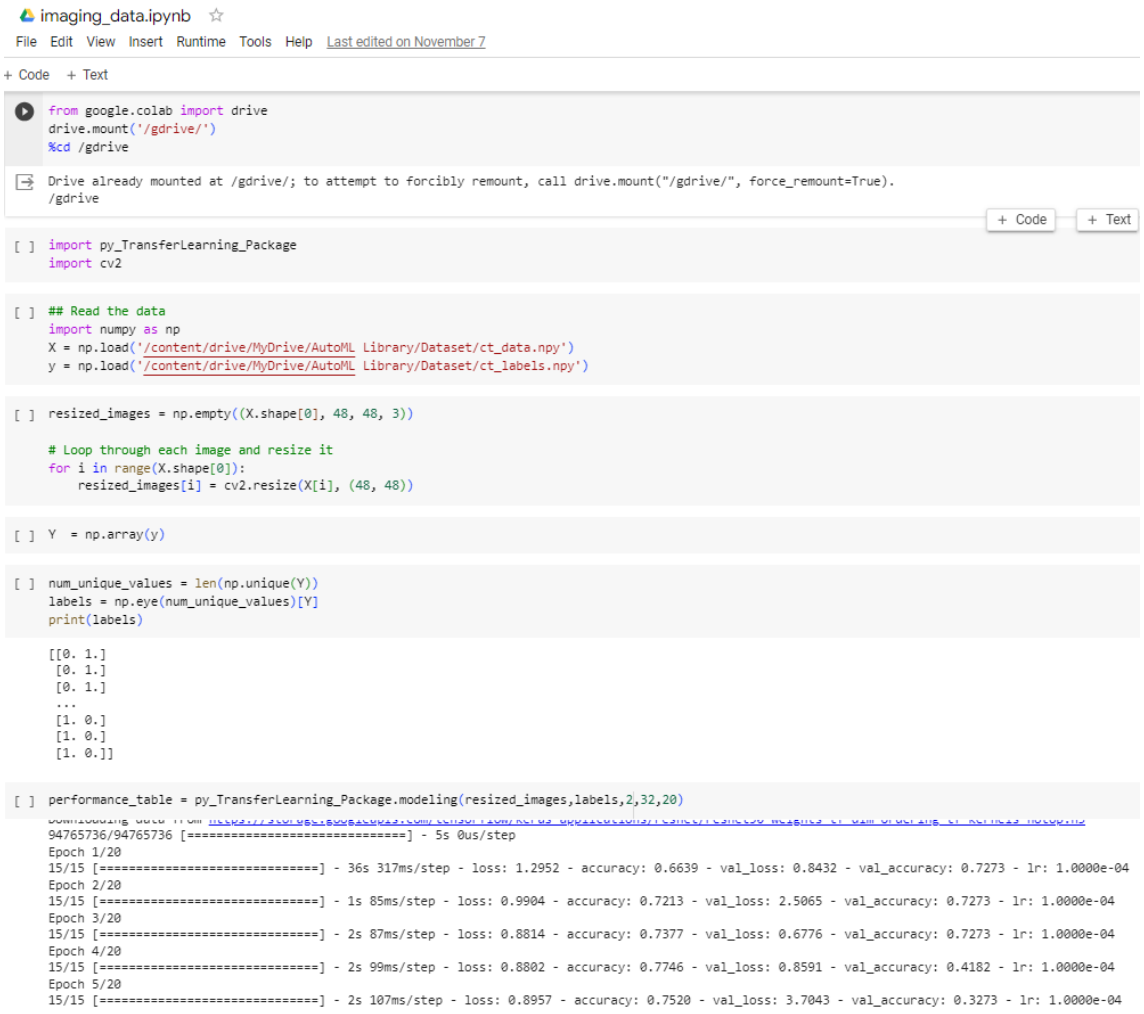

Figure 16: Pneumonia classification

Model	Accuracy	Precision	Recall	F1 Score
VGGNet19	0.800000	0.796167	0.800000	0.797786
MobileNetV2	0.781818	0.770563	0.781818	0.752066
DenseNet-121	0.763636	0.821612	0.763636	0.689782
ResNet50	0.727273	0.528926	0.727273	0.612440
EfficientNet-B7	0.727273	0.528926	0.727273	0.612440

Figure 17: Sample output from the Pneumonia classification use case