Configuration Manual

MSc Data Analytics
Research Project Configuration Manual

Akshen Doke
Student ID: x18191592

School of Computing
National College of Ireland

Supervisor: - Prof. Rashmi Gupta

National College of Ireland

MSc Project Configuration Manual

School of Computing

Student Name:	Akshen Doke		
Student ID:	x18191592		
Programme:	MSc in Data Analytics	**Year:**	2020
Module:	Research Project		
Supervisor:	Prof. Rashmi Gupta		
Submission Due Date:	1st February 2021.		
Project Title:	Image Classification: Detection of covid19, normal and pneumonia from chest x-ray image dataset using ensemble methods.		
Word Count:	3207	**Page Count**	22

Signature: Akshen Doke

Date: 1st February 2021

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):	

1. Hardware and Software Requirements

For this project, all compute intensive tasks like modelling, data visualization and prediction was done on a cloud service called Google Colab[1] which was accessed using a MacBook Air. Only the data downloaded from various data sources were organized properly in their respective folders and converted from jpeg to png and was renamed on local device (MacBook Air) using bash program before uploading it to the cloud.

Table 1: Cloud Setup Option

Processor	On-demand
Graphic Card	TPU and GPU option available
RAM	Min 8Gb-Max 32GB
HDD	12GB free space

Bash scripts for data format changing and renaming.

Changing the format

```
for i in *.jpeg; do
   sips -s format png $i --out pngs
   done
echo “Operation Over”
```

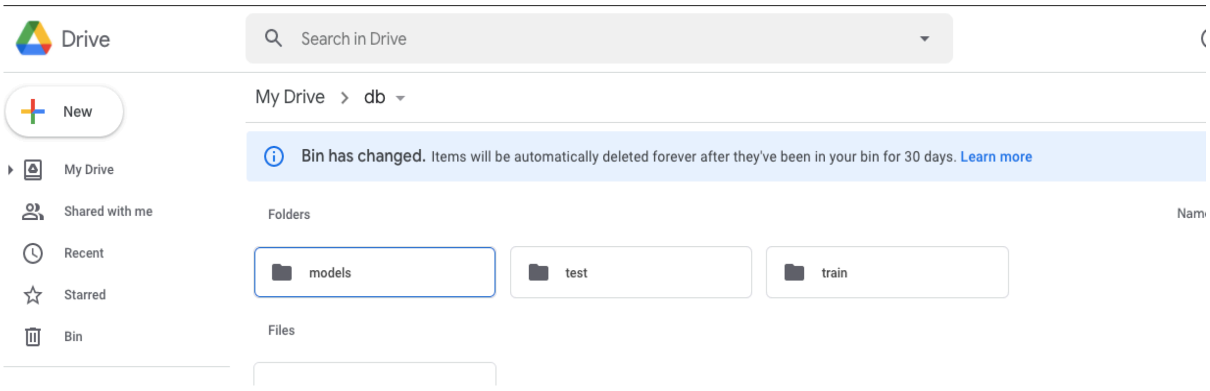
Renaming images

```
count = 0
for i in *.png; do
   mv “$i” “normal-img${count}.png”;
   let count++;
done
echo “Operation Complete..”
```

[1] https://research.google.com/colaboratory/faq.html

2. Google Collaboratory (Colab) Setup

Since this research was carried out using Google Colab's Cloud infrastructure, we need to first upload our dataset to Google drive which can be connected to our notebook (code platform of colab) where we are going to code and use the data.



Bin has changed. Items will be automatically deleted forever after they've been in your bin for 30 days. Learn more

FIGURE 1: Google Drive

We need three folders, one in which we are going to store our training data, second our test data and third for the models on which the training is going to be happening.
The train and test folders had random images from that dataset and were divided locally and then uploaded while the models folder was create online.

As mentioned in (Google, n.d.) Google Colab is an Infrastructure and Software as a Service free to use provided by Google for tasks related to machine learning, data analytics and artificial intelligence in python and its related libraries.

List of libraries and packages used

- Python 3.6.9
- Keras 2.4.0
- Matplotlib
- os
- tensorflow
- sklearn, numpy

To mount the drive to our notebook we use the code given below

G-Drive mounting

```
from google.colab import drive
drive.mount("/content/drive/")
```

After our drive is mounted successfully we can set paths for our train and test files, also import the required libraries and functions for our project.

Importing required libraries and functions

```
import os
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive
from tensorflow import keras
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from keras.utils.vis_utils import plot_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from glob import glob
from tensorflow.keras.models import load_model
```

To get maximum speed and utilization of our notebook we change our runtime to GPU from None, this will make our program execution faster while we train and run our predictions on the dataset.

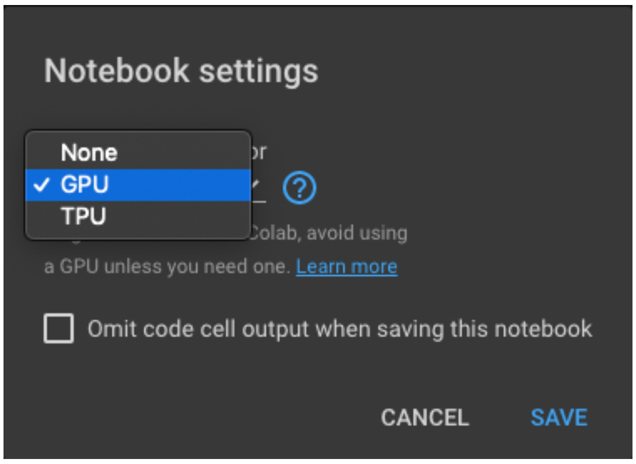

Notebook settings
None
✓ GPU
TPU
Colab, avoid using
a GPU unless you need one. Learn more
Omit code cell output when saving this notebook
CANCEL
SAVE

FIGURE 2: Setting Notebook runtime to GPU

3. Data Preparation and Visualization

Next, we set paths for our train and test datasets in the respect variable names.

Setting path to variables

```
train_path = '/content/drive/My Drive/db/train/'
test_path = '/content/drive/My Drive/db/test/'
folders = glob('/content/drive/My Drive/db/test/*')
```

Now we need to calculate the overall count of each set of images and represent it visually for that we use python based library called matplotlib

Counting datasets and plotting

```
count = {‘covid’: 0, ‘normal’: 0, ‘pneumonia’:0}
for i in count.keys():
  train_path +=i
  test_path +=i
  path, dirs, Trfiles = next(os.walk(train_path))
  path, dirs, Tsfiles = next(os.walk(test_path))
  count[i] += len(Trfiles) + len(Tsfiles)
  train_path = ‘/content/drive/My Drive/db/train/’
  test_path = ‘/content/drive/My Drive/db/test/’

keys = count.keys()
values = count.values()
colors = [‘c’, ‘g’, ‘y’]
plt.rcParams.update({‘font.size’: 14})
plt.pie(values, labels=keys, colors=colors, startangle=360,
      explode=(0.2,0,0), autopct= ‘%1.2f%%’)
plt.title(‘DATA’, fontdict = {‘fontsize’: 21})
plt.show()
```

The output of Data spread which we get is

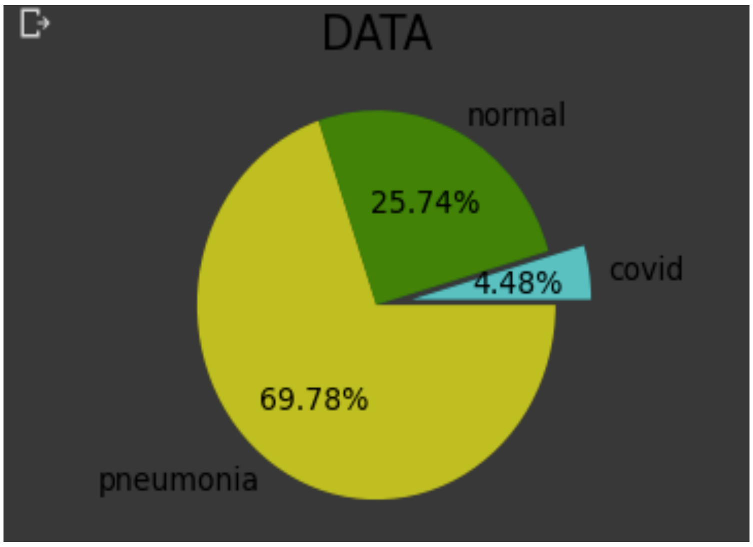


FIGURE 3: Pie plot of Dataset

As we can see the count of covid is relatively low, in order to balance this out we would be using data augmentation techniques while training our model.

4. Implementation of Models

Since we are going to make use of ensemble methods for prediction, we would be training around 7 models using which we would be performing the ensemble based prediction.
For the first 5 models, we would be using transfer learning methodology via which a previously trained/optimized model on a large dataset can be inherited and reutilized on other datasets, the advantage of using such a method is that since these models are trained and optimized on large and complex datasets, their architecture can quickly adapt to most of the image datasets and reduce the huge overhead time of creating a convolutional neural network from scratch.

Keras[2] package has numerous such models which can be inherited via transfer learning and reused.

4.1 Image Augmentation and rescaling

Certain methods would be common throughout the model training process like image rescaling and augmentation which is shown below

Data Augmentation

```
# Use the Image Data Generator to import the images from the dataset
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

# Make sure you provide the same target size as initialied for the image
size
training_set = train_datagen.flow_from_directory('/content/drive/My
      Drive/db/train', target_size = (IMAGE_SIZE[0], IMAGE_SIZE[1]),
      batch_size = 32, class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('/content/drive/My
      Drive/db/test', target_size = (IMAGE_SIZE[0], IMAGE_SIZE[1]),
      batch_size = 32, class_mode = 'categorical')
```

4.2 Common Packages and libraries

Other imports

```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import
ImageDataGenerator,load_img
import numpy as np
from glob import glob
```

[2] https://keras.io/about/

a. DenseNet201

Below is the code for implementation of DenseNet201 model which we import from keras package and train our dataset on.

Building the DenseNet Model

```
from tensorflow.keras.applications.densenet import DenseNet201
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.applications.densenet import decode_predictions
IMAGE_SIZE = [224, 224]
densenet201 = DenseNet201(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
# don't train existing weights
for layer in densenet201.layers:
    layer.trainable = False

x = Flatten()(densenet201.output)
prediction = Dense(len(folders), activation='softmax')(x)

# # create a model object
model = Model(inputs=densenet201.input, outputs=prediction)
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Once the model is build and compiled, we begin the training process, we can optimize the parameters while training our model in order to get better output.

Training and saving the densenet model.

```
densenet_model = model.fit(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)

# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('saved_model/densenet201.h5')

from google.colab import files
files.download("saved_model/densenet201.h5")
```

We also save and download the model which we will be using later on for our ensemble of models. Here on, same steps would be repeated for all the models mentioned below.

b. VGG 16

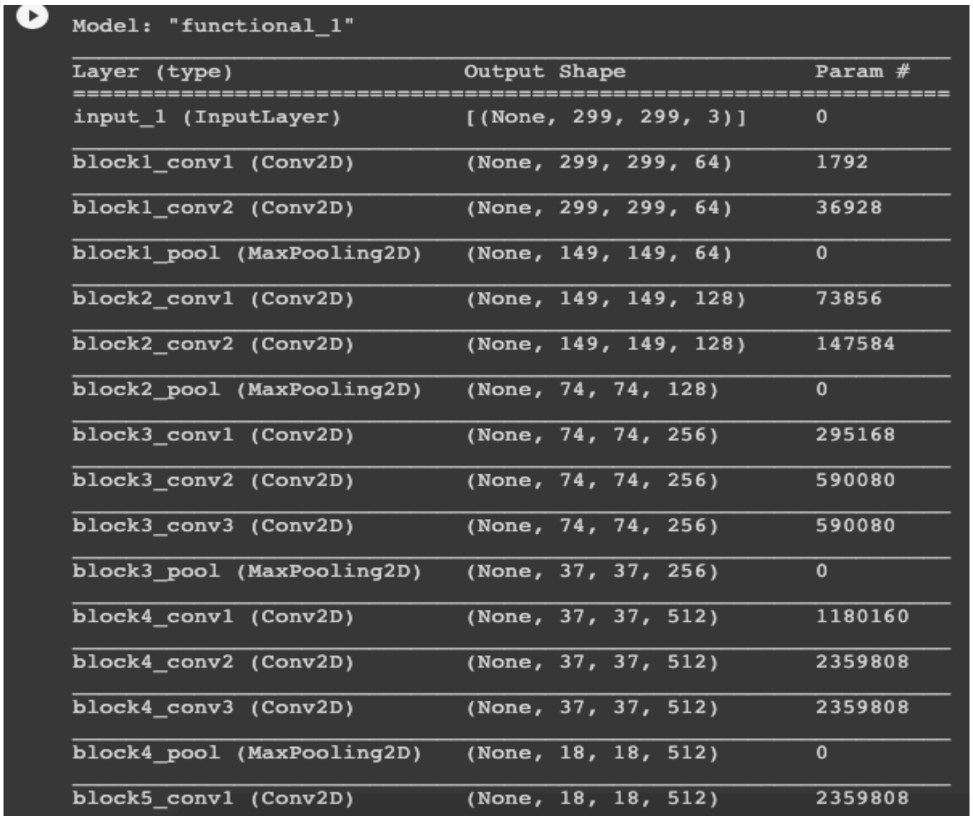
Building the VGG 16 Model

```
from tensorflow.keras.applications.vgg16 import VGG16
IMAGE_SIZE = [299, 299]
vgg_net = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
for layer in vgg_net.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/My Drive/db/train/*')
x = Flatten()(vgg_net.output)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vgg_net.input, outputs=prediction)
model.summary()
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Training and Saving Model

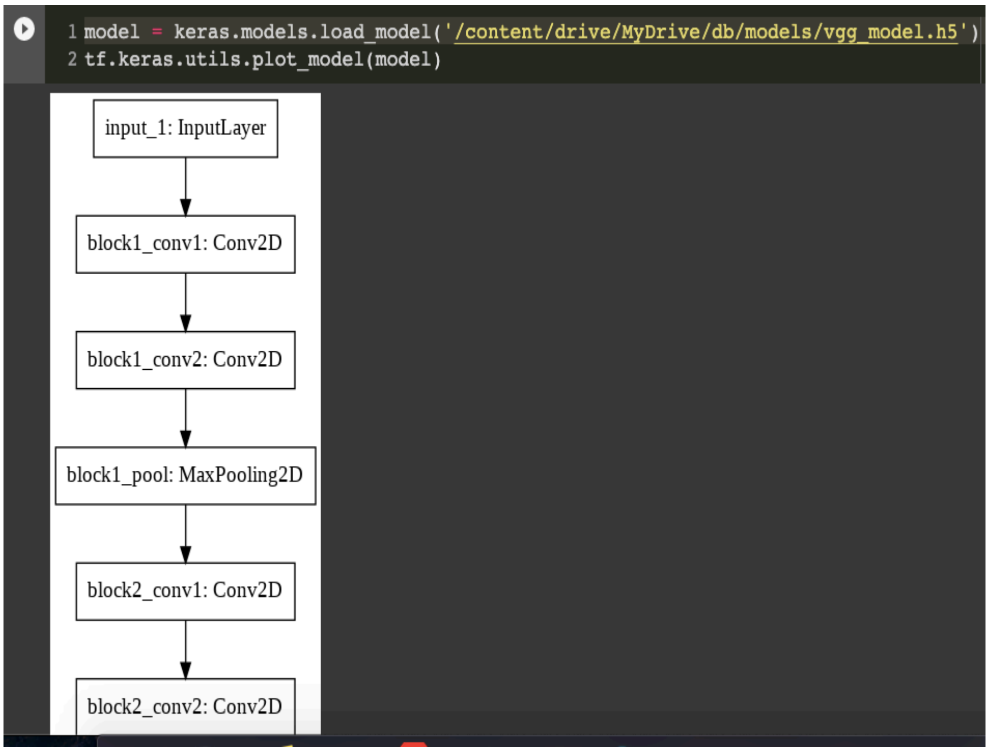
```
# fit the model
# It will take some time to execute
vgg_model = model.fit(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
# save it as a h5 file
from google.colab import files
# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('saved_model/vgg_model.h5')
files.download("saved_model/vgg_model.h5")
```

Some other features worth mentioning which can help us improve the performance and accuracy of our models is that we can take a peek in to the model architecture by using a built in method called `model.summary()` which summarizes the architecture of the model in our case VGG16 in a textual format and another function which gives a plot of our layer stack is `tf.keras.utils.plot_model(model)` output of both functions is given below.



```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 299, 299, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 299, 299, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 299, 299, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 149, 149, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 149, 149, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 149, 149, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 74, 74, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 74, 74, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 74, 74, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 74, 74, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 37, 37, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 37, 37, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 18, 18, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 18, 18, 512)       2359808
```

FIGURE 4: Summary of VGG 16



```
model = keras.models.load_model('/content/drive/MyDrive/db/models/vgg_model.h5')
tf.keras.utils.plot_model(model)
```

FIGURE 5: Architecture Plot for VGG16 Model

Also, we have another technique to see the output of the prediction layers by plotting a heatmap around the input image. This technique is called “Grad-CAM” And the code and output for it is given below

GRAD-CAM settings

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Display
from IPython.display import Image
import matplotlib.pyplot as plt
import matplotlib.cm as cm

img_size = (331, 331)
preprocess_input = keras.applications.nasnet.preprocess_input

last_conv_layer_name = "activation_259"
classifier_layer_names = [
    "flatten",
    "dense",
]
# The local path to our target image
img_path = "/content/drive/MyDrive/db/test/covid/covid68.png"
display(Image(img_path))
```

GRAD-CAM algorithm implementation

```
def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = keras.preprocessing.image.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array

def make_gradcam_heatmap(
    img_array, model, last_conv_layer_name, classifier_layer_names
):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer
    last_conv_layer = model.get_layer(last_conv_layer_name)
    last_conv_layer_model = keras.Model(model.inputs,
last_conv_layer.output)

    # Second, we create a model that maps the activations of the last conv
    # layer to the final class predictions
    classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:])
    x = classifier_input
    for layer_name in classifier_layer_names:
        x = model.get_layer(layer_name)(x)
    classifier_model = keras.Model(classifier_input, x)
    # Then, we compute the gradient of the top predicted class for our
input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make the tape
watch it
```

```
        last_conv_layer_output = last_conv_layer_model(img_array)
        tape.watch(last_conv_layer_output)
        # Compute class predictions
        preds = classifier_model(last_conv_layer_output)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]

    # This is the gradient of the top predicted class with regard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the
      gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted
class
    last_conv_layer_output = last_conv_layer_output.numpy()[0]
    pooled_grads = pooled_grads.numpy()

    for i in range(pooled_grads.shape[-1]):
        last_conv_layer_output[:, :, i] *= pooled_grads[i]

    # The channel-wise mean of the resulting feature map
    # is our heatmap of class activation
    heatmap = np.mean(last_conv_layer_output, axis=-1)

    # For visualization purpose, we will also normalize the heatmap between
      0 & 1
    heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    return heatmap
```

GRAD-CAM HeatMap

```
#Prepare image
img_array = preprocess_input(get_img_array(img_path, size=img_size))

# Print what the top predicted class is
preds = model.predict(img_array)
# Generate class activation heatmap
heatmap = make_gradcam_heatmap(
    img_array, model, last_conv_layer_name, classifier_layer_names
)
# Display heatmap
plt.matshow(heatmap)
plt.show()
```

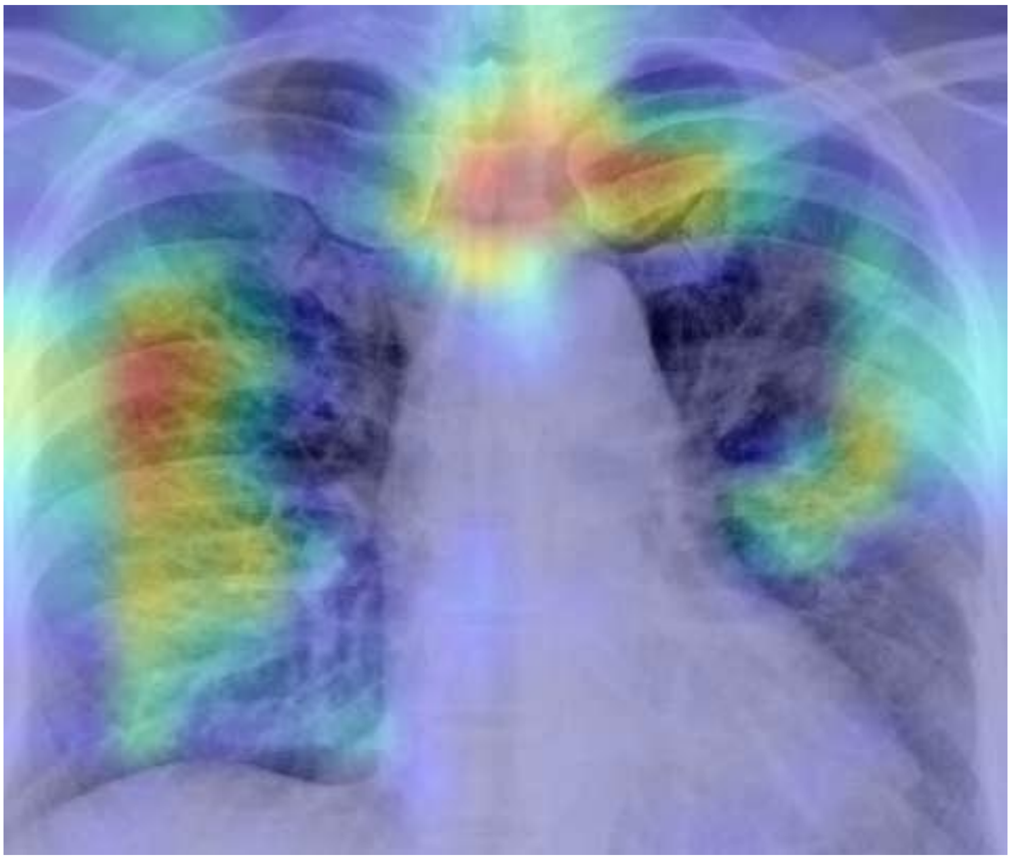


FIGURE 6:HeatMap Over Image

This technique can be applied on individual models but can’t be implemented on the overall output of the ensemble networks which we are going to create.

c. NasNet

Building NASNET Model

```
IMAGE_SIZE = (331, 331,3)
nasNet = NASNetLarge(input_shape=IMAGE_SIZE, weights='imagenet',
include_top=False)
# don't train existing weights
for layer in nasNet.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/My Drive/db/train/*')
x = Flatten()(nasNet.output)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=nasNet.input, outputs=prediction)
#model.summary()
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Training and saving Nasnet

```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'saved_model/'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
# fit the model
# Run the cell. It will take some time to execute
nasnet_model = model.fit(
  training_set,
  validation_data=test_set,
  epochs=15,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
```

d. Xception

Building Xception Model

```
# re-size all the images to this
IMAGE_SIZE = [299, 299]
xceptionNet = Xception(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
for layer in xceptionNet.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/My Drive/db/train/*')
x = Flatten()(xceptionNet.output)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=xceptionNet.input, outputs=prediction)
model.summary()
```

Compiling and Training Xception Model

```
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'saved_model/'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
# fit the model It will take some time to execute
xception_model = model.fit(
  training_set,
  validation_data=test_set,
  epochs=15,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
```

e. Resnet

Building a Resnet

```
# re-size all the images to this
IMAGE_SIZE = [224, 224]
resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
for layer in resnet.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/My Drive/db/train/*')
x = Flatten()(resnet.output)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=resnet.input, outputs=prediction)
x = Flatten()(resnet.output)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=resnet.input, outputs=prediction)
model.summary()
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Training of Resnet Model

```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'saved_model/'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

# fit the model It will take some time to execute
resnet_model = model.fit(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
```

f. MyModel

In case of this model, we create it from scratch and train it on our data, the performance of this model was close to 90 % similar to our other models but since it is only trained on our dataset, the overall performance in comparison to other models might differ when other datasets are taken into consideration.

Building the custom model

```
      myModel = Sequential()
      myModel.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,
      3),padding="same",      activation="relu"))
      myModel.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
      activation="relu"))
      myModel.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
      myModel.add(Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
      padding='same'))
      myModel.add(BatchNormalization())
      myModel.add(Activation('relu'))
      myModel.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
      padding='same'))
      myModel.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
      activation="relu"))
      myModel.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
      activation="relu"))
      myModel.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
      myModel.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1),
      padding='same'))
      myModel.add(BatchNormalization())
      myModel.add(Activation('relu'))
      myModel.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
      padding='same'))
      #Passing it to a Fully Connected layer
      myModel.add(Flatten())
      # 1st Fully Connected Layer
      myModel.add(Dense(4096, input_shape=(224,224,3,)))
      myModel.add(BatchNormalization())
      myModel.add(Activation('relu'))
      # Add Dropout to prevent overfitting
      myModel.add(Dropout(0.4))
      #2nd Fully Connected Layer
      myModel.add(Dense(1000))
      myModel.add(BatchNormalization())
```

```
myModel.add(Activation('relu'))
#Add Dropout
myModel.add(Dropout(0.2))
#Output Layer
myModel.add(Dense(10))
myModel.add(BatchNormalization())
myModel.add(Dense(len(folders), activation='softmax'))
myModel.summary()
myModel.compile(loss = keras.losses.categorical_crossentropy,
optimizer= 'adam', metrics=['accuracy'])
```

Training the Model

```
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'saved_model/'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

# This will take some time to execute
mymodel_ready = myModel.fit(
  training_set,
  validation_data=test_set,
  epochs=15,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
```

g. AlexNet

Building Alexnet Model from scratch

```
#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(150,150,3),
kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
```

```
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(150,150,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
# Add Dropout to prevent overfitting
AlexNet.add(Dropout(0.4))

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#Output Layer
AlexNet.add(Dense(10))
AlexNet.add(BatchNormalization())
AlexNet.add(Dense(len(folders), activation='softmax'))
```

Training Alexnet Model

```
AlexNet.compile(loss = keras.losses.categorical_crossentropy,
optimizer= 'adam', metrics=['accuracy'])
from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = 'saved_model/'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
# fit the model
alexnet_model = AlexNet.fit(
  training_set,
  validation_data=test_set,
  epochs=15,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set),
  callbacks=[model_checkpoint_callback]
)
```

5. Implementation and Evaluation of Ensemble Networks.

Ensemble is a collection of the above mentioned models, the input image is given to each model and output of each is stored in a list and the majority is regarded as the final outcome for that input Image. Here we implement two techniques of ensemble networks first one is based on voting and second one is based on weighted voting.

Each model created above has its own function within which we import the trained model for that type and pass on our data to it which then returns output for the same.

Example: Resnet Function

Function for Resnet loading and prediction

```
# Resnet Model call
def resnet(img_path, img_size):
  # load all images into a list
  from tensorflow.keras.applications.resnet import preprocess_input
  images_gen = []
  dirs = ['covid/', 'normal/', 'pneumonia/']
  for next_path in dirs:
    next_path = os.path.join(img_path, next_path)
    for img in os.listdir(next_path):
      img = os.path.join(next_path, img)
      img = preprocess_input(get_img_array(img, size=img_size))
      images_gen.append(img)

  model =
keras.models.load_model('/content/drive/MyDrive/db/models/resnet.h5')
  images_gen = np.vstack(images_gen)
  preds = model.predict(images_gen)
  predicted_values = np.argmax(preds,axis=1)
  print('Done...........Resnet')
  return predicted_values
```

We call all our defined functions and save their output in respective variables.

Calling models

```
resnet_predictions = resnet('/content/drive/My Drive/db/test/',(224,
224))
alexnet_predictions = alexnet('/content/drive/My
Drive/db/test/',(150, 150))
densenet_predictions = densenet('/content/drive/My
Drive/db/test/',(224, 224))
nassnet_predictions = nasnet('/content/drive/My Drive/db/test/',(331,
331))
xception_predictions = xception('/content/drive/My
Drive/db/test/',(299, 299))
vgg_predictions = vgg16('/content/drive/My Drive/db/test/',(299,
299))
mymodel_predictions = myModel('/content/drive/My
Drive/db/test/',(224, 224))
```

Then we merge them in a list and for each input we calculate the prediction based on voting and weighted voting algorithm.
Note: We've passed the complete directory of our test data instead of a single image in order to evaluate the ensembles properly.

Creating Ensembles of Model

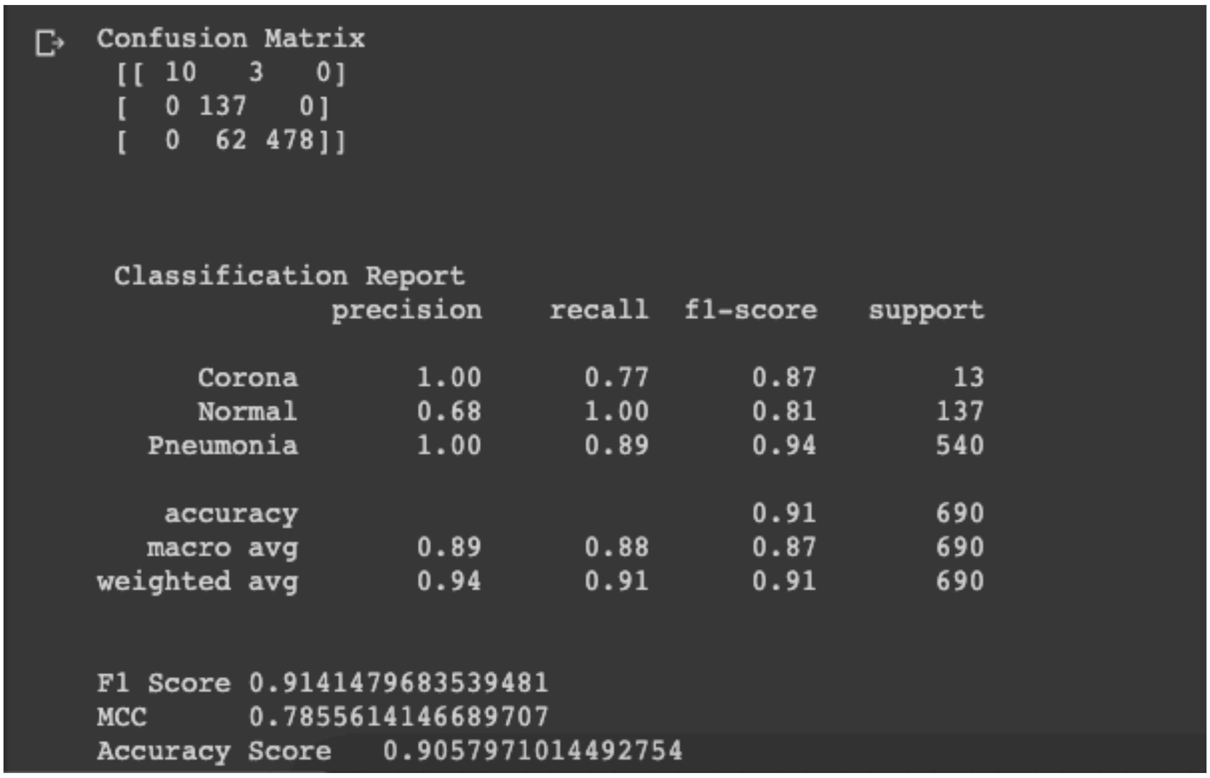
```
model_preds = np.vstack((resnet_predictions, alexnet_predictions,
densenet_predictions, nassnet_predictions, xception_predictions,
vgg_predictions, mymodel_predictions)).T
model_predictions_weights = []
model_predictions = []

for i in model_preds:
  preds = list(i)
  model_predictions.append(max(set(preds), key=preds.count))
  for j in range(len(i)):
    if j==3 or j == 4 or j == 2:
      tmp = i[j]
      i = np.append(i, tmp)
      i = np.append(i, tmp)
  preds_weights = list(i)
  model_predictions_weights.append(max(set(preds_weights),
      key=preds_weights.count))
```

Evaluation

For our ensemble based on voting we get the following metrics

Evaluation Metrics for Voting Based Ensemble Network

```
from sklearn.metrics import classification_report, confusion_matrix,
multilabel_confusion_matrix
from sklearn.metrics import f1_score, accuracy_score,
matthews_corrcoef
target_names = ['Corona', 'Normal', 'Pneumonia']
cm = confusion_matrix(test_set.classes, model_predictions)
print('Confusion Matrix \n', cm)
print('\n\n\n','Classification Report')
print(classification_report(test_set.classes, model_predictions,
target_names=target_names), '\n')
print('F1 Score', f1_score(test_set.classes,
model_predictions_weights, average='weighted'))
print('MCC \t' , matthews_corrcoef(test_set.classes,
model_predictions))
print('Accuracy Score \t', accuracy_score(test_set.classes,
model_predictions))
```



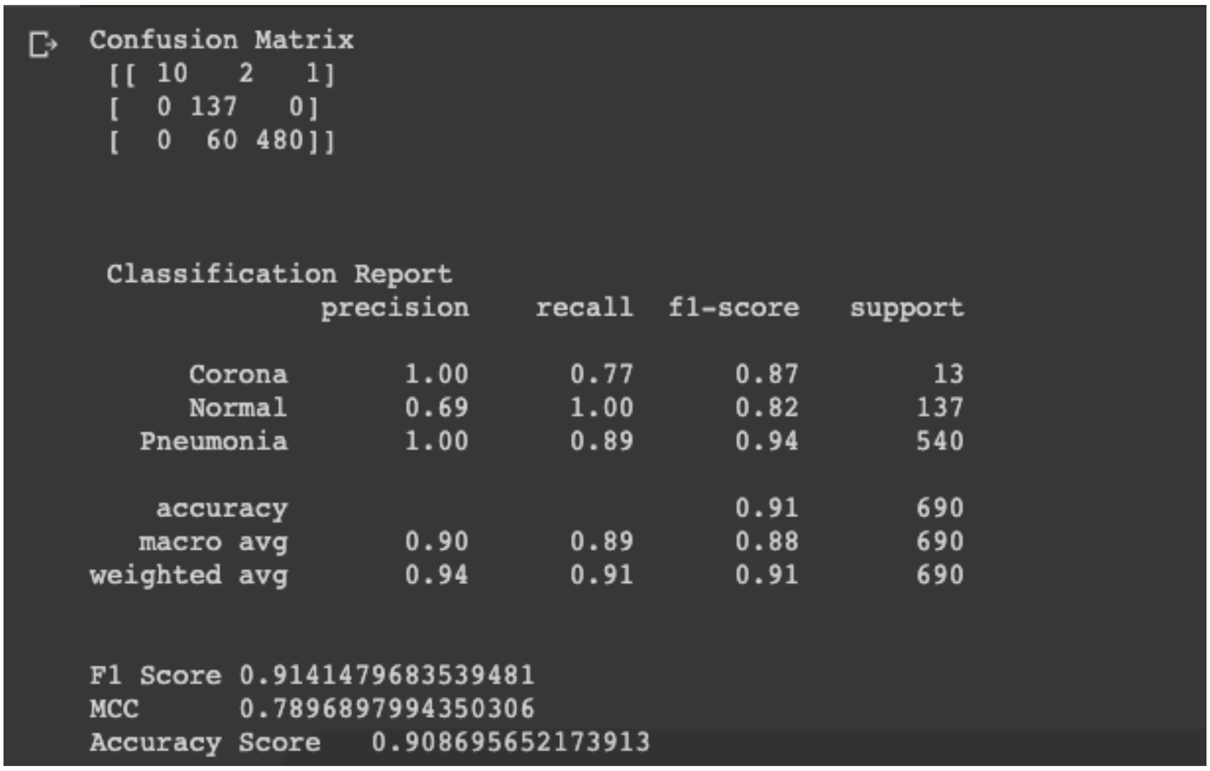
```
Confusion Matrix
 [[ 10   3   0]
 [  0 137   0]
 [  0  62 478]]



 Classification Report
               precision    recall  f1-score   support

      Corona       1.00      0.77      0.87        13
      Normal       0.68      1.00      0.81       137
   Pneumonia       1.00      0.89      0.94       540

    accuracy                           0.91       690
   macro avg       0.89      0.88      0.87       690
weighted avg       0.94      0.91      0.91       690


F1 Score 0.9141479683539481
MCC      0.7855614146689707
Accuracy Score   0.9057971014492754
```

FIGURE 7: Result of Evaluation Metrics Voting Based

For ensemble based on weight increment, we get the following output

Metrics for Weighted Voting Based Ensemble Network

```
cm = confusion_matrix(test_set.classes, model_predictions_weights)
print('Confusion Matrix \n', cm)
print('\n\n\n','Classification Report')
print(classification_report(test_set.classes,
model_predictions_weights, target_names=target_names), '\n')
print('F1 Score',f1_score(test_set.classes,
model_predictions_weights, average='weighted'))
print('MCC \t', matthews_corrcoef(test_set.classes,
model_predictions_weights))
print('Accuracy Score \t', accuracy_score(test_set.classes,
model_predictions_weights))
```



```
Confusion Matrix
 [[ 10   2   1]
 [  0 137   0]
 [  0  60 480]]


 Classification Report
              precision    recall  f1-score   support

      Corona       1.00      0.77      0.87        13
      Normal       0.69      1.00      0.82       137
   Pneumonia       1.00      0.89      0.94       540

    accuracy                           0.91       690
   macro avg       0.90      0.89      0.88       690
weighted avg       0.94      0.91      0.91       690


F1 Score 0.9141479683539481
MCC      0.7896897994350306
Accuracy Score   0.908695652173913
```

FIGURE 8: Result of Evaluation from Weighted Voted Based Ensemble Network

All the code mentioned in the screenshots above are provided with the ICT solution for this project.

Works Cited

Google, n.d. *Collaboratory : - Frequently Asked Questions.* [Online]
Available at: https://research.google.com/colaboratory/faq.html
[Accessed 8 December 2020].

Chollet, F., 2020. *Grad-CAM class activation visualization.* [Online]
Available at: https://keras.io/examples/vision/grad_cam/
[Accessed 10 December 2020].

Kumar, D. V., 2020. *Hands-on Guide To Implementing AlexNet With Keras For Multi-Class Image Classification.* [Online]
Available at: https://analyticsindiamag.com/hands-on-guide-to-implementing-alexnet-with-keras-for-multi-class-image-classification/
[Accessed 10 November 2020].