A picture containing graphical user interface

Description automatically generated **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

A picture containing text, clipart

Description automatically generated

**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 | | |
|  |  | | |

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:** | 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]](#footnote-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 | page13image60933248page13image38216000  On-demand |
| Graphic Card | page13image60947072page13image60947456  TPU and GPU option available  page13image60948032page13image60937856 |
| 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. **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.

Graphical user interface, application

Description automatically generated

**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.

Graphical user interface, application

Description automatically generated

**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

Chart, pie chart

Description automatically generated

**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.

1. **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]](#footnote-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

* 1. 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.

* 1. 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

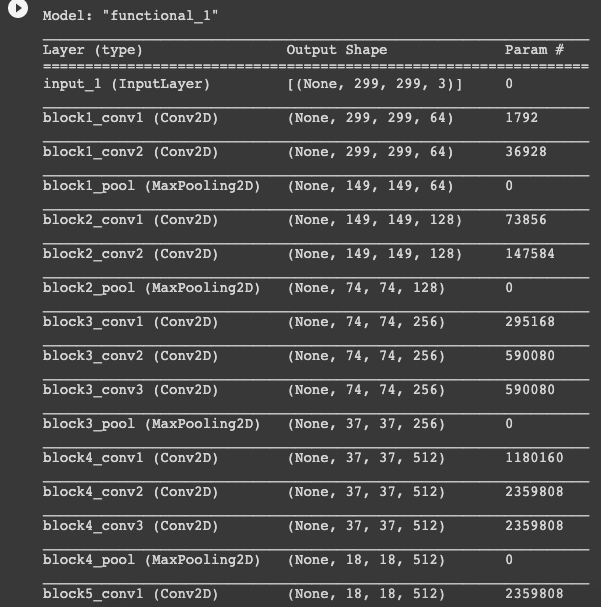
**# Save the entire model as a SavedModel.**

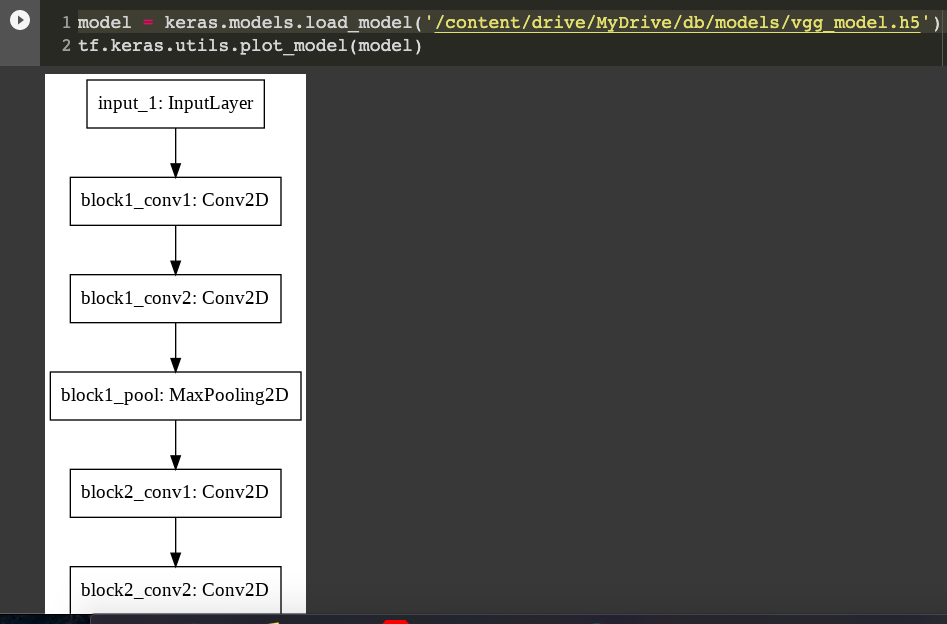
!mkdir -p saved\_model

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.

**FIGURE 4: Summary of VGG 16**



**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(

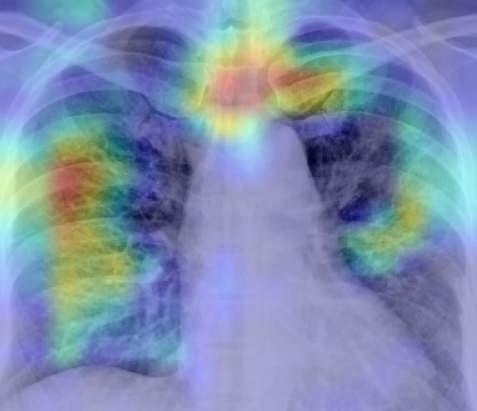
img\_array, model, last\_conv\_layer\_name, classifier\_layer\_names

)

# 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.

* 1. 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]

)

* 1. 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]

)

* 1. 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]

)

* 1. 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]

)

* 1. 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]

)

1. 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))

Calendar

Description automatically generated

**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))

Calendar

Description automatically generated

**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].

1. <https://research.google.com/colaboratory/faq.html> [↑](#footnote-ref-1)
2. <https://keras.io/about/> [↑](#footnote-ref-2)