Configuration Manual

MSc Research Project
Data Analytics

Lavneet Janeja
Student ID: x18199445

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	LAVNEET JANEJA
Student ID:	X18199445
Programme:	MSc. DATA ANALYTICS **Year:** 2019 – 2020
Module:	MSc. RESEARCH PROJECT
Lecturer:	Dr Catherine Mulwa
Submission Due Date:	17/08/2020
Project Title:	Identification of defects in the fabrics using deep Convolutional Neural Networks

Word Count:1077 **Page Count: 14**

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: ………

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

Lavneet Janeja
Student ID: x18199445

1 Introduction

The main objective of carrying out fabric defect detection is to develop a model which is capable of identifying the defects in the fabrics and classify them at a higher detection rate which eventually will boost up the accuracy and make the model more efficient than the state of the art.
The document contains complete step of procedures that needed to be followed for executing five pre-trained models for identifying the defects in fabrics. The manual starts from hardware configurations required for setting up the environment and ends at showing the prediction at the testing part.

2 Setting up hardware

The configuration of the hardware (laptop) which is used for implementing the project is mentioned in figure 1. It is a Lenovo laptop installed with windows 10 operating system having 8 GB of RAM and has an intel i5-9300H processor with a processing speed of 2.4 GHz.

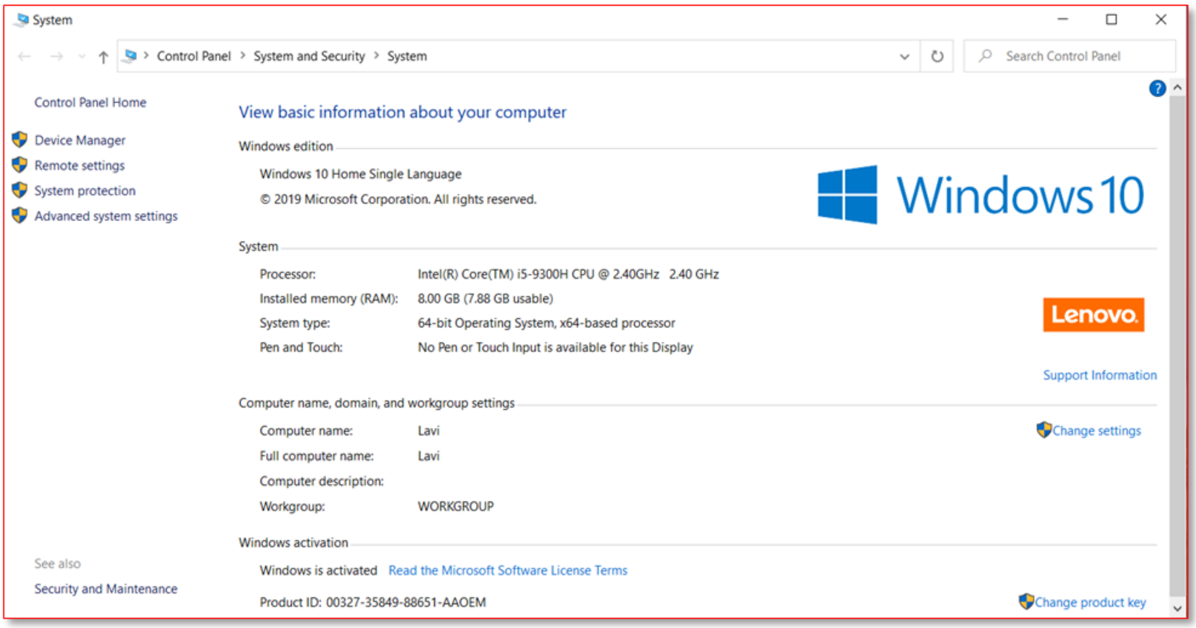


Figure 1: Hardware configuration

3 Setting up the environment

Two environments were used for implementing the models viz.

a. Spyder (Anaconda 3)
b. Google Colaboratory

First we need to install Anaconda3 using Anaconda navigator figure2. We have strictly used only conda environment in the command line terminal and have not used pip3 anywhere.

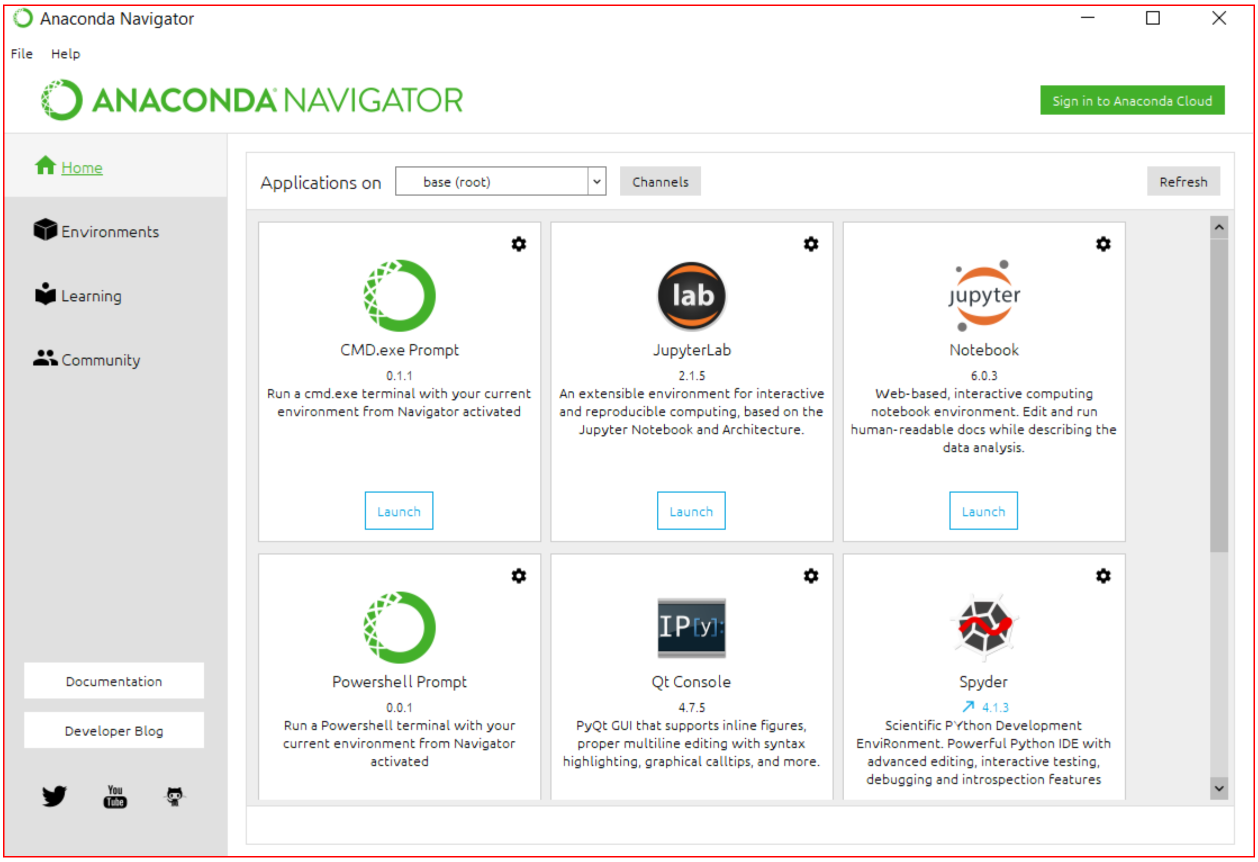


Figure 2: Overview of anaconda navigator

Now we install and launch spyder for running python. We have used python 3.7.7. Once the spyder is installed we create a new environment in spyder to install packages only for tensorflow and keras figure 3.

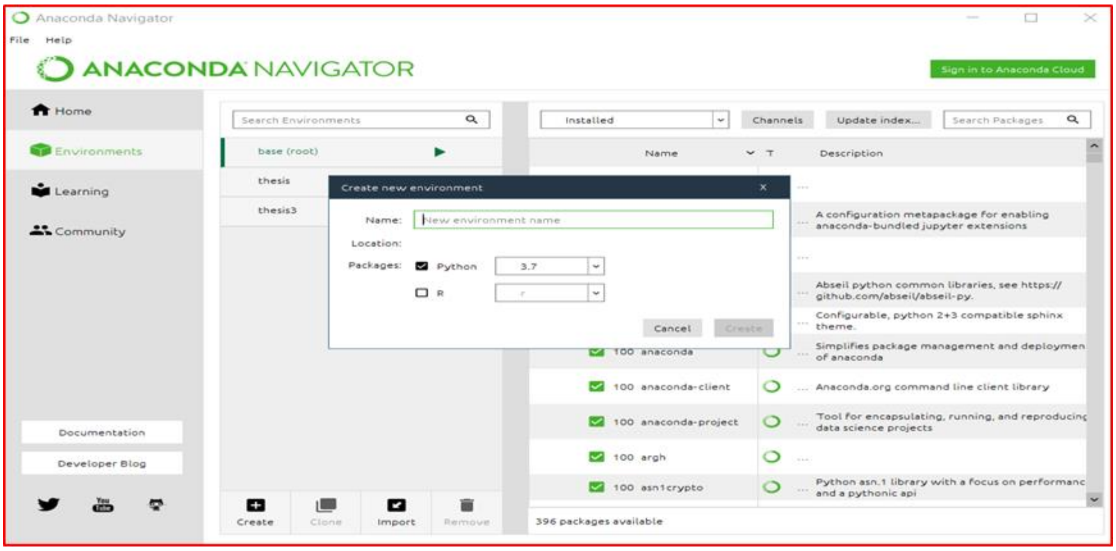
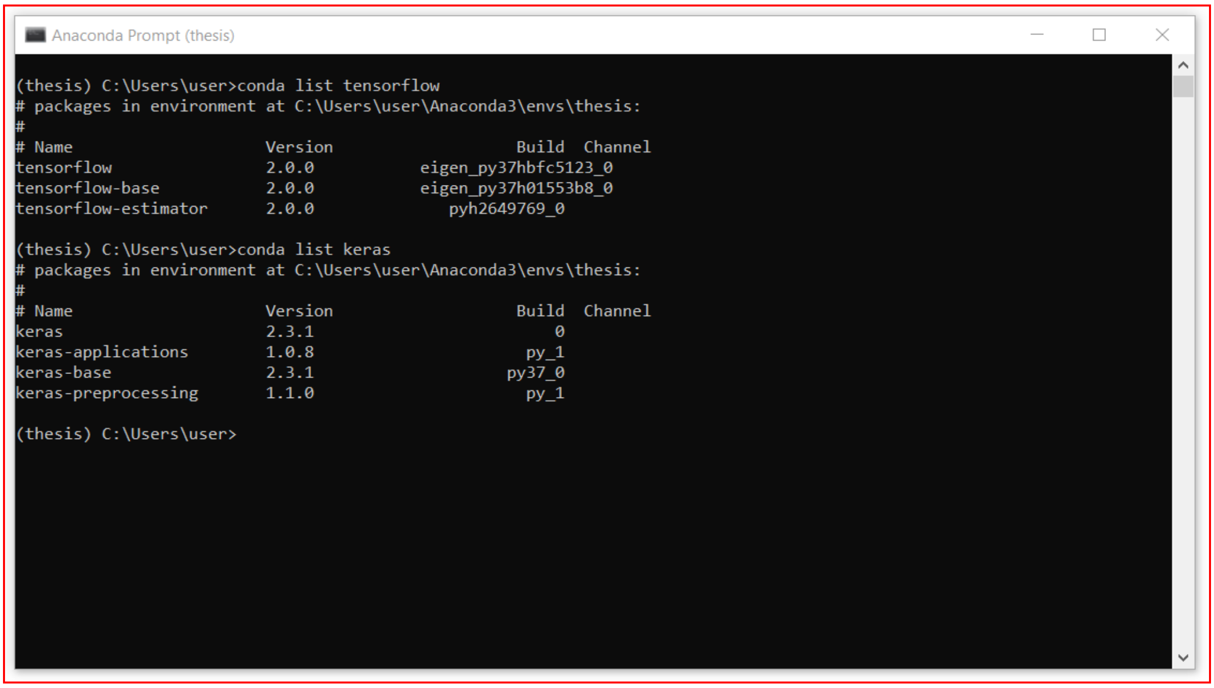


Figure 3: Setting up a new environment

We created a new environment with the name '**thesis**' and installed all necessary classes and libraries. Two of the most important packages that needed to be installed were Tensorflow and Keras figure 4.



```
Anaconda Prompt (thesis)

(thesis) C:\Users\user>conda list tensorflow
# packages in environment at C:\Users\user\Anaconda3\envs\thesis:
#
# Name                    Version                   Build  Channel
tensorflow                2.0.0           eigen_py37hbfc5123_0
tensorflow-base           2.0.0           eigen_py37h01553b8_0
tensorflow-estimator      2.0.0              pyh2649769_0

(thesis) C:\Users\user>conda list keras
# packages in environment at C:\Users\user\Anaconda3\envs\thesis:
#
# Name                    Version                   Build  Channel
keras                     2.3.1                         0
keras-applications        1.0.8                      py_1
keras-base                2.3.1                    py37_0
keras-preprocessing       1.1.0                      py_1

(thesis) C:\Users\user>
```

Figure 4: Installing Tensorflow amd Keras

4. Data Source

The dataset was fetched from www.kaggle.com

5. Code Implementation

Before implementing the codes, all the necessary packages are needed to be imported in the spyder environment for all the five models [1] [2]. Out of five models, four models (VGG16, VGG19, MobileNet and DCCNN) use tensorflow at the backend and keras at the frontend. Packages required to be imported for running these four models are mentioned in figure 5.

```
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import pickle
import pandas as pd
from tensorflow.keras.applications.xception import Xception, preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, Conv2D
from tensorflow.keras.layers import MaxPooling2D, Flatten, Add
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import add
from torchvision import datasets, transforms, models
import cv2
import multiprocessing as mp
import pickle
```

Figure 5: Importing the libraries for VGG16, VGG19, MobileNet and DCCNN

On the other hand only AlexNet uses only keras without using tensorflow at the backend. The packages needed to be imported for this model is mentioned in figure 6.

```python
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import pickle
import pandas as pd
from tensorflow.keras.applications.xception import Xception, preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import add
from torchvision import datasets, transforms, models
import cv2
import multiprocessing as mp
import pickle
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.layers import Input, Flatten, Dense
from keras.models import Model
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import numpy as np

```

Figure 6: Importing libraries for AlexNet

Once libraries and classes are imported then we can download the data from the below url “https://www.kaggle.com/mhskjelvareid/dagm-2007-competition-dataset-optical-inspection”. Then it is imported into python and rescaled using training and validation datagen by using the following code in figure 7 (It is valid for all the five models).

```python
dataset_images = 'Train'
dataset_test = 'Test'
#testing_data = datasets.ImageFolder('test', transform=transform)


train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        dataset_images,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        dataset_test,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

```

Figure 7: Loading images to train and test generator

Then we move on to training and testing phase, the top convolutional layers of each of the model needs to be freezed so that there weights don't get changed/ manipulated while training the model with our dataset. Figure 8 illustrates how the weights of MobileNet (picked at random out of 5 models) are freezed by running a for loop and iterating through all the layers of the model and making the training layers to be false.

```
img_rows, img_cols = 224, 224

# Reload mobilenet without the top or FC layers
MobileNet = MobileNet(weights = 'imagenet',
                      include_top = False,
                      input_shape = (img_rows, img_cols, 3))

#Here we freeze the last 4 layers as its trainable as true by default
for layers in MobileNet.layers:
    layers.trainable = False
```

Figure 8: Setting convolutional blocks to false

Then from which ever layer we want the training to start happening again, we just need to mention the name of the layer to be true inside layer.name. For Example in VGG16 and VGG19 we want the training to resume from the first convolution of the fifth block. "block5_conv1", as depicted in figure 9.

```
for layer in model_vgg19_conv.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Figure 9: Setting the threshold till where the convolutional blocks would be false

One additional step is required in DCCNN i.e. to explicitly extract the features of VGG16 (the shallow channel) so that it can be further clubbed to three convolutional blocks (figure 10).

```python
def extract_features1(imageList):
    model =  VGG16(weights='imagenet', include_top=False, pooling = 'avg')
    features = list()
    for img in imageList:
        image = Image.fromarray(img, 'RGB')
        image = image.resize((224,224))
        image = np.expand_dims(image, axis=0)
        image = preprocess_input(image)
        feature = model.predict(image)
        features.append(feature)
    features = np.array(features)
    features = features.reshape((len(imageList), 512))
    return features
```

Figure 10: Extracting features for DCCNN

Next we define an Image Data_Generator for parsing a set of 100 images in a batch along with the label whether it is defected or not (Figure 11).

```python
def data_generator(image_generator):
    while 1:
        temp = image_generator.next()
        #ftrs = extract_features1(temp[0])
        yield ([temp[0]], temp[1])
```

Figure 11: Defining Image_Data_Generator

Finally we define the model. For VGG16, MobileNet and VGG19 it is only five line code where we define the input shape of the image, the actual model to be defined, then the model is flattened, added dense layer to it and finally complied (figure 12).

```python
input = Input(shape=(224,224,3),name = 'image_input')

output_vgg19_conv = model_vgg19_conv(input)



x = Flatten(name='flatten')(output_vgg19_conv)
x = Dense(4096, activation='relu', name='fc1')(x)
x = Dense(2048, activation='relu', name='fc2')(x)
x = Dense(1, activation='sigmoid', name='predictions')(x)

#Create your own model
my_model = Model(input=input, output=x)



my_model.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
#In the summary, weights and layers from VGG part will be hidden, but they will be fit during the training
my_model.summary()
```

Figure 12: Defining the model for VGG16

For AlexNet the model is not available inside the Keras package so we have to write the whole set of architecture block instead (figure 13).

```
#Instantiate an empty model
model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(filters=96, input_shape=(224,224,3), kernel_size=(11,11), strides=(4,4), padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))

# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1), padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))

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

# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation('relu'))

# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))

# Passing it to a Fully Connected layer
model.add(Flatten())
# 1st Fully Connected Layer
model.add(Dense(4096, input_shape=(224*224*3,)))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))

# 2nd Fully Connected Layer
model.add(Dense(2048))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))

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

# Output Layer
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary()
```

Figure 13: Defining model for AlexNet

Similarly as DCCNN is our developed model, so we had to define a model with whole set of block by combining VGG16 with another channel of convolutions. In figure 14, we can see the whole set of architectural coding block.

```
# define the architectural model block
def define_model():
    #cnn model
    inputs11 = Input(shape=(224, 224, 3))
    layer1 = Conv2D(32, (3,3), activation='relu')(inputs11)
    layer2 = MaxPooling2D((2,2))(layer1)
    layer3 = Conv2D(64, (3,3), activation='relu')(layer2)
    layer4 = MaxPooling2D((2,2))(layer3)
    layer5 = Conv2D(128, (3,3), activation='relu')(layer4)
    layer6 = MaxPooling2D((2,2))(layer5)
    layer7 = Conv2D(128, (3,3), activation='relu')(layer6)
    layer8 = MaxPooling2D((2,2))(layer7)
    layer9 = Flatten()(layer8)
    layer10 = Dropout(0.5)(layer9)
    layer11 = Dense(256, activation='relu')(layer10)
    # features from VGG CNN model squeezed from 2048 to 256 nodes
    inputs21 = Input(shape=(512,))
    fe21 = Dropout(0.5)(inputs21)
    fe22 = Dense(256, activation='relu')(fe21)
    #combining both cnn models
    decoder = add([layer11, fe22])
    decoder2 = Dropout(0.5)(decoder)
    decoder3 = Dense(256, activation='relu')(decoder2)
    decoder4 = Dense(128, activation='relu')(decoder3)
    outputs = Dense(1, activation='sigmoid')(decoder4)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs11, inputs21], outputs=outputs)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
    # summarize model
    print(model.summary())
    plot_model(model, to_file='model_fabric.png', show_shapes=True)
    return model

model = define_model()
```

Figure 14: Defining model for DCCNN

Finally after defining and compiling the models, they were trained upon 10 epochs with a batch size of 100. The steps per epochs were validation set (4995) divided with the batch size (100). The models were trained using model.fit_generator() function as depicted in figure 15. Once the models were trained the training and validation accuracy / loss were dumped in a pickle file.

```
epochs = 10
batch = 100
# steps_per_epoch = 1006 // batch
steps_per_epoch = 4995 // batch
#os.mkdir("C:/Users/user/Desktop/Moodle/Research project/DCCNN")
for i in range(epochs):
    training_generator = data_generator(train_generator)
    testing_generator = data_generator(test_generator)
    mp.set_start_method('spawn', force=True)
    history = model.fit_generator(training_generator, validation_data = testing_generator, epochs=epochs, steps_per_epoch= steps_per_epoch,
    model.save("C:/Users/user/Desktop/Moodle/Research project/DCCNN/model_TUE_" + str(i) + ".h5")

pickle.dump(history.history, open('history_DCCNN.pkl', 'wb'))
```

Figure 15: Training the model

After the model is trained the validation and training accuracy / loss needs to be visualized using matplotlib as shown in figure 16.

```
import matplotlib.pyplot as plt
plt.plot(training_acc, label='TRAINING ACCURACY')
plt.plot(validation_acc, label='VALIDATION ACCURACY')
plt.legend()
plt.show()

plt.plot(validation_loss, label='VALIDATION LOSS')
plt.plot(training_loss, label='TRAINING LOSS')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['acc'], label= 'testing loss')
plt.plot(history.history['accuracy'])
plt.legend()
plt.show()
```

Figure 16: Plotting loss and accuracy using matplotlib

The accuracy and loss function are visualized in the form of line chart. We just randomly took two visualizations from two different models (AlexNet- showing accuracy and MobileNet- showing loss) in figure 17 and figure 18 respectively.

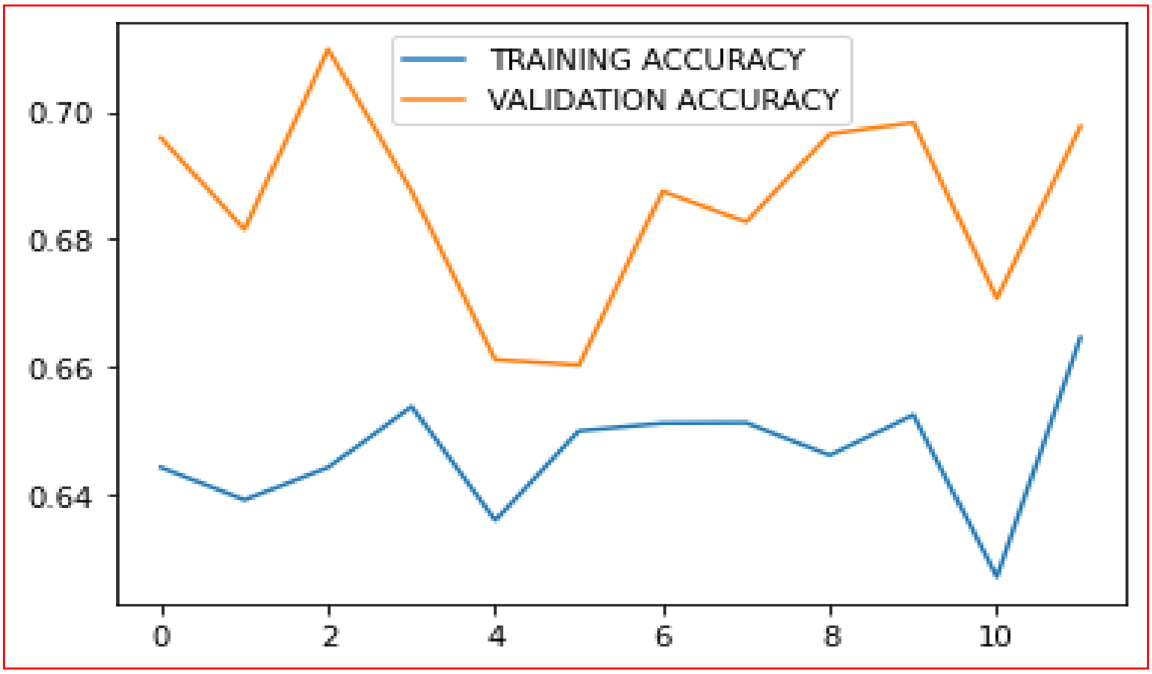


Figure 17: Accuracy graph of AlexNet

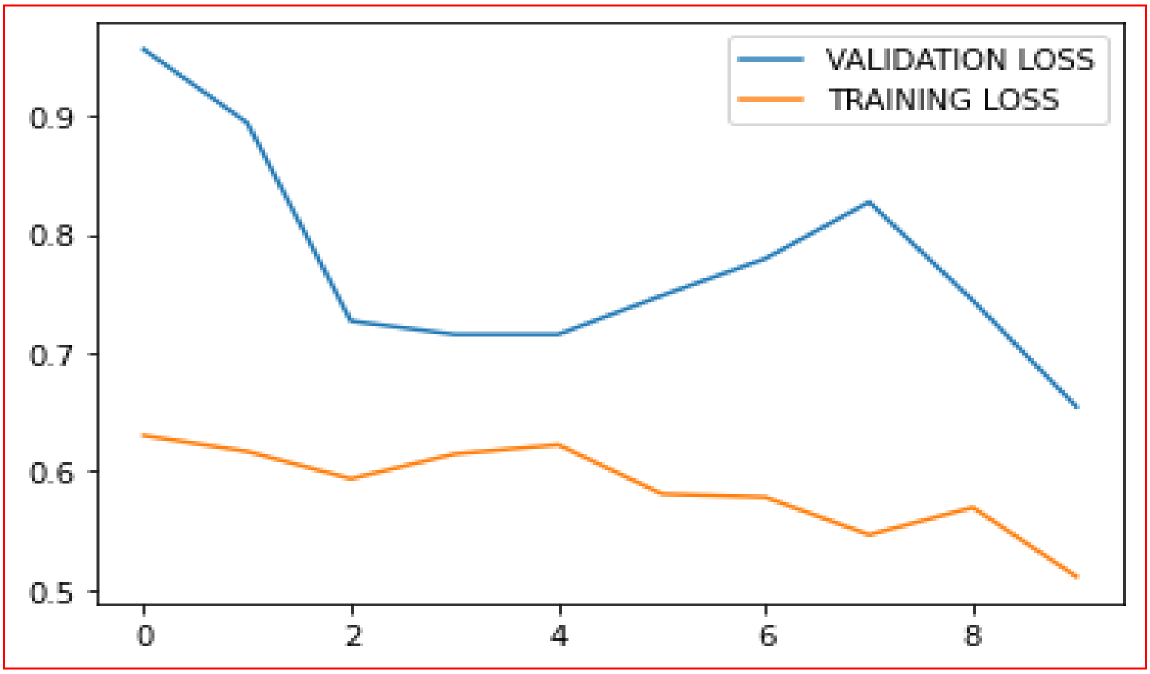


Figure 18: Loss graph of MobileNet

6. Testing

After training the model if you have closed the environment then you need to extract the features again testing purpose. Once the features are extracted again, then we are going to predict the result of the model. An example of DCCNN is shown that how do we make the prediction on the basis of features extracted (shown in figure 19).

```python
#-----------------------------------------------------------------------------
#TESTING



def extract_features1(img):
    model =  VGG16(weights='imagenet', include_top=False, pooling = 'avg')
    image = Image.fromarray(img, 'RGB')
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    feature = model.predict(image)
    return feature

def generate_prediction(model, photo1, img):
    pred = model.predict((img, photo_vgg), verbose=0)
    return pred

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        dataset_test,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

model = load_model('C:/Users/user/Desktop/Moodle/Research project/DCCNN/model_1.h5')



lst = []
for i in tqdm(range(13)):
    a = next(test_generator)
    for j in range(len(a[1])):
        photo_vgg = extract_features1(a[0][j])
        description = generate_prediction(model, photo_vgg, a[0][j].reshape((1,224,224,3)))
        lst.append((description, a[1][j]))
print("\n\n")
print(lst)
```

Figure 19: Extracting features for testing and implementing the prediction

Finally a confusion matrix is plotted with a threshold of 0.5 to check that how efficient our model managed to predict the results (Figure 20).

```python

#set threshold here
threshold = 0.5
predList = list()
predprobList = list()
trueList = list()
for i in lst:
    predprobList.append(np.ravel(i[0])[0])
    trueList.append(i[1])
predList = np.array(predprobList)>threshold
#confusion matrix
from sklearn.metrics import confusion_matrix
cm  = confusion_matrix(trueList,predList)
tn, fp, fn, tp = cm.ravel()
```

Figure 20: Creating a confusion matrix

References

[1] https://keras.io/preprocessing/image/

[2] Ketkar, Nikhil. (2017). Introduction to Keras. 10.1007/978-1-4842-2766-4_7.