Configuration Manual

MSc Research Project
Data Analytics

Arpita Mitra
Student ID: x21116211

School of Computing
National College of Ireland

Supervisor: Bharat Agarwal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Arpita Mitra
Student ID:	x21116211
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Bharat Agarwal
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	539
Page Count:	11

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:	31st January 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

Arpita Mitra
x21116211

1 Introduction

The scripts employed in this study have certain instructions that must be followed, which are outlined in the configuration manual. You will find that following the instructions in this tutorial will help you successfully execute the code. In addition to that, this documentation includes information on the hardware design of the machine on which the code was implemented. In addition to this, the system's most basic configuration requirements are also listed.

2 System Specification

The below table 1 and 2 are listed with all the hardware and software specifications for the platform upon which the research work is being carried out.

2.1 Hardware Specification

Hardware used	Specification
Processor	Intel(R) Core(TM) i7 8th generation
RAM	16 GB
Hard Drive	1 TB

Table 1: Hardware Specification

2.2 Software Specification

Software used	Version
Operating System	Windows-10 Home, 64 bit Operating System
Anaconda Navigator	2022.05
Python	3.9.12
Power Bi	2.100.1401.0

Table 2: Software Specification

3 Environment Set Up

The entire research was done by writing Python code in Jupyter Notebook on Anaconda Navigator. First, Anaconda Navigator needs to be installed.

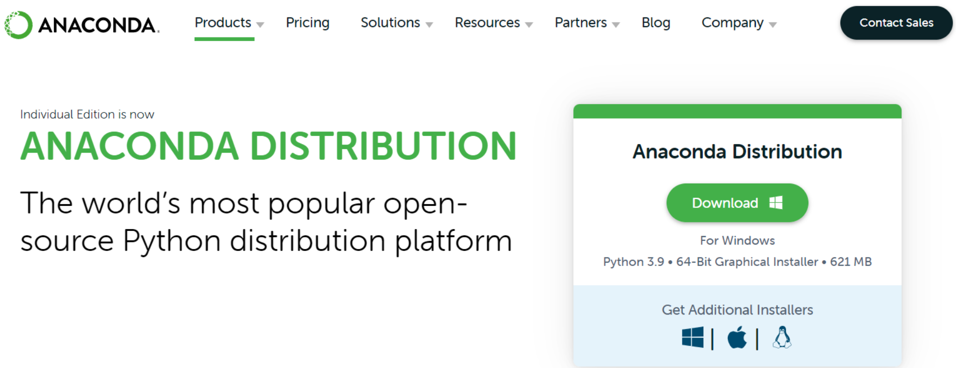

Figure 1: Anaconda Navigator Installation

The Anaconda Navigator can be accessed through Start, then select Anaconda Navigator after the installation has been completed successfully. Then Jupyter notebook can be launched from Anaconda Navigator.

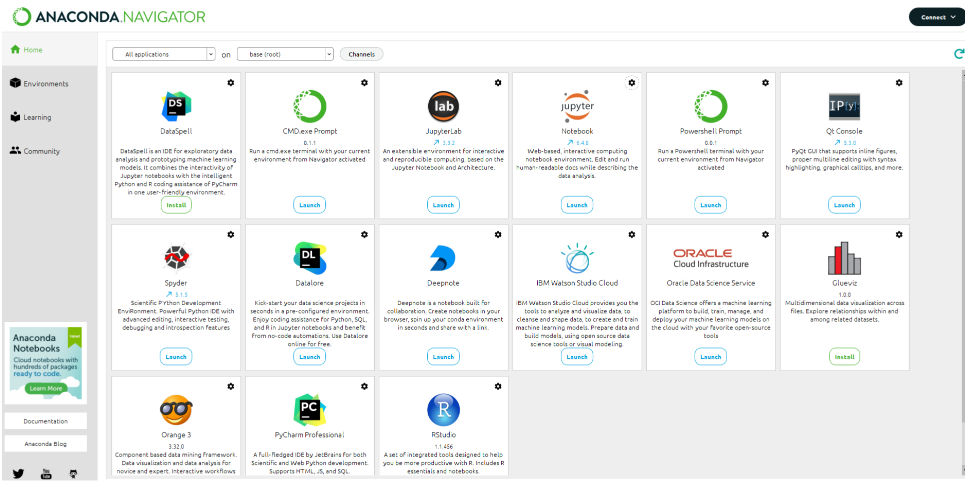

Figure 2: Jupyter Access

4 Data Source

Dataset has been collected from Kaggle. This research project has been carried out using the dataset [1] created by Ali et al. (2022) which was made publicly available by the authors for future research.

[1] https://www.kaggle.com/datasets/nafin59/monkeypox-skin-lesion-dataset?datasetId=2308447

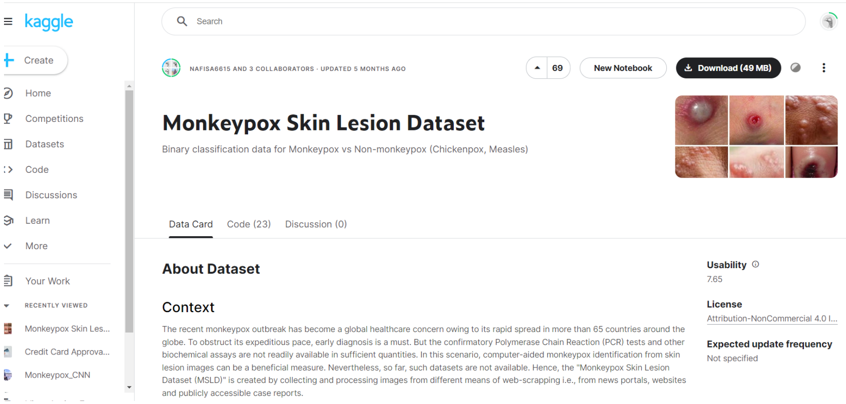

Figure 3: Monkeypox Dataset

The dataset has been downloaded in a zip format. Further, it was unzipped in the local drive and then the entire research was performed using the same dataset.

5 Implementation

The design of this study, as well as its execution, makes use of the following libraries mentioned in table 3 that has to be installed to conduct this research study.

Libraries used	Version
Numpy	1.21.5
Pandas	1.4.2
Scikit-learn	1.0.2
Keras	2.11.0
TensorFlow	2.11.0
Matplotlib	3.5.1

Table 3: Libraries used in this study

The study procedure, along with its methodology, will be broken down into its component parts in the following sections.

5.1 Blocks of Code:

Import Libraries

The below screenshot depicts all the necessary libraries that are used in this study.

```python
# Import relevant libraries and packages
import glob
import os
from PIL import Image
from PIL import ImageFilter
import pandas as pd
import numpy as np
import shutil
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scikitplot as skplt
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, load_model
from keras.callbacks import EarlyStopping
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import optimizers
from keras.applications.vgg19 import VGG19
from sklearn.metrics import confusion_matrix, precision_score, recall_score
import matplotlib.pyplot as plt
np.random.seed(123)
import itertools
from sklearn.metrics import f1_score
```

Figure 4: Import Libraries

Data Pre-processing

The below block of code creates the train, validation and test folders inside the root directory of each class of images and then data was shuffled randomly and split into 70:15:15 ratio, and then it was stored in its respective directories.

```python
#Funtion to shuffle and split the data, and then store them in respective directories

def get_value_split():

    # CREATE TRAIN/ VAL/ TEST FOLDERS INSIDE THE ROOT DIRECTORY OF EACH CLASS
    root_dir = 'C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images'
    monkeypox = '/Monkeypox_augmented'
    others = '/Others_augmented'

    os.makedirs(root_dir +'/train' + monkeypox)
    os.makedirs(root_dir +'/train' + others)
    os.makedirs(root_dir +'/val' + monkeypox)
    os.makedirs(root_dir +'/val' + others)
    os.makedirs(root_dir +'/test' + monkeypox)
    os.makedirs(root_dir +'/test' + others)

    # DATA PARTITION AFTER SHUFFLING THE DATA RANDOMLY (into 70, 15, 15)
    for partitions in [monkeypox, others]:
        src = "C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images" + partitions

        allFileNames = os.listdir(src)
        np.random.shuffle(allFileNames)
        train_FileNames, validation_FileNames, test_FileNames = np.split(np.array(allFileNames),
                                                          [int(len(allFileNames)*0.7),
                                                           int(len(allFileNames)*0.85)])

        # FOLDER NAMES
        train_FileNames = [src+'/'+ name for name in train_FileNames.tolist()]
        validation_FileNames = [src+'/' + name for name in validation_FileNames.tolist()]
        test_FileNames = [src+'/' + name for name in test_FileNames.tolist()]

        #COUNT OF IMAGES OF EACH CLASS IN RESPECTIVE FOLDERS AFTER PARTITION
        print('Total images: ', len(allFileNames))
        print('Training: ', len(train_FileNames))
        print('Validation: ', len(validation_FileNames))
        print('Testing: ', len(test_FileNames))

        # COPYPASTE THE IMAGES
        for name in train_FileNames:
            shutil.copy(name, "C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/train"+partitions)

        for name in validation_FileNames:
            shutil.copy(name, "C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/val"+partitions)

        for name in test_FileNames:
            shutil.copy(name, "C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/test"+partitions)
```

```python
get_value_split() #RUN ONLY ONCE TO  CREATE THE FOLDERS AND RANDOMLY DIVIDE THE IMAGES INTO THE FOLDERS
```

Figure 5: Shuffle & split the data

To create more complexity in the model, more blurry images were added to the existing data.

```
#CRAETION OF MORE BLURRED AND UNSHARPENED IMAGES OF MONKEYPOX IN TRAINING DATA

os.chdir('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/train/Monkeypox_augmented')
filelist = glob.glob("*.jpg")
count = 0
for imagefile in filelist:
    img = Image.open(imagefile)
    img = img.convert("RGB")
    img_blur = img.filter(ImageFilter.GaussianBlur)
    img_unsharp = img.filter(ImageFilter.UnsharpMask)
    img_blur.save(str(count) + 'bl_' + imagefile)
    img_unsharp.save(str(count) + 'un_' + imagefile)
```

```
#CRAETION OF MORE BLURRED AND UNSHARPENED IMAGES OF NON-MONKEYPOX IN TRAINING DATA
os.chdir('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/train/Others_augmented')
filelist = glob.glob("*.jpg")
count = 0
for imagefile in filelist:
    img = Image.open(imagefile)
    img = img.convert("RGB")
    img_blur = img.filter(ImageFilter.GaussianBlur)
    img_unsharp = img.filter(ImageFilter.UnsharpMask)
    img_blur.save(str(count) + 'bl_' + imagefile)
    img_unsharp.save(str(count) + 'un_' + imagefile)
```

Figure 6: Adding blurry images to dataset

CNN Model

CNN Model building

```
#GET THE MODEL TO OUTPUT 3D FEATURE MAPS (HEIGHT, WIDTH, FEATURES)
model = Sequential()

# Layer 1
model.add(Conv2D(32, (3, 3), input_shape=(228, 228, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 2
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 3
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# APPLY THE FLATTENING FUNCTION TO CONVERT 3D FEATURE MAPS INTO 1D FEATURE VECTORS
model.add(Flatten())

# ADD 2 FINAL DENSE LAYERS TO ADD A CLASSIFIER TO THE CONVOLUTIONAL BASE
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# COMPILE THE MODEL
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
# PRINTING THE MODEL SUMMARY
print(model.summary())
```

Figure 7: CNN Model

Data Augmentation & Training of CNN Model

Data Augmentation & Training of the model

```
# DEFINE BATCH SIZE & TARGET SIZE
batch_size = 50 # batch size defines how many images will  be sent over the network at a time
target_size = (228, 228) #image size

# CONFIGURE AUGMENTATION FOR TRAINING ADVERSITY

# ImageDataGenerator rescales the pixels between zero and one
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# SET AUGMENTATION FOR TESTING
test_datagen = ImageDataGenerator(rescale=1./255)

# READ PICTURES IN TRAINING DIRECTORY AND GENERATE BATCHES OF IMAGE DATA
train_generator = train_datagen.flow_from_directory(
        'C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/train',
        target_size=target_size,
        batch_size=batch_size,
        class_mode='binary')

# SAME GENERATOR AS ABOVE, BUT FOR VALIDATION DATA
validation_generator = test_datagen.flow_from_directory(
        'C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/val',
        target_size=target_size,
        batch_size=batch_size,
        class_mode='binary')
```

```
Found 6699 images belonging to 2 classes.
Found 479 images belonging to 2 classes.
```

```
# SET STOP TO 5 EPOCHS TO PREVENT OVERFITTING OF THE MODEL
callback = EarlyStopping(
    monitor='val_acc',
    restore_best_weights=True,
    patience=5
)
```

```
# SAVE WEIGHTS AFTER IMPLEMETING CALLBACK TO LATER COMPARE MODEL
history = model.fit(
        train_generator,
        steps_per_epoch=6699 // batch_size, #6699 IS THE TRAIN_GENERATOR RESULT
        epochs=30,  # NUMBER OF EPOCH MEANS THE RUNNING SPEED OF THE MODEL
        validation_data=validation_generator,
        validation_steps=479 // batch_size, #479 IS VALIDATION_GENERATOR RESULT
        callbacks=[callback]
)
# CREATE DIRECTORY TO SAVE RESULTS OF THE MODEL
os.mkdir('C:/Users/arpit/Model_Results')
model.save('Model_Results/model_01.h5')
```

Figure 8: Training of CNN model

Training of the CNN model will take ˜ 3 hours to complete.

Validation of CNN Model

Validation of CNN Model

```
# VALIDATION GENERATOR WITH BATCH SIZE = 1
validation_generator = test_datagen.flow_from_directory(
        'C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/val',
        target_size=target_size,
        batch_size=1,
        class_mode='binary')

# EXAMPLES FROM VALIDATION SET AND LABELS PREDICTED FROM BASELINE CNN MODEL
for _ in range(5):
    x = next(validation_generator)[0]
    print('Monkeypox_augmented' if model.predict(x)[0][0] > 0.5 else 'Others_augmented')
    plt.imshow(x.reshape((228, 228, 3)))
    plt.show()
    print()
```

Figure 9: Validation of CNN model

VGG 19 Model

VGG 19 Model

```python
# Build VGG19 structure
cnn_base = VGG19(weights='imagenet',
                 include_top=False,
                 input_shape=(228, 228, 3))
print('VGG19 model is loaded')
print(cnn_base.summary())
```

Figure 10: Structure of VGG 19 Model

The bottlenecked characteristics of the model was retained after the extraction process, and a classifier comprising final dense layers has been appended to the model. Using the below blocks of code, features and labels had been extracted and applied to train, validation and test set and the same was saved in a form of Numpy array for later use.

Feature Extraction

```python
# Build extraction function to get features and labels
def feature_extraction(directory, sample_amount):
    features = np.zeros(shape=(sample_amount, 7, 7, 512))
    labels = np.zeros(shape=(sample_amount))
    datagen = ImageDataGenerator(rescale=1./255)
    generator = datagen.flow_from_directory(
        directory, target_size=(228, 228),
        batch_size = batch_size,
        class_mode='binary')

    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = cnn_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i = i + 1
        if i * batch_size >= sample_amount:
            break
    return features, labels
```

Train the model

```python
# Apply extraction function to 3 datasets
train_features, train_labels = feature_extraction(training_folder, train_samples)
validation_features, validation_labels = feature_extraction(validation_folder, validation_samples)
test_features, test_labels = feature_extraction(testing_folder, test_samples)
```

Figure 11: Feature Extraction

```python
# Save the extracted features and labels in a directory
os.mkdir('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked')
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/train_features.npy', train_feature
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/train_labels.npy', train_labels)
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/validation_features.npy', validati
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/validation_labels.npy', validation_
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/test_features.npy', test_features)
np.save('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/test_labels.npy', test_labels)
```

Figure 12: Saving Extracted Features

VGG 19 Models Validation

VGG-19 Model 1

Validation

VGG-19 Model 1 (with 2 dense layers)

```
# Build classifier on top of  VGG19
model = Sequential()

# Add dense layers on top of VGG19
# 1
model.add(Dense(256, activation='relu', input_dim=reshape_y))
# 2
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=50,
                    validation_data=(validation_features, validation_labels))

# Save VGG19 results
model.save('Model_Results/model_VGG_01.h5')
```

Figure 13: VGG 19 Model 1

Previously extracted saved features were reloaded to use in other models as they have dropout layers appended to them.

VGG-19 Model 2

VGG-19 Model 2 (3 dense layers + 1 Dropout layer)

```
count = 2
```

```
# Deeper VGG19 network
model = Sequential()
# 1
model.add(Dense(256, activation='relu', input_dim=train_features.shape[1]))
# 2
model.add(Dropout(0.2))
# 3
model.add(Dense(64, activation='relu'))
# 4
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=50,
                    validation_data=(validation_features, validation_labels))
model.save(f'Model_Results/model_VGG_0{count}.h5')
pd.DataFrame(history.history).plot(figsize=(5, 5))
plt.title(f'Pre-Trained Training Performance: Model {count}')
plt.xlabel('Epoch')
plt.ylabel('Metric')
count += 1
plt.show()
```

Figure 14: VGG 19 Model 2

VGG-19 Model 3

VGG-19 Model 3 (Reduced learning rate 1e-2)

```
# Reduce learning rate to 1e-2
model = Sequential()
# 1
model.add(Dense(256, activation='relu', input_dim=train_features.shape[1]))
# 2
model.add(Dropout(0.2))
# 3
model.add(Dense(64, activation='relu'))
# 4
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=1e-2),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=50,
                    validation_data=(validation_features, validation_labels))
model.save(f'Model_Results/model_VGG_0{count}.h5')
pd.DataFrame(history.history).plot(figsize=(5, 5))
plt.title(f'Pre-Trained Training Performance: Model {count}')
plt.xlabel('Epoch')
plt.ylabel('Metric')
count += 1
plt.show()
```

Figure 15: VGG 19 Model 3

VGG-19 Model 4

VGG-19 Model 4 (with Adam Optimizer)

```
# Try adam optimizer
model = Sequential()
# 1
model.add(Dense(256, activation='relu', input_dim=train_features.shape[1]))
# 2
model.add(Dropout(0.2))
# 3
model.add(Dense(64, activation='relu'))
# 4
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=50,
                    validation_data=(validation_features, validation_labels))
model.save(f'Model_Results/model_VGG_0{count}.h5')
pd.DataFrame(history.history).plot(figsize=(5, 5))
plt.title(f'Pre-Trained Training Performance: Model {count}')
plt.xlabel('Epoch')
plt.ylabel('Metric')
count += 1
plt.show()
```

Figure 16: VGG 19 Model 4

VGG-19 Model 5

VGG-19 Model 5 (with 2 dense layers + 1 Dropout layer)

```
# Shallower network
model = Sequential()
# 1
model.add(Dense(256, activation='relu', input_dim=train_features.shape[1]))
# 2
model.add(Dropout(0.2))
# 3
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=5e-6),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=50,
                    validation_data=(validation_features, validation_labels))
model.save(f'Model_Results/model_VGG_0{count}.h5')
pd.DataFrame(history.history).plot(figsize=(5, 5))
plt.title(f'Pre-Trained Training Performance: Model {count}')
plt.xlabel('Epoch')
plt.ylabel('Metric')
count += 1
plt.show()
```

Figure 17: VGG 19 Model 5

Testing the Models

```
# Print scores of baseline CNN model usine ImagedataGenerator
model_baseline.evaluate(test_generator,
                        steps=480 // 50)
```

```
9/9 [==============================] - 6s 703ms/step - loss: 0.2413 - accuracy: 0.9000

[0.24128085374832153, 0.8999999761581421]
```

```
# Get pretrained test features containing the weights
test_features = np.load('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/test_features.npy')
test_labels = np.load('C:/Users/arpit/Research/Research_Data/Augmented Images/Augmented Images/bottlenecked/test_labels.npy')
```

```
# Print model scores
for model_name, model in models_VGG.items():
    print(model_name  + ' Evaluation')
    print(model.evaluate(test_features, test_labels))
    print()
```

```
model_VGG_01.h5 Evaluation
15/15 [==============================] - 1s 11ms/step - loss: 0.2298 - acc: 0.9187
[0.22976429760456085, 0.918749988079071]

model_VGG_02.h5 Evaluation
15/15 [==============================] - 0s 10ms/step - loss: 0.3039 - acc: 0.9000
[0.3039029836654663, 0.8999999761581421]

model_VGG_03.h5 Evaluation
15/15 [==============================] - 0s 7ms/step - loss: 0.3260 - acc: 0.8917
[0.3260074257850647, 0.8916666507720947]

model_VGG_04.h5 Evaluation
15/15 [==============================] - 0s 6ms/step - loss: 0.3645 - acc: 0.9000
[0.3644777536392212, 0.8999999761581421]

model_VGG_05.h5 Evaluation
15/15 [==============================] - 0s 6ms/step - loss: 0.2900 - acc: 0.8979
[0.2900329828262329, 0.8979166746139526]
```

Figure 18: Model Testing

References

Ali, S. N., Ahmed, M. T., Paul, J., Jahan, T., Sani, S. M. S., Noor, N. and Hasan, T. (2022). Monkeypox skin lesion detection using deep learning models: A preliminary feasibility study, *arXiv preprint arXiv:2207.03342* .