Advanced Weapon Detection and Classification Using Fine-Tuned Transfer Learning Models

MSc Research Project
Data Analytics

Brandon Craig D’souza
Student ID: x23100125

School of Computing
National College of Ireland

Supervisor: Prof. Abdul Qayum

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Brandon Craig D'souza

Student ID: x23100125

Programme: Msc Data Analytics **Year:** 2023-24

Module: Msc Research Project

Supervisor: Prof. Abdul Qayum

Submission Due Date: 12/08/2024

Project Title: Advanced Weapon Detection and Classification Using Fine-Tuned Transfer Learning Models

Word Count: 1643 **Page Count** 21

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: Brandon Craig D'souza

Date: 12/08/2024

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):	

Advanced Weapon Detection and Classification Using Fine-Tuned Transfer Learning Models

Brandon Craig D’souza
Student ID: x23100125

1 Environment requirements

The configuration manual discusses all the hardware and software requirements for the research work. This will help anyone who needs to replicate the project making it easy to follow the instructions.

2 System specifications

2.1 Hardware requirements

The hardware requirements necessary to run the project is below:

- **Processor:** Intel Core i7.
- **System Memory:** 256 GB SSD + 1 TB HDD
- **RAM:** 32 GB

2.2 Software requirements

The software requirements necessary to run the project are discussed below:

- **Windows Version:** Windows 11.
- **Development Environment and software:** Google colab, Visual Studio Code.

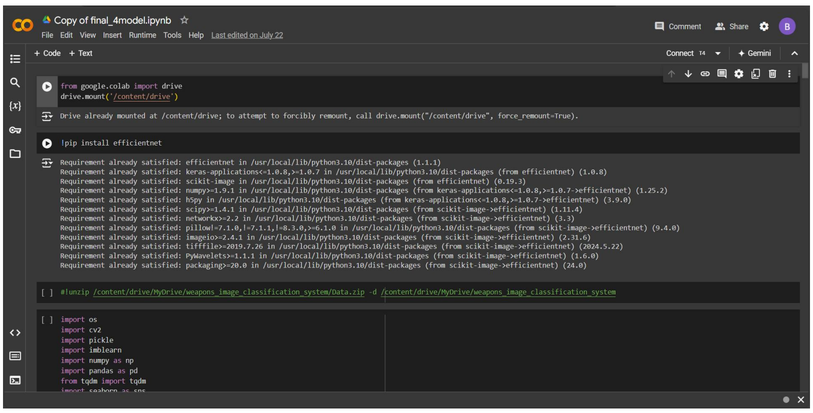


Figure 1: Google colab

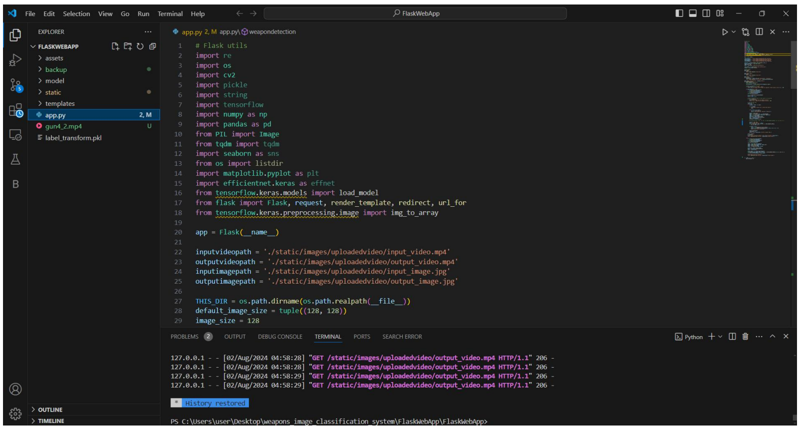


Figure 2: Visual Studio code

- **Programming/Scripting language:** Python
- **Cloud Storage:** Google Drive

2.3 Setting up Google Golab

If a user doesnt have Google Colab setup on their Google drive then follow the steps below:

- Go to https://www.google.com/drive/ and login.
- Click on new from the left pane.
- Click on More.
- Search for Google Colaboratory and select.
- If it isn't available please search in more apps and then search for Google Colaboratory.
- Create a new folder named as 'weapons_image_classification_system'
- Upload and place the 'x23100125.ipynb' in the above created folder.
- Press Run all to run the entire project seemlessly.

2.4 Setting up Visual Studio Code

- Go to https://code.visualstudio.com/download and download VScode for you Required system.
- Install and open VSCode.
- Install latest Python from https://www.python.org/downloads/ .
- In VSCode, search for the python library extension and download it.

3 Dataset Information

- The dataset is got from https://images.cv/dataset-categories/weapons for knife, pistol, rifles and swords.
- For easier usage , the data is clubbed into one data file as 'Data.zip' which can be downloaded from the following link https://drive.google.com/file/d/178WKRPuIcDXGW-YUsfburuWsgBTDEI83/view?usp=drive_link
- Download and place this zip in the drive folder created.
- Run the following code snippet to unzip the Data.zip file in google drive:

```
unzip /content/drive/MyDrive/weapons_image_classification_system/Data.zip -d /content/drive/MyDrive/weapons_image_classification_system
```

Figure 3: Unzip Dataset file

3.1 Libraries

```
import os
import cv2
import pickle
import imblearn
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from os import listdir
from keras import backend as K
from keras.models import Model
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from keras.utils import plot_model
from keras.models import Sequential
import efficientnet.keras as effnet
from keras.layers import MaxPooling2D
from imblearn.over_sampling import SMOTE
from tensorflow.keras.layers import Input
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Lambda, multiply
from tensorflow.keras.applications.xception import Xception
from sklearn.metrics import precision_recall_fscore_support
from keras.layers import Activation, Flatten, Dropout, Dense
from tensorflow.keras.preprocessing.image import img_to_array
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from sklearn.metrics import classification_report,confusion_matrix
from keras.layers import Conv2D, MaxPool2D, BatchNormalization, AveragePooling2D, GlobalAveragePooling2D
```

Figure 4: Libraries

4 Project Implementation

4.1 Setting up essential parameters

```
image_size=128
default_image_size = tuple((128, 128))
labels = os.listdir('/content/drive/MyDrive/weapons_image_classification_system/Data')
directory_root = '/content/drive/MyDrive/weapons_image_classification_system/Data'
classes = len(labels)
BATCHZ_SIZE=64
```

Figure 5: Variable initialization

From the above Figure 5, image size and default images are resized to 128x128 pixels. ‘labels’ will list the contents of the directory . ‘directory_root’ will sets the root directory for the dataset.’classes’ will be the number of classes used in the dataset by counting the number of ‘labels’. the ‘batch_size’ is the number of samples that will be processed in each training and that is set to 64.

4.2 Loading Dataset using OpenCV

```
def image_array(dir):
    try:
        img = cv2.imread(dir)
        if img is not None :
            img = cv2.resize(img, default_image_size)
            return img_to_array(img)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

```
image_list, label_list = [], []

try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for classes_folder in root_dir :
        image_classes_folder_list = os.listdir('/content/drive/MyDrive/weapons_image_classification_system/Data')

    for classes_folder in image_classes_folder_list :
            # remove .DS_Store from list
            if classes_folder == ".DS_Store" :
                image_classes_folder_list.remove(classes_folder)

    for  image_classes_folder in image_classes_folder_list:
            print(f"[INFO] Processing { image_classes_folder} ...")
            image_classes_image_list = listdir(f"{directory_root}/{ image_classes_folder}/")

            for single_image_classes_image in image_classes_image_list :
                if single_image_classes_image == ".DS_Store" :
                    image_classes_image_list.remove(single_image_classes_image)

            for image in image_classes_image_list[:2500]:
                image_directory = f"{directory_root}/{ image_classes_folder}/{image}"
                if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True or image_directory.endswith(".png") == True or image_directory.endswith(".PNG") == True:
                    image_list.append(image_array(image_directory))
                    label_list.append( image_classes_folder)
    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")
```

```
[INFO] Loading images ...
[INFO] Processing knife ...
[INFO] Processing sword ...
[INFO] Processing rifle ...
[INFO] Processing pistol ...
[INFO] Image loading completed
```

Figure 6: Dataset Loading with OpenCV

This will load, pre-process and label images from a the specified directory in order to prepare it for training the machine learning model that will be done later.

4.3 EDA or Data Visualization

```
#Data Visualization
import matplotlib.image as mpimg
plt.figure(figsize = (10, 10))
image_count = 1
BASE_URL = '/content/drive/MyDrive/weapons_image_classification_system/Data/'
for directory in os.listdir('/content/drive/MyDrive/weapons_image_classification_system/Data'):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(3, 2, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory)
```

Figure 7: Data Visualization

This will get a sample image from each category in the dataset. This will be plotted in a 3x2 grid using Matplotlib. The output is shown in Figure 8 below.

Figure 8: Data Visualization Output

4.4 Count Plot and balancing the data using SMOTE .

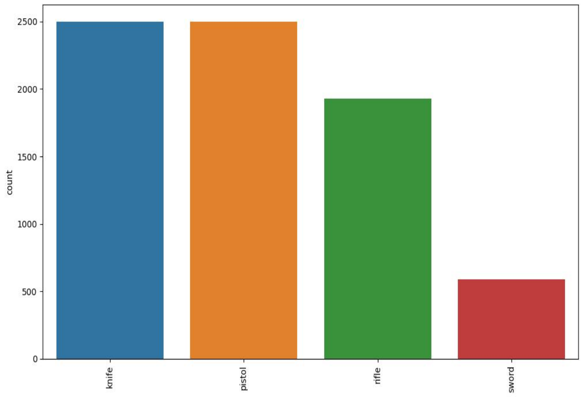


Figure 9: Visualizations of Count of Classes

As it can be seen from Figure 9 that the dataset doesn’t contain an equal number of images for each class. Handling imbalances is crucial when training any Deep learning model especially when there is a need for every class to be trained properly (Chawla et al., 2002). Hence by applying SMOTE oversampling as seen in Figure 10, one can avoid the issue of imbalance and increase the performance the model.

```
[ ] #data balancing
    X = X.reshape(-1, image_size * image_size * 3)
    X.shape
    oversample = SMOTE()
    X, Y = oversample.fit_resample(X, Y)
    X = X.reshape(-1,image_size,image_size,3)
```

Figure 10: SMOTE oversampling

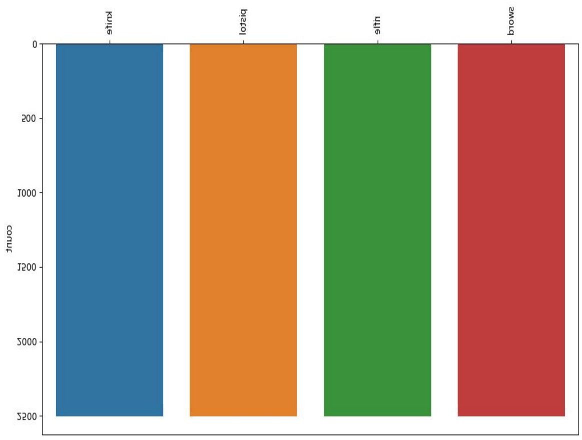


Figure 11: After Sampling/Data Balancing

By the use of SMOTE oversampling, all the 4 classes now have equal number of sample images which can be seen in the Figure 11.

```
[ ] class_labels = LabelBinarizer()
    Y = class_labels.fit_transform(Y)
    pickle.dump(class_labels,open('/content/drive/MyDrive/weapons_image_classification_system/label_transform.pkl', 'wb'))
    n_classes = len(class_labels.classes_)

[ ] cls = len(class_labels.classes_)
    print(class_labels.classes_)

    ['knife' 'pistol' 'rifle' 'sword']
```

Figure 12: LabelBinarizer

The LabelBinarizer converts the categorical values of the labels like 'knife','pistol','rifle' and 'sword' into numerical values that will be suitable for the ML model.

4.5 Splitting the data into Training, Validation and testing:

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1)

[ ] X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, stratify=y_train)

[ ] X_train.shape
```

Figure 13: Data splitting

From the Figure 13 , we can see that the data will be split in the ratio of 80:10:10 respectively.

4.6 Deep Learning models used:

A) CNN Model:

```
#CNN
depth=3
model = Sequential()
inputShape = (image_size, image_size, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, image_size, image_size)
    chanDim = 1
model.add(Conv2D(128, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(cls))
model.add(Activation("sigmoid"))
model.summary()
```

Figure 14: CNN model

Figure 14 is the initialization and also for defining the architecture for the CNN model by making the use of Keras library(He et al., 2016) .

Model Summary:

```
+ Code   + Text
dropout_1 (Dropout)          (None, 14, 14, 64)       0

conv2d_3 (Conv2D)            (None, 14, 14, 64)       36928

activation_3 (Activation)    (None, 14, 14, 64)       0

batch_normalization_3 (Bat   (None, 14, 14, 64)       256
chNormalization)

conv2d_4 (Conv2D)            (None, 14, 14, 32)       18464

activation_4 (Activation)    (None, 14, 14, 32)       0

batch_normalization_4 (Bat   (None, 14, 14, 32)       128
chNormalization)

max_pooling2d_2 (MaxPoolin   (None, 4, 4, 32)         0
g2D)

dropout_2 (Dropout)          (None, 4, 4, 32)         0

flatten (Flatten)            (None, 512)              0

dense (Dense)                (None, 1024)             525312

activation_5 (Activation)    (None, 1024)             0

batch_normalization_5 (Bat   (None, 1024)             4096
chNormalization)

dropout_3 (Dropout)          (None, 1024)             0

dense_1 (Dense)              (None, 4)                4100

activation_6 (Activation)    (None, 4)                0

=================================================================
Total params: 815524 (3.11 MB)
Trainable params: 812644 (3.10 MB)
Non-trainable params: 2880 (11.25 KB)
```

Figure 15: CNN model summary

This gives the layer by layer summary of the CNN model which shows the type , output shape and number of parameters of each layer and which of them can be trained and untrained.

Model Training:

```
history = model.fit(x=X_train, y=y_train, batch_size=64, epochs=10, validation_data=(X_val, y_val))

Epoch 1/10
127/127 [==============================] - 43s 185ms/step - loss: 1.2871 - accuracy: 0.5521 - val_loss: 1.4329 - val_accuracy: 0.5644
Epoch 2/10
127/127 [==============================] - 20s 159ms/step - loss: 0.9021 - accuracy: 0.6672 - val_loss: 0.7426 - val_accuracy: 0.7044
Epoch 3/10
127/127 [==============================] - 19s 152ms/step - loss: 0.7534 - accuracy: 0.7151 - val_loss: 0.6214 - val_accuracy: 0.7511
Epoch 4/10
127/127 [==============================] - 19s 149ms/step - loss: 0.6783 - accuracy: 0.7419 - val_loss: 0.9027 - val_accuracy: 0.6800
Epoch 5/10
127/127 [==============================] - 19s 152ms/step - loss: 0.6034 - accuracy: 0.7702 - val_loss: 0.6559 - val_accuracy: 0.7456
Epoch 6/10
127/127 [==============================] - 19s 151ms/step - loss: 0.5840 - accuracy: 0.7757 - val_loss: 0.5859 - val_accuracy: 0.7822
Epoch 7/10
127/127 [==============================] - 19s 152ms/step - loss: 0.5282 - accuracy: 0.7969 - val_loss: 0.5604 - val_accuracy: 0.7933
Epoch 8/10
127/127 [==============================] - 19s 152ms/step - loss: 0.4766 - accuracy: 0.8180 - val_loss: 0.5078 - val_accuracy: 0.8078
Epoch 9/10
127/127 [==============================] - 19s 150ms/step - loss: 0.4425 - accuracy: 0.8367 - val_loss: 0.5135 - val_accuracy: 0.8067
Epoch 10/10
127/127 [==============================] - 19s 153ms/step - loss: 0.4119 - accuracy: 0.8442 - val_loss: 0.6329 - val_accuracy: 0.7756
```

Figure 14: Fitting the model

The CNN model will be trained over 10 epochs which displays the training as well as validation datasets at each epoch.

Accuracy and loss Graph:

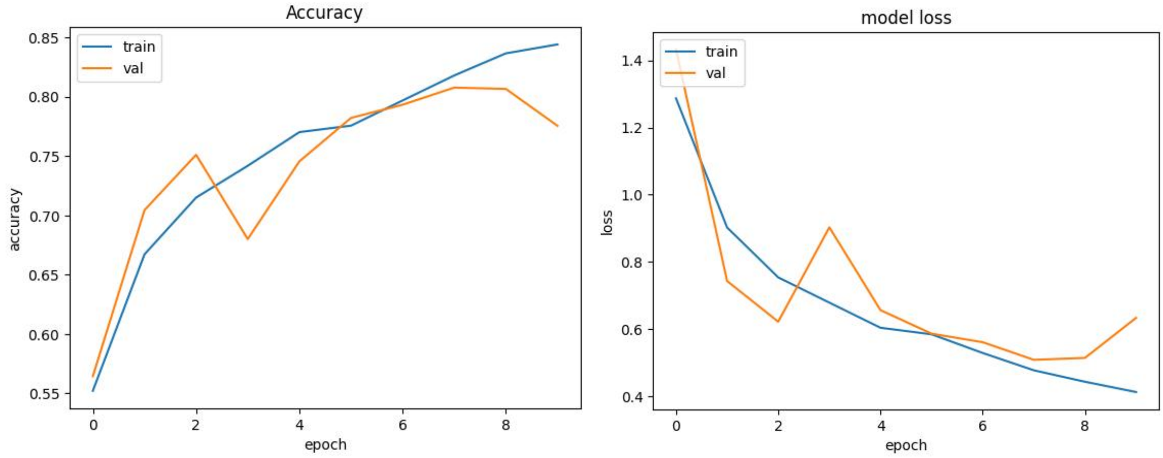


Figure 15: Plotting accuracy and loss Graphs

Two graphs are plotted to show the training and validation accuracy as well as model loss over the epochs.

Confusion Matrix:

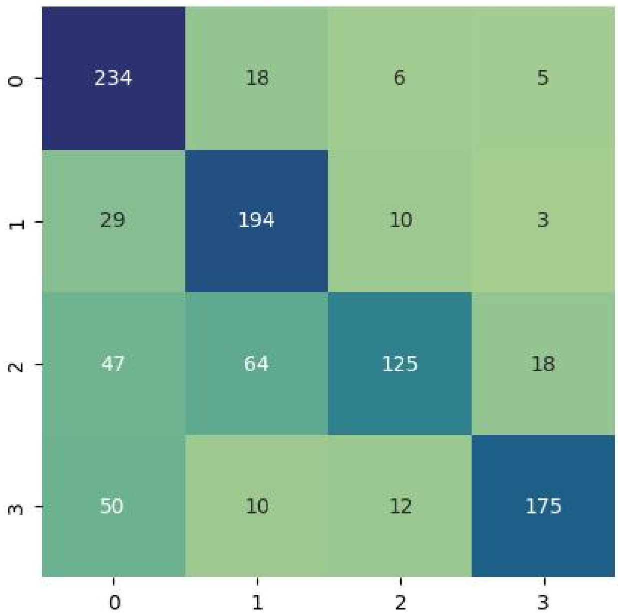


Figure 16: Confusion matrix for CNN model

Figure 16 shows the confusion matrix for the CNN model. The matrix shows the information of the actual classes vs predicted classes.

Classification report, Specitivity and Sensitivity:

```
print(classification_report(y_test_new,pred))

              precision    recall  f1-score   support

           0       0.65      0.89      0.75       263
           1       0.68      0.82      0.74       236
           2       0.82      0.49      0.61       254
           3       0.87      0.71      0.78       247

    accuracy                           0.73      1000
   macro avg       0.75      0.73      0.72      1000
weighted avg       0.75      0.73      0.72      1000
```

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(cls):
    prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                      pred==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.829037	0.889734
1	1	0.879581	0.822034
2	2	0.962466	0.492126
3	3	0.965471	0.708502

Figure 17: Model Output

Figure 17 shows the evaluation of the CNN model by various metrics such as sensitivity, specificity, precision, recall, and F1-score

B) Xception Model:

```
# Xception Backbone
xception = Xception(input_shape=(image_size,image_size,3), weights='imagenet', include_top=False)

# Freeze the model weights
xception.trainable = True

# The Xception Model baseline
model = Sequential([
    xception,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(n_classes, activation='softmax')
])

# Compile the Baseline
model.compile(
    loss='categorical_crossentropy',
    optimizer="adam",
    metrics=['accuracy']
)
```

Figure 18: Xception Model

In Figure 18, this code initializes the Xception model with pre-trained weights from ImageNet (Chollet, 2017).

Model Training:

```
Epoch 1/10
127/127 [==============================] - 80s 357ms/step - loss: 0.3748 - accuracy: 0.8677 - val_loss: 1.3771 - val_accuracy: 0.7189
Epoch 2/10
127/127 [==============================] - 40s 314ms/step - loss: 0.1450 - accuracy: 0.9493 - val_loss: 0.2659 - val_accuracy: 0.9300
Epoch 3/10
127/127 [==============================] - 40s 314ms/step - loss: 0.1120 - accuracy: 0.9638 - val_loss: 0.3642 - val_accuracy: 0.8900
Epoch 4/10
127/127 [==============================] - 40s 314ms/step - loss: 0.0704 - accuracy: 0.9768 - val_loss: 0.3508 - val_accuracy: 0.9200
Epoch 5/10
127/127 [==============================] - 40s 315ms/step - loss: 0.0616 - accuracy: 0.9786 - val_loss: 0.2135 - val_accuracy: 0.9311
Epoch 6/10
127/127 [==============================] - 40s 314ms/step - loss: 0.0567 - accuracy: 0.9814 - val_loss: 0.4087 - val_accuracy: 0.9100
Epoch 7/10
127/127 [==============================] - 40s 315ms/step - loss: 0.0494 - accuracy: 0.9836 - val_loss: 0.2345 - val_accuracy: 0.9411
Epoch 8/10
127/127 [==============================] - 40s 315ms/step - loss: 0.0438 - accuracy: 0.9859 - val_loss: 0.4023 - val_accuracy: 0.9100
Epoch 9/10
127/127 [==============================] - 40s 315ms/step - loss: 0.0394 - accuracy: 0.9879 - val_loss: 0.3110 - val_accuracy: 0.9222
Epoch 10/10
127/127 [==============================] - 40s 316ms/step - loss: 0.0235 - accuracy: 0.9914 - val_loss: 0.3450 - val_accuracy: 0.9244
```

Figure 19: Fitting the Xception model

The Xception model will be trained over 10 epochs which displays the training as well as validation datasets at each epoch.

Accuracy and loss Graph:

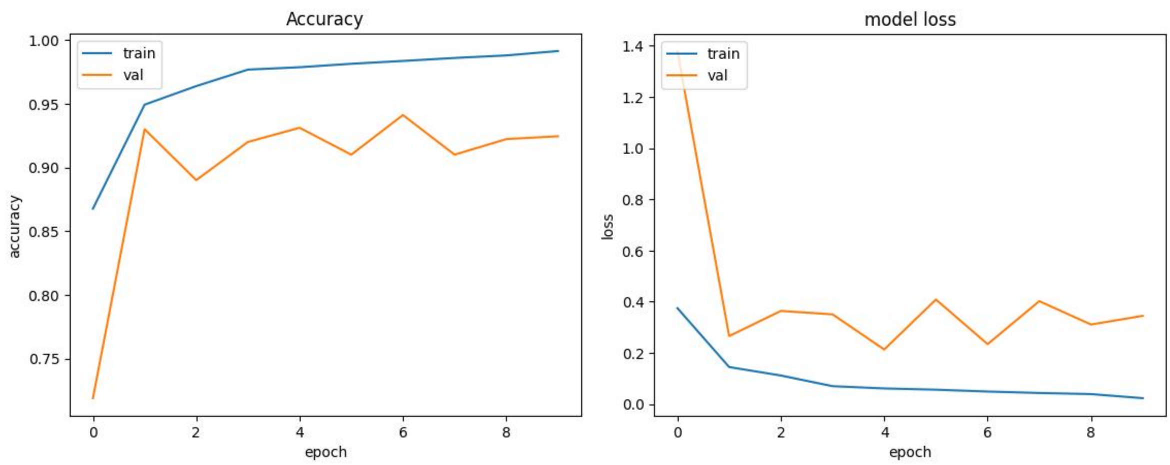


Figure 20: Accuracy and loss Graph for Xception Model

In Figure 20, the two graphs are plotted to show the training and validation accuracy as well as model loss over the epochs.

Confusion Matrix:

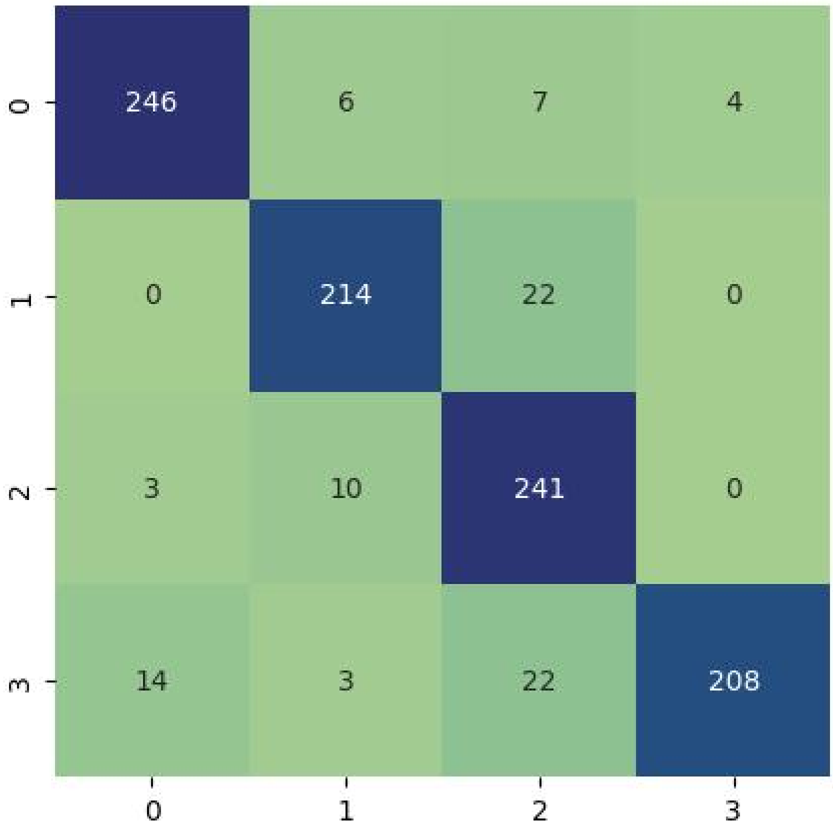


Figure 22: Confusion matrix for Xception Model

FIgure 22 shows the confusion matrix of the Xception model on the actual classes versus the predicted classes.

Classification report, Specitivity and Sensitivity:

```
print(classification_report(y_test_new,pred))

              precision    recall  f1-score   support

           0       0.94      0.94      0.94       263
           1       0.92      0.91      0.91       236
           2       0.83      0.95      0.88       254
           3       0.98      0.84      0.91       247

    accuracy                           0.91      1000
   macro avg       0.92      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```

```
[ ] from sklearn.metrics import precision_recall_fscore_support
    res = []
    for l in range(cls):
        prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                          pred==l,
                                                          pos_label=True,average=None)
        res.append([l,recall[0],recall[1]])

    pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.976934	0.935361
1	1	0.975131	0.906780
2	2	0.931635	0.948819
3	3	0.994688	0.842105

Figure 23:Xception Model Output

Figure 23 shows the evaluation of the Xcception model by various metrics such as sensitivity, specificity, precision, recall, and F1-score

C) EfficientNet B2 Model:

```
[ ] base_model = effnet.EfficientNetB2(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
    model = base_model.output
    model = GlobalAveragePooling2D()(model)
    model = Dense(cls, activation='softmax')(model)
    model = Model(inputs = base_model.input, outputs=model)

    Downloading data from https://github.com/Callidior/keras-applications/releases/download/efficientnet/efficientnet-b2_weights_tf_dim_ordering_tf_kernels_autoaugment_notop.h5
    31936256/31936256 [==============================] - 2s 0us/step

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

Figure 24:EfficientNet B2 Model

Figure 24 is the initialization and also for defining the architecture for the EfficientNet B2 model (Tan and Le, 2019).

Model Summary:

```
block7b_se_expand (Conv2D)   (None, 1, 1, 2112)   187968   ['block7b_se_reduce[0][0]']

block7b_se_excite (Multipl   (None, 4, 4, 2112)   0        ['block7b_activation[0][0]',
y)                                                          'block7b_se_expand[0][0]']

block7b_project_conv (Conv   (None, 4, 4, 352)    743424   ['block7b_se_excite[0][0]']
2D)

block7b_project_bn (BatchN   (None, 4, 4, 352)    1408     ['block7b_project_conv[0][0]']
ormalization)

block7b_drop (FixedDropout   (None, 4, 4, 352)    0        ['block7b_project_bn[0][0]']
)

block7b_add (Add)            (None, 4, 4, 352)    0        ['block7b_drop[0][0]',
                                                            'block7a_project_bn[0][0]']

top_conv (Conv2D)            (None, 4, 4, 1408)   495616   ['block7b_add[0][0]']

top_bn (BatchNormalization   (None, 4, 4, 1408)   5632     ['top_conv[0][0]']
)

top_activation (Activation   (None, 4, 4, 1408)   0        ['top_bn[0][0]']
)

global_average_pooling2d_1   (None, 1408)         0        ['top_activation[0][0]']
 (GlobalAveragePooling2D)

dense_3 (Dense)              (None, 4)            5636     ['global_average_pooling2d_1[0
                                                           ][0]']

==================================================================================================
Total params: 7774198 (29.66 MB)
Trainable params: 7706630 (29.40 MB)
Non-trainable params: 67568 (263.94 KB)
__________________________________________________________________________________________________
```

Figure 25:EfficientNet B2 Model Summary

This gives the layer by layer summary of the Efficientnet B2 model which shows the type , output shape and number of parameters of each layer and which of them can be trained and untrained.

Model Training:

```
history = model.fit(x=X_train, y=y_train, batch_size=64, epochs=10, validation_data=(X_val, y_val))

Epoch 1/10
127/127 [==============================] - 88s 299ms/step - loss: 0.3396 - accuracy: 0.8788 - val_loss: 0.2604 - val_accuracy: 0.9211
Epoch 2/10
127/127 [==============================] - 31s 242ms/step - loss: 0.1212 - accuracy: 0.9552 - val_loss: 0.2419 - val_accuracy: 0.9356
Epoch 3/10
127/127 [==============================] - 31s 246ms/step - loss: 0.0845 - accuracy: 0.9694 - val_loss: 0.2368 - val_accuracy: 0.9356
Epoch 4/10
127/127 [==============================] - 31s 246ms/step - loss: 0.0696 - accuracy: 0.9759 - val_loss: 0.2759 - val_accuracy: 0.9300
Epoch 5/10
127/127 [==============================] - 31s 242ms/step - loss: 0.0641 - accuracy: 0.9791 - val_loss: 0.3997 - val_accuracy: 0.9233
Epoch 6/10
127/127 [==============================] - 35s 272ms/step - loss: 0.0621 - accuracy: 0.9798 - val_loss: 0.2030 - val_accuracy: 0.9422
Epoch 7/10
127/127 [==============================] - 31s 247ms/step - loss: 0.0626 - accuracy: 0.9804 - val_loss: 0.4195 - val_accuracy: 0.8900
Epoch 8/10
127/127 [==============================] - 31s 241ms/step - loss: 0.0435 - accuracy: 0.9859 - val_loss: 0.2516 - val_accuracy: 0.9433
Epoch 9/10
127/127 [==============================] - 31s 247ms/step - loss: 0.0354 - accuracy: 0.9880 - val_loss: 0.1727 - val_accuracy: 0.9522
Epoch 10/10
127/127 [==============================] - 31s 245ms/step - loss: 0.0242 - accuracy: 0.9930 - val_loss: 0.2328 - val_accuracy: 0.9356
```

Figure 26: Fitting the EfficientNet B2 model

The EfficientNet B2 model will be trained over 10 epochs which displays the training as well as validation datasets at each epoch as seen in Figure 26.

Accuracy and loss Graph:

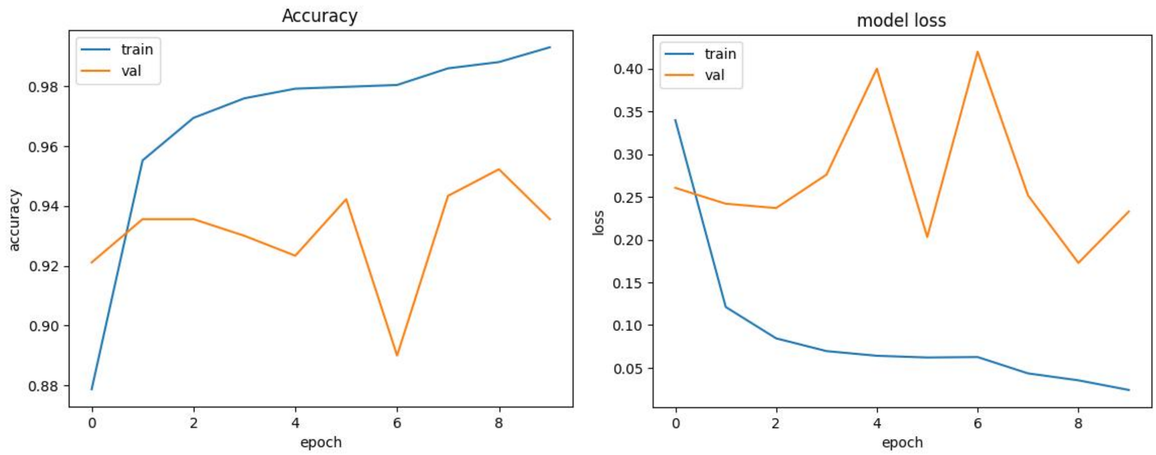


Figure 27: Accuracy and Loss Graph for the EfficientNet B2 model

In Figure 27, the two graphs are plotted to show the training and validation accuracy as well as model loss over the epoch.

Confusion Matrix:

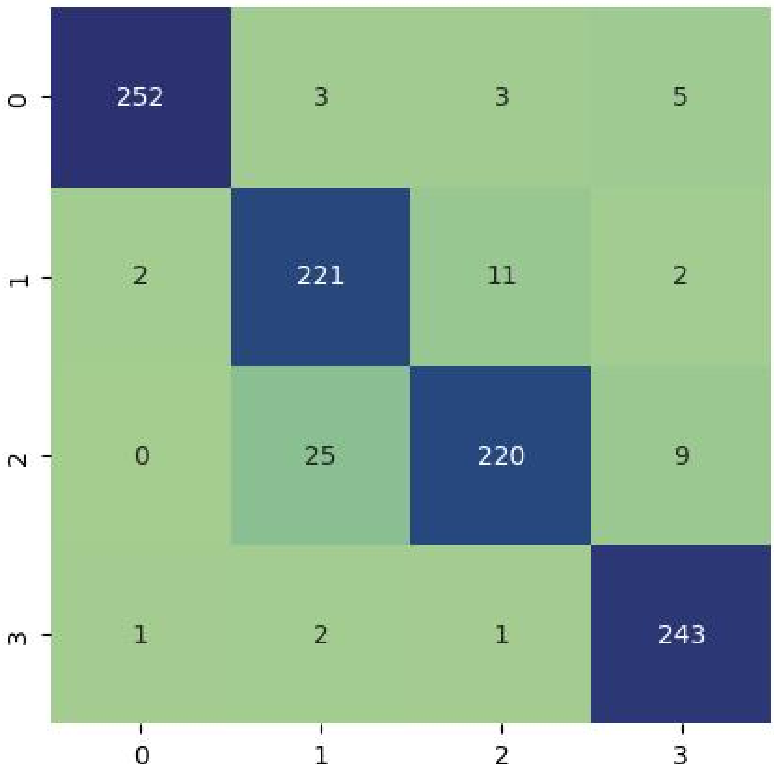


Figure 28: Confusion matrix for the EfficientNet B2 model

FIgure 28 shows the confusion matrix of the EfficientNet B2 model on the actual classes versus the predicted classes.

Classification report, Specitivity and Sensitivity:

```
print(classification_report(y_test_new,pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.96      0.97       263
           1       0.88      0.94      0.91       236
           2       0.94      0.87      0.90       254
           3       0.94      0.98      0.96       247

    accuracy                           0.94      1000
   macro avg       0.94      0.94      0.94      1000
weighted avg       0.94      0.94      0.94      1000
```

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(cls):
    prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                      pred==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.995929	0.958175
1	1	0.960733	0.936441
2	2	0.979893	0.866142
3	3	0.978752	0.983806

Figure 29 : EfficientNet B2 Model Output

Figure 29 shows the evaluation of the EfficientNet B2 model by various metrics such as sensitivity, specificity, precision, recall, and F1-score

D) EfficientNet B2 Model with Attention baseline:

```
in_lay = Input(shape=(128,128,3))
base_model = effnet.EfficientNetB2(weights=None, include_top=False, input_shape=(image_size, image_size, 3))

base_model.load_weights("/content/drive/MyDrive/weapons_image_classification_system/efficientnet-b2_imagenet_1000_notop.h5")
pt_features = base_model(in_lay)
bn_features = BatchNormalization()(pt_features)
pt_depth = base_model.get_output_shape_at(0)[-1]
```

```
# here we do an attention mechanism to turn pixels in the GAP on an off
attn_layer = Conv2D(64, kernel_size = (1,1), padding = 'same', activation = 'relu')(Dropout(0.5)(bn_features))
attn_layer = Conv2D(16, kernel_size = (1,1), padding = 'same', activation = 'relu')(attn_layer)
attn_layer = Conv2D(8, kernel_size = (1,1), padding = 'same', activation = 'relu')(attn_layer)
attn_layer = Conv2D(1,
                    kernel_size = (1,1),
                    padding = 'valid',
                    activation = 'sigmoid')(attn_layer)
# fan it out to all of the channels
up_c2_w = np.ones((1, 1, 1, pt_depth))
up_c2 = Conv2D(pt_depth, kernel_size = (1,1), padding = 'same',
               activation = 'linear', use_bias = False, weights = [up_c2_w])
up_c2.trainable = False
attn_layer = up_c2(attn_layer)

mask_features = multiply([attn_layer, bn_features])
gap_features = GlobalAveragePooling2D()(mask_features)
gap_mask = GlobalAveragePooling2D()(attn_layer)
# to account for missing values from the attention model
gap = Lambda(lambda x: x[0]/x[1], name = 'RescaleGAP')([gap_features, gap_mask])
gap_dr = Dropout(0.25)(gap)
dr_steps = Dropout(0.25)(Dense(128, activation = 'relu')(gap_dr))
out_layer = Dense(cls, activation = 'softmax')(dr_steps)
model = Model(inputs = [in_lay], outputs = [out_layer])
```

Figure 30 : EfficientNet B2 Model with attention baseline Output

Model Summary:

```
 conv2d_1 (Conv2D)           (None, 4, 4, 16)      1040      ['conv2d[0][0]']

 conv2d_2 (Conv2D)           (None, 4, 4, 8)       136       ['conv2d_1[0][0]']

 conv2d_3 (Conv2D)           (None, 4, 4, 1)       9         ['conv2d_2[0][0]']

 conv2d_4 (Conv2D)           (None, 4, 4, 1408)    1408      ['conv2d_3[0][0]']

 multiply (Multiply)         (None, 4, 4, 1408)    0         ['conv2d_4[0][0]',
                                                              'batch_normalization[0][0]']

 global_average_pooling2d (  (None, 1408)          0         ['multiply[0][0]']
 GlobalAveragePooling2D)

 global_average_pooling2d_1  (None, 1408)          0         ['conv2d_4[0][0]']
  (GlobalAveragePooling2D)

 RescaleGAP (Lambda)         (None, 1408)          0         ['global_average_pooling2d[0][
                                                             0]',
                                                              'global_average_pooling2d_1[0
                                                             ][0]']

 dropout_1 (Dropout)         (None, 1408)          0         ['RescaleGAP[0][0]']

 dense (Dense)               (None, 128)           180352    ['dropout_1[0][0]']

 dropout_2 (Dropout)         (None, 128)           0         ['dense[0][0]']

 dense_1 (Dense)             (None, 4)             516       ['dropout_2[0][0]']

==================================================================================================
Total params: 8047831 (30.70 MB)
Trainable params: 7976039 (30.43 MB)
Non-trainable params: 71792 (280.44 KB)
```

Figure 31 : EfficientNet B2 with Attention baseline Model Summary

This gives the layer by layer summary of the Efficientnet B2 model with Attention baseline which shows the type , output shape and number of parameters of each layer and which of them can be trained and untrained.

Model Training:

```
[ ] history = model.fit(x=X_train, y=y_train, batch_size=64, epochs=10, validation_data=(X_val, y_val))

Epoch 1/10
127/127 [==============================] - 95s 294ms/step - loss: 0.4261 - accuracy: 0.8437 - val_loss: 0.3272 - val_accuracy: 0.9056
Epoch 2/10
127/127 [==============================] - 30s 237ms/step - loss: 0.1653 - accuracy: 0.9405 - val_loss: 0.1836 - val_accuracy: 0.9456
Epoch 3/10
127/127 [==============================] - 30s 239ms/step - loss: 0.1126 - accuracy: 0.9616 - val_loss: 0.2754 - val_accuracy: 0.9311
Epoch 4/10
127/127 [==============================] - 31s 246ms/step - loss: 0.0781 - accuracy: 0.9721 - val_loss: 0.2346 - val_accuracy: 0.9489
Epoch 5/10
127/127 [==============================] - 31s 242ms/step - loss: 0.0591 - accuracy: 0.9804 - val_loss: 0.3518 - val_accuracy: 0.9422
Epoch 6/10
127/127 [==============================] - 31s 242ms/step - loss: 0.0714 - accuracy: 0.9772 - val_loss: 0.2463 - val_accuracy: 0.9444
Epoch 7/10
127/127 [==============================] - 31s 244ms/step - loss: 0.0755 - accuracy: 0.9747 - val_loss: 0.3231 - val_accuracy: 0.9344
Epoch 8/10
127/127 [==============================] - 31s 247ms/step - loss: 0.0663 - accuracy: 0.9775 - val_loss: 0.2604 - val_accuracy: 0.9378
Epoch 9/10
127/127 [==============================] - 32s 248ms/step - loss: 0.0511 - accuracy: 0.9831 - val_loss: 0.3580 - val_accuracy: 0.9411
Epoch 10/10
127/127 [==============================] - 31s 243ms/step - loss: 0.0556 - accuracy: 0.9825 - val_loss: 0.1883 - val_accuracy: 0.9533
```

Figure 32 :Fitting the EfficientNet B2 Model with Attention baseline

The EfficientNet B2 model with Attention baseline will be trained over 10 epochs which displays the training as well as validation datasets at each epoch as seen in Figure 32.

Accuracy and loss Graph:

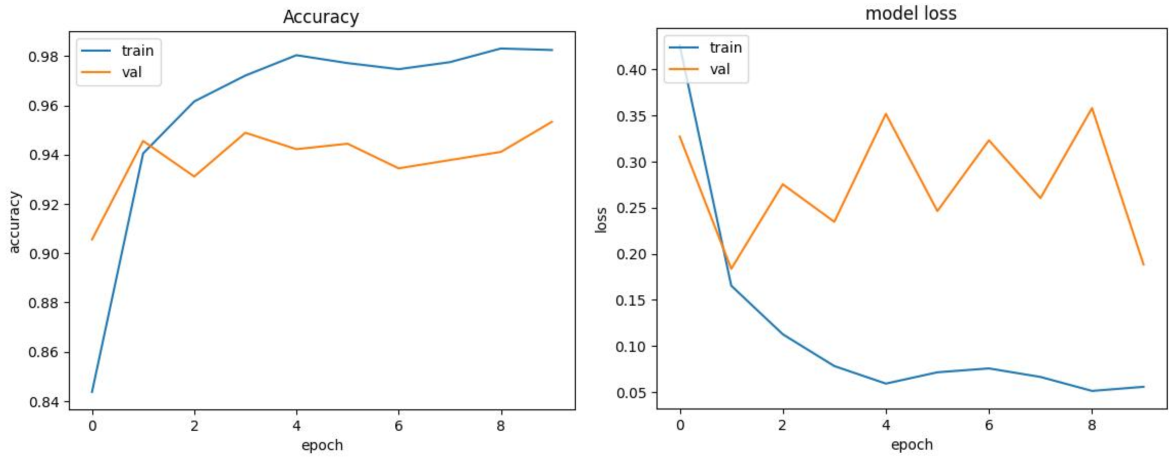


Figure 33 : EfficientNet B2 Model with Attention baseline Graphs

In Figure 33, the two graphs are plotted to show the training and validation accuracy as well as model loss over the epoch.

Confusion Matrix:

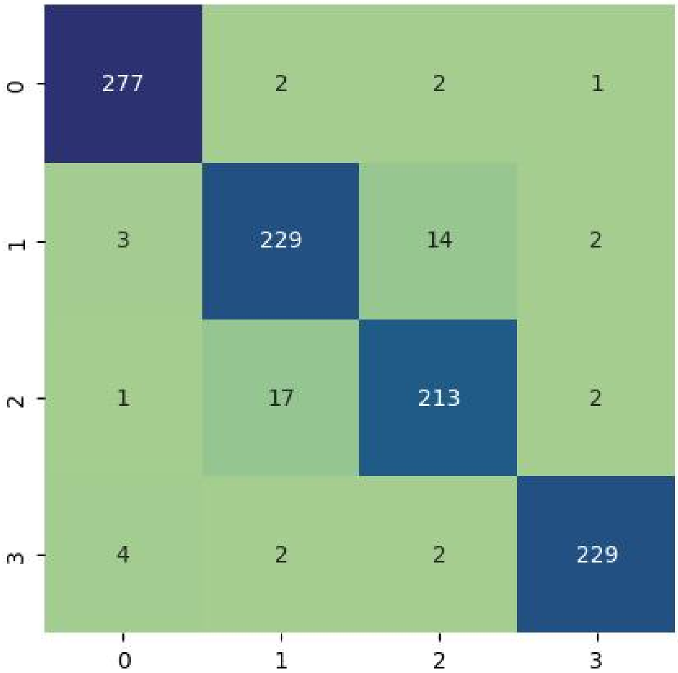


Figure 34 : EfficientNet B2 Model with Attention baseline Confusion matrix

Figure 34 shows the confusion matrix of the EfficientNet B2 Model with Attention baseline on the actual classes versus the predicted classes.

Classification report, Specitivity and Sensitivity:

```
[ ]  print(classification_report(y_test_new,pred))

              precision    recall  f1-score   support

           0       0.97      0.98      0.98       282
           1       0.92      0.92      0.92       248
           2       0.92      0.91      0.92       233
           3       0.98      0.97      0.97       237

    accuracy                           0.95      1000
   macro avg       0.95      0.95      0.95      1000
weighted avg       0.95      0.95      0.95      1000
```

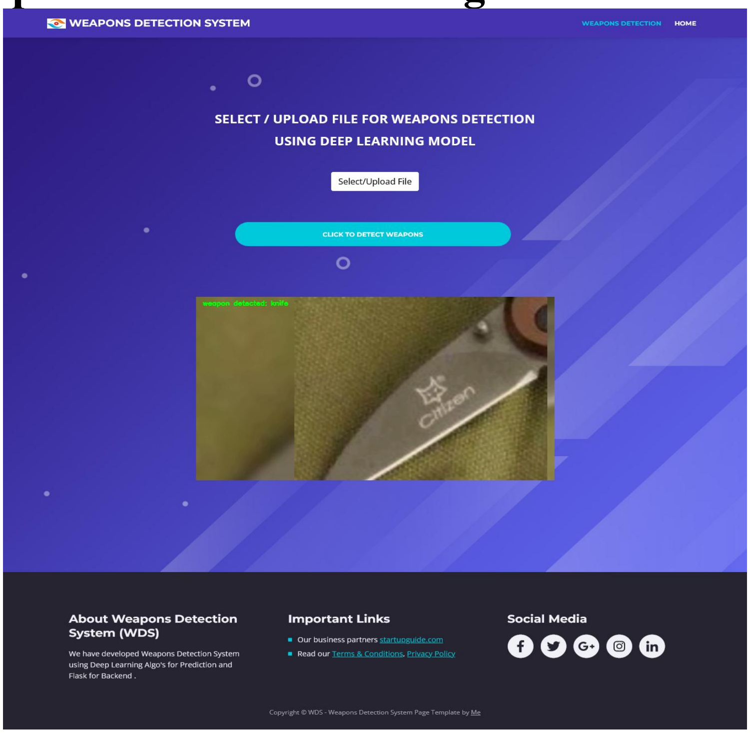
```python
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(cls):
     prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                        pred==l,
                                                        pos_label=True,average=None)
     res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.988858	0.982270
1	1	0.972074	0.923387
2	2	0.976532	0.914163
3	3	0.993447	0.966245

Figure 35 : EfficientNet B2 Model with Attention baseline Model Output

Figure 35 shows the evaluation of the EfficientNet B2 model by various metrics such as sensitivity, specificity, precision, recall, and F1-score.

5 Web Application GUI for testing.



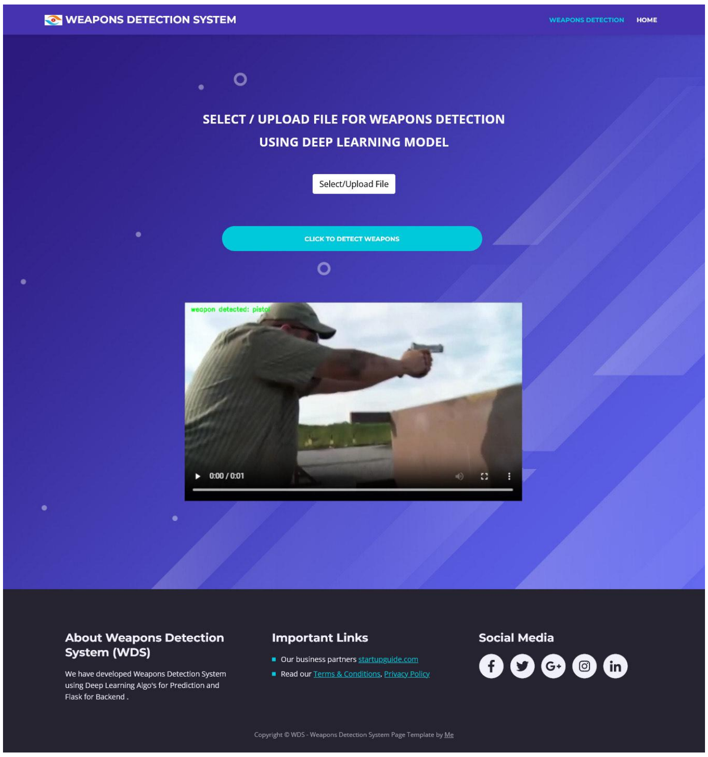


Figure 36 : Flask Web App GUI

A web application is created to test images or videos. To run the Local Web Application follow the below steps:
1) Unzip the Attached FlaskWebApp.zip
2) Open the file named app.py
3) Run the following python file and you can see a link of the hosted website in the command prompt or you can visit http://127.0.0.1:5000/
4) The user can upload a file and then click detect weapon where the created EfficientNet model will run on the file and an output will be generated by predicting the weapon as seen in Figure 36.

6 Conculsion

Following all the above mentioned steps, the code for this research can be implemented by replicating the steps to get the same results and understand the working of this project better.

References

He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).

Tan, M. and Le, Q., 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, pp.321-357.