AN AI-BASED THREAT DETECTOR FOR INTELLIGENCE SURVEILLANCE CAMERA

Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

Submitted by

NAFIZE AHAMED F

(Reg. No.: 124003188, Computer Science & Engineering)

NITHARSHAN V

(Reg. No.: 124003204, Computer Science & Engineering)

VISHVAA K

(Reg. No.: 124003367, Computer Science & Engineering)

May 2024

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA-613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR -613 401

Bonafide Certificate

This is to certify that the project report titled "**An AI-Based Threat Detector for Intelligence Surveillance Camera**" submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. Computer Science and Engineering to the SASTRA Deemed to be University, is a bonafide record of the work done by **Mr. NAFIZE AHAMED F (Reg. No.: 124003188)**, **Mr. NITHARSHAN V(Reg. No.: 124003204)** and **Mr. VISHVAA K (Reg. No.: 124003367)** during the final semester of the academic year **2023-24**, in the **School of Computing**, under my situation. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other similar title to any candidate of any University.

Signature of Project Supervisor : B.Kannan.

Name with Affiliation : Dr. Kannan Balasubramaniyan

Date :

Project Viva Voce held on _____.

Examiner 1 **Examiner 2**

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

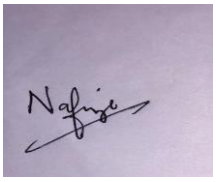
THANJAVUR – 613 401

Bonafide Certificate

Declaration

We declare that the project report titled "**An AI Based Threat Detector for Intelligence Surveillance Camera**" submitted by us is an original work done by us under the guidance of **Dr. Kannan Balasubramaniyan, Professor, School of Computing, SASTRA Deemed to be University** during the academic year 2023-24, in the **School of Computing**. This work is original and wherever we have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other similar title to any candidate of any University.

Signature of the candidates :



Name of the candidates : Nafize Ahamed F Nitharshan V Vishvaa K

Date : 20.04.2024

ACKNOWLEDGEMENTS

We would like to thank our Honourable Chancellor Prof. R. Sethuraman for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honourable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, of Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Students Welfare.

Our guide **Dr. Kannan Balasubramanian** - a professor, at the School of Computing, was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through the project.

Table of Contents

List of Figures

List of Tables

ABBREVIATIONS

CNN	Convolution Neural Network
DL	Deep learning
CSS	Cascading Style Sheets
UI	User Interface
JSON	JavaScript Object Notation
RESNET	Residual Network
AI	Artificial Intelligence
OS	Operating System
HDMI	High-Definition Multimedia Interface

ABSTRACT

In the realm of security surveillance, the traditional approach often demands continuous human observation, predisposing to lapses in concentration and potential loss of critical information. To mitigate these challenges and usher in a new era of efficiency and effectiveness, the development of an intelligent surveillance system is imperative. This system, fortified by cutting-edge technologies and innovative architectures, promises to revolutionize the landscape of security monitoring. At the heart of this paradigm shift lies the integration of deep learning, frontend and backend development, and image processing. Each component plays a pivotal role in crafting a comprehensive solution that not only automates threat detection but also augments the capabilities of security personnel. Deep learning emerges as the cornerstone of this endeavor, offering sophisticated algorithms capable of discerning intricate patterns and anomalies within surveillance footage. Leveraging techniques such as convolutional neural networks (CNNs), the system can identify objects, recognize activities, and flag potential security threats in real time.

The web-based architecture caters to applications requiring extensive API development, offering scalability and interoperability with a diverse range of systems. On the other hand, the OpenCV-based application for Raspberry Pi targets edge computing environments, enabling real-time processing of surveillance footage directly on the device. The integration of deep learning, frontend and backend development, image processing, and innovative architectures heralds a groundbreaking solution to the challenges plaguing traditional surveillance methods. By harnessing the power of these technologies, the intelligent surveillance system emerges as the definitive answer to the imperative need for automated threat detection.

Keywords: OpenCV, DL, CNN, API.

CHAPTER 1

INTRODUCTION

The evolution of security surveillance is indeed fascinating, and our project is at the forefront of this technological revolution. By harnessing the power of deep learning, OpenCV, and Django, we are aiming to transform the traditional method of human-monitored surveillance into an automated, computer-assisted process. This approach not only enhances efficiency but also accuracy in threat detection. Deep learning algorithms can tirelessly analyze video frames for potential threats, such as individuals carrying weapons, and promptly alert the necessary authorities. The integration of OpenCV allows for real-time image processing, which is crucial for identifying critical moments that require immediate attention. Meanwhile, Django's robust web framework can facilitate backend operations, ensuring that alerts are managed and disseminated securely and reliably. This synergy of technologies could redefine safety measures and response times, providing a more secure environment in public and private spaces. The project could serve as a blueprint for the future of surveillance, where computers augment human capabilities to create a safer society.

1.1 Threats

Many types of threats can appear on camera. Our project is only about alerting users to physical threats. The thing that I'm referring to is physical threat means an individual who possesses a weapon that can hurt others.In order to enforce the law and manage a civil, criminal department a successfully developed city should have to ensure the safety of each and every individual who is living in the city. One of the main problems for such developed cities who want to ensure that they should have installed CCTV cameras in each and every corner of the city. the usual method or should I say the method we have been using until now is assigning a person to a certain area of CCTV cameras as a watcher and that person is responsible for the security on that area where it is impossible to maintain eye contact with the screen each and every second of the nine to five job. Because there may appear to be human error or external disturbance. He/she will have an eye strain some seconds may or probably get missed but for a crime to occur that few seconds is enough. This is the problem with the current security surveillance model. In this model the THREAT appears and it will be seen by the human manually after that the action will be taken or the human misses the crime that occurred then the complaint is raised then the action will be taken. Our project is to eliminate the second part of the statement I mentioned above.

1.2 Applications

From the start of any era, there are many fascinating inventions made by humans from wheels to computers. Each of those inventions played a vital role in advancing that current era to be a greater one. Those advancements were made to make human life easier, more specifically make human tasks more efficient and reduce the risk of making errors. not every advancement was good, not every era was great.

In our era, the greatest advancement is AI(Artificial Intelligence). For our cause and our goal, we are going to use AI to create an application that reduces human error and compensates for human negligence in the surveillance department.

I want to tell more about the application based on the needs and specifications. Since we are using AI which is deep learning, that application has to be based on Python. That application has to run 24/7 which means it needs to use a lightweight environment. It has to be able to have a camera port (input), HDMI(output), and other qualities that are cheap and Python-based. All the specifications point to Raspberry Pi for deployment which can run 24/7 and it's cheap and lightweight raspbian-os which is a Python environment-based OS is also lightweight. We are going to develop a software application for Raspberry Pi.

We are gonna create an application that can become a stand-alone application where it can be deployed even in a low-end processor and even work without the screen

CHAPTER 2

OBJECTIVES

The typical security surveillance method frequently requires some human observation to be done continuously, which puts the workers at risk of attentional slips and maybe losing some important data. So, an intelligent surveillance system development is essential for addressing these issues by providing a solution to replace human surveillance into camera-based surveillance for detecting the threats around the environment.

2.1 System Architecture

We are proposing two different architectures for our execution one of them is web-based and the other one is purely for Python-based environments

Two types

1) Web-based architecture
2) python environment architecture

2.1.1 Web-based Architecture

In this architecture, we are going to use web technologies for building the front end. First of all we are going to use React JS. It is a web framework that can be used to build front-end application functionalities with low lines of code and to obtain maximum functionality. After that, we are going to use tailwind CSS for UI design. Tailwind CSS is a CSS framework where the designing can be done by just defining class names in divs and can be used for simple animation. After that, we are going to use node js for accessing the camera more specifically we are going to use react webcam by executing the command npm install react webcam. the selected or captured image will be appended to form data, converted to JSON, and sent to the backend

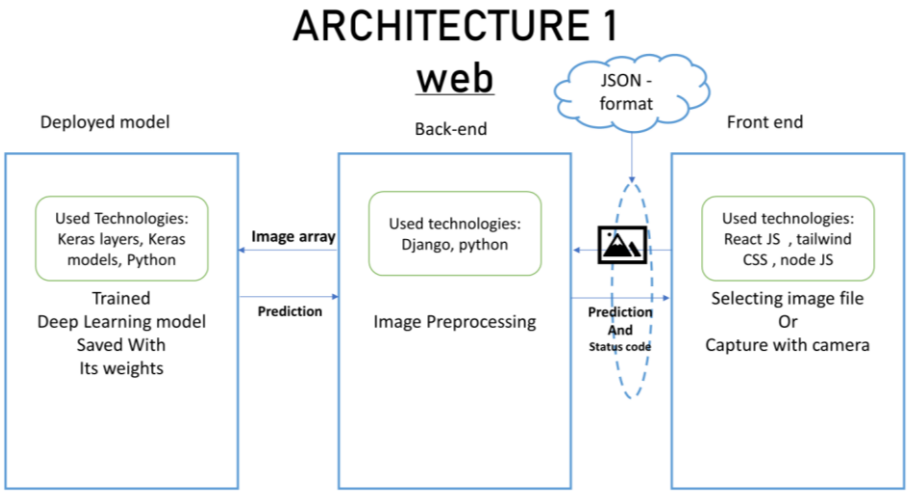


Fig. 2.1 Web Architecture

Django, a web framework, is utilized to manage requests from the front end, process images, and predict weapons using a saved DL model. Images are sent from the frontend to the backend in JSON format. The images undergo preprocessing within the backend and are subsequently analyzed using the saved DL model for weapon prediction. The resulting predictions are then packaged into a JSON response and sent a reply to the front end for display or further processing.

2.1.2 Python Environment Based Architecture

In the Python environment, such as that on a Raspberry Pi, captured images from the camera are directed into the system. These images undergo preprocessing, a crucial step where they are refined and optimized for analysis. Once polished, they are presented to a trained deep learning (DL) model, eagerly awaiting their arrival. This DL model, a product of rigorous training and countless iterations, eagerly consumes the preprocessed images. Through its complex network of neurons and connections, it interprets the nuances of the images, extracting patterns and features with remarkable precision. Finally, the model delivers its verdict: prediction results that shed light on the content of the images.

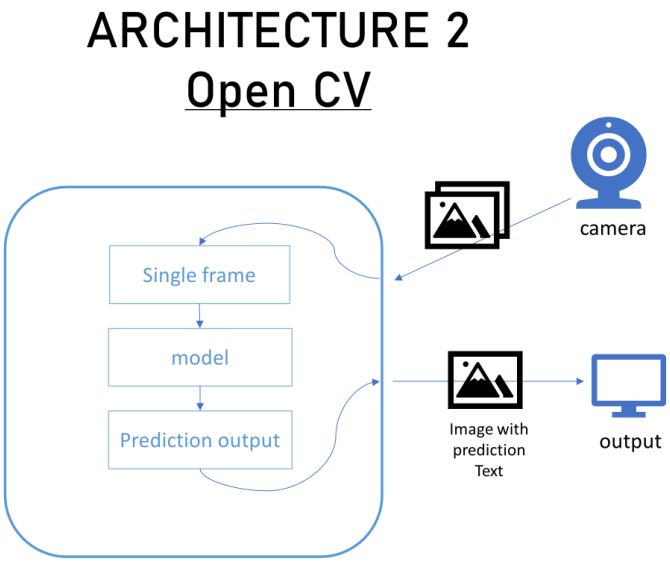


Fig. 2.2 OpenCV Architecture

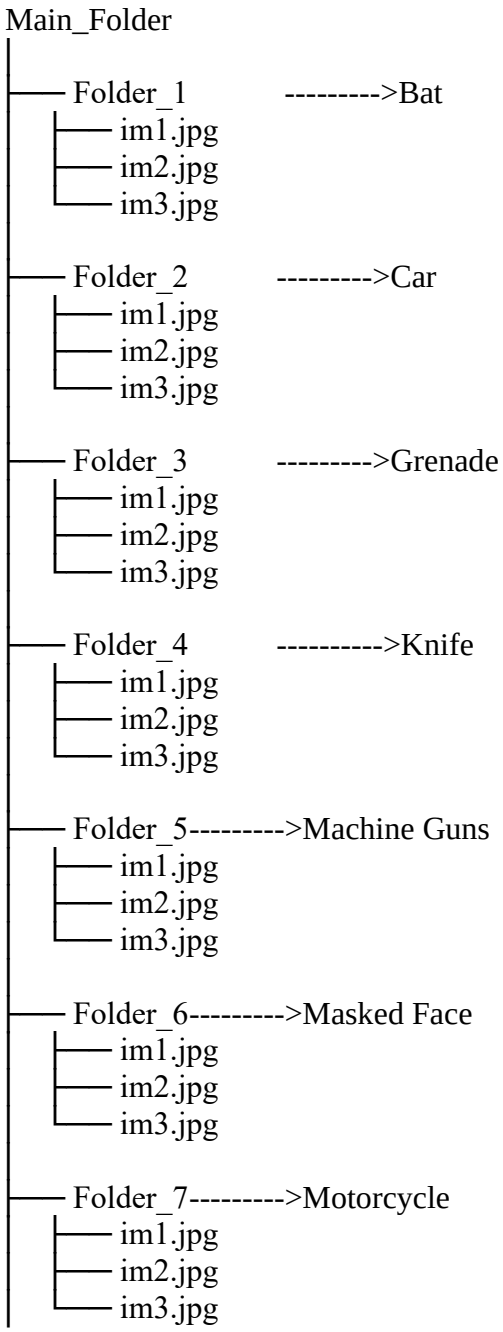
These insights, derived from the depths of the DL model's computations, offer valuable information and understanding. In this Python-powered ecosystem, the Raspberry Pi serves as both host and facilitator, orchestrating the flow of data from camera to prediction. With each step, it harnesses the power of Python to unlock the potential hidden within the captured images, paving the way for discoveries and advancements.

CHAPTER 3

EXPERIMENTAL WORK

3.1 Dataset

Since we are using an image dataset, we are not using CSV files or any other Excel files our dataset is a folder that is composed of other subfolders where each folder is composed of multiple images. Each folder represents a class. A class folder contains images of weapons or any other threat-posing object. for example. Guns. Knife



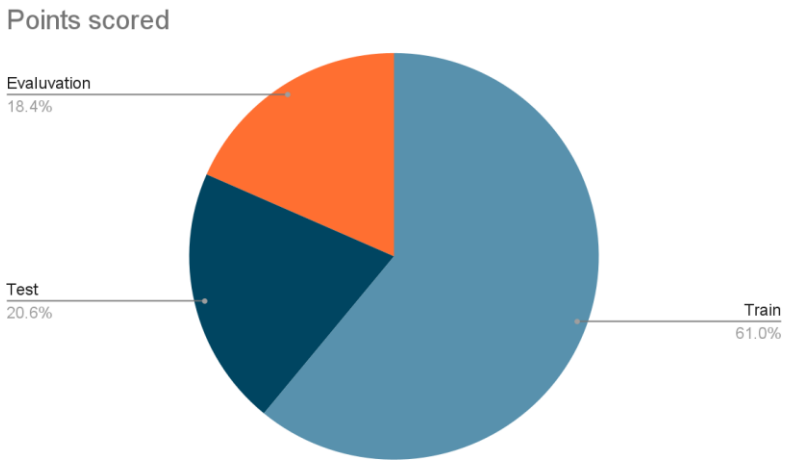


Fig 3.1 Dataset Pie Chart Analysis

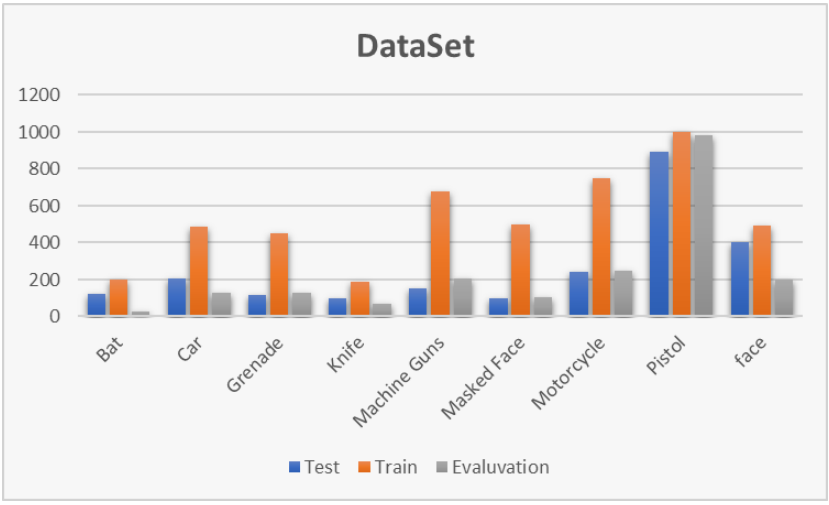
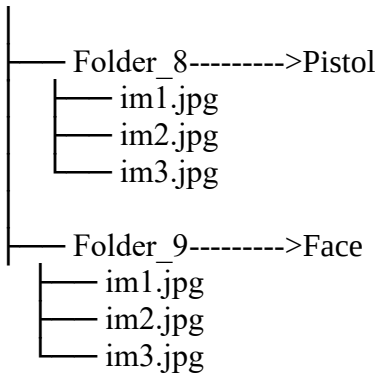


Fig 3.2 Dataset Bar Chart Analysis



Dataset: Train, Test, Eval

Our main dataset folder contains totally three main folder

Named

1)Train

2)Test

3)Eval

where each folder above has the same structure as the figure I mentioned above

3.2 Data preprocessing

Image preprocessing plays an important role in training deep-learning models for image-based tasks. It involves a series of transformation processes applied to input images before feeding them into the model. Mostly the image preprocessing in the deep-learning model involves Resizing, Normalization, Data Augmentation, Rotation, brightness changing

1)Initial image

Fig 3.3 Original Image

2) Flipping

Fig 3.4 Flipped Image

3)Random Rotation

Fig 3.5 Rotated Image

4) Random Zoom

Fig 3.6 Random Zoom

5)Random Translation

Fig 3.7 Translated Image

6)Contrast

Fig 3.8 Contrast Image

7) Random Brightness

Fig 3.9 Brightness

8) Resize

Fig 3.10 Resize

Another preprocessing is also there is normalization which is done by converting the image into a numpy array and ranging them between 1-255.

These are the basic image preprocessing that is done in our dataset to improve the accuracy of the prediction and to obtain a better model.

3.3 Models

Here, we came to the most important part of the project: the DL models where the prediction and the replacement of the human watcher will happen. here we are going to use CNN models.

We have implemented several deep learning models namely,

1. CNN

   This CNN model consists of multiple layers such as convolution layers, and max-pooling layers which work together to process and understand the input images.
2. Google net

   It is also called Inception V1.it is also a CNN model for image classification. the architecture uses 1x1 convolution and global average pooling
3. Resnet 18

   The word Resnet stands for Residual Network architecture which is commonly used in deep learning for classification of images. The number 18 represents the number of layers it is computationally efficient
4. Resnet 50

   It consists of 50 layers. And can provide more accurate results.

Max-pooling layer

Before going into the max-pooling layer we have to understand what a pooling layer is. The purpose of a pooling layer is to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

There are three major types of pooling layer

1)max -pooling
2)average pooling
3)global pooling

What we are using is the max pooling layer. A max-pooling layer is a critical component in the architecture of a Convolutional Neural Network (CNN). It serves the purpose of reducing the spatial dimensions of the input feature map, making the computation more manageable, and reducing the risk of overfitting by providing an abstracted form of the representation. This layer operates by sliding a filter, often referred to as a kernel, across the input feature map and taking the maximum value within the region covered by the filter. This process results in a new feature map that contains only the most activated features, which are generally the most important features for making predictions. Max pooling layers are known for their ability to make the detection of

features in the input invariant to scale and translation, contributing to the robustness of the model. The implementation of max pooling in various deep learning frameworks, such as Keras, is straightforward, involving only a few lines of code to define the layer and its parameters like pool size and stride.

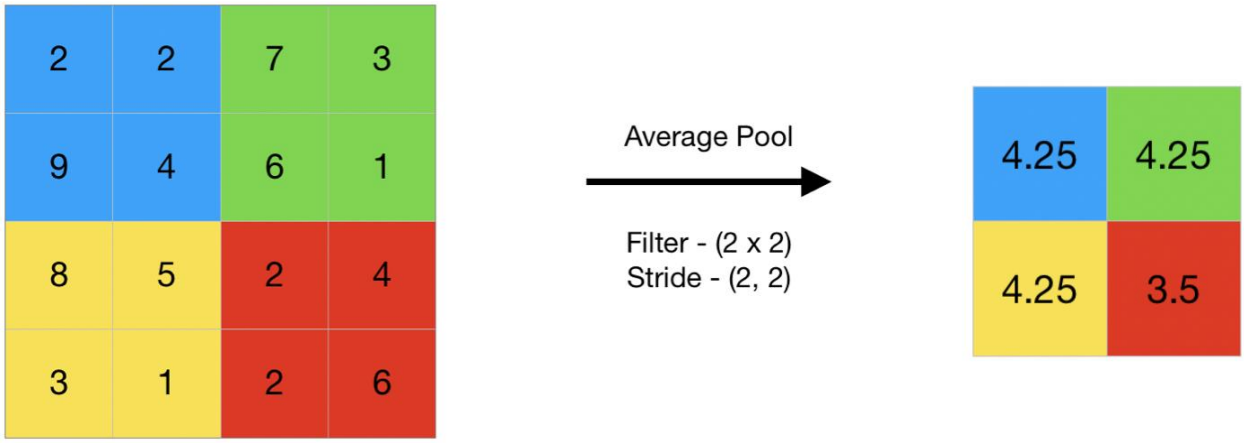


Fig 3.11 Max-pooling layer

Convolution layer

A convolution layer is a fundamental component of a Convolutional Neural Network (CNN), primarily used in the field of deep learning for analyzing visual imagery. The layer's main function is to perform a mathematical operation called convolution, which involves sliding a filter or kernel over the input data to produce a feature map. This feature map is a processed version of the input that highlights certain features, making it easier for the network to learn from the data. Convolution layers can detect edges, shapes, and textures in images, contributing significantly to the network's ability to recognize objects and patterns. By stacking multiple convolution layers with different filters, a CNN can learn a hierarchy of features, from simple to complex, enabling it to perform tasks like image and video recognition, and image classification. The power of convolution layers lies in their ability to learn these filters automatically during the training process, tailored to the specific task at hand.

CNN Model:

A convolutional Neural Network (CNN) is a deep learning algorithm used for image recognition and classification processes. It consists of multiple layers which work together to process and understand the input images. Here it consists of convolution layers, max-pooling layers, batch normalization layers, dropout layers used to prevent the chance of overfitting, a flatten layer, and dense layers.

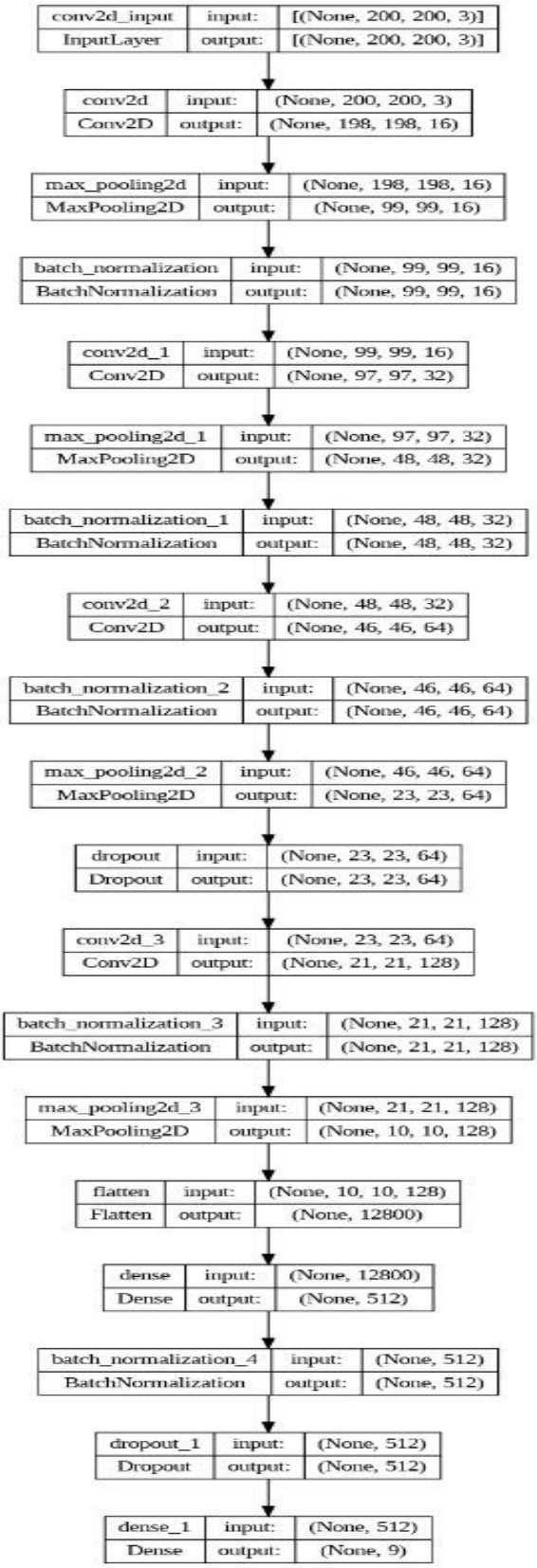


Fig 3.12 Model Architecture

The structure above has some advantages over other models. This architecture follows a pattern of increasing complexity in terms of the number of filters and decreasing spatial dimensions after each convolutional layer. This allows the model to learn increasingly abstract and high-level features as it progresses through the layers. Batch normalization layers are inserted after each convolutional layer and before the activation function. This normalizes the activations of the previous layer, which helps alleviate issues like vanishing gradients and accelerates the training process. It also acts as a form of regularization, reducing the risk of overfitting. Dropout randomly deactivates a certain percentage of neurons during training, forcing the network to learn more robust features and reducing the risk of overfitting by preventing the co-adaptation of neurons. Use of data augmentation techniques such as random flipping, rotation, and zooming helps increase the diversity of the training dataset. This makes the model more robust to variations in the input data and helps prevent overfitting.ReLU (Rectified Linear Unit) activation functions are used throughout the model, except for the output layer where SoftMax activation is used. ReLU is widely used in deep learning models due to its simplicity and effectiveness in alleviating the vanishing gradient problem. The model has a moderate depth with several convolutional layers followed by dense layers. This depth allows the model to capture both low-level and high-level features in the input images, making it suitable for a wide range of image classification tasks.

Resnet-18 Model:

ResNet stands for "Residual Network," and it's designed to address the vanishing gradient problem encountered in very deep neural networks. ResNet-18 specifically is one of the smaller versions of the ResNet family, developed by Microsoft Research. It consists of 18 layers, including convolutional layers, pooling layers, and fully connected layers. The architecture employs residual connections, where shortcut connections skip one or more layers, allowing the model to learn residual functions rather than directly approximating the desired underlying mapping. ResNet-18 is often used in image classification, object detection, and image segmentation.

Resnet-50 Model:

ResNet-50 is a deeper variant of the ResNet architecture compared to ResNet-18. It's also a convolutional neural network (CNN) designed to address the challenges of training very deep neural networks by introducing residual connections. In essence, ResNet-50 is not just a deeper version of its predecessors; it's a powerhouse of convolutional neural network architecture, offering unparalleled accuracy and performance in tasks where complexity and detail are paramount.

ResNet-50 emerges as the beacon of precision in the realm of visual recognition, its towering depth serving as a beacon of accuracy in tasks like image classification, object detection, and image segmentation. With a discerning eye for subtle nuances, it navigates through the intricate tapestry of images, unraveling complex patterns and delineating objects with unparalleled precision.

Yet, amidst its prowess lies a hunger for computational resources, a voracious appetite demanding substantial computing power for both training and inference. Despite this appetite, organizations and research institutions armed with robust computational infrastructure embrace ResNet-50 as an indispensable ally in their quest for excellence. It's a cornerstone, propelling them towards the zenith of achievement, where state-of-the-art results in visual recognition tasks await those bold enough to harness its power.

Google Net Model:

GoogleNet, also known as Inception v1, is a deep convolutional neural network (CNN) architecture developed by researchers at Google. It was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014. GoogleNet achieved state-of-the-art performance on the ImageNet dataset while being computationally more efficient compared to other deeper architectures. Its inception modules allowed it to capture a rich hierarchy of features while keeping the number of parameters relatively low. Despite its success, GoogleNet has evolved over the years, with subsequent versions (e.g., Inception v2, Inception v3, etc.) introducing improvements in architecture and performance. These advancements have led to even more efficient and accurate models for various computer vision tasks.

GoogleNet bursts onto the scene with its groundbreaking architecture, redefining the landscape of convolutional neural networks (CNNs) with its ingenious design.

Inception Modules, the cornerstone of GoogleNet's brilliance, revolutionize feature extraction by orchestrating parallel convolutions across varying filter sizes, from the minute 1x1 to the substantial 5x5. By seamlessly merging outputs, GoogleNet adeptly captures spatial intricacies, enabling it to discern features across multiple scales with remarkable efficiency.

Embarking on a journey of depth, GoogleNet's architecture delves into the abyss of complexity with its stacked inception modules. This deep structure empowers the network to ascend the hierarchical ladder of features, navigating through layers of increasing sophistication to extract the essence of visual data.

Pooling layers, both max and average, pepper the network, skillfully downsizing feature maps while spotlighting dominant features. These layers act as gatekeepers, ushering only the most salient information forward, thereby streamlining subsequent processing.

As the journey nears its conclusion, a Global Average Pooling layer emerges, a beacon of computational elegance. By condensing feature maps to a compact 1x1 size through meticulous averaging, GoogleNet achieves a feat of dimensionality reduction, mitigating parameter overload and computational burdens.

Tradition meets innovation as fully connected layers weave their magic, seamlessly blending learned features to orchestrate predictions. In the realm of classification, the alchemy of the SoftMax activation function reigns supreme, transforming raw scores into probabilities with finesse and certainty.

But the saga of GoogleNet doesn't end there. Enter the auxiliary classifiers, a stroke of genius in the battle against the vanishing gradient. Nestled within intermediate layers, these classifiers lend a helping hand, providing additional supervision and guiding the network toward discriminative enlightenment.

In essence, GoogleNet isn't just another convolutional neural network; it's a testament to ingenuity, a symphony of innovation orchestrated to unravel the complexities of visual data with precision and grace.

CHAPTER 4

RESULTS AND DISCUSSION

This part is about the comparisons and analytics of the models and experimental things that got as much distance from the start of this project. Since we have done four models for our research purposes we have to analyze these four models to pick a model that can predict well.to pick that specific model we have several ways to select the right model. After training the model successfully we have to evaluate the model to get some metrics out of it. The first one we are going to analyze is the confusion matrix. if someone wants to know what a confusion matrix is. It is in a table with the original label in the y-axis and the predicted label in the x-axis.we compare model predictions and true labels. It gives us insight on the classes that data need to be improved

4.1 Metrics

4.1.1 Confusion matrix

CNN

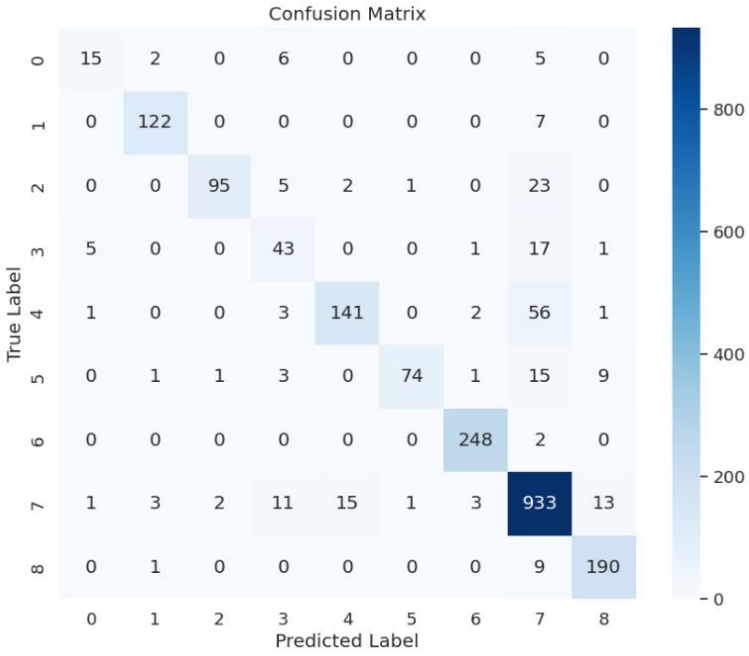


Fig 4.1 CNN Confusion Matrix

Google Net

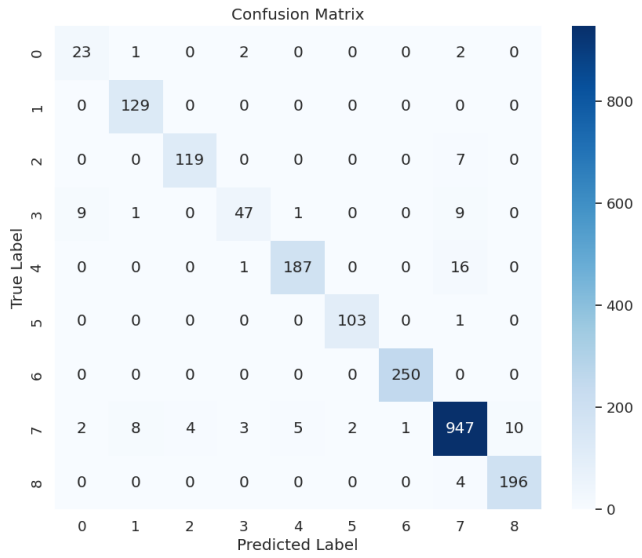


Fig 4.2 Google net Confusion Matrix

Resnet 18

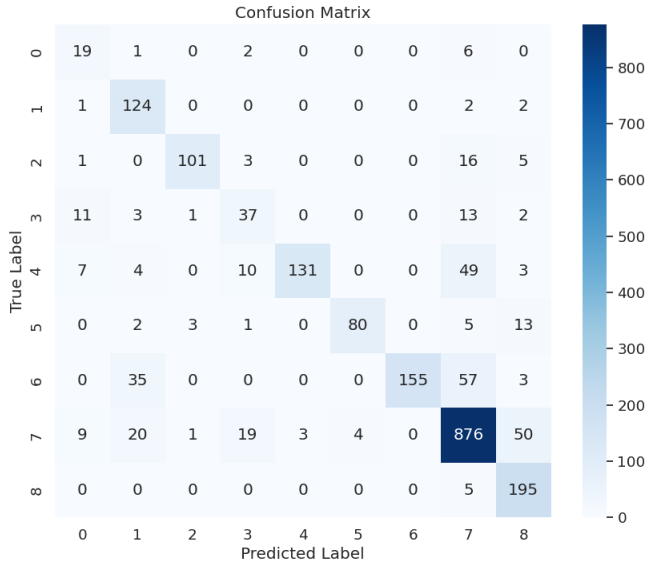


Fig 4.3 Resnet 18 Confusion Matrix

Resnet 50

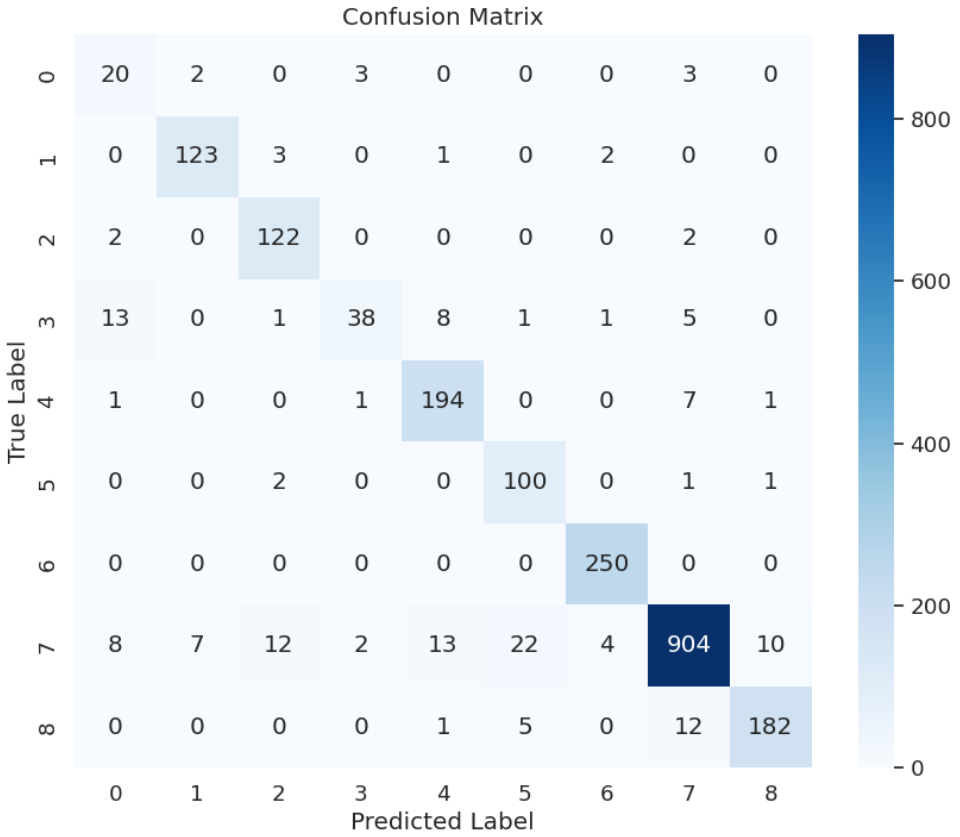


Fig 4.4 Resnet 50 Confusion Matrix

After seeing the confusion matrix of all the models, we trained we can see that the Google Net model is better than other models

TP -> True Positive(correct prediction of the correct class)

FP ->False Positive (incorrectly predicts as positive)

TN ->True Negative(correct prediction of negative class)

FN -> False Negative(incorrectly predicts as negative)

1) Precision

It measures the accuracy of the positive predictions made by the model

P=TP/(TP+FP)

Higher precision means low FP errors

2)Recall

It is also known as Sensitivity or True Positive Rate (TPR)

R=TP/(TP+FN)

Higher the recall. Higher positive samples detected

3)F1 Score

It is the harmonic mean of precision and recall

F1 = 2 x (P X R)/(P+R)

0.9>best

0.8>good

0.7>need improvements

Models	Accuracy	F1 score	Recall	Precision
CNN	85.7	87	87	88
GoogleNet	95.0	93.1	92.9	93.4
Resnet 50	91.9	87.6	88.1	88.9
Resnet 18	91.2	92	92	92.4

Table 4.1

As you can see in the table Google Net also has the best overall F1 score out of all models Hence, we are selecting Google Net.

CHAPTER 5

APPENDIX

5.1 SOURCE CODE

5.1.1 Deep Learning

a) CNN Model

0.1 CNN

clone git for dataset

[29]:
```
!git clone https://github.com/nitharshanv/sem-8-major-project.git
```

```
fatal: destination path 'sem-8-major-project' already exists and is not an empty
directory.
```

load the data

[36]:
```
import keras

batch_size=32
image_size=(224,224)
train_dir = r"/kaggle/input/threat/new hawk/Hawk_Eye_dataset/train"
test_dir = "/kaggle/input/threat/new hawk/Hawk_Eye_dataset/test"



train_ds=keras.utils.
  ↪image_dataset_from_directory(train_dir,validation_split=None,subset=None,seed=1337,image_si

test_ds=keras.utils.
  ↪image_dataset_from_directory(test_dir,validation_split=None,subset=None,seed=1337,image_siz
val_dir=r"/kaggle/input/threat/new hawk/Hawk_Eye_dataset/validation"
val_ds=keras.utils.
  ↪image_dataset_from_directory(val_dir,validation_split=None,subset=None,seed=1337,image_size
```

```
Found 6925 files belonging to 9 classes.
Found 2346 files belonging to 9 classes.
Found 2090 files belonging to 9 classes.
```

[37]:
```
train_ds=train_ds.concatenate(test_ds)
test_ds=val_ds
```

[38]:
```
len(train_ds)
```

[38]: 291

model

[43]:
```
from keras.models import Sequential
from keras.layers import␣
 ↪Conv2D,Flatten,Dropout,Dense,MaxPooling2D,BatchNormalization,Flatten
from keras import layers
model = Sequential()

model.add(layers.RandomFlip("horizontal"))
model.add(layers.RandomTranslation(0.2, 0.2))
model.add(layers.RandomRotation(0.1))
model.add(layers.RandomZoom(0.2))
model.add(layers.Resizing(224,224))
#model.add(layers.Rescaling(1./255))
################################################
model.add(Conv2D(filters=16,kernel_size=(3,3),activation='relu',input_shape␣
 ↪=(200,200,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

################################################
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='relu',input_shape␣
 ↪=(97,97,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

################################################
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',input_shape␣
 ↪=(46,46,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
################################################

model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu',input_shape␣
 ↪=(21,21,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=2))

################################################

model.add(Flatten())

model.add(Dense(512,activation='relu'))

model.add(BatchNormalization())
```

```
model.add(Dropout(0.2))

model.add(Dense(9,activation='softmax'))
```

1 Training

[53]:
```
import tensorflow as tf
tf.__version__
import keras
from keras.models import Sequential
from keras import layers
from keras.layers import␣
 ↪Conv2D,Flatten,Dropout,Dense,MaxPooling2D,BatchNormalization,Flatten




opt=keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    name="adam"
)

model.compile(optimizer=opt,
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

import tensorflow as tf


epochs = 50
history = model.fit(
  train_ds,
  validation_data=test_ds,
  epochs=epochs
```

```
model.summary()
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
random_flip_7 (RandomFlip)	(None, 224, 224, 3)	0
random_translation_7 (RandomTranslation)	(None, 224, 224, 3)	0
random_rotation_7 (RandomRotation)	(None, 224, 224, 3)	0
random_zoom_7 (RandomZoom)	(None, 224, 224, 3)	0
resizing_7 (Resizing)	(None, 224, 224, 3)	0
conv2d_28 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_28 (MaxPooling2D)	(None, 111, 111, 16)	0
batch_normalization_35 (BatchNormalization)	(None, 111, 111, 16)	64
conv2d_29 (Conv2D)	(None, 109, 109, 32)	4,640
max_pooling2d_29 (MaxPooling2D)	(None, 54, 54, 32)	0
conv2d_30 (Conv2D)	(None, 52, 52, 64)	18,496
batch_normalization_37 (BatchNormalization)	(None, 52, 52, 64)	256
max_pooling2d_30 (MaxPooling2D)	(None, 26, 26, 64)	0
dropout_14 (Dropout)	(None, 26, 26, 64)	0
conv2d_31 (Conv2D)	(None, 24, 24, 128)	73,856
batch_normalization_38 (BatchNormalization)	(None, 24, 24, 128)	512
max_pooling2d_31 (MaxPooling2D)	(None, 12, 12, 128)	0
flatten_7 (Flatten)	(None, 18432)	0
dense_14 (Dense)	(None, 512)	9,437,696
batch_normalization_39 (BatchNormalization)	(None, 512)	2,048
dropout_15 (Dropout)	(None, 512)	0
dense_15 (Dense)	(None, 9)	4,617

```
class_names=val_ds.class_names
print(class_names)
path='/kaggle/input/threat/new hawk/Hawk_Eye_dataset/validation/Car/04574.jpg'
img = tf.keras.utils.load_img(
   path, target_size=(224,224)
)



img_array = np.expand_dims(img, axis=0)
predictions = model.predict(img_array)
score = predictions[0]
val = np.argmax(score)
print(score)


print(
    "This image most likely belongs to {} "
    .format(class_names[np.argmax(score)])
)
```

```
['Bat', 'Car', 'Grenade', 'Knife', 'Machine Guns', 'Masked Face', 'Motorcycle',
'Pistol', 'face']
1/1             0s 20ms/step
[9.1526931e-07 9.9870622e-01 3.1539066e-08 1.8725141e-05 1.1575899e-07
 2.0858188e-11 2.3791163e-04 1.0359474e-03 3.8271133e-10]
This image most likely belongs to Car
```

[60]:
```
img
```

[60]:

b) Google Net model

[1]:
```
!git clone https://github.com/nitharshanv/sem-8-major-project.git
```

```
Cloning into 'sem-8-major-project'…
remote: Enumerating objects: 12887, done.
remote: Counting objects: 100% (2043/2043), done.
remote: Compressing objects: 100% (1693/1693), done.
remote: Total 12887 (delta 31), reused 1999 (delta 11), pack-reused 10844
Receiving objects: 100% (12887/12887), 625.35 MiB | 36.85 MiB/s, done.
Resolving deltas: 100% (50/50), done.
Updating files: 100% (12031/12031), done.
```

[31]:
```
import keras

batch_size=150
image_size=(224,224)
train_dir = r"/content/sem-8-major-project/new hawk/Hawk_Eye_dataset/train"
test_dir = "/content/sem-8-major-project/new hawk/Hawk_Eye_dataset/test"



train_ds1=keras.utils.
  ↪image_dataset_from_directory(train_dir,validation_split=None,subset=None,seed=1337,image_si

test_ds1=keras.utils.
  ↪image_dataset_from_directory(test_dir,validation_split=None,subset=None,seed=1337,image_siz
val_dir=r"/content/sem-8-major-project/new hawk/Hawk_Eye_dataset/validation"
val_ds=keras.utils.
  ↪image_dataset_from_directory(val_dir,validation_split=None,subset=None,seed=1337,image_size
```

```
Found 6925 files belonging to 9 classes.
Found 2346 files belonging to 9 classes.
Found 2090 files belonging to 9 classes.
```

[32]:
```
train_ds=train_ds1.concatenate(test_ds1)
test_ds=val_ds
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def create_model():
    input_1_unnormalized = keras.Input(shape=(224,224,3),␣
 ↪name="input_1_unnormalized")
    input_1 = keras.layers.Normalization(axis=(1,2,3),␣
 ↪name="input_1_")(input_1_unnormalized)
    conv1_prepadded = layers.ZeroPadding2D(padding=((3,3),(3,3)))(input_1)
    conv1 = layers.Conv2D(64, (7,7), strides=(2,2),␣
 ↪name="conv1_")(conv1_prepadded)
    bn_conv1 = layers.BatchNormalization(epsilon=0.001000,␣
 ↪name="bn_conv1_")(conv1)
    activation_1_relu = layers.ReLU()(bn_conv1)
    max_pooling2d_1_prepadded = layers.
 ↪ZeroPadding2D(padding=((1,1),(1,1)))(activation_1_relu)
    max_pooling2d_1 = layers.MaxPool2D(pool_size=(3,3),␣
 ↪strides=(2,2))(max_pooling2d_1_prepadded)
    res2a_branch2a = layers.Conv2D(64, (1,1),␣
 ↪name="res2a_branch2a_")(max_pooling2d_1)
    bn2a_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
 ↪name="bn2a_branch2a_")(res2a_branch2a)
    activation_2_relu = layers.ReLU()(bn2a_branch2a)
    res2a_branch2b = layers.Conv2D(64, (3,3), padding="same",␣
 ↪name="res2a_branch2b_")(activation_2_relu)
    bn2a_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
 ↪name="bn2a_branch2b_")(res2a_branch2b)
    activation_3_relu = layers.ReLU()(bn2a_branch2b)
    res2a_branch2c = layers.Conv2D(256, (1,1),␣
 ↪name="res2a_branch2c_")(activation_3_relu)
    res2a_branch1 = layers.Conv2D(256, (1,1),␣
 ↪name="res2a_branch1_")(max_pooling2d_1)
    bn2a_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
 ↪name="bn2a_branch2c_")(res2a_branch2c)
    bn2a_branch1 = layers.BatchNormalization(epsilon=0.001000,␣
 ↪name="bn2a_branch1_")(res2a_branch1)
    add_1 = layers.Add()([bn2a_branch2c, bn2a_branch1])
```

```
  activation_4_relu = layers.ReLU()(add_1)
  res2b_branch2a = layers.Conv2D(64, (1,1),␣
↪name="res2b_branch2a_")(activation_4_relu)
  bn2b_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2b_branch2a_")(res2b_branch2a)
  activation_5_relu = layers.ReLU()(bn2b_branch2a)
  res2b_branch2b = layers.Conv2D(64, (3,3), padding="same",␣
↪name="res2b_branch2b_")(activation_5_relu)
  bn2b_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2b_branch2b_")(res2b_branch2b)
  activation_6_relu = layers.ReLU()(bn2b_branch2b)
  res2b_branch2c = layers.Conv2D(256, (1,1),␣
↪name="res2b_branch2c_")(activation_6_relu)
  bn2b_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2b_branch2c_")(res2b_branch2c)
  add_2 = layers.Add()([bn2b_branch2c, activation_4_relu])
  activation_7_relu = layers.ReLU()(add_2)
  res2c_branch2a = layers.Conv2D(64, (1,1),␣
↪name="res2c_branch2a_")(activation_7_relu)
  bn2c_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2c_branch2a_")(res2c_branch2a)
  activation_8_relu = layers.ReLU()(bn2c_branch2a)
  res2c_branch2b = layers.Conv2D(64, (3,3), padding="same",␣
↪name="res2c_branch2b_")(activation_8_relu)
  bn2c_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2c_branch2b_")(res2c_branch2b)
  activation_9_relu = layers.ReLU()(bn2c_branch2b)
  res2c_branch2c = layers.Conv2D(256, (1,1),␣
↪name="res2c_branch2c_")(activation_9_relu)
  bn2c_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn2c_branch2c_")(res2c_branch2c)
  add_3 = layers.Add()([bn2c_branch2c, activation_7_relu])
  activation_10_relu = layers.ReLU()(add_3)
  res3a_branch2a = layers.Conv2D(128, (1,1), strides=(2,2),␣
↪name="res3a_branch2a_")(activation_10_relu)
  bn3a_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3a_branch2a_")(res3a_branch2a)
  activation_11_relu = layers.ReLU()(bn3a_branch2a)
  res3a_branch2b = layers.Conv2D(128, (3,3), padding="same",␣
↪name="res3a_branch2b_")(activation_11_relu)
  bn3a_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3a_branch2b_")(res3a_branch2b)
  activation_12_relu = layers.ReLU()(bn3a_branch2b)
  res3a_branch2c = layers.Conv2D(512, (1,1),␣
↪name="res3a_branch2c_")(activation_12_relu)
```

```
    res3a_branch1 = layers.Conv2D(512, (1,1), strides=(2,2),␣
↪name="res3a_branch1_")(activation_10_relu)
    bn3a_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3a_branch2c_")(res3a_branch2c)
    bn3a_branch1 = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3a_branch1_")(res3a_branch1)
    add_4 = layers.Add()([bn3a_branch2c, bn3a_branch1])
    activation_13_relu = layers.ReLU()(add_4)
    res3b_branch2a = layers.Conv2D(128, (1,1),␣
↪name="res3b_branch2a_")(activation_13_relu)
    bn3b_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3b_branch2a_")(res3b_branch2a)
    activation_14_relu = layers.ReLU()(bn3b_branch2a)
    res3b_branch2b = layers.Conv2D(128, (3,3), padding="same",␣
↪name="res3b_branch2b_")(activation_14_relu)
    bn3b_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3b_branch2b_")(res3b_branch2b)
    activation_15_relu = layers.ReLU()(bn3b_branch2b)
    res3b_branch2c = layers.Conv2D(512, (1,1),␣
↪name="res3b_branch2c_")(activation_15_relu)
    bn3b_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3b_branch2c_")(res3b_branch2c)
    add_5 = layers.Add()([bn3b_branch2c, activation_13_relu])
    activation_16_relu = layers.ReLU()(add_5)
    res3c_branch2a = layers.Conv2D(128, (1,1),␣
↪name="res3c_branch2a_")(activation_16_relu)
    bn3c_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3c_branch2a_")(res3c_branch2a)
    activation_17_relu = layers.ReLU()(bn3c_branch2a)
    res3c_branch2b = layers.Conv2D(128, (3,3), padding="same",␣
↪name="res3c_branch2b_")(activation_17_relu)
    bn3c_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3c_branch2b_")(res3c_branch2b)
    activation_18_relu = layers.ReLU()(bn3c_branch2b)
    res3c_branch2c = layers.Conv2D(512, (1,1),␣
↪name="res3c_branch2c_")(activation_18_relu)
    bn3c_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3c_branch2c_")(res3c_branch2c)
    add_6 = layers.Add()([bn3c_branch2c, activation_16_relu])
    activation_19_relu = layers.ReLU()(add_6)
    res3d_branch2a = layers.Conv2D(128, (1,1),␣
↪name="res3d_branch2a_")(activation_19_relu)
    bn3d_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3d_branch2a_")(res3d_branch2a)
    activation_20_relu = layers.ReLU()(bn3d_branch2a)
```

```
  res3d_branch2b = layers.Conv2D(128, (3,3), padding="same",␣
↪name="res3d_branch2b_")(activation_20_relu)
  bn3d_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3d_branch2b_")(res3d_branch2b)
  activation_21_relu = layers.ReLU()(bn3d_branch2b)
  res3d_branch2c = layers.Conv2D(512, (1,1),␣
↪name="res3d_branch2c_")(activation_21_relu)
  bn3d_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn3d_branch2c_")(res3d_branch2c)
  add_7 = layers.Add()([bn3d_branch2c, activation_19_relu])
  activation_22_relu = layers.ReLU()(add_7)
  res4a_branch2a = layers.Conv2D(256, (1,1), strides=(2,2),␣
↪name="res4a_branch2a_")(activation_22_relu)
  bn4a_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4a_branch2a_")(res4a_branch2a)
  activation_23_relu = layers.ReLU()(bn4a_branch2a)
  res4a_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4a_branch2b_")(activation_23_relu)
  bn4a_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4a_branch2b_")(res4a_branch2b)
  activation_24_relu = layers.ReLU()(bn4a_branch2b)
  res4a_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4a_branch2c_")(activation_24_relu)
  res4a_branch1 = layers.Conv2D(1024, (1,1), strides=(2,2),␣
↪name="res4a_branch1_")(activation_22_relu)
  bn4a_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4a_branch2c_")(res4a_branch2c)
  bn4a_branch1 = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4a_branch1_")(res4a_branch1)
  add_8 = layers.Add()([bn4a_branch2c, bn4a_branch1])
  activation_25_relu = layers.ReLU()(add_8)
  res4b_branch2a = layers.Conv2D(256, (1,1),␣
↪name="res4b_branch2a_")(activation_25_relu)
  bn4b_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4b_branch2a_")(res4b_branch2a)
  activation_26_relu = layers.ReLU()(bn4b_branch2a)
  res4b_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4b_branch2b_")(activation_26_relu)
  bn4b_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4b_branch2b_")(res4b_branch2b)
  activation_27_relu = layers.ReLU()(bn4b_branch2b)
  res4b_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4b_branch2c_")(activation_27_relu)
  bn4b_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4b_branch2c_")(res4b_branch2c)
  add_9 = layers.Add()([bn4b_branch2c, activation_25_relu])
```

```
  activation_28_relu = layers.ReLU()(add_9)
  res4c_branch2a = layers.Conv2D(256, (1,1),␣
↪name="res4c_branch2a_")(activation_28_relu)
  bn4c_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4c_branch2a_")(res4c_branch2a)
  activation_29_relu = layers.ReLU()(bn4c_branch2a)
  res4c_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4c_branch2b_")(activation_29_relu)
  bn4c_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4c_branch2b_")(res4c_branch2b)
  activation_30_relu = layers.ReLU()(bn4c_branch2b)
  res4c_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4c_branch2c_")(activation_30_relu)
  bn4c_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4c_branch2c_")(res4c_branch2c)
  add_10 = layers.Add()([bn4c_branch2c, activation_28_relu])
  activation_31_relu = layers.ReLU()(add_10)
  res4d_branch2a = layers.Conv2D(256, (1,1),␣
↪name="res4d_branch2a_")(activation_31_relu)
  bn4d_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4d_branch2a_")(res4d_branch2a)
  activation_32_relu = layers.ReLU()(bn4d_branch2a)
  res4d_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4d_branch2b_")(activation_32_relu)
  bn4d_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4d_branch2b_")(res4d_branch2b)
  activation_33_relu = layers.ReLU()(bn4d_branch2b)
  res4d_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4d_branch2c_")(activation_33_relu)
  bn4d_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4d_branch2c_")(res4d_branch2c)
  add_11 = layers.Add()([bn4d_branch2c, activation_31_relu])
  activation_34_relu = layers.ReLU()(add_11)
  res4e_branch2a = layers.Conv2D(256, (1,1),␣
↪name="res4e_branch2a_")(activation_34_relu)
  bn4e_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4e_branch2a_")(res4e_branch2a)
  activation_35_relu = layers.ReLU()(bn4e_branch2a)
  res4e_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4e_branch2b_")(activation_35_relu)
  bn4e_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4e_branch2b_")(res4e_branch2b)
  activation_36_relu = layers.ReLU()(bn4e_branch2b)
  res4e_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4e_branch2c_")(activation_36_relu)
```

```
  bn4e_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4e_branch2c_")(res4e_branch2c)
  add_12 = layers.Add()([bn4e_branch2c, activation_34_relu])
  activation_37_relu = layers.ReLU()(add_12)
  res4f_branch2a = layers.Conv2D(256, (1,1),␣
↪name="res4f_branch2a_")(activation_37_relu)
  bn4f_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4f_branch2a_")(res4f_branch2a)
  activation_38_relu = layers.ReLU()(bn4f_branch2a)
  res4f_branch2b = layers.Conv2D(256, (3,3), padding="same",␣
↪name="res4f_branch2b_")(activation_38_relu)
  bn4f_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4f_branch2b_")(res4f_branch2b)
  activation_39_relu = layers.ReLU()(bn4f_branch2b)
  res4f_branch2c = layers.Conv2D(1024, (1,1),␣
↪name="res4f_branch2c_")(activation_39_relu)
  bn4f_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn4f_branch2c_")(res4f_branch2c)
  add_13 = layers.Add()([bn4f_branch2c, activation_37_relu])
  activation_40_relu = layers.ReLU()(add_13)
  res5a_branch2a = layers.Conv2D(512, (1,1), strides=(2,2),␣
↪name="res5a_branch2a_")(activation_40_relu)
  bn5a_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5a_branch2a_")(res5a_branch2a)
  activation_41_relu = layers.ReLU()(bn5a_branch2a)
  res5a_branch2b = layers.Conv2D(512, (3,3), padding="same",␣
↪name="res5a_branch2b_")(activation_41_relu)
  bn5a_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5a_branch2b_")(res5a_branch2b)
  activation_42_relu = layers.ReLU()(bn5a_branch2b)
  res5a_branch2c = layers.Conv2D(2048, (1,1),␣
↪name="res5a_branch2c_")(activation_42_relu)
  res5a_branch1 = layers.Conv2D(2048, (1,1), strides=(2,2),␣
↪name="res5a_branch1_")(activation_40_relu)
  bn5a_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5a_branch2c_")(res5a_branch2c)
  bn5a_branch1 = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5a_branch1_")(res5a_branch1)
  add_14 = layers.Add()([bn5a_branch2c, bn5a_branch1])
  activation_43_relu = layers.ReLU()(add_14)
  res5b_branch2a = layers.Conv2D(512, (1,1),␣
↪name="res5b_branch2a_")(activation_43_relu)
  bn5b_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5b_branch2a_")(res5b_branch2a)
  activation_44_relu = layers.ReLU()(bn5b_branch2a)
```

```
    res5b_branch2b = layers.Conv2D(512, (3,3), padding="same",␣
↪name="res5b_branch2b_")(activation_44_relu)
    bn5b_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5b_branch2b_")(res5b_branch2b)
    activation_45_relu = layers.ReLU()(bn5b_branch2b)
    res5b_branch2c = layers.Conv2D(2048, (1,1),␣
↪name="res5b_branch2c_")(activation_45_relu)
    bn5b_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5b_branch2c_")(res5b_branch2c)
    add_15 = layers.Add()([bn5b_branch2c, activation_43_relu])
    activation_46_relu = layers.ReLU()(add_15)
    res5c_branch2a = layers.Conv2D(512, (1,1),␣
↪name="res5c_branch2a_")(activation_46_relu)
    bn5c_branch2a = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5c_branch2a_")(res5c_branch2a)
    activation_47_relu = layers.ReLU()(bn5c_branch2a)
    res5c_branch2b = layers.Conv2D(512, (3,3), padding="same",␣
↪name="res5c_branch2b_")(activation_47_relu)
    bn5c_branch2b = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5c_branch2b_")(res5c_branch2b)
    activation_48_relu = layers.ReLU()(bn5c_branch2b)
    res5c_branch2c = layers.Conv2D(2048, (1,1),␣
↪name="res5c_branch2c_")(activation_48_relu)
    bn5c_branch2c = layers.BatchNormalization(epsilon=0.001000,␣
↪name="bn5c_branch2c_")(res5c_branch2c)
    add_16 = layers.Add()([bn5c_branch2c, activation_46_relu])
    activation_49_relu = layers.ReLU()(add_16)
    avg_pool = layers.GlobalAveragePooling2D(keepdims=True)(activation_49_relu)
    fc1000 = layers.Reshape((-1,), name="fc1000_preFlatten1")(avg_pool)
    fc1000 = layers.Dense(9, name="fc1000_")(fc1000)
    fc1000_softmax = layers.Softmax()(fc1000)

    model = keras.Model(inputs=[input_1_unnormalized], outputs=[fc1000_softmax])
    return model
```

:

```
#    This file was created by
#    MATLAB Deep Learning Toolbox Converter for TensorFlow Models.
#    12-Apr-2024 10:49:57

#import myModel.model
import os

def load_model(load_weights=True, debug=False):
    m =create_model()
    if load_weights:
        loadWeights(m, debug=debug)
```

```
    return m

## Utility functions:

import tensorflow as tf
import h5py

def loadWeights(model, filename= "/content/weights.h5", debug=False):
    with h5py.File(filename, 'r') as f:
        # Every layer is an h5 group. Ignore non-groups (such as /0)
        for g in f:
            if isinstance(f[g], h5py.Group):
                group = f[g]
                layerName = group.attrs['Name']
                numVars = int(group.attrs['NumVars'])
                if debug:
                    print("layerName:", layerName)
                    print("    numVars:", numVars)
                # Find the layer index from its namevar
                layerIdx = layerNum(model, layerName)
                layer = model.layers[layerIdx]
                if debug:
                    print("    layerIdx=", layerIdx)
                # Every weight is an h5 dataset in the layer group. Read the␣
 ↪weights
                # into a list in the correct order
                weightList = [0]*numVars
                for d in group:
                    dataset = group[d]
                    varName = dataset.attrs['Name']
                    shp     = intList(dataset.attrs['Shape'])
                    weightNum = int(dataset.attrs['WeightNum'])
                    # Read the weight and put it into the right position in the␣
 ↪list
                    if debug:
                        print("    varName:", varName)
                        print("        shp:", shp)
                        print("        weightNum:", weightNum)
                    weightList[weightNum] = tf.constant(dataset[()], shape=shp)
                # Assign the weights into the layer
                for w in range(numVars):
                    if debug:
                        print("Copying variable of shape:")
                        print(weightList[w].shape)
                    layer.variables[w].assign(weightList[w])
                    if debug:
                        print("Assignment successful.")
```

```
                        print("Set variable value:")
                        print(layer.variables[w])
                # Finalize layer state
                if hasattr(layer, 'finalize_state'):
                    layer.finalize_state()

def layerNum(model, layerName):
    # Returns the index to the layer
    layers = model.layers
    for i in range(len(layers)):
        if layerName==layers[i].name:
            return i
    print("")
    print("WEIGHT LOADING FAILED. MODEL DOES NOT CONTAIN LAYER WITH NAME: ",␣
 ↪layerName)
    print("")
    return -1

def intList(myList):
    # Converts a list of numbers into a list of ints.
    return list(map(int, myList))
```

```
: import tensorflow as tf
tf.__version__
import keras
from keras.models import Sequential
from keras import layers
from keras.layers import␣
 ↪Conv2D,Flatten,Dropout,Dense,MaxPooling2D,BatchNormalization,Flatten




#model=create_model()
model=load_model()
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

import tensorflow as tf


epochs = 30
history = model.fit(
  train_ds,
```

```
  validation_data=test_ds,
  epochs=epochs
)
```

```
<ipython-input-43-23111b96cb07>:26: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  numVars = int(group.attrs['NumVars'])
<ipython-input-43-23111b96cb07>:42: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  weightNum = int(dataset.attrs['WeightNum'])
```

```
Epoch 1/30
63/63 [==============================] - 164s 2s/step - loss: 0.4143 - accuracy:
0.8735 - val_loss: 20.3348 - val_accuracy: 0.1737
Epoch 2/30
63/63 [==============================] - 101s 2s/step - loss: 0.1644 - accuracy:
0.9482 - val_loss: 1.5760 - val_accuracy: 0.6019
Epoch 3/30
63/63 [==============================] - 101s 2s/step - loss: 0.0933 - accuracy:
0.9698 - val_loss: 0.5358 - val_accuracy: 0.8632
Epoch 4/30
63/63 [==============================] - 102s 2s/step - loss: 0.0758 - accuracy:
0.9770 - val_loss: 3.1708 - val_accuracy: 0.5541
Epoch 5/30
63/63 [==============================] - 101s 2s/step - loss: 0.0544 - accuracy:
```

```
Epoch 28/30
63/63 [==============================] - 99s 2s/step - loss: 0.0095 - accuracy:
0.9972 - val_loss: 0.2621 - val_accuracy: 0.9459
Epoch 29/30
63/63 [==============================] - 98s 2s/step - loss: 0.0039 - accuracy:
0.9987 - val_loss: 0.2759 - val_accuracy: 0.9498
Epoch 30/30
63/63 [==============================] - 99s 2s/step - loss: 0.0029 - accuracy:
0.9991 - val_loss: 0.2196 - val_accuracy: 0.9574
```

```
print(class_names)
path='/content/sem-8-major-project/new hawk/Hawk_Eye_dataset/validation/Pistol/
  ↪265.jpg'
img = tf.keras.utils.load_img(
    path, target_size=(224,224)
)
```

```
img_array = np.expand_dims(img, axis=0)
predictions = model.predict(img_array)
```

```
score = predictions[0]
val = np.argmax(score)


print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
['Bat', 'Car', 'Grenade', 'Knife', 'Machine Guns', 'Masked Face', 'Motorcycle',
'Pistol', 'face']
1/1 [==============================] - 0s 77ms/step
This image most likely belongs to Pistol with a 100.00 percent confidence.
```

[67]:
```
img
```

[67]:

c)Resnet-18 model:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def create_model():
    data_unnormalized = keras.Input(shape=(224,224,3), name="data_unnormalized")
    data = keras.layers.Normalization(axis=(1,2,3), name="data_")(data_unnormalized)
    conv1_prepadded = layers.ZeroPadding2D(padding=((3,3),(3,3)))(data)
    conv1 = layers.Conv2D(64, (7,7), strides=(2,2), name="conv1_")(conv1_prepadded)
    bn_conv1 = layers.BatchNormalization(epsilon=0.000010, name="bn_conv1_")(conv1)
    conv1_relu = layers.ReLU()(bn_conv1)
    pool1_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(conv1_relu)
    pool1 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(pool1_prepadded)
    res2a_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(pool1)
    res2a_branch2a = layers.Conv2D(64, (3,3), name="res2a_branch2a_")(res2a_branch2a_prepadded)
    bn2a_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn2a_branch2a_")(res2a_branch2a)
    res2a_branch2a_relu = layers.ReLU()(bn2a_branch2a)
    res2a_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res2a_branch2a_relu)
    res2a_branch2b = layers.Conv2D(64, (3,3), name="res2a_branch2b_")(res2a_branch2b_prepadded)
    bn2a_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn2a_branch2b_")(res2a_branch2b)
    res2a = layers.Add()([bn2a_branch2b, pool1])
    res2a_relu = layers.ReLU()(res2a)
    res2b_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res2a_relu)
    res2b_branch2a = layers.Conv2D(64, (3,3), name="res2b_branch2a_")(res2b_branch2a_prepadded)
    bn2b_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn2b_branch2a_")(res2b_branch2a)
    res2b_branch2a_relu = layers.ReLU()(bn2b_branch2a)
    res2b_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res2b_branch2a_relu)
    res2b_branch2b = layers.Conv2D(64, (3,3), name="res2b_branch2b_")(res2b_branch2b_prepadded)
    bn2b_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn2b_branch2b_")(res2b_branch2b)
    res2b = layers.Add()([bn2b_branch2b, res2a_relu])
    res2b_relu = layers.ReLU()(res2b)
    res3a_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res2b_relu)
    res3a_branch2a = layers.Conv2D(128, (3,3), strides=(2,2), name="res3a_branch2a_")(res3a_branch2a_prepadded)
    bn3a_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn3a_branch2a_")(res3a_branch2a)
    res3a_branch2a_relu = layers.ReLU()(bn3a_branch2a)
    res3a_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res3a_branch2a_relu)
    res3a_branch2b = layers.Conv2D(128, (3,3), name="res3a_branch2b_")(res3a_branch2b_prepadded)
```

```
bn3a_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn3a_branch2b_")(res3a_branch2b)
res3a_branch1 = layers.Conv2D(128, (1,1), strides=(2,2), name="res3a_branch1_")(res2b_relu)
bn3a_branch1 = layers.BatchNormalization(epsilon=0.000010, name="bn3a_branch1_")(res3a_branch1)
res3a = layers.Add()([bn3a_branch2b, bn3a_branch1])
res3a_relu = layers.ReLU()(res3a)
res3b_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res3a_relu)
res3b_branch2a = layers.Conv2D(128, (3,3), name="res3b_branch2a_")(res3b_branch2a_prepadded)
bn3b_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn3b_branch2a_")(res3b_branch2a)
res3b_branch2a_relu = layers.ReLU()(bn3b_branch2a)
res3b_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res3b_branch2a_relu)
res3b_branch2b = layers.Conv2D(128, (3,3), name="res3b_branch2b_")(res3b_branch2b_prepadded)
bn3b_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn3b_branch2b_")(res3b_branch2b)
res3b = layers.Add()([bn3b_branch2b, res3a_relu])
res3b_relu = layers.ReLU()(res3b)
res4a_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res3b_relu)
res4a_branch2a = layers.Conv2D(256, (3,3), strides=(2,2), name="res4a_branch2a_")(res4a_branch2a_prepadded)
bn4a_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn4a_branch2a_")(res4a_branch2a)
res4a_branch2a_relu = layers.ReLU()(bn4a_branch2a)
res4a_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res4a_branch2a_relu)
res4a_branch2b = layers.Conv2D(256, (3,3), name="res4a_branch2b_")(res4a_branch2b_prepadded)
bn4a_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn4a_branch2b_")(res4a_branch2b)
res4a_branch1 = layers.Conv2D(256, (1,1), strides=(2,2), name="res4a_branch1_")(res3b_relu)
bn4a_branch1 = layers.BatchNormalization(epsilon=0.000010, name="bn4a_branch1_")(res4a_branch1)
res4a = layers.Add()([bn4a_branch2b, bn4a_branch1])
res4a_relu = layers.ReLU()(res4a)
res4b_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res4a_relu)
res4b_branch2a = layers.Conv2D(256, (3,3), name="res4b_branch2a_")(res4b_branch2a_prepadded)
bn4b_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn4b_branch2a_")(res4b_branch2a)
res4b_branch2a_relu = layers.ReLU()(bn4b_branch2a)
res4b_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res4b_branch2a_relu)
res4b_branch2b = layers.Conv2D(256, (3,3), name="res4b_branch2b_")(res4b_branch2b_prepadded)
bn4b_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn4b_branch2b_")(res4b_branch2b)
res4b = layers.Add()([bn4b_branch2b, res4a_relu])
res4b_relu = layers.ReLU()(res4b)
res5a_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res4b_relu)
res5a_branch2a = layers.Conv2D(512, (3,3), strides=(2,2), name="res5a_branch2a_")(res5a_branch2a_prepadded)
bn5a_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn5a_branch2a_")(res5a_branch2a)
```

```
    res5a_branch2a_relu = layers.ReLU()(bn5a_branch2a)
    res5a_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res5a_branch2a_relu)
    res5a_branch2b = layers.Conv2D(512, (3,3), name="res5a_branch2b_")(res5a_branch2b_prepadded)
    bn5a_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn5a_branch2b_")(res5a_branch2b)
    res5a_branch1 = layers.Conv2D(512, (1,1), strides=(2,2), name="res5a_branch1_")(res4b_relu)
    bn5a_branch1 = layers.BatchNormalization(epsilon=0.000010, name="bn5a_branch1_")(res5a_branch1)
    res5a = layers.Add()([bn5a_branch2b, bn5a_branch1])
    res5a_relu = layers.ReLU()(res5a)
    res5b_branch2a_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res5a_relu)
    res5b_branch2a = layers.Conv2D(512, (3,3), name="res5b_branch2a_")(res5b_branch2a_prepadded)
    bn5b_branch2a = layers.BatchNormalization(epsilon=0.000010, name="bn5b_branch2a_")(res5b_branch2a)
    res5b_branch2a_relu = layers.ReLU()(bn5b_branch2a)
    res5b_branch2b_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(res5b_branch2a_relu)
    res5b_branch2b = layers.Conv2D(512, (3,3), name="res5b_branch2b_")(res5b_branch2b_prepadded)
    bn5b_branch2b = layers.BatchNormalization(epsilon=0.000010, name="bn5b_branch2b_")(res5b_branch2b)
    res5b = layers.Add()([bn5b_branch2b, res5a_relu])
    res5b_relu = layers.ReLU()(res5b)
    pool5 = layers.GlobalAveragePooling2D(keepdims=True)(res5b_relu)
    fc1000 = layers.Reshape((-1,), name="fc1000_preFlatten1")(pool5)
    fc1000 = layers.Dense(9, name="fc1000_")(fc1000)
    prob = layers.Softmax()(fc1000)

    model = keras.Model(inputs=[data_unnormalized], outputs=[prob])
    return model
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def create_model():
    input_1_unnormalized = keras.Input(shape=(224,224,3), name="input_1_unnormalized")
    input_1 = keras.layers.Normalization(axis=(1,2,3), name="input_1_")(input_1_unnormalized)
    conv1_prepadded = layers.ZeroPadding2D(padding=((3,3),(3,3)))(input_1)
    conv1 = layers.Conv2D(64, (7,7), strides=(2,2), name="conv1_")(conv1_prepadded)
    bn_conv1 = layers.BatchNormalization(epsilon=0.001000, name="bn_conv1_")(conv1)
    activation_1_relu = layers.ReLU()(bn_conv1)
    max_pooling2d_1_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(activation_1_relu)
    max_pooling2d_1 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(max_pooling2d_1_prepadded)
    res2a_branch2a = layers.Conv2D(64, (1,1), name="res2a_branch2a_")(max_pooling2d_1)
    bn2a_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn2a_branch2a_")(res2a_branch2a)
    activation_2_relu = layers.ReLU()(bn2a_branch2a)
    res2a_branch2b = layers.Conv2D(64, (3,3), padding="same", name="res2a_branch2b_")(activation_2_relu)
    bn2a_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn2a_branch2b_")(res2a_branch2b)
    activation_3_relu = layers.ReLU()(bn2a_branch2b)
    res2a_branch2c = layers.Conv2D(256, (1,1), name="res2a_branch2c_")(activation_3_relu)
    res2a_branch1 = layers.Conv2D(256, (1,1), name="res2a_branch1_")(max_pooling2d_1)
    bn2a_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn2a_branch2c_")(res2a_branch2c)
    bn2a_branch1 = layers.BatchNormalization(epsilon=0.001000, name="bn2a_branch1_")(res2a_branch1)
    add_1 = layers.Add()([bn2a_branch2c, bn2a_branch1])
    activation_4_relu = layers.ReLU()(add_1)
    res2b_branch2a = layers.Conv2D(64, (1,1), name="res2b_branch2a_")(activation_4_relu)
    bn2b_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn2b_branch2a_")(res2b_branch2a)
    activation_5_relu = layers.ReLU()(bn2b_branch2a)
    res2b_branch2b = layers.Conv2D(64, (3,3), padding="same", name="res2b_branch2b_")(activation_5_relu)
    bn2b_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn2b_branch2b_")(res2b_branch2b)
    activation_6_relu = layers.ReLU()(bn2b_branch2b)
    res2b_branch2c = layers.Conv2D(256, (1,1), name="res2b_branch2c_")(activation_6_relu)
    bn2b_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn2b_branch2c_")(res2b_branch2c)
    add_2 = layers.Add()([bn2b_branch2c, activation_4_relu])
    activation_7_relu = layers.ReLU()(add_2)
    res2c_branch2a = layers.Conv2D(64, (1,1), name="res2c_branch2a_")(activation_7_relu)
    bn2c_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn2c_branch2a_")(res2c_branch2a)
```

```
activation_8_relu = layers.ReLU()(bn2c_branch2a)
res2c_branch2b = layers.Conv2D(64, (3,3), padding="same", name="res2c_branch2b_")(activation_8_relu)
bn2c_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn2c_branch2b_")(res2c_branch2b)
activation_9_relu = layers.ReLU()(bn2c_branch2b)
res2c_branch2c = layers.Conv2D(256, (1,1), name="res2c_branch2c_")(activation_9_relu)
bn2c_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn2c_branch2c_")(res2c_branch2c)
add_3 = layers.Add()([bn2c_branch2c, activation_7_relu])
activation_10_relu = layers.ReLU()(add_3)
res3a_branch2a = layers.Conv2D(128, (1,1), strides=(2,2), name="res3a_branch2a_")(activation_10_relu)
bn3a_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn3a_branch2a_")(res3a_branch2a)
activation_11_relu = layers.ReLU()(bn3a_branch2a)
res3a_branch2b = layers.Conv2D(128, (3,3), padding="same", name="res3a_branch2b_")(activation_11_relu)
bn3a_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn3a_branch2b_")(res3a_branch2b)
activation_12_relu = layers.ReLU()(bn3a_branch2b)
res3a_branch2c = layers.Conv2D(512, (1,1), name="res3a_branch2c_")(activation_12_relu)
res3a_branch1 = layers.Conv2D(512, (1,1), strides=(2,2), name="res3a_branch1_")(activation_10_relu)
bn3a_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn3a_branch2c_")(res3a_branch2c)
bn3a_branch1 = layers.BatchNormalization(epsilon=0.001000, name="bn3a_branch1_")(res3a_branch1)
add_4 = layers.Add()([bn3a_branch2c, bn3a_branch1])
activation_13_relu = layers.ReLU()(add_4)
res3b_branch2a = layers.Conv2D(128, (1,1), name="res3b_branch2a_")(activation_13_relu)
bn3b_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn3b_branch2a_")(res3b_branch2a)
activation_14_relu = layers.ReLU()(bn3b_branch2a)
res3b_branch2b = layers.Conv2D(128, (3,3), padding="same", name="res3b_branch2b_")(activation_14_relu)
bn3b_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn3b_branch2b_")(res3b_branch2b)
activation_15_relu = layers.ReLU()(bn3b_branch2b)
res3b_branch2c = layers.Conv2D(512, (1,1), name="res3b_branch2c_")(activation_15_relu)
bn3b_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn3b_branch2c_")(res3b_branch2c)
add_5 = layers.Add()([bn3b_branch2c, activation_13_relu])
activation_16_relu = layers.ReLU()(add_5)
res3c_branch2a = layers.Conv2D(128, (1,1), name="res3c_branch2a_")(activation_16_relu)
bn3c_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn3c_branch2a_")(res3c_branch2a)
activation_17_relu = layers.ReLU()(bn3c_branch2a)
res3c_branch2b = layers.Conv2D(128, (3,3), padding="same", name="res3c_branch2b_")(activation_17_relu)
bn3c_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn3c_branch2b_")(res3c_branch2b)
activation_18_relu = layers.ReLU()(bn3c_branch2b)
res3c_branch2c = layers.Conv2D(512, (1,1), name="res3c_branch2c_")(activation_18_relu)
```

```
bn3c_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn3c_branch2c_")(res3c_branch2c)
add_6 = layers.Add()([bn3c_branch2c, activation_16_relu])
activation_19_relu = layers.ReLU()(add_6)
res3d_branch2a = layers.Conv2D(128, (1,1), name="res3d_branch2a_")(activation_19_relu)
bn3d_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn3d_branch2a_")(res3d_branch2a)
activation_20_relu = layers.ReLU()(bn3d_branch2a)
res3d_branch2b = layers.Conv2D(128, (3,3), padding="same", name="res3d_branch2b_")(activation_20_relu)
bn3d_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn3d_branch2b_")(res3d_branch2b)
activation_21_relu = layers.ReLU()(bn3d_branch2b)
res3d_branch2c = layers.Conv2D(512, (1,1), name="res3d_branch2c_")(activation_21_relu)
bn3d_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn3d_branch2c_")(res3d_branch2c)
add_7 = layers.Add()([bn3d_branch2c, activation_19_relu])
activation_22_relu = layers.ReLU()(add_7)
res4a_branch2a = layers.Conv2D(256, (1,1), strides=(2,2), name="res4a_branch2a_")(activation_22_relu)
bn4a_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4a_branch2a_")(res4a_branch2a)
activation_23_relu = layers.ReLU()(bn4a_branch2a)
res4a_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4a_branch2b_")(activation_23_relu)
bn4a_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4a_branch2b_")(res4a_branch2b)
activation_24_relu = layers.ReLU()(bn4a_branch2b)
res4a_branch2c = layers.Conv2D(1024, (1,1), name="res4a_branch2c_")(activation_24_relu)
res4a_branch1 = layers.Conv2D(1024, (1,1), strides=(2,2), name="res4a_branch1_")(activation_22_relu)
bn4a_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4a_branch2c_")(res4a_branch2c)
bn4a_branch1 = layers.BatchNormalization(epsilon=0.001000, name="bn4a_branch1_")(res4a_branch1)
add_8 = layers.Add()([bn4a_branch2c, bn4a_branch1])
activation_25_relu = layers.ReLU()(add_8)
res4b_branch2a = layers.Conv2D(256, (1,1), name="res4b_branch2a_")(activation_25_relu)
bn4b_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4b_branch2a_")(res4b_branch2a)
activation_26_relu = layers.ReLU()(bn4b_branch2a)
res4b_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4b_branch2b_")(activation_26_relu)
bn4b_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4b_branch2b_")(res4b_branch2b)
activation_27_relu = layers.ReLU()(bn4b_branch2b)
res4b_branch2c = layers.Conv2D(1024, (1,1), name="res4b_branch2c_")(activation_27_relu)
bn4b_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4b_branch2c_")(res4b_branch2c)
add_9 = layers.Add()([bn4b_branch2c, activation_25_relu])
activation_28_relu = layers.ReLU()(add_9)
res4c_branch2a = layers.Conv2D(256, (1,1), name="res4c_branch2a_")(activation_28_relu)
bn4c_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4c_branch2a_")(res4c_branch2a)
```

```
activation_29_relu = layers.ReLU()(bn4c_branch2a)
res4c_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4c_branch2b_")(activation_29_relu)
bn4c_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4c_branch2b_")(res4c_branch2b)
activation_30_relu = layers.ReLU()(bn4c_branch2b)
res4c_branch2c = layers.Conv2D(1024, (1,1), name="res4c_branch2c_")(activation_30_relu)
bn4c_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4c_branch2c_")(res4c_branch2c)
add_10 = layers.Add()([bn4c_branch2c, activation_28_relu])
activation_31_relu = layers.ReLU()(add_10)
res4d_branch2a = layers.Conv2D(256, (1,1), name="res4d_branch2a_")(activation_31_relu)
bn4d_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4d_branch2a_")(res4d_branch2a)
activation_32_relu = layers.ReLU()(bn4d_branch2a)
res4d_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4d_branch2b_")(activation_32_relu)
bn4d_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4d_branch2b_")(res4d_branch2b)
activation_33_relu = layers.ReLU()(bn4d_branch2b)
res4d_branch2c = layers.Conv2D(1024, (1,1), name="res4d_branch2c_")(activation_33_relu)
bn4d_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4d_branch2c_")(res4d_branch2c)
add_11 = layers.Add()([bn4d_branch2c, activation_31_relu])
activation_34_relu = layers.ReLU()(add_11)
res4e_branch2a = layers.Conv2D(256, (1,1), name="res4e_branch2a_")(activation_34_relu)
bn4e_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4e_branch2a_")(res4e_branch2a)
activation_35_relu = layers.ReLU()(bn4e_branch2a)
res4e_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4e_branch2b_")(activation_35_relu)
bn4e_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4e_branch2b_")(res4e_branch2b)
activation_36_relu = layers.ReLU()(bn4e_branch2b)
res4e_branch2c = layers.Conv2D(1024, (1,1), name="res4e_branch2c_")(activation_36_relu)
bn4e_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4e_branch2c_")(res4e_branch2c)
add_12 = layers.Add()([bn4e_branch2c, activation_34_relu])
activation_37_relu = layers.ReLU()(add_12)
res4f_branch2a = layers.Conv2D(256, (1,1), name="res4f_branch2a_")(activation_37_relu)
bn4f_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn4f_branch2a_")(res4f_branch2a)
activation_38_relu = layers.ReLU()(bn4f_branch2a)
res4f_branch2b = layers.Conv2D(256, (3,3), padding="same", name="res4f_branch2b_")(activation_38_relu)
bn4f_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn4f_branch2b_")(res4f_branch2b)
activation_39_relu = layers.ReLU()(bn4f_branch2b)
res4f_branch2c = layers.Conv2D(1024, (1,1), name="res4f_branch2c_")(activation_39_relu)
bn4f_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn4f_branch2c_")(res4f_branch2c)
add_13 = layers.Add()([bn4f_branch2c, activation_37_relu])
```

```
activation_40_relu = layers.ReLU()(add_13)
res5a_branch2a = layers.Conv2D(512, (1,1), strides=(2,2), name="res5a_branch2a_")(activation_40_relu)
bn5a_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn5a_branch2a_")(res5a_branch2a)
activation_41_relu = layers.ReLU()(bn5a_branch2a)
res5a_branch2b = layers.Conv2D(512, (3,3), padding="same", name="res5a_branch2b_")(activation_41_relu)
bn5a_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn5a_branch2b_")(res5a_branch2b)
activation_42_relu = layers.ReLU()(bn5a_branch2b)
res5a_branch2c = layers.Conv2D(2048, (1,1), name="res5a_branch2c_")(activation_42_relu)
res5a_branch1 = layers.Conv2D(2048, (1,1), strides=(2,2), name="res5a_branch1_")(activation_40_relu)
bn5a_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn5a_branch2c_")(res5a_branch2c)
bn5a_branch1 = layers.BatchNormalization(epsilon=0.001000, name="bn5a_branch1_")(res5a_branch1)
add_14 = layers.Add()([bn5a_branch2c, bn5a_branch1])
activation_43_relu = layers.ReLU()(add_14)
res5b_branch2a = layers.Conv2D(512, (1,1), name="res5b_branch2a_")(activation_43_relu)
bn5b_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn5b_branch2a_")(res5b_branch2a)
activation_44_relu = layers.ReLU()(bn5b_branch2a)
res5b_branch2b = layers.Conv2D(512, (3,3), padding="same", name="res5b_branch2b_")(activation_44_relu)
bn5b_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn5b_branch2b_")(res5b_branch2b)
activation_45_relu = layers.ReLU()(bn5b_branch2b)
res5b_branch2c = layers.Conv2D(2048, (1,1), name="res5b_branch2c_")(activation_45_relu)
bn5b_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn5b_branch2c_")(res5b_branch2c)
add_15 = layers.Add()([bn5b_branch2c, activation_43_relu])
activation_46_relu = layers.ReLU()(add_15)
res5c_branch2a = layers.Conv2D(512, (1,1), name="res5c_branch2a_")(activation_46_relu)
bn5c_branch2a = layers.BatchNormalization(epsilon=0.001000, name="bn5c_branch2a_")(res5c_branch2a)
activation_47_relu = layers.ReLU()(bn5c_branch2a)
res5c_branch2b = layers.Conv2D(512, (3,3), padding="same", name="res5c_branch2b_")(activation_47_relu)
bn5c_branch2b = layers.BatchNormalization(epsilon=0.001000, name="bn5c_branch2b_")(res5c_branch2b)
activation_48_relu = layers.ReLU()(bn5c_branch2b)
res5c_branch2c = layers.Conv2D(2048, (1,1), name="res5c_branch2c_")(activation_48_relu)
bn5c_branch2c = layers.BatchNormalization(epsilon=0.001000, name="bn5c_branch2c_")(res5c_branch2c)
add_16 = layers.Add()([bn5c_branch2c, activation_46_relu])
activation_49_relu = layers.ReLU()(add_16)
avg_pool = layers.GlobalAveragePooling2D(keepdims=True)(activation_49_relu)
fc1000 = layers.Reshape((-1,), name="fc1000_preFlatten1")(avg_pool)
fc1000 = layers.Dense(9, name="fc1000_")(fc1000)
fc1000_softmax = layers.Softmax()(fc1000)
```

5.1.3 Backend (Django)

```
# views.py
from django.shortcuts import render
from django.shortcuts import render
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from django.http import HttpResponse
import os
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
def modelling():
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    def create_model():
        data_unnormalized = keras.Input(shape=(224, 224, 3), name="data_unnormalized")
        data = keras.layers.Normalization(axis=(1, 2, 3), name="data_")(data_unnormalized)
        conv1_7x7_s2_prepadded = layers.ZeroPadding2D(padding=((3, 3), (3, 3)))(data)
        conv1_7x7_s2 = layers.Conv2D(64, (7, 7), strides=(2, 2), name="conv1_7x7_s2_")(conv1_7x7_s2_prepadded)
        conv1_relu_7x7 = layers.ReLU()(conv1_7x7_s2)
        pool1_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0, 1), (0, 1)))(conv1_relu_7x7)
        pool1_3x3_s2 = layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2))(pool1_3x3_s2_prepadded)
```

```
CCNormLayer = layers.Lambda(

  lambda X: tf.nn.local_response_normalization(X, depth_radius=2.000000, bias=1.000000, alpha=0.000020,

                                               beta=0.750000))

pool1_norm1 = CCNormLayer(pool1_3x3_s2)

conv2_3x3_reduce = layers.Conv2D(64, (1, 1), name="conv2_3x3_reduce_")(pool1_norm1)

conv2_relu_3x3_reduce = layers.ReLU()(conv2_3x3_reduce)

conv2_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(conv2_relu_3x3_reduce)

conv2_3x3 = layers.Conv2D(192, (3, 3), name="conv2_3x3_")(conv2_3x3_prepadded)

conv2_relu_3x3 = layers.ReLU()(conv2_3x3)

CCNormLayer = layers.Lambda(

  lambda X: tf.nn.local_response_normalization(X, depth_radius=2.000000, bias=1.000000, alpha=0.000020,

                                               beta=0.750000))

conv2_norm2 = CCNormLayer(conv2_relu_3x3)

pool2_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0, 1), (0, 1)))(conv2_norm2)

pool2_3x3_s2 = layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2))(pool2_3x3_s2_prepadded)

inception_3a_1x1 = layers.Conv2D(64, (1, 1), name="inception_3a_1x1_")(pool2_3x3_s2)

inception_3a_relu_1x1 = layers.ReLU()(inception_3a_1x1)

inception_3a_3x3_reduce = layers.Conv2D(96, (1, 1), name="inception_3a_3x3_reduce_")(pool2_3x3_s2)

inception_3a_relu_3x3_reduce = layers.ReLU()(inception_3a_3x3_reduce)

inception_3a_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_3a_relu_3x3_reduce)

inception_3a_3x3 = layers.Conv2D(128, (3, 3), name="inception_3a_3x3_")(inception_3a_3x3_prepadded)
```

```
inception_3a_relu_3x3 = layers.ReLU()(inception_3a_3x3)

inception_3a_5x5_reduce = layers.Conv2D(16, (1, 1), name="inception_3a_5x5_reduce_")(pool2_3x3_s2)

inception_3a_relu_5x5_reduce = layers.ReLU()(inception_3a_5x5_reduce)

inception_3a_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_3a_relu_5x5_reduce)

inception_3a_5x5 = layers.Conv2D(32, (5, 5), name="inception_3a_5x5_")(inception_3a_5x5_prepadded)

inception_3a_relu_5x5 = layers.ReLU()(inception_3a_5x5)

inception_3a_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(pool2_3x3_s2)

inception_3a_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_3a_pool_prepadded)

inception_3a_pool_proj = layers.Conv2D(32, (1, 1), name="inception_3a_pool_proj_")(inception_3a_pool)

inception_3a_relu_pool_proj = layers.ReLU()(inception_3a_pool_proj)

inception_3a_output = layers.Concatenate(axis=-1)(

  [inception_3a_relu_1x1, inception_3a_relu_3x3, inception_3a_relu_5x5, inception_3a_relu_pool_proj])

inception_3b_1x1 = layers.Conv2D(128, (1, 1), name="inception_3b_1x1_")(inception_3a_output)

inception_3b_relu_1x1 = layers.ReLU()(inception_3b_1x1)

inception_3b_3x3_reduce = layers.Conv2D(128, (1, 1), name="inception_3b_3x3_reduce_")(inception_3a_output)

inception_3b_relu_3x3_reduce = layers.ReLU()(inception_3b_3x3_reduce)

inception_3b_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_3b_relu_3x3_reduce)

inception_3b_3x3 = layers.Conv2D(192, (3, 3), name="inception_3b_3x3_")(inception_3b_3x3_prepadded)

inception_3b_relu_3x3 = layers.ReLU()(inception_3b_3x3)

inception_3b_5x5_reduce = layers.Conv2D(32, (1, 1), name="inception_3b_5x5_reduce_")(inception_3a_output)
```

```
inception_3b_relu_5x5_reduce = layers.ReLU()(inception_3b_5x5_reduce)

inception_3b_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_3b_relu_5x5_reduce)

inception_3b_5x5 = layers.Conv2D(96, (5, 5), name="inception_3b_5x5_")(inception_3b_5x5_prepadded)

inception_3b_relu_5x5 = layers.ReLU()(inception_3b_5x5)

inception_3b_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_3a_output)

inception_3b_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_3b_pool_prepadded)

inception_3b_pool_proj = layers.Conv2D(64, (1, 1), name="inception_3b_pool_proj_")(inception_3b_pool)

inception_3b_relu_pool_proj = layers.ReLU()(inception_3b_pool_proj)

inception_3b_output = layers.Concatenate(axis=-1)(

  [inception_3b_relu_1x1, inception_3b_relu_3x3, inception_3b_relu_5x5, inception_3b_relu_pool_proj])

pool3_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_3b_output)

pool3_3x3_s2 = layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2))(pool3_3x3_s2_prepadded)

inception_4a_1x1 = layers.Conv2D(192, (1, 1), name="inception_4a_1x1_")(pool3_3x3_s2)

inception_4a_relu_1x1 = layers.ReLU()(inception_4a_1x1)

inception_4a_3x3_reduce = layers.Conv2D(96, (1, 1), name="inception_4a_3x3_reduce_")(pool3_3x3_s2)

inception_4a_relu_3x3_reduce = layers.ReLU()(inception_4a_3x3_reduce)

inception_4a_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4a_relu_3x3_reduce)

inception_4a_3x3 = layers.Conv2D(208, (3, 3), name="inception_4a_3x3_")(inception_4a_3x3_prepadded)

inception_4a_relu_3x3 = layers.ReLU()(inception_4a_3x3)

inception_4a_5x5_reduce = layers.Conv2D(16, (1, 1), name="inception_4a_5x5_reduce_")(pool3_3x3_s2)
```

```
inception_4a_relu_5x5_reduce = layers.ReLU()(inception_4a_5x5_reduce)

inception_4a_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_4a_relu_5x5_reduce)

inception_4a_5x5 = layers.Conv2D(48, (5, 5), name="inception_4a_5x5_")(inception_4a_5x5_prepadded)

inception_4a_relu_5x5 = layers.ReLU()(inception_4a_5x5)

inception_4a_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(pool3_3x3_s2)

inception_4a_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_4a_pool_prepadded)

inception_4a_pool_proj = layers.Conv2D(64, (1, 1), name="inception_4a_pool_proj_")(inception_4a_pool)

inception_4a_relu_pool_proj = layers.ReLU()(inception_4a_pool_proj)

inception_4a_output = layers.Concatenate(axis=-1)(

  [inception_4a_relu_1x1, inception_4a_relu_3x3, inception_4a_relu_5x5, inception_4a_relu_pool_proj])

inception_4b_1x1 = layers.Conv2D(160, (1, 1), name="inception_4b_1x1_")(inception_4a_output)

inception_4b_relu_1x1 = layers.ReLU()(inception_4b_1x1)

inception_4b_3x3_reduce = layers.Conv2D(112, (1, 1), name="inception_4b_3x3_reduce_")(inception_4a_output)

inception_4b_relu_3x3_reduce = layers.ReLU()(inception_4b_3x3_reduce)

inception_4b_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4b_relu_3x3_reduce)

inception_4b_3x3 = layers.Conv2D(224, (3, 3), name="inception_4b_3x3_")(inception_4b_3x3_prepadded)

inception_4b_relu_3x3 = layers.ReLU()(inception_4b_3x3)

inception_4b_5x5_reduce = layers.Conv2D(24, (1, 1), name="inception_4b_5x5_reduce_")(inception_4a_output)

inception_4b_relu_5x5_reduce = layers.ReLU()(inception_4b_5x5_reduce)

inception_4b_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_4b_relu_5x5_reduce)
```

```
    inception_4b_5x5 = layers.Conv2D(64, (5, 5), name="inception_4b_5x5_")(inception_4b_5x5_prepadded)

    inception_4b_relu_5x5 = layers.ReLU()(inception_4b_5x5)

    inception_4b_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4a_output)

    inception_4b_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_4b_pool_prepadded)

    inception_4b_pool_proj = layers.Conv2D(64, (1, 1), name="inception_4b_pool_proj_")(inception_4b_pool)

    inception_4b_relu_pool_proj = layers.ReLU()(inception_4b_pool_proj)

    inception_4b_output = layers.Concatenate(axis=-1)(

      [inception_4b_relu_1x1, inception_4b_relu_3x3, inception_4b_relu_5x5, inception_4b_relu_pool_proj])

    inception_4c_1x1 = layers.Conv2D(128, (1, 1), name="inception_4c_1x1_")(inception_4b_output)

    inception_4c_relu_1x1 = layers.ReLU()(inception_4c_1x1)

    inception_4c_3x3_reduce = layers.Conv2D(128, (1, 1), name="inception_4c_3x3_reduce_")(inception_4b_output)

    inception_4c_relu_3x3_reduce = layers.ReLU()(inception_4c_3x3_reduce)

    inception_4c_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4c_relu_3x3_reduce)

    inception_4c_3x3 = layers.Conv2D(256, (3, 3), name="inception_4c_3x3_")(inception_4c_3x3_prepadded)

    inception_4c_relu_3x3 = layers.ReLU()(inception_4c_3x3)

    inception_4c_5x5_reduce = layers.Conv2D(24, (1, 1), name="inception_4c_5x5_reduce_")(inception_4b_output)

    inception_4c_relu_5x5_reduce = layers.ReLU()(inception_4c_5x5_reduce)

    inception_4c_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_4c_relu_5x5_reduce)

    inception_4c_5x5 = layers.Conv2D(64, (5, 5), name="inception_4c_5x5_")(inception_4c_5x5_prepadded)

    inception_4c_relu_5x5 = layers.ReLU()(inception_4c_5x5)
```

```
    inception_4c_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4b_output)

    inception_4c_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_4c_pool_prepadded)

    inception_4c_pool_proj = layers.Conv2D(64, (1, 1), name="inception_4c_pool_proj_")(inception_4c_pool)

    inception_4c_relu_pool_proj = layers.ReLU()(inception_4c_pool_proj)

    inception_4c_output = layers.Concatenate(axis=-1)(

      [inception_4c_relu_1x1, inception_4c_relu_3x3, inception_4c_relu_5x5, inception_4c_relu_pool_proj])

    inception_4d_1x1 = layers.Conv2D(112, (1, 1), name="inception_4d_1x1_")(inception_4c_output)

    inception_4d_relu_1x1 = layers.ReLU()(inception_4d_1x1)

    inception_4d_3x3_reduce = layers.Conv2D(144, (1, 1), name="inception_4d_3x3_reduce_")(inception_4c_output)

    inception_4d_relu_3x3_reduce = layers.ReLU()(inception_4d_3x3_reduce)

    inception_4d_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4d_relu_3x3_reduce)

    inception_4d_3x3 = layers.Conv2D(288, (3, 3), name="inception_4d_3x3_")(inception_4d_3x3_prepadded)

    inception_4d_relu_3x3 = layers.ReLU()(inception_4d_3x3)

    inception_4d_5x5_reduce = layers.Conv2D(32, (1, 1), name="inception_4d_5x5_reduce_")(inception_4c_output)

    inception_4d_relu_5x5_reduce = layers.ReLU()(inception_4d_5x5_reduce)

    inception_4d_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_4d_relu_5x5_reduce)

    inception_4d_5x5 = layers.Conv2D(64, (5, 5), name="inception_4d_5x5_")(inception_4d_5x5_prepadded)

    inception_4d_relu_5x5 = layers.ReLU()(inception_4d_5x5)

    inception_4d_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4c_output)

    inception_4d_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_4d_pool_prepadded)
```

```
    inception_4d_pool_proj = layers.Conv2D(64, (1, 1), name="inception_4d_pool_proj_")(inception_4d_pool)

    inception_4d_relu_pool_proj = layers.ReLU()(inception_4d_pool_proj)

    inception_4d_output = layers.Concatenate(axis=-1)(

      [inception_4d_relu_1x1, inception_4d_relu_3x3, inception_4d_relu_5x5, inception_4d_relu_pool_proj])

    inception_4e_1x1 = layers.Conv2D(256, (1, 1), name="inception_4e_1x1_")(inception_4d_output)

    inception_4e_relu_1x1 = layers.ReLU()(inception_4e_1x1)

    inception_4e_3x3_reduce = layers.Conv2D(160, (1, 1), name="inception_4e_3x3_reduce_")(inception_4d_output)

    inception_4e_relu_3x3_reduce = layers.ReLU()(inception_4e_3x3_reduce)

    inception_4e_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4e_relu_3x3_reduce)

    inception_4e_3x3 = layers.Conv2D(320, (3, 3), name="inception_4e_3x3_")(inception_4e_3x3_prepadded)

    inception_4e_relu_3x3 = layers.ReLU()(inception_4e_3x3)

    inception_4e_5x5_reduce = layers.Conv2D(32, (1, 1), name="inception_4e_5x5_reduce_")(inception_4d_output)

    inception_4e_relu_5x5_reduce = layers.ReLU()(inception_4e_5x5_reduce)

    inception_4e_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_4e_relu_5x5_reduce)

    inception_4e_5x5 = layers.Conv2D(128, (5, 5), name="inception_4e_5x5_")(inception_4e_5x5_prepadded)

    inception_4e_relu_5x5 = layers.ReLU()(inception_4e_5x5)

    inception_4e_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_4d_output)

    inception_4e_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_4e_pool_prepadded)

    inception_4e_pool_proj = layers.Conv2D(128, (1, 1), name="inception_4e_pool_proj_")(inception_4e_pool)

    inception_4e_relu_pool_proj = layers.ReLU()(inception_4e_pool_proj)
```

```
inception_4e_output = layers.Concatenate(axis=-1)(
    [inception_4e_relu_1x1, inception_4e_relu_3x3, inception_4e_relu_5x5, inception_4e_relu_pool_proj])

pool4_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_4e_output)

pool4_3x3_s2 = layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2))(pool4_3x3_s2_prepadded)

inception_5a_1x1 = layers.Conv2D(256, (1, 1), name="inception_5a_1x1_")(pool4_3x3_s2)

inception_5a_relu_1x1 = layers.ReLU()(inception_5a_1x1)

inception_5a_3x3_reduce = layers.Conv2D(160, (1, 1), name="inception_5a_3x3_reduce_")(pool4_3x3_s2)

inception_5a_relu_3x3_reduce = layers.ReLU()(inception_5a_3x3_reduce)

inception_5a_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_5a_relu_3x3_reduce)

inception_5a_3x3 = layers.Conv2D(320, (3, 3), name="inception_5a_3x3_")(inception_5a_3x3_prepadded)

inception_5a_relu_3x3 = layers.ReLU()(inception_5a_3x3)

inception_5a_5x5_reduce = layers.Conv2D(32, (1, 1), name="inception_5a_5x5_reduce_")(pool4_3x3_s2)

inception_5a_relu_5x5_reduce = layers.ReLU()(inception_5a_5x5_reduce)

inception_5a_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_5a_relu_5x5_reduce)

inception_5a_5x5 = layers.Conv2D(128, (5, 5), name="inception_5a_5x5_")(inception_5a_5x5_prepadded)

inception_5a_relu_5x5 = layers.ReLU()(inception_5a_5x5)

inception_5a_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(pool4_3x3_s2)

inception_5a_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_5a_pool_prepadded)

inception_5a_pool_proj = layers.Conv2D(128, (1, 1), name="inception_5a_pool_proj_")(inception_5a_pool)

inception_5a_relu_pool_proj = layers.ReLU()(inception_5a_pool_proj)
```

```
inception_5a_output = layers.Concatenate(axis=-1)(
    [inception_5a_relu_1x1, inception_5a_relu_3x3, inception_5a_relu_5x5, inception_5a_relu_pool_proj])

inception_5b_1x1 = layers.Conv2D(384, (1, 1), name="inception_5b_1x1_")(inception_5a_output)

inception_5b_relu_1x1 = layers.ReLU()(inception_5b_1x1)

inception_5b_3x3_reduce = layers.Conv2D(192, (1, 1), name="inception_5b_3x3_reduce_")(inception_5a_output)

inception_5b_relu_3x3_reduce = layers.ReLU()(inception_5b_3x3_reduce)

inception_5b_3x3_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_5b_relu_3x3_reduce)

inception_5b_3x3 = layers.Conv2D(384, (3, 3), name="inception_5b_3x3_")(inception_5b_3x3_prepadded)

inception_5b_relu_3x3 = layers.ReLU()(inception_5b_3x3)

inception_5b_5x5_reduce = layers.Conv2D(48, (1, 1), name="inception_5b_5x5_reduce_")(inception_5a_output)

inception_5b_relu_5x5_reduce = layers.ReLU()(inception_5b_5x5_reduce)

inception_5b_5x5_prepadded = layers.ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_5b_relu_5x5_reduce)

inception_5b_5x5 = layers.Conv2D(128, (5, 5), name="inception_5b_5x5_")(inception_5b_5x5_prepadded)

inception_5b_relu_5x5 = layers.ReLU()(inception_5b_5x5)

inception_5b_pool_prepadded = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_5a_output)

inception_5b_pool = layers.MaxPool2D(pool_size=(3, 3), strides=(1, 1))(inception_5b_pool_prepadded)

inception_5b_pool_proj = layers.Conv2D(128, (1, 1), name="inception_5b_pool_proj_")(inception_5b_pool)

inception_5b_relu_pool_proj = layers.ReLU()(inception_5b_pool_proj)

inception_5b_output = layers.Concatenate(axis=-1)(
    [inception_5b_relu_1x1, inception_5b_relu_3x3, inception_5b_relu_5x5, inception_5b_relu_pool_proj])
```

```
        pool5_7x7_s1 = layers.GlobalAveragePooling2D(keepdims=True)(inception_5b_output)
        pool5_drop_7x7_s1 = layers.Dropout(0.400000)(pool5_7x7_s1)
        loss3_classifier = layers.Reshape((-1,), name="loss3_classifier_preFlatten1")(pool5_drop_7x7_s1)
        loss3_classifier = layers.Dense(9, name="loss3_classifier_")(loss3_classifier)
        prob = layers.Softmax()(loss3_classifier)
        model = keras.Model(inputs=[data_unnormalized], outputs=[prob])
        return model
    model = create_model();
    model.load_weights("/workspaces/HackEye-Backend/MainProject/weights1.h5")
    print('model loaded')
    return model
model = modelling();
@csrf_exempt
def upload_image(request):
    if request.method == 'POST':
        image_file = request.FILES.get('image')
        print(image_file)
        if image_file:
            # Process the image or save it
            save_path = '/workspaces/HackEye-Backend/MainProject/files'
            os.makedirs(save_path, exist_ok=True)
            with open(os.path.join(save_path, image_file.name), 'wb') as f:
                for chunk in image_file.chunks():
                    f.write(chunk)
            class_names = ['Bat', 'Car', 'Grenade', 'Knife', 'Machine Guns', 'Masked Face', 'Motorcycle', 'Pistol', 'face']
```

```
            print(class_names)
            path = os.path.join(save_path, image_file.name)
            img = tf.keras.utils.load_img(
                path, target_size=(224, 224)
            )
            # img_array = tf.keras.utils.img_to_array(img)
            # img_array = tf.expand_dims(img_array, 0) # Create a batch
            img_array = np.expand_dims(img, axis=0)
            predictions = model.predict(img_array)
            score = predictions[0]
            val = np.argmax(score)
            print(
                "This image most likely belongs to {} with a {:.2f} percent confidence."
                    .format(class_names[np.argmax(score)], 100 * np.max(score))
            )


            return JsonResponse(data={'status': class_names[np.argmax(score)], 'message': "{:.2f}".format(round(100 * np.max(score),2))})
        else:
            return JsonResponse({'status': 'error', 'message': 'No image received'})
    return JsonResponse({'status': 'error', 'message': 'Invalid request'})
def test(request):
    return HttpResponse("Any kind of HTML Here")
```

5.1.4 OpenCV

```
import cv2

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

def create_model():

    data_unnormalized = keras.Input(shape=(224,224,3), name="data_unnormalized")

    data = keras.layers.Normalization(axis=(1,2,3), name="data_")(data_unnormalized)

    conv1_7x7_s2_prepadded = layers.ZeroPadding2D(padding=((3,3),(3,3)))(data)

    conv1_7x7_s2 = layers.Conv2D(64, (7,7), strides=(2,2), name="conv1_7x7_s2_")(conv1_7x7_s2_prepadded)

    conv1_relu_7x7 = layers.ReLU()(conv1_7x7_s2)

    pool1_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0,1),(0,1)))(conv1_relu_7x7)

    pool1_3x3_s2 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(pool1_3x3_s2_prepadded)

    CCNormLayer = layers.Lambda(lambda X: tf.nn.local_response_normalization(X, depth_radius=2.000000, bias=1.000000, alpha=0.000020, beta=0.750000))

    pool1_norm1 = CCNormLayer(pool1_3x3_s2)

    conv2_3x3_reduce = layers.Conv2D(64, (1,1), name="conv2_3x3_reduce_")(pool1_norm1)

    conv2_relu_3x3_reduce = layers.ReLU()(conv2_3x3_reduce)

    conv2_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(conv2_relu_3x3_reduce)

    conv2_3x3 = layers.Conv2D(192, (3,3), name="conv2_3x3_")(conv2_3x3_prepadded)

    conv2_relu_3x3 = layers.ReLU()(conv2_3x3)

    CCNormLayer = layers.Lambda(lambda X: tf.nn.local_response_normalization(X, depth_radius=2.000000, bias=1.000000, alpha=0.000020, beta=0.750000))

    conv2_norm2 = CCNormLayer(conv2_relu_3x3)
```

```
    pool2_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0,1),(0,1)))(conv2_norm2)

    pool2_3x3_s2 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(pool2_3x3_s2_prepadded)

    inception_3a_1x1 = layers.Conv2D(64, (1,1), name="inception_3a_1x1_")(pool2_3x3_s2)

    inception_3a_relu_1x1 = layers.ReLU()(inception_3a_1x1)

    inception_3a_3x3_reduce = layers.Conv2D(96, (1,1), name="inception_3a_3x3_reduce_")(pool2_3x3_s2)

    inception_3a_relu_3x3_reduce = layers.ReLU()(inception_3a_3x3_reduce)

    inception_3a_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_3a_relu_3x3_reduce)

    inception_3a_3x3 = layers.Conv2D(128, (3,3), name="inception_3a_3x3_")(inception_3a_3x3_prepadded)

    inception_3a_relu_3x3 = layers.ReLU()(inception_3a_3x3)

    inception_3a_5x5_reduce = layers.Conv2D(16, (1,1), name="inception_3a_5x5_reduce_")(pool2_3x3_s2)

    inception_3a_relu_5x5_reduce = layers.ReLU()(inception_3a_5x5_reduce)

    inception_3a_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_3a_relu_5x5_reduce)

    inception_3a_5x5 = layers.Conv2D(32, (5,5), name="inception_3a_5x5_")(inception_3a_5x5_prepadded)

    inception_3a_relu_5x5 = layers.ReLU()(inception_3a_5x5)

    inception_3a_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(pool2_3x3_s2)

    inception_3a_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_3a_pool_prepadded)

    inception_3a_pool_proj = layers.Conv2D(32, (1,1), name="inception_3a_pool_proj_")(inception_3a_pool)

    inception_3a_relu_pool_proj = layers.ReLU()(inception_3a_pool_proj)

    inception_3a_output = layers.Concatenate(axis=-1)([inception_3a_relu_1x1, inception_3a_relu_3x3, inception_3a_relu_5x5, inception_3a_relu_pool_proj])
```

```
    inception_3b_1x1 = layers.Conv2D(128, (1,1), name="inception_3b_1x1_")(inception_3a_output)

    inception_3b_relu_1x1 = layers.ReLU()(inception_3b_1x1)

    inception_3b_3x3_reduce = layers.Conv2D(128, (1,1), name="inception_3b_3x3_reduce_")(inception_3a_output)

    inception_3b_relu_3x3_reduce = layers.ReLU()(inception_3b_3x3_reduce)

    inception_3b_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_3b_relu_3x3_reduce)

    inception_3b_3x3 = layers.Conv2D(192, (3,3), name="inception_3b_3x3_")(inception_3b_3x3_prepadded)

    inception_3b_relu_3x3 = layers.ReLU()(inception_3b_3x3)

    inception_3b_5x5_reduce = layers.Conv2D(32, (1,1), name="inception_3b_5x5_reduce_")(inception_3a_output)

    inception_3b_relu_5x5_reduce = layers.ReLU()(inception_3b_5x5_reduce)

    inception_3b_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_3b_relu_5x5_reduce)

    inception_3b_5x5 = layers.Conv2D(96, (5,5), name="inception_3b_5x5_")(inception_3b_5x5_prepadded)

    inception_3b_relu_5x5 = layers.ReLU()(inception_3b_5x5)

    inception_3b_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_3a_output)

    inception_3b_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_3b_pool_prepadded)

    inception_3b_pool_proj = layers.Conv2D(64, (1,1), name="inception_3b_pool_proj_")(inception_3b_pool)

    inception_3b_relu_pool_proj = layers.ReLU()(inception_3b_pool_proj)

    inception_3b_output = layers.Concatenate(axis=-1)([inception_3b_relu_1x1, inception_3b_relu_3x3, inception_3b_relu_5x5, inception_3b_relu_pool_proj])

    pool3_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0,1),(0,1)))(inception_3b_output)

    pool3_3x3_s2 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(pool3_3x3_s2_prepadded)
```

```
    inception_4a_1x1 = layers.Conv2D(192, (1,1), name="inception_4a_1x1_")(pool3_3x3_s2)

    inception_4a_relu_1x1 = layers.ReLU()(inception_4a_1x1)

    inception_4a_3x3_reduce = layers.Conv2D(96, (1,1), name="inception_4a_3x3_reduce_")(pool3_3x3_s2)

    inception_4a_relu_3x3_reduce = layers.ReLU()(inception_4a_3x3_reduce)

    inception_4a_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4a_relu_3x3_reduce)

    inception_4a_3x3 = layers.Conv2D(208, (3,3), name="inception_4a_3x3_")(inception_4a_3x3_prepadded)

    inception_4a_relu_3x3 = layers.ReLU()(inception_4a_3x3)

    inception_4a_5x5_reduce = layers.Conv2D(16, (1,1), name="inception_4a_5x5_reduce_")(pool3_3x3_s2)

    inception_4a_relu_5x5_reduce = layers.ReLU()(inception_4a_5x5_reduce)

    inception_4a_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_4a_relu_5x5_reduce)

    inception_4a_5x5 = layers.Conv2D(48, (5,5), name="inception_4a_5x5_")(inception_4a_5x5_prepadded)

    inception_4a_relu_5x5 = layers.ReLU()(inception_4a_5x5)

    inception_4a_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(pool3_3x3_s2)

    inception_4a_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_4a_pool_prepadded)

    inception_4a_pool_proj = layers.Conv2D(64, (1,1), name="inception_4a_pool_proj_")(inception_4a_pool)

    inception_4a_relu_pool_proj = layers.ReLU()(inception_4a_pool_proj)

    inception_4a_output = layers.Concatenate(axis=-1)([inception_4a_relu_1x1, inception_4a_relu_3x3, inception_4a_relu_5x5, inception_4a_relu_pool_proj])

    inception_4b_1x1 = layers.Conv2D(160, (1,1), name="inception_4b_1x1_")(inception_4a_output)

    inception_4b_relu_1x1 = layers.ReLU()(inception_4b_1x1)

    inception_4b_3x3_reduce = layers.Conv2D(112, (1,1), name="inception_4b_3x3_reduce_")(inception_4a_output)
```

```
    inception_4b_relu_3x3_reduce = layers.ReLU()(inception_4b_3x3_reduce)

    inception_4b_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4b_relu_3x3_reduce)

    inception_4b_3x3 = layers.Conv2D(224, (3,3), name="inception_4b_3x3_")(inception_4b_3x3_prepadded)

    inception_4b_relu_3x3 = layers.ReLU()(inception_4b_3x3)

    inception_4b_5x5_reduce = layers.Conv2D(24, (1,1), name="inception_4b_5x5_reduce_")(inception_4a_output)

    inception_4b_relu_5x5_reduce = layers.ReLU()(inception_4b_5x5_reduce)

    inception_4b_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_4b_relu_5x5_reduce)

    inception_4b_5x5 = layers.Conv2D(64, (5,5), name="inception_4b_5x5_")(inception_4b_5x5_prepadded)

    inception_4b_relu_5x5 = layers.ReLU()(inception_4b_5x5)

    inception_4b_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4a_output)

    inception_4b_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_4b_pool_prepadded)

    inception_4b_pool_proj = layers.Conv2D(64, (1,1), name="inception_4b_pool_proj_")(inception_4b_pool)

    inception_4b_relu_pool_proj = layers.ReLU()(inception_4b_pool_proj)

    inception_4b_output = layers.Concatenate(axis=-1)([inception_4b_relu_1x1, inception_4b_relu_3x3, inception_4b_relu_5x5, inception_4b_relu_pool_proj])

    inception_4c_1x1 = layers.Conv2D(128, (1,1), name="inception_4c_1x1_")(inception_4b_output)

    inception_4c_relu_1x1 = layers.ReLU()(inception_4c_1x1)

    inception_4c_3x3_reduce = layers.Conv2D(128, (1,1), name="inception_4c_3x3_reduce_")(inception_4b_output)

    inception_4c_relu_3x3_reduce = layers.ReLU()(inception_4c_3x3_reduce)

    inception_4c_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4c_relu_3x3_reduce)

    inception_4c_3x3 = layers.Conv2D(256, (3,3), name="inception_4c_3x3_")(inception_4c_3x3_prepadded)
```

```
    inception_4c_relu_3x3 = layers.ReLU()(inception_4c_3x3)

    inception_4c_5x5_reduce = layers.Conv2D(24, (1,1), name="inception_4c_5x5_reduce_")(inception_4b_output)

    inception_4c_relu_5x5_reduce = layers.ReLU()(inception_4c_5x5_reduce)

    inception_4c_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_4c_relu_5x5_reduce)

    inception_4c_5x5 = layers.Conv2D(64, (5,5), name="inception_4c_5x5_")(inception_4c_5x5_prepadded)

    inception_4c_relu_5x5 = layers.ReLU()(inception_4c_5x5)

    inception_4c_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4b_output)

    inception_4c_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_4c_pool_prepadded)

    inception_4c_pool_proj = layers.Conv2D(64, (1,1), name="inception_4c_pool_proj_")(inception_4c_pool)

    inception_4c_relu_pool_proj = layers.ReLU()(inception_4c_pool_proj)

    inception_4c_output = layers.Concatenate(axis=-1)([inception_4c_relu_1x1, inception_4c_relu_3x3, inception_4c_relu_5x5, inception_4c_relu_pool_proj])

    inception_4d_1x1 = layers.Conv2D(112, (1,1), name="inception_4d_1x1_")(inception_4c_output)

    inception_4d_relu_1x1 = layers.ReLU()(inception_4d_1x1)

    inception_4d_3x3_reduce = layers.Conv2D(144, (1,1), name="inception_4d_3x3_reduce_")(inception_4c_output)

    inception_4d_relu_3x3_reduce = layers.ReLU()(inception_4d_3x3_reduce)

    inception_4d_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4d_relu_3x3_reduce)

    inception_4d_3x3 = layers.Conv2D(288, (3,3), name="inception_4d_3x3_")(inception_4d_3x3_prepadded)

    inception_4d_relu_3x3 = layers.ReLU()(inception_4d_3x3)

    inception_4d_5x5_reduce = layers.Conv2D(32, (1,1), name="inception_4d_5x5_reduce_")(inception_4c_output)

    inception_4d_relu_5x5_reduce = layers.ReLU()(inception_4d_5x5_reduce)
```

```
    inception_4d_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_4d_relu_5x5_reduce)

    inception_4d_5x5 = layers.Conv2D(64, (5,5), name="inception_4d_5x5_")(inception_4d_5x5_prepadded)

    inception_4d_relu_5x5 = layers.ReLU()(inception_4d_5x5)

    inception_4d_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4c_output)

    inception_4d_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_4d_pool_prepadded)

    inception_4d_pool_proj = layers.Conv2D(64, (1,1), name="inception_4d_pool_proj_")(inception_4d_pool)

    inception_4d_relu_pool_proj = layers.ReLU()(inception_4d_pool_proj)

    inception_4d_output = layers.Concatenate(axis=-1)([inception_4d_relu_1x1, inception_4d_relu_3x3, inception_4d_relu_5x5, inception_4d_relu_pool_proj])

    inception_4e_1x1 = layers.Conv2D(256, (1,1), name="inception_4e_1x1_")(inception_4d_output)

    inception_4e_relu_1x1 = layers.ReLU()(inception_4e_1x1)

    inception_4e_3x3_reduce = layers.Conv2D(160, (1,1), name="inception_4e_3x3_reduce_")(inception_4d_output)

    inception_4e_relu_3x3_reduce = layers.ReLU()(inception_4e_3x3_reduce)

    inception_4e_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4e_relu_3x3_reduce)

    inception_4e_3x3 = layers.Conv2D(320, (3,3), name="inception_4e_3x3_")(inception_4e_3x3_prepadded)

    inception_4e_relu_3x3 = layers.ReLU()(inception_4e_3x3)

    inception_4e_5x5_reduce = layers.Conv2D(32, (1,1), name="inception_4e_5x5_reduce_")(inception_4d_output)

    inception_4e_relu_5x5_reduce = layers.ReLU()(inception_4e_5x5_reduce)

    inception_4e_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_4e_relu_5x5_reduce)

    inception_4e_5x5 = layers.Conv2D(128, (5,5), name="inception_4e_5x5_")(inception_4e_5x5_prepadded)

    inception_4e_relu_5x5 = layers.ReLU()(inception_4e_5x5)
```

```
    inception_4e_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_4d_output)

    inception_4e_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_4e_pool_prepadded)

    inception_4e_pool_proj = layers.Conv2D(128, (1,1), name="inception_4e_pool_proj_")(inception_4e_pool)

    inception_4e_relu_pool_proj = layers.ReLU()(inception_4e_pool_proj)

    inception_4e_output = layers.Concatenate(axis=-1)([inception_4e_relu_1x1, inception_4e_relu_3x3, inception_4e_relu_5x5, inception_4e_relu_pool_proj])

    pool4_3x3_s2_prepadded = layers.ZeroPadding2D(padding=((0,1),(0,1)))(inception_4e_output)

    pool4_3x3_s2 = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(pool4_3x3_s2_prepadded)

    inception_5a_1x1 = layers.Conv2D(256, (1,1), name="inception_5a_1x1_")(pool4_3x3_s2)

    inception_5a_relu_1x1 = layers.ReLU()(inception_5a_1x1)

    inception_5a_3x3_reduce = layers.Conv2D(160, (1,1), name="inception_5a_3x3_reduce_")(pool4_3x3_s2)

    inception_5a_relu_3x3_reduce = layers.ReLU()(inception_5a_3x3_reduce)

    inception_5a_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_5a_relu_3x3_reduce)

    inception_5a_3x3 = layers.Conv2D(320, (3,3), name="inception_5a_3x3_")(inception_5a_3x3_prepadded)

    inception_5a_relu_3x3 = layers.ReLU()(inception_5a_3x3)

    inception_5a_5x5_reduce = layers.Conv2D(32, (1,1), name="inception_5a_5x5_reduce_")(pool4_3x3_s2)

    inception_5a_relu_5x5_reduce = layers.ReLU()(inception_5a_5x5_reduce)

    inception_5a_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_5a_relu_5x5_reduce)

    inception_5a_5x5 = layers.Conv2D(128, (5,5), name="inception_5a_5x5_")(inception_5a_5x5_prepadded)

    inception_5a_relu_5x5 = layers.ReLU()(inception_5a_5x5)
```

```
    inception_5a_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(pool4_3x3_s2)

    inception_5a_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_5a_pool_prepadded)

    inception_5a_pool_proj = layers.Conv2D(128, (1,1), name="inception_5a_pool_proj_")(inception_5a_pool)

    inception_5a_relu_pool_proj = layers.ReLU()(inception_5a_pool_proj)

    inception_5a_output = layers.Concatenate(axis=-1)([inception_5a_relu_1x1, inception_5a_relu_3x3, inception_5a_relu_5x5, inception_5a_relu_pool_proj])

    inception_5b_1x1 = layers.Conv2D(384, (1,1), name="inception_5b_1x1_")(inception_5a_output)

    inception_5b_relu_1x1 = layers.ReLU()(inception_5b_1x1)

    inception_5b_3x3_reduce = layers.Conv2D(192, (1,1), name="inception_5b_3x3_reduce_")(inception_5a_output)

    inception_5b_relu_3x3_reduce = layers.ReLU()(inception_5b_3x3_reduce)

    inception_5b_3x3_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_5b_relu_3x3_reduce)

    inception_5b_3x3 = layers.Conv2D(384, (3,3), name="inception_5b_3x3_")(inception_5b_3x3_prepadded)

    inception_5b_relu_3x3 = layers.ReLU()(inception_5b_3x3)

    inception_5b_5x5_reduce = layers.Conv2D(48, (1,1), name="inception_5b_5x5_reduce_")(inception_5a_output)

    inception_5b_relu_5x5_reduce = layers.ReLU()(inception_5b_5x5_reduce)

    inception_5b_5x5_prepadded = layers.ZeroPadding2D(padding=((2,2),(2,2)))(inception_5b_relu_5x5_reduce)

    inception_5b_5x5 = layers.Conv2D(128, (5,5), name="inception_5b_5x5_")(inception_5b_5x5_prepadded)

    inception_5b_relu_5x5 = layers.ReLU()(inception_5b_5x5)

    inception_5b_pool_prepadded = layers.ZeroPadding2D(padding=((1,1),(1,1)))(inception_5a_output)

    inception_5b_pool = layers.MaxPool2D(pool_size=(3,3), strides=(1,1))(inception_5b_pool_prepadded)
```

```
    inception_5b_pool_proj = layers.Conv2D(128, (1,1), name="inception_5b_pool_proj_")(inception_5b_pool)

    inception_5b_relu_pool_proj = layers.ReLU()(inception_5b_pool_proj)

    inception_5b_output = layers.Concatenate(axis=-1)([inception_5b_relu_1x1, inception_5b_relu_3x3, inception_5b_relu_5x5, inception_5b_relu_pool_proj])

    pool5_7x7_s1 = layers.GlobalAveragePooling2D(keepdims=True)(inception_5b_output)

    pool5_drop_7x7_s1 = layers.Dropout(0.400000)(pool5_7x7_s1)

    loss3_classifier = layers.Reshape((-1,), name="loss3_classifier_preFlatten1")(pool5_drop_7x7_s1)

    loss3_classifier = layers.Dense(9, name="loss3_classifier_")(loss3_classifier)

    prob = layers.Softmax()(loss3_classifier)

    model = keras.Model(inputs=[data_unnormalized], outputs=[prob])

    return model

model=create_model()

model.load_weights("weights1.h5")

print('model loaded')

import tensorflow as tf

from tensorflow import image

import numpy as np

class_names = ['Bat', 'Car', 'Grenade', 'Knife', 'Machine Guns', 'Masked Face', 'Motorcycle', 'Pistol', 'face']

print(class_names)

path="armas (2596).jpg"

img = tf.keras.utils.load_img(

  path, target_size=(224,224)

)

def fun(img):

    img_array = np.expand_dims(img, axis=0)
```

```
    predictions = model.predict(img_array)
    score = predictions[0]
    # val = np.argmax(score)
    print(class_names[np.argmax(score)])
    print(np.max(score))
    if np.max(score) <0.9:
        return False
    else:
        return True


net = cv2.dnn.readNet(r"C:\Users\hemna\Downloads\Weapon-Detection-with-yolov3-master\weapon_detection\yolov3_training_2000.weights", r"C:\Users\hemna\Downloads\Weapon-Detection-with-yolov3-master\weapon_detection\yolov3_testing.cfg")
classes = ["Weapon"]
output_layer_names = net.getUnconnectedOutLayersNames()
colors = np.random.uniform(0, 255, size=(len(classes), 3))
# Enter file name for example "ak47.jpg" or press "Enter" to start webcam
def value():
    val = input("Enter file name or press enter to start webcam : \n")
    if val == "":
        val = 0
    return val
# for video capture
cap = cv2.VideoCapture(value())
while True:
    _, img = cap.read()
    if not _:
```

```
    print("Error: Failed to read a frame from the video source.")
    break
height, width, channels = img.shape
# Detecting objects
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layer_names)
# Showing information on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
```

```
            class_ids.append(class_id)
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    print(indexes)
    if indexes == 0: print("weapon detected in frame")
    font = cv2.FONT_HERSHEY_PLAIN
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[class_ids[i]]
            # Crop the image
            cropped_img = img[y:y+h, x:x+w]
            cropped_img = cv2.resize(cropped_img, (224, 224), interpolation =
cv2.INTER_LINEAR)
            if fun(cropped_img):
                cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
                cv2.putText(img, label, (x, y + 30), font, 3, color, 3)


            # Optionally save the cropped image
            cv2.imwrite(r'C:\Users\hemna\Desktop\ScreenRecorder\cropped_image1.jpg',
img)
    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

5.2 OUTPUT SNAPSHOTS

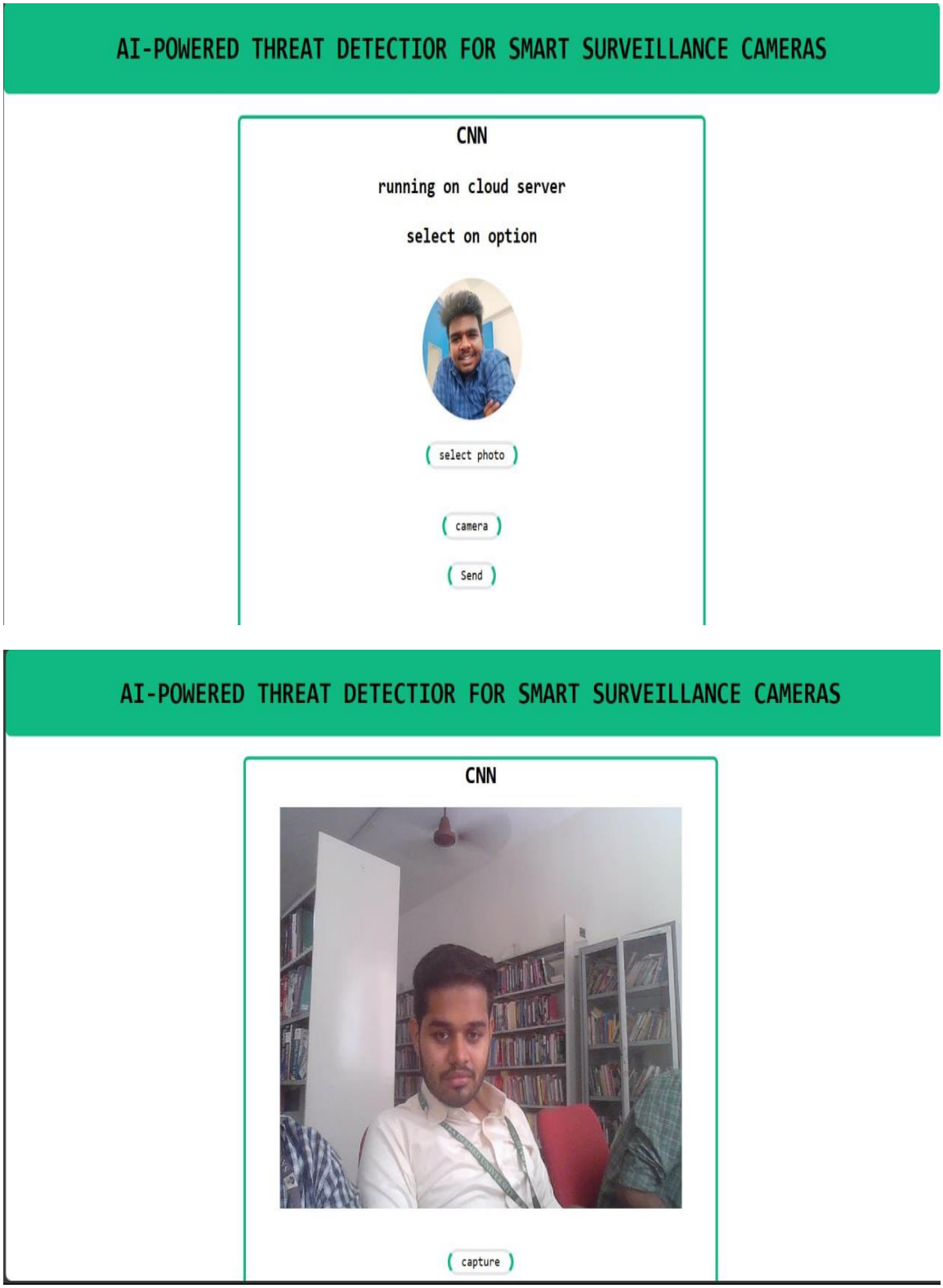


Fig 5.1 Web Output-1

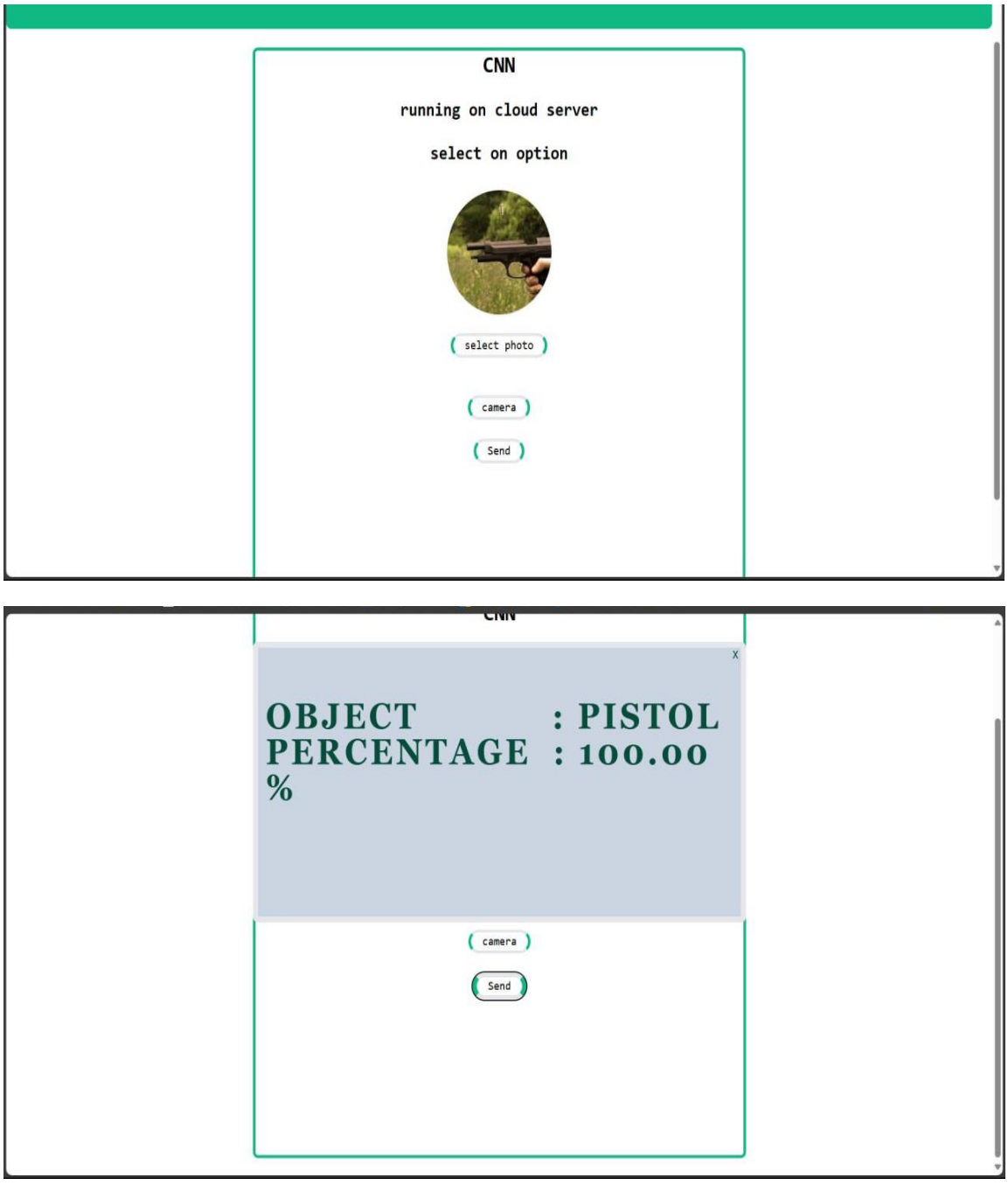


Fig 5.1 web output-2

Fig 5.3 OpenCV output

CHAPTER 6

CONCLUSION AND FUTURE WORK

This project outlines the creation and deployment of an AI-powered threat detection system tailored for real-time video surveillance. The prototype showcased its ability to be directly installed on the camera at the edge, achieving exceptional prediction accuracy and rapid response times. By doing so, the system holds promise in significantly bolstering security personnel's capabilities to identify various weapons in real time, potentially thwarting criminal activities. Our experiments rigorously assessed the system's performance and classification accuracy, with a specific emphasis on evaluating prediction time at the camera side.

Looking ahead, we plan to augment our data models by integrating real-time CCTV crime footage, thereby enhancing our application's real-time operational capabilities. This iterative process will substantially refine the system's accuracy and sensitivity to potential threats. Additionally, we aim to develop our models for object segmentation, empowering them to store and meticulously analyze pixel-level data for even more precise threat detection.

Furthermore, upon the camera's detection of a threat, our system will swiftly dispatch a burst of alert notifications to designated personnel or activate the alarm section within the deployment environment. This burst of alerts ensures immediate attention to potential security breaches. Moreover, the system will precisely pinpoint the section of the CCTV camera feed where the threat is detected, enabling swift response and intervention in the identified area.

CHAPTER 7

REFERENCES

1. V. Jadhav, R. Deshmukh, P. Gupta, S. Ghogale and M. Bodireddy," Weapon Detection from Surveillance Footage in Real-Time Using Deep Learning, in IEEE Access, pp. 1-6, doi: 10.1109/ICCUBEA58933.2023.10392003.
2. E. D and N. P. G. Bhavani, "An Effective DNN Based ResNet Approach for Satellite Image Classification," 2023 4th International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2023, pp. 1055-1062, doi: 10.1109/ICOSEC58147.2023.10276330.
3. P. M, S. Sreekumar and A. S, "Detection of Covid-19 from the Chest X-Ray Images: A Comparison Study between CNN and Resnet-50," 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon), Mysuru, India, 2022, pp. 1-7, doi: 10.1109/MysuruCon55714.2022.9972488.
4. M. T. Bhatti, M. G. Khan, M. Aslam, and M. J. Fiaz, "Weapon Detection in Real-Time CCTV Videos Using Deep Learning," in IEEE Access, vol. 9, pp. 34366-34382, 2021, doi: 10.1109/ACCESS.2021.3059170. Keywords:
5. M. Grega, S. Łach and R. Sieradzki, "Automated recognition of firearms in surveillance video," 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), San Diego, CA, USA, 2013, pp. 45-50, doi: 10.1109/CogSIMA.2013.6523822.
6. U. V. Navalgund and P. K., "Crime Intention Detection System Using Deep Learning," 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET), Kottayam, India, 2018, pp. 1-6, doi: 10.1109/ICCSDET.2018.8821168.
7. P. T, R. Thangaraj, P. P, U. R. M, and B. Vadivelu, "Real-Time Handgun Detection in Surveillance Videos based on Deep Learning Approach," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2022, pp. 689-693, doi: 10.1109/ICAAIC53929.2022.9793288.
8. A. A. Ahmed and M. Echi, "Hawk-Eye: An AI-Powered Threat Detector for Intelligent Surveillance Cameras," in IEEE Access, vol. 9, pp. 63283-63293, 2021, doi: 10.1109/ACCESS.2021.3074319.

DrillBit

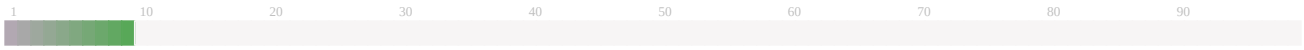
The Report is Generated by DrillBit Plagiarism Detection Software

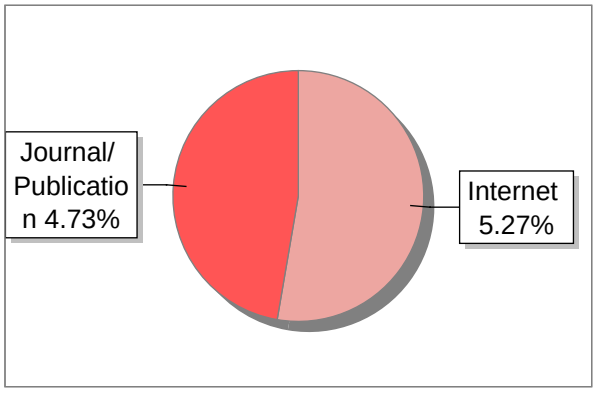
Submission Information

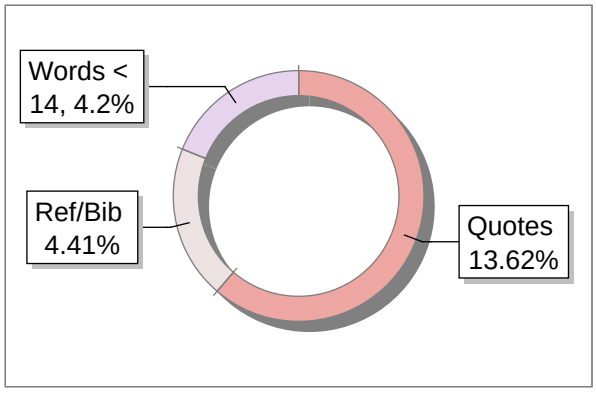
Author Name	Nafize
Title	Threatdetector
Paper/Submission ID	1681025
Submitted by	kannanb@cse.sastra.edu
Submission Date	2024-04-22 14:49:16
Total Pages	74
Document type	Project Work

Result Information

Similarity **10 %**







Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Sources: Less than 14 Words %	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File