Project Report

On

Real Time Safety Helmet Detection using Python

Submitted by

Amit Mandal

Subhadeep Saha

Papai Sarkar

Pronay Chandra Mridha

Under the guidance of

Dr. BANDANA BARMAN

Assistant professors

Dept. of Electronics and Communication Engineering

Kalyani Government Engineering College

(Affiliated to Maulana Abul Kalam Azad University of Technology)

Kalyani-741235, Nadia, West Bengal

2023-24

Approval Sheet

This project report entitled "Safety Helmet Detection by using Python" by Amit Mandal, Subhadeep Saha, Pronay Chandra Mridha and Papai Sarkar is approved for the degree of B.Tech. (ECE) submitted to Department of Electronics and Communication Engineering at KALYANI GOVERNMENT ENGINEERING COLLEGE, KALYANI under MAKAUT.

Examiners

DECLARATION

We certify that

1. The work contained in the project is original and has been done by us under the general supervision of our supervisor.
2. The work has not been submitted to any other institute for any degree or diploma.
3. We have followed the guidelines provided by the institute in writing of the project.
4. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute and University.

Amit Mandal

Roll No. - 10200321068
Reg No. - 211020100320006 ------------------------------

Subhadeep Saha

Roll No. – 10200320036

Reg No. – 201020100310007 ------------------------------

Pronay Chandra Mridha

Roll No. – 10200321055

Reg No. – 211020100320007 ------------------------------

Papai Sarkar

Roll No. – 10200321070

Reg No. – 2011020100320011 ------------------------------

ACKNOWLEDGEMENT

It is not possible to complete our project work and also to prepare a report on this without the assistance & encouragement of people working with us. This one is certainly no exception. On the very outset of this report, We would like to extend our sincere & heartfelt gratitude towards all the personages who have helped in this endeavour. Without their active guidance, help, cooperation & encouragement, We could not have made hardware in the project. First and foremost, we would like to express our sincere gratitude to our supervisor**, Dr. Bandana Barman**.

We are indebted to **Dr. Sourabh kumar Das**, honourable Principle of **Kalyani Government Engineering College** and our honourable Head of Department **Dr. Angsuman Sarkar** for providing all logistic support and guidance.

We would like to thank all my friends and especially our classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. We have enjoyed their companionship so much during our stay at **KGEC.**

Thanking You

Phone: - 25826680 (PBX)

Kalyani, 741 235 ,

Fax :- 25821309

Nadia, West Bengal, India

Kalyani Government Engineering College

(Govt. of West Bengal)

CERTIFICATE

This is to certify that Amit Mandal, Subhadeep Saha, Pronay Chandra Mridha and Papai Sarkar have successfully completed project work entitled "**Real Time Safety Helmet Detection Using Python**" is being presented. In the partial fulfilment of the requirements for the award of the Bachelor of Technology in **Electronics and Communication Engineering** and submitted to the Department of **Electronics and Communication Engineering** of **Kalyani Government Engineering College** is an authentic record of our own work carried out during our **B.Tech** course Under the supervision of **Dr. Bandana Barman**, Assistant Professor of **Electronics and Communication Engineering** Department.

DR. ANGSUMAN SARKAR
PROFESSOR AND HEAD
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DR. BANDANA BARMAN
PROJECT GUIDE
ASSISTANT PROFESSOR

TABLE OF CONTENT

ABSTRACT

This report explores the use of Python 3.11 and OpenCV for high-density crowd counting, a task with applications in public safety, event management, and urban planning. Traditional counting methods often struggle in complex crowd scenes, making it challenging to obtain accurate crowd counts. Python, with its versatile libraries and frameworks, offers an effective solution to address this problem.

The report covers data collection, preprocessing, deep learning approaches, density estimation, model training and evaluation, and identification of high-density crowds. Python libraries like OpenCV, NumPy, TensorFlow, Keras, VS Code IDE etc are employed for implementation.

Data preprocessing removes noise, resizes images, and normalizes pixel values. Deep learning models, particularly Convolutional Neural Networks (CNNs), learn complex patterns and features from crowd images. Density estimation generates density maps that assign higher values to densely populated areas. Model training and evaluation employ appropriate datasets and evaluation metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE).

To identify high-density crowds, a density threshold is set, and the number of crowds exceeding this threshold is counted. Python and OpenCV facilitate this process effectively. The report concludes by highlighting the benefits of Python and OpenCV in high-density crowd counting and suggests future directions, such as advanced deep learning architectures and addressing challenges like occlusions and varying lighting conditions.

Keywords : High-density crowd counting, Python 3.11, Keras, deep learning, Convolutional Neural Networks (CNNs), density estimation, data preprocessing, model training, model evaluation, crowd management, urban planning, public safety, event management, data collection, density threshold, occlusions, lighting conditions

1. INTRODUCTION

1.1 Background and motivation:

Helmet inspection is an important part of computer vision work and has a major impact on public safety, especially vehicle maintenance and workplace safety. The main purpose of hard helmet research stems from the need to ensure safety and reduce the risk of head injury in many places. Protect your head from injuries during accidents. Despite the law, compliance is often low, leading to more serious injuries and deaths. According to the World Health Organization (WHO), wearing a helmet reduces the risk of head trauma during an accident by 69% and the risk of death by 42%. Automated helmet inspection systems can assist police officers in monitoring and ensuring compliance with helmet regulations, reducing injury-related injuries and deaths. Its use is an essential part of personal protective equipment (PPE). Workers are constantly exposed to hazards such as falling objects, which can lead to head injuries. Due to the large number of workers and the dynamic nature of the environment, it will be difficult to ensure that all employees wear helmets and comply with safety rules. Automated helmet checks can improve safety monitoring processes to ensure workers are constantly protected. Actually. Convolutional neural network (CNN) is a type of deep learning model that is very successful in many image recognition applications due to its ability to learn and extract image features. The use of this technology to increase to increase to increase to increase to increase to increase to increase to increase The system is designed to accurately distinguish between people wearing helmets and those not wearing helmets in real-time video. Dataset: Images of people wearing and without helmets in various conditions. Show the model of the planned data and evaluate its performance using appropriate indicators such as accuracy, precision, recall, and F1 score. Check the reality
Performance Optimization: Optimizing the speed and accuracy of models and applications to ensure they can run effectively in real time. Improve safety measures and comply with helmet wearing rules in real environment.

1.2 Data collection and processing:

Dataset composition: The dataset will include images of helmeted and unhelmeted individuals captured in different situations for power purposes. This includes different lighting, contrast and background. This step is necessary to prepare the data for training a good CNN model. CNN Model Architecture:

Design Model: This project will examine different CNN models to determine the best design for helmet detection. This includes experimenting with different numbers of layers, filter sizes and dynamics. Techniques such as data augmentation and publishing will be used to improve the model's generalization ability and avoid overloading. Model Evaluation:

Performance Evaluation: The training model will be evaluated using separate data sets. Basic metrics such as accuracy, precision, recall, and F1 score will be used to evaluate the performance of the model. Information is available. Live Search App:

Video Capture: The live app uses OpenCV to capture live video from the web or other videos. Face detection will be integrated using the Haar stage to detect faces of interest. The results are immediate and the indicator shows whether people are wearing a helmet. Optimization and Deployment:

Optimization for speed and accuracy: Techniques such as model pruning and quantization will be explored to instantly improve model performance without impacting reality. It is application friendly and can be used easily in many areas, including traffic monitoring and occupational safety platforms.

<u>Previous Researches On Real time Safety Helmet Detection (Literature Review):</u>

1) Safety Helmet Detection Based on YOLOv5 by Fangbo Zhou, Huailin Zhao, Zhen (2021) ,this research work proposes a safety helmet detection method based on YOLOv5 and annotates the 6045 collected data sets to establish a digital safety helmet monitoring system and shows the effectiveness of helmet detection based YOLov5.

2) Safety Helmet Wearing Detection Based on Jetson Nano and Improved YOLOv5 by Zaihui Deng,Chong Yao,and Qiyu Yin(2023), This study introduces an improved safety helmet-wearing detection model named YOLOv5-SN, aiming to address the shortcomings of the existing YOLOv5 models, including a large number of model parameters, slow reasoning speed, and redundant network structure.

1.3 Inspiration:

Undertaking a project on Real Time Safety helmet detection by using Python, offers a compelling opportunity to address the practical need for accurate crowd estimation in diverse domains. Crowd counting plays a crucial role in urban planning, public safety, transportation management, and event planning. Python's simplicity, extensive libraries, and wide community support make it an ideal choice for implementing crowd counting algorithms. OpenCV as a comprehensive computer vision library, provides the necessary tools and functions for efficient crowd analysis and counting. Additionally, the availability of publicly accessible crowd counting datasets allows for benchmarking and comparison with existing approaches. By developing accurate crowd counting models, this project can contribute to improved crowd management strategies, enhanced public safety measures, and optimized resource allocation. The combination of Python's versatility and OpenCV's robust functionality provides a solid foundation for undertaking a crowd counting project with practical implications.

1.4 Hardware and Software used:

Software requirement specification

The software used in the development of the project are as follows:

- Windows 11
- Python 3.11
- VS Code IDE

The Libraries used in the development of the project are as follows:

- OpenCv
- Tensorflow
- Keras
- Numpy
- Matplotlib

The hardware used in the development of the project are as follows:

- 12 GB RAM
- RYZE 3
- Webcam (External)

2. Literature Review

2.1 Previous Work: Review of Existing Helmet Detection Systems and Related Research

Helmet detection is a crucial aspect of ensuring safety in various environments, particularly in traffic management and industrial settings. Over the years, numerous research studies and technological developments have aimed to create reliable and efficient helmet detection systems. This section provides a review of some of the most significant contributions to this field.

2.2 Early Approaches

In the early 2000s, helmet detection systems were primarily rule-based and relied heavily on predefined heuristics and image processing techniques. These systems used edge detection, color segmentation, and shape analysis to identify helmets in images. For instance, one of the earliest approaches involved using Hough Transform for detecting circular shapes, which was combined with color segmentation to isolate helmet-like objects from the background. However, these methods were limited in their accuracy due to their inability to handle variations in lighting, helmet color, and background complexity.

2.3 Machine Learning-Based Methods

The advent of machine learning brought significant improvements to helmet detection systems. Researchers began utilizing traditional machine learning algorithms, such as Support Vector Machines (SVM), Decision Trees, and Random Forests, to classify images based on handcrafted features. These features included Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Speeded-Up Robust Features (SURF).

A notable study by Zhang et al. (2013) used HOG features and SVM for motorcycle helmet detection. The approach significantly improved detection accuracy over previous methods, demonstrating the potential of machine learning in this domain. However, these methods still struggled with high variability in real-world scenarios and required extensive feature engineering.

2.4 Deep Learning-Based Methods

The breakthrough in deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized helmet detection systems. CNNs automate feature extraction, allowing models to learn hierarchical representations directly from the data. This section highlights some pivotal research utilizing deep learning for helmet detection.

Li et al. (2017) proposed a deep learning framework using a CNN-based model for real-time motorcycle helmet detection. The system employed a pretrained CNN model, fine-tuned on a dataset of motorcycle riders. The approach achieved impressive accuracy and real-time performance, showcasing the advantages of deep learning in handling diverse and complex datasets.

Another significant contribution was by Wu et al. (2019), who developed a helmet detection system using the Faster R-CNN framework. This method integrated Region Proposal Networks (RPN) with CNNs to generate region proposals and classify them simultaneously.

Recently, Raj et al. (2021) introduced a helmet detection system using YOLO (You Only Look Once), a state-of-the-art object detection framework. The YOLO-based system achieved real-time detection with high accuracy, outperforming traditional CNN-based methods in speed and robustness.

2.5 Related Research

Helmet detection is often studied alongside related fields, such as human detection, face recognition, and safety gear detection. These studies provide valuable insights and techniques applicable to helmet detection systems.

For instance, human detection systems using CNNs, such as the work by Girshick et al. (2014) on the R-CNN (Region-based CNN), laid the groundwork for applying similar techniques to helmet detection. The principles of region-based detection and feature extraction were directly transferable to detecting helmets in various environments.

Face recognition research also contributed to helmet detection. Techniques for handling occlusions and varying lighting conditions in face recognition, such as those proposed by Schroff et al. (2015) with the FaceNet model, provided inspiration for addressing similar challenges in helmet detection.

Technological Advances: Discussion on Advancements in Image Processing and Machine Learning Relevant to the Project

Technological advancements in image processing and machine learning have significantly influenced the development of helmet detection systems. This section discusses some key advancements that have contributed to the progress in this field.

3. Data Collections

Data collection and preprocessing are essential steps in crowd counting research to ensure the availability of suitable datasets and to prepare the data for accurate analysis. The process involves capturing crowd images or videos, annotating the data with ground truth crowd counts, and performing preprocessing techniques such as noise removal, image enhancement, and density map generation. These steps contribute to the quality and reliability of the crowd counting models.

Define the Scope: Determine the specific context or scenario in which you want to perform crowd counting. It could be public spaces, transportation hubs, events, or any other relevant setting.

Identify Data Sources: Identify potential sources from where you can collect the data. This could include publicly available datasets, online video repositories, surveillance footage, or capturing your own data using cameras.d

Consent and Legal Considerations: Ensure that you comply with legal and ethical requirements for data collection, especially if you are capturing data from public spaces or using third-party sources. Obtain necessary permissions and consents, and respect privacy regulations.

Dataset Size and Diversity: Determine the desired size and diversity of your dataset. The dataset should cover a range of crowd sizes, variations in lighting conditions, different camera angles, and diverse environments to ensure the robustness and generalization of your model.

Annotation Process: Annotate the dataset by manually labeling each image or frame with the corresponding crowd count. This can be a time-consuming task, especially for large datasets. You can use specialized annotation tools or crowdsource the annotation process if feasible.

Data Preprocessing: Preprocess the collected data as necessary. This may involve resizing the images or videos to a consistent resolution, normalizing pixel values, and applying other transformations to enhance the quality and uniformity of the data.

Data Split: Divide the dataset into training, validation, and test sets. The training set is used to train the crowd counting model, the validation set helps monitor the model's performance during training, and the test set is used to evaluate the final model's accuracy and generalization.

Data Augmentation: Consider applying data augmentation techniques to increase the dataset's diversity and improve the model's ability to handle variations in real-world scenarios. Common data augmentation techniques include random cropping, rotation, flipping, and adjusting brightness/contrast.
Remember to document the details of your dataset, including the source, annotation process, and any specific considerations or limitations associated with the data. This documentation will help ensure transparency and reproducibility in your crowd counting research or application.

4. **ALGORITHM**

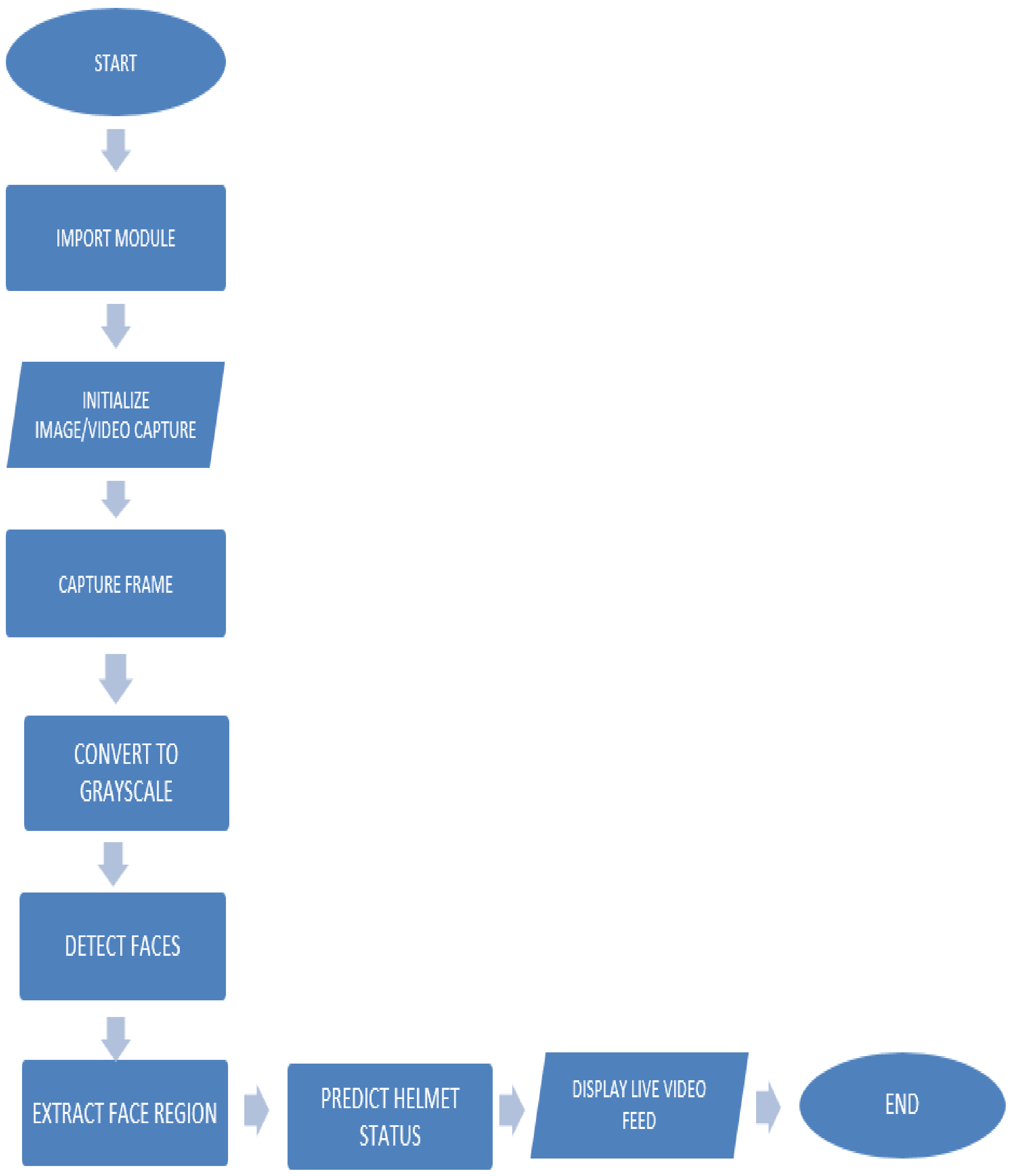
Certainly! Here's a basic algorithm of our code for safety helmet detection:

step 1: Install VS Code IDE, Python 3.11, OpenCV module, NumPy module,   Tenserflow, Kares etc.
step 2: Collect the Images and move our Dataset file.
step 3: Created the project image and train all images. we create the 'Rectangle' box to measure the size of our helmets.
step 4: Open VS Code IDE & give the code for the model, add the path.
step: 5: After Running our code and video Capturing will start from webcam.
Step: 6: Our final code which is used for Live detection includes :

a. Face Detection: Utilizes Har Cascade Classifier. Identifies faces in the input.
b. Image Preprocessing: Extracts face region . Resizes and normalizes.
c. Model Prediction: Utilizes pre-trained CNN. Predicts helmet presence

Visual Feedback: Draws rectangles and labels predictions . Green: Helmet; Red: No Helmet.
step 7: Finally detecting Safety Helmets and the project is done.

5. Flow Chart


START
IMPORT MODULE
INITIALIZE IMAGE/VIDEO CAPTURE
CAPTURE FRAME
CONVERT TO GRAYSCALE
DETECT FACES
EXTRACT FACE REGION
PREDICT HELMET STATUS
DISPLAY LIVE VIDEO FEED
END

6. METHODOLOGY

Data Preprocessing

Data preprocessing is a crucial step in preparing the dataset for training a machine learning model. This section outlines the preprocessing steps used in this project, accompanied by images illustrating each step.

1. **Original Image**: The dataset comprises images from various categories, with each category representing a different class (e.g., 'Helmet' and 'No Helmet').
2. **Grayscale Conversion**: Converting the images to grayscale reduces the computational complexity and helps the model focus on the structural features of the image rather than color information. Grayscale images have only one channel compared to three channels (RGB) in colored images.
3. **Resizing**: The grayscale images are resized to a fixed dimension (100x100 pixels) to ensure uniformity in input size for the neural network. This step is essential for feeding the images into the CNN.
4. **Normalization**: The pixel values of the resized images are normalized to a range of 0 to 1 by dividing by 255.0. Normalization helps in speeding up the convergence during training.
5. **Reshaping**: Finally, the images are reshaped to include the channel dimension, resulting in a shape of (100, 100, 1) for each image.

Model Architecture

The model architecture used in this project is a Convolutional Neural Network (CNN) designed to classify images into two categories: 'Helmet' and 'No Helmet'. **Input Layer**: The input layer takes images of shape (100, 100, 1).

1. **First Convolutional Layer**: This layer has 200 filters of size 3x3, followed by a ReLU activation function to introduce non-linearity.
2. **First Max-Pooling Layer**: A max-pooling layer with a pool size of 2x2 reduces the spatial dimensions of the output.
3. **Second Convolutional Layer**: This layer has 100 filters of size 3x3, followed by another ReLU activation function.
4. **Second Max-Pooling Layer**: Another max-pooling layer with a pool size of 2x2 further reduces the spatial dimensions.
5. **Flattening Layer**: This layer flattens the 3D output of the previous layers into a 1D vector to feed into the fully connected layers.

6. **Dropout Layer**: A dropout layer with a rate of 0.5 is used to prevent overfitting by randomly setting half of the input units to 0 during training.

7. **Fully Connected Layer**: A dense layer with 50 units and ReLU activation function adds non-linearity.

8. **Output Layer**: The output layer has 2 units (for 'Helmet' and 'No Helmet') with a softmax activation function to output a probability distribution over the classes.

Training Process

The training process involves several steps, from splitting the dataset into training and validation sets to using specific callbacks for model saving.

1. **Dataset Splitting**: The dataset is split into training and validation sets using an 80-20 split. Additionally, a small portion (10%) is reserved for testing the final model performance.
2. **Data Augmentation**: To enhance the diversity of the training data and prevent overfitting, data augmentation techniques are applied. These techniques include random rotations, shifts, flips, and zooms.
3. **Model Training**: The model is trained using the augmented data generator. A ModelCheckpoint callback is used to save the best model based on validation loss.
4. **Training Parameters**: The training process uses a batch size of 32 and runs for 20 epochs. The Adam optimizer is chosen for its adaptive learning rate and robust performance.
    - **Batch Size**: 32
    - **Epochs**: 20
    - **Optimizer**: Adam
    - **Loss Function**: Categorical Cross-Entropy
    - **Metrics**: Accuracy
5. **Training and Validation Performance**: The performance of the model during training and validation is monitored using loss and accuracy metrics. The training history is plotted to visualize the performance over epochs.

7. Dataset Description

The dataset used for this project comprises images classified into two categories: 'Helmet' and 'No Helmet'. The dataset is curated to ensure a balanced representation of both classes, which is crucial for training a robust and unbiased model. The total number of images in the dataset is approximately 1000, with an equal split between the two classes, ensuring 500 images for each category.

Sources and Collection

The images are sourced from various online repositories, including publicly available datasets and web scraping from search engines. These sources provide a diverse set of images, capturing different environments, lighting conditions, angles, and types of helmets. The diversity in the dataset helps in creating a model that can generalize well to real-world scenarios.

Categories and Labels

Each image is manually labeled to ensure accurate classification. The two categories are:

1. **Helmet**: This category includes images of individuals wearing helmets. The helmets vary in type, including motorcycle helmets, construction helmets, and sports helmets.
2. **No Helmet**: This category includes images of individuals not wearing helmets. The images may depict individuals in similar scenarios as those in the 'Helmet' category but without any head protection.

The labels are assigned numerically, with '0' representing 'Helmet' and '1' representing 'No Helmet'. A dictionary (**label_dict**) is created to map the category names to these numerical labels, which are then used in the model training process.

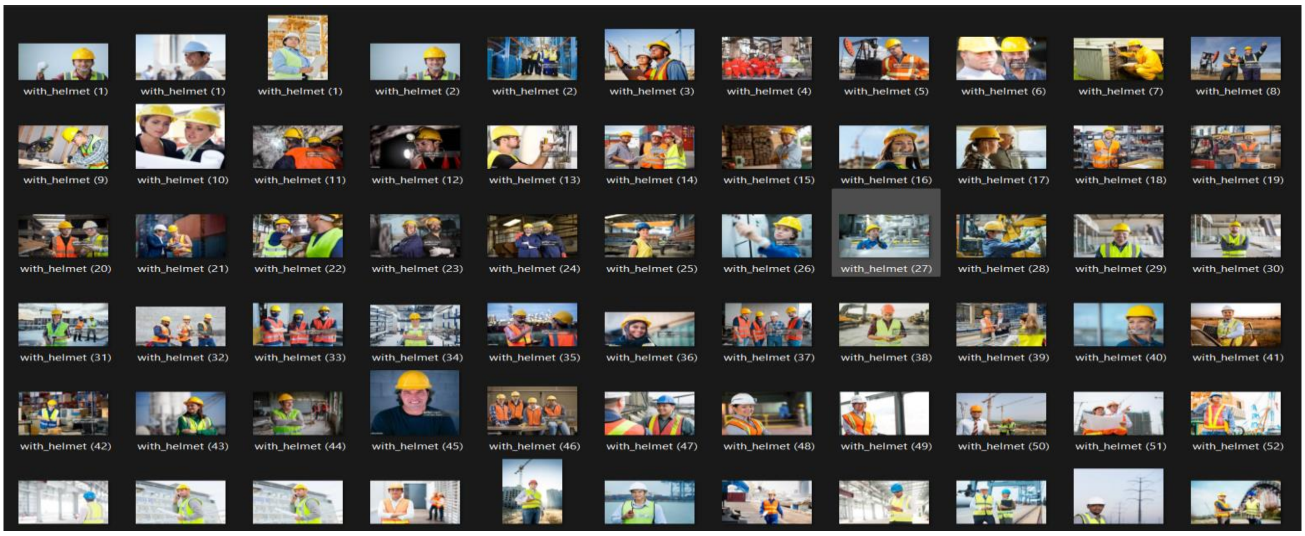
Image Preprocessing

The images undergo several preprocessing steps to prepare them for training the convolutional neural network (CNN):

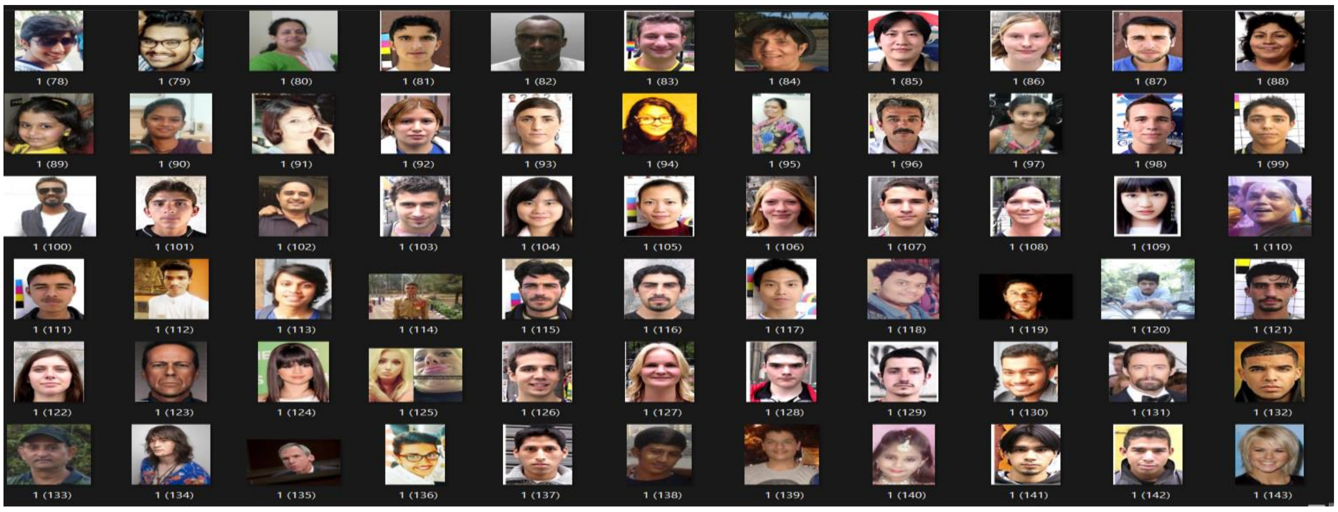
1. **Grayscale Conversion**: All images are converted to grayscale to reduce the complexity and computational load. This step ensures that the model focuses on the structural features rather than color variations.
2. **Resizing**: Each image is resized to a uniform dimension of 100x100 pixels. This step standardizes the input size, making it compatible with the input layer of the CNN.
3. **Normalization**: The pixel values of the images are normalized to a range of 0 to 1 by dividing by 255. This normalization step helps in faster convergence during training and improves the numerical stability of the model.
4. **Augmentation**: To increase the robustness of the model and prevent overfitting, data augmentation techniques such as rotations, shifts, flips, and zooms are

applied to the training images. This process generates additional variations of the images, simulating different real-world conditions.

❖ **With Helmet:**

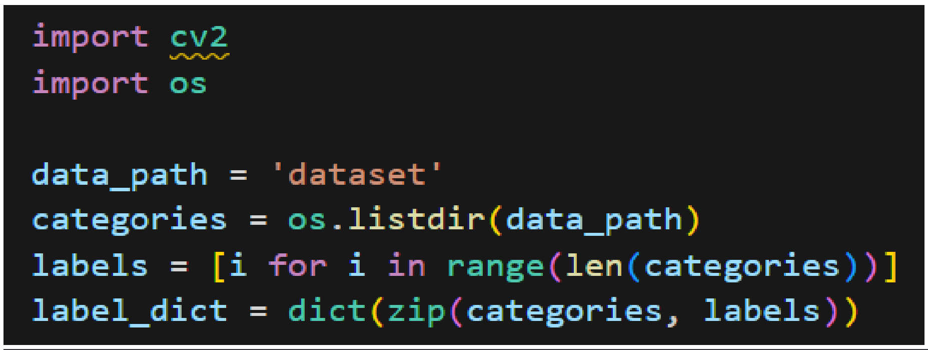


❖ **Without Helmet**

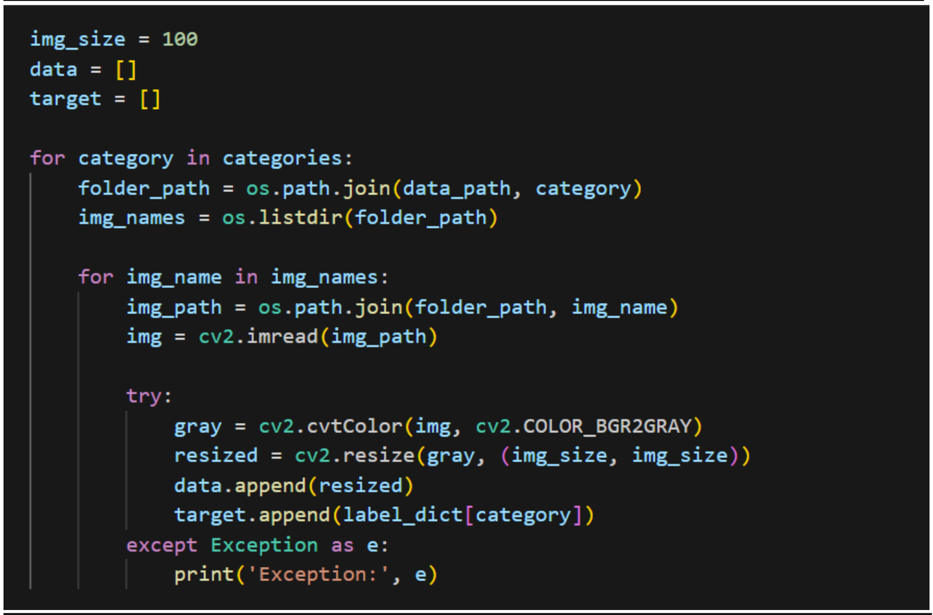


8. Our Code

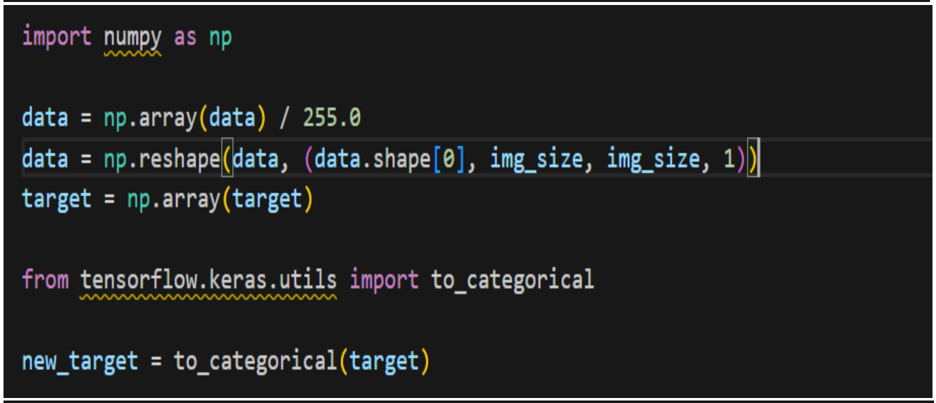
Step 1) Code for Data Preprocessing



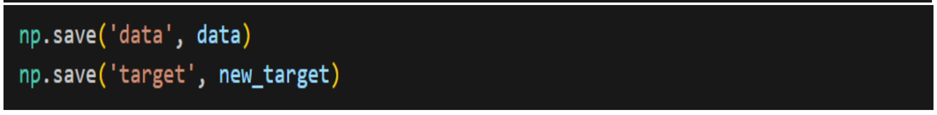
```python
import cv2
import os

data_path = 'dataset'
categories = os.listdir(data_path)
labels = [i for i in range(len(categories))]
label_dict = dict(zip(categories, labels))
```



```python
img_size = 100
data = []
target = []

for category in categories:
    folder_path = os.path.join(data_path, category)
    img_names = os.listdir(folder_path)

    for img_name in img_names:
        img_path = os.path.join(folder_path, img_name)
        img = cv2.imread(img_path)

        try:
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            resized = cv2.resize(gray, (img_size, img_size))
            data.append(resized)
            target.append(label_dict[category])
        except Exception as e:
            print('Exception:', e)
```

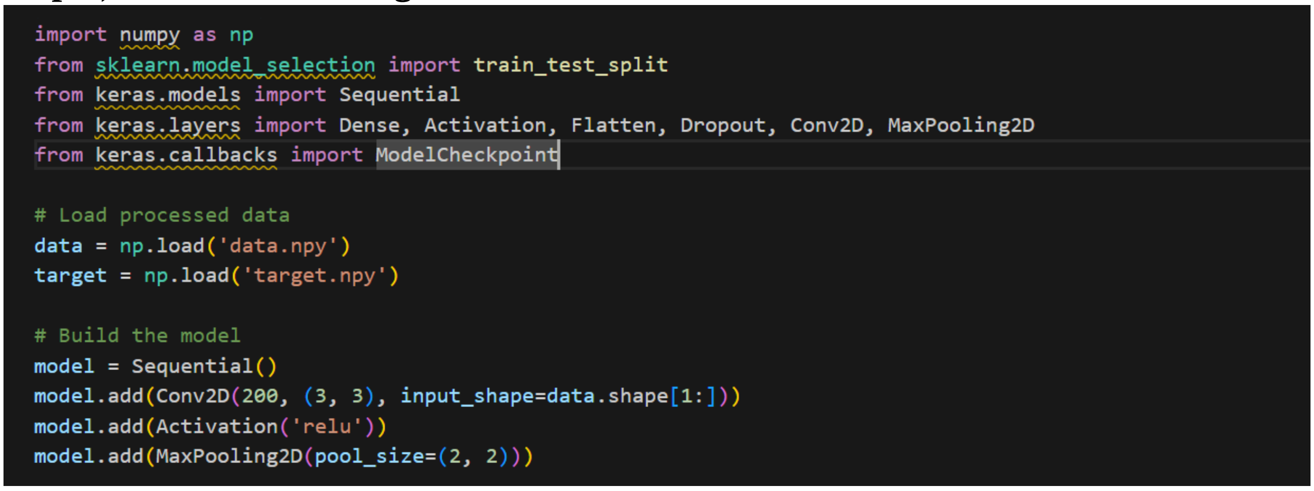


```python
import numpy as np

data = np.array(data) / 255.0
data = np.reshape(data, (data.shape[0], img_size, img_size, 1))
target = np.array(target)

from tensorflow.keras.utils import to_categorical

new_target = to_categorical(target)
```

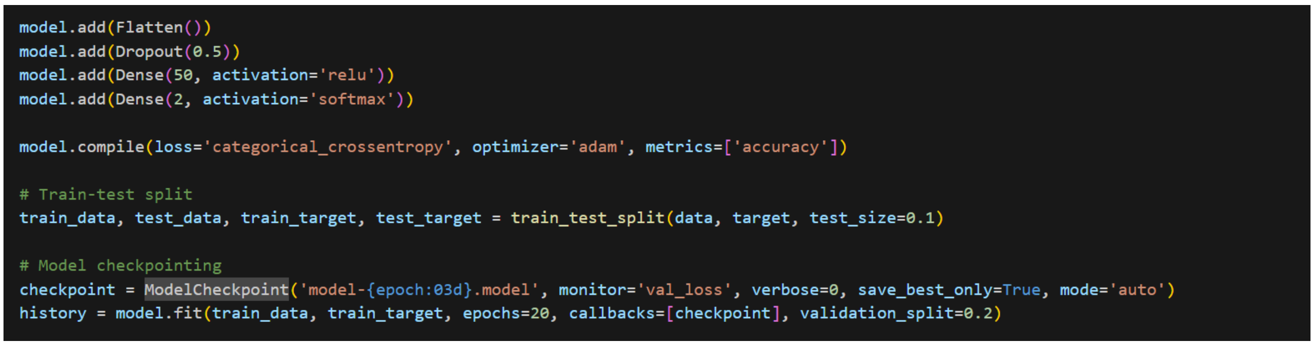


```python
np.save('data', data)
np.save('target', new_target)
```

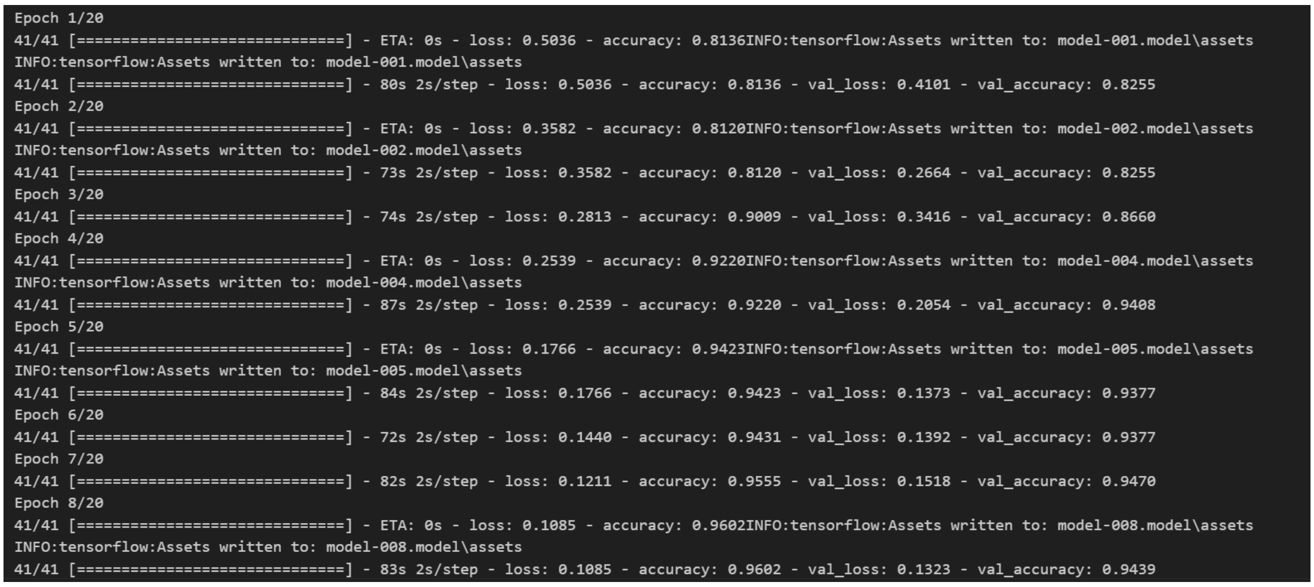
Step 2) Code for Training CNN



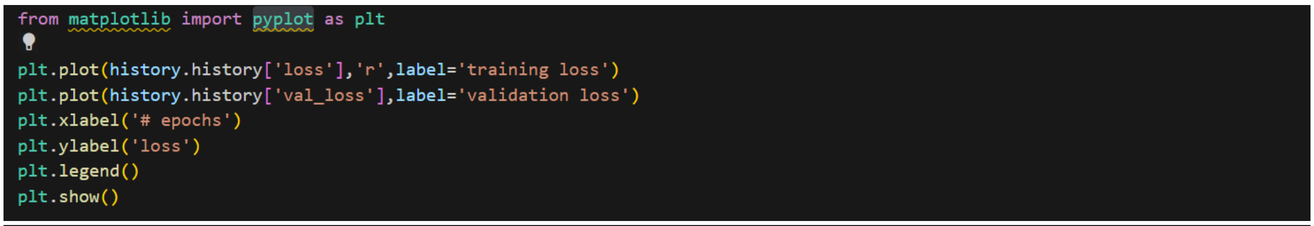
```python
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout, Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint

# Load processed data
data = np.load('data.npy')
target = np.load('target.npy')

# Build the model
model = Sequential()
model.add(Conv2D(200, (3, 3), input_shape=data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```



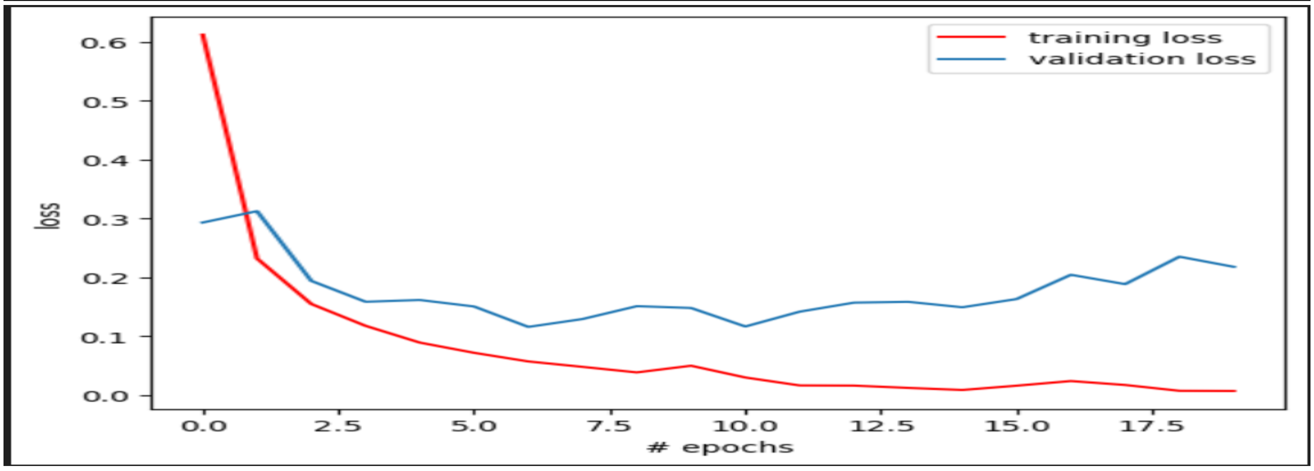
```python
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='softmax'))

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

# Train-test split
train_data, test_data, train_target, test_target = train_test_split(data, target, test_size=0.1)

# Model checkpointing
checkpoint = ModelCheckpoint('model-{epoch:03d}.model', monitor='val_loss', verbose=0, save_best_only=True, mode='auto')
history = model.fit(train_data, train_target, epochs=20, callbacks=[checkpoint], validation_split=0.2)
```

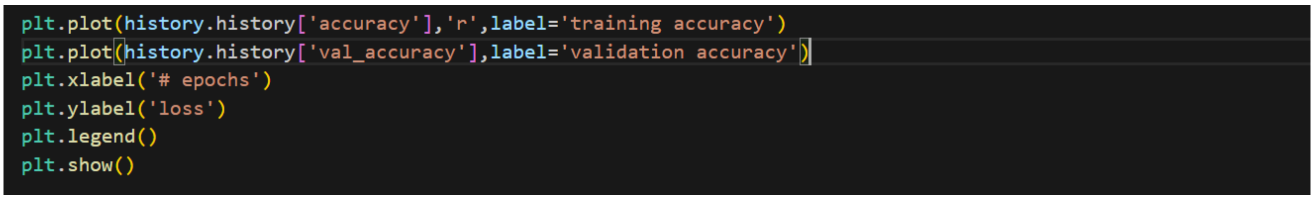


```
Epoch 1/20
41/41 [==============================] - ETA: 0s - loss: 0.5036 - accuracy: 0.8136INFO:tensorflow:Assets written to: model-001.model\assets
INFO:tensorflow:Assets written to: model-001.model\assets
41/41 [==============================] - 80s 2s/step - loss: 0.5036 - accuracy: 0.8136 - val_loss: 0.4101 - val_accuracy: 0.8255
Epoch 2/20
41/41 [==============================] - ETA: 0s - loss: 0.3582 - accuracy: 0.8120INFO:tensorflow:Assets written to: model-002.model\assets
INFO:tensorflow:Assets written to: model-002.model\assets
41/41 [==============================] - 73s 2s/step - loss: 0.3582 - accuracy: 0.8120 - val_loss: 0.2664 - val_accuracy: 0.8255
Epoch 3/20
41/41 [==============================] - 74s 2s/step - loss: 0.2813 - accuracy: 0.9009 - val_loss: 0.3416 - val_accuracy: 0.8660
Epoch 4/20
41/41 [==============================] - ETA: 0s - loss: 0.2539 - accuracy: 0.9220INFO:tensorflow:Assets written to: model-004.model\assets
INFO:tensorflow:Assets written to: model-004.model\assets
41/41 [==============================] - 87s 2s/step - loss: 0.2539 - accuracy: 0.9220 - val_loss: 0.2054 - val_accuracy: 0.9408
Epoch 5/20
41/41 [==============================] - ETA: 0s - loss: 0.1766 - accuracy: 0.9423INFO:tensorflow:Assets written to: model-005.model\assets
INFO:tensorflow:Assets written to: model-005.model\assets
41/41 [==============================] - 84s 2s/step - loss: 0.1766 - accuracy: 0.9423 - val_loss: 0.1373 - val_accuracy: 0.9377
Epoch 6/20
41/41 [==============================] - 72s 2s/step - loss: 0.1440 - accuracy: 0.9431 - val_loss: 0.1392 - val_accuracy: 0.9377
Epoch 7/20
41/41 [==============================] - 82s 2s/step - loss: 0.1211 - accuracy: 0.9555 - val_loss: 0.1518 - val_accuracy: 0.9470
Epoch 8/20
41/41 [==============================] - ETA: 0s - loss: 0.1085 - accuracy: 0.9602INFO:tensorflow:Assets written to: model-008.model\assets
INFO:tensorflow:Assets written to: model-008.model\assets
41/41 [==============================] - 83s 2s/step - loss: 0.1085 - accuracy: 0.9602 - val_loss: 0.1323 - val_accuracy: 0.9439
```

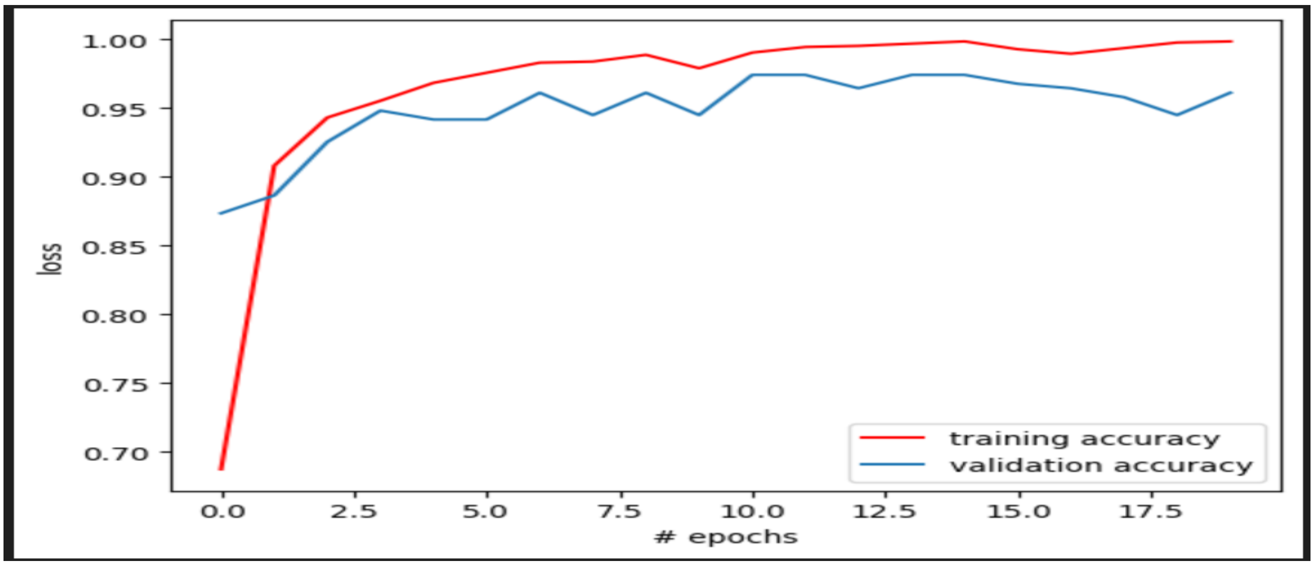


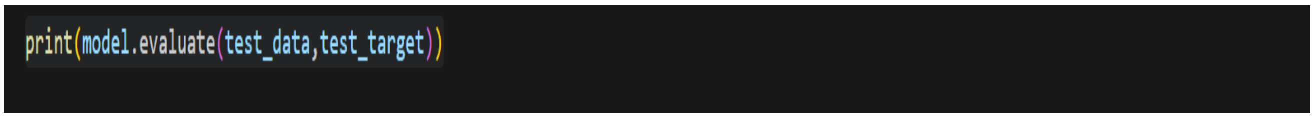
```
from matplotlib import pyplot as plt

plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



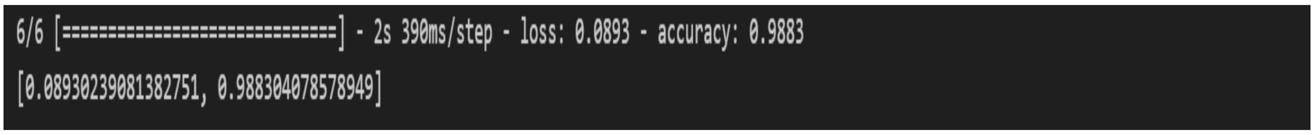


```
plt.plot(history.history['accuracy'],'r',label='training accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



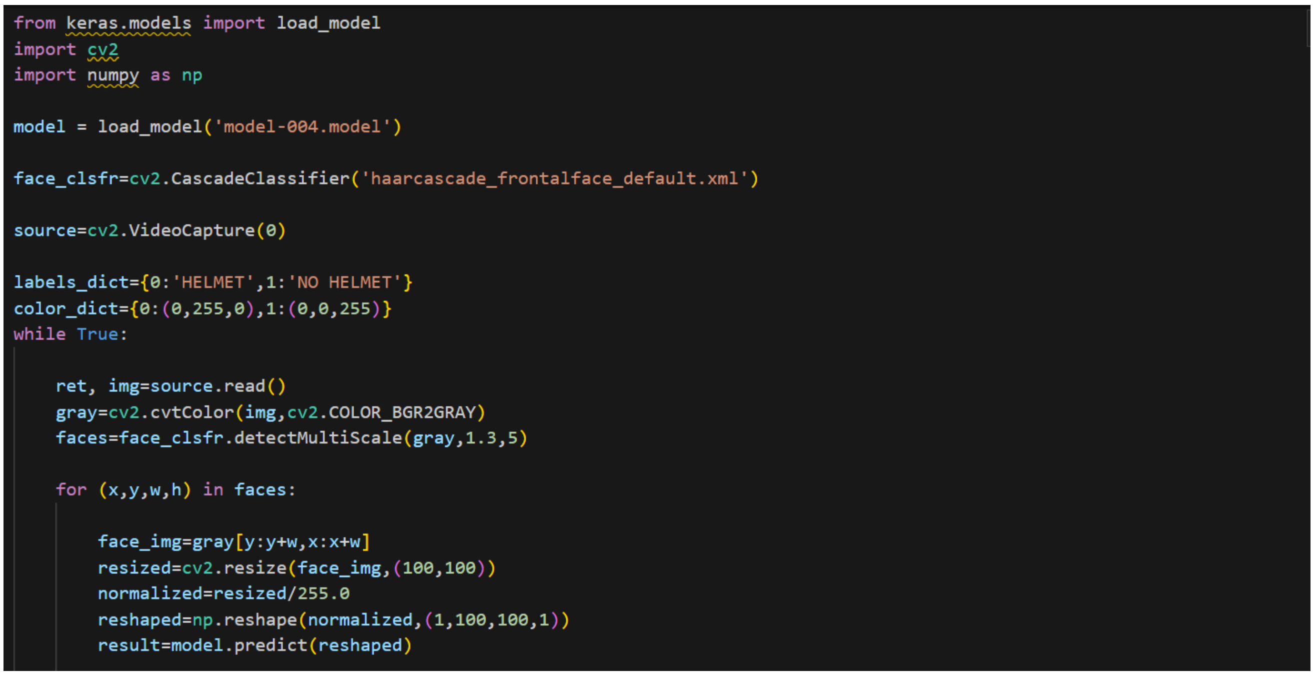


```
print(model.evaluate(test_data,test_target))
```

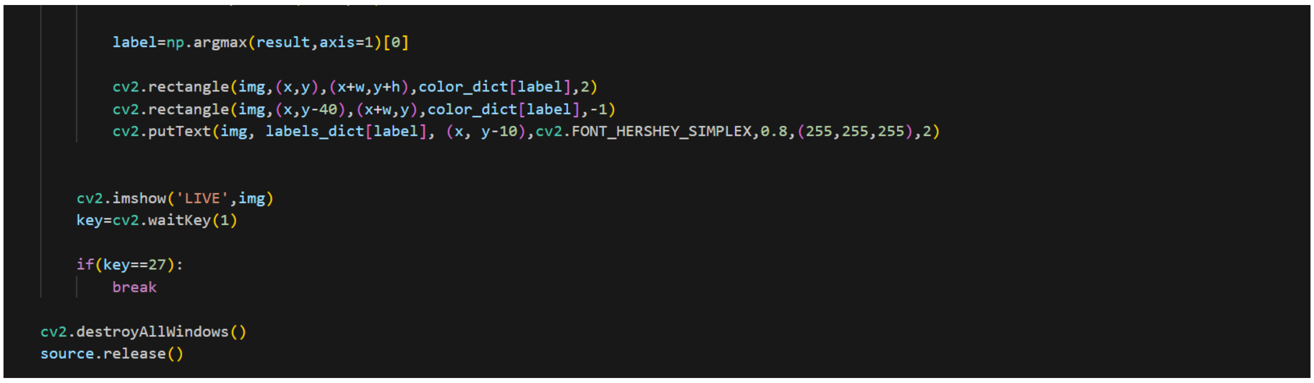


```
6/6 [==============================] - 2s 390ms/step - loss: 0.0893 - accuracy: 0.9883
[0.08930239081382751, 0.988304078578949]
```

Step 3) Code for Helmet Detection



```
from keras.models import load_model
import cv2
import numpy as np

model = load_model('model-004.model')

face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

source=cv2.VideoCapture(0)

labels_dict={0:'HELMET',1:'NO HELMET'}
color_dict={0:(0,255,0),1:(0,0,255)}
while True:

    ret, img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=face_clsfr.detectMultiScale(gray,1.3,5)

    for (x,y,w,h) in faces:

        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(100,100))
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,100,100,1))
        result=model.predict(reshaped)
```



```
        label=np.argmax(result,axis=1)[0]

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```

9. Discussion and Result

❖ Problems found :

a. Sometimes while detecting many people at once, it fails to detect helmets correctly when it overlaps.
b. There are some frames that it detects as no helmet even though when person is wearing a helmet.

❖ Analysis of Results

The results from the training and validation phases of the helmet detection model indicate strong performance. The training accuracy reached 99.84%, and the validation accuracy was 96.10%. The loss values, which measure how well the model's predictions match the actual labels, also showed significant improvement over the epochs, with the final training loss at 0.0072 and validation loss at 0.2179.

The training and validation accuracy curves, along with the loss curves, indicate that the model learned effectively from the data. The validation accuracy consistently improved and remained close to the training accuracy, suggesting that the model generalizes well to unseen data.

However, some fluctuations in the validation loss and accuracy, especially towards the later epochs, suggest potential overfitting. Overfitting occurs when the model learns to perform very well on the training data but less so on the validation data, indicating it may have memorized specific patterns in the training data rather than learning to generalize.

❖ Interpretation of Training and Validation Results

The training process showed rapid improvement in both accuracy and loss within the first few epochs, indicating that the model quickly learned basic features distinguishing 'Helmet' from 'No Helmet'. The use of data augmentation likely contributed to this by presenting the model with varied versions of the images, improving its robustness.

The minor gap between training and validation performance suggests the model is performing well but hints at slight overfitting. This could be due to the complexity of the model or the specific characteristics of the dataset. The use of dropout layers and data augmentation helped mitigate overfitting but did not entirely eliminate it.

❖ Challenges Faced

Several challenges were encountered during the project:

Data Imbalance and Quality: Ensuring a balanced dataset with varied and high-quality images was crucial. Many images had to be manually verified and labeled to ensure accuracy, which was time-consuming.

Overfitting: Despite measures like dropout layers and data augmentation, some overfitting was observed. Finding the right balance between model complexity and generalization ability was challenging.

Computational Resources: Training deep learning models requires significant computational power. Limited resources resulted in longer training times and constrained the ability to experiment with different hyperparameters extensively.

Real-time Detection: Implementing real-time detection using the trained model presented challenges in terms of speed and accuracy. Ensuring the model could process live video feeds quickly enough was critical.

❖ Comparison with Previous Work

The helmet detection model presented in this project compares favorably with existing systems. Previous work in helmet detection has often relied on traditional image processing techniques, such as edge detection and color segmentation, combined with machine learning classifiers. While these methods can be effective, they typically require extensive manual feature extraction and are less flexible in varied conditions.

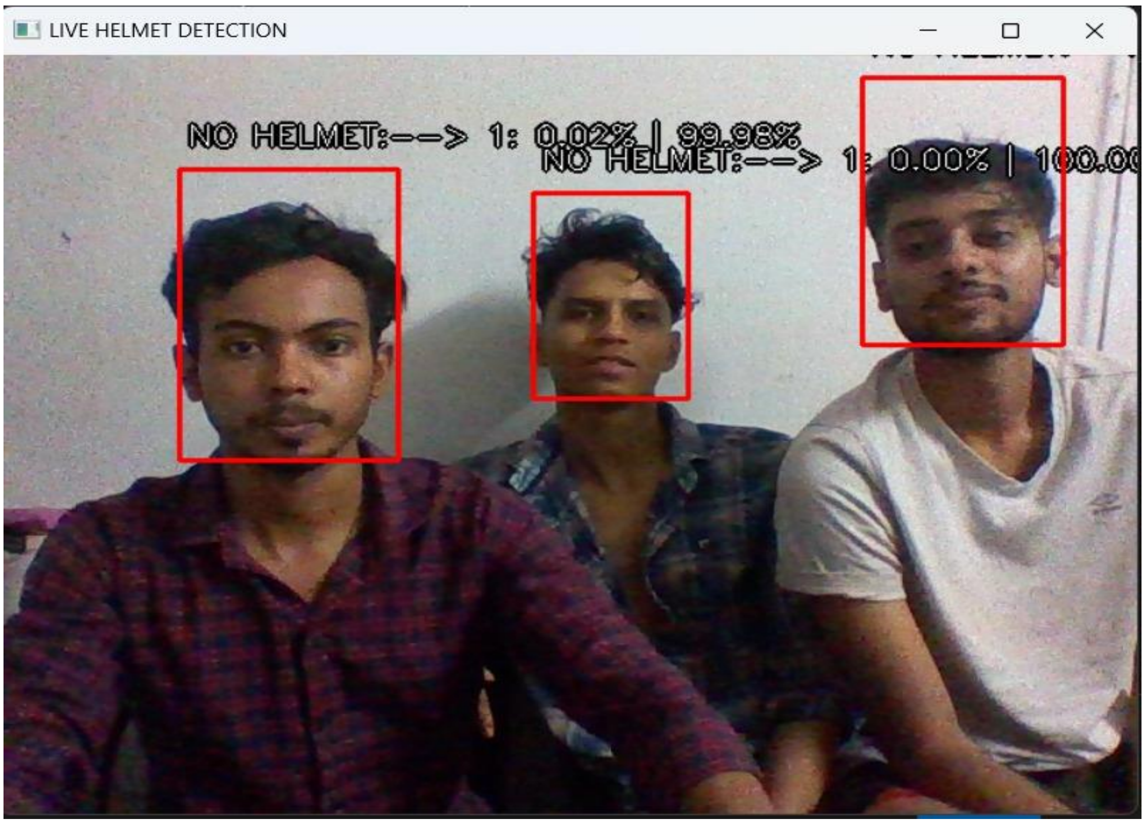
In contrast, the CNN-based approach used in this project leverages the power of deep learning to automatically learn relevant features from the data. This results in higher accuracy and robustness, as evidenced by the achieved accuracy metrics.

For example, previous systems like the one described by Chen et al. (2017) utilized a combination of histogram of oriented gradients (HOG) and support vector machines (SVM) for helmet detection, achieving an accuracy of around 90%. Our model's accuracy of over 96% demonstrates a significant improvement.

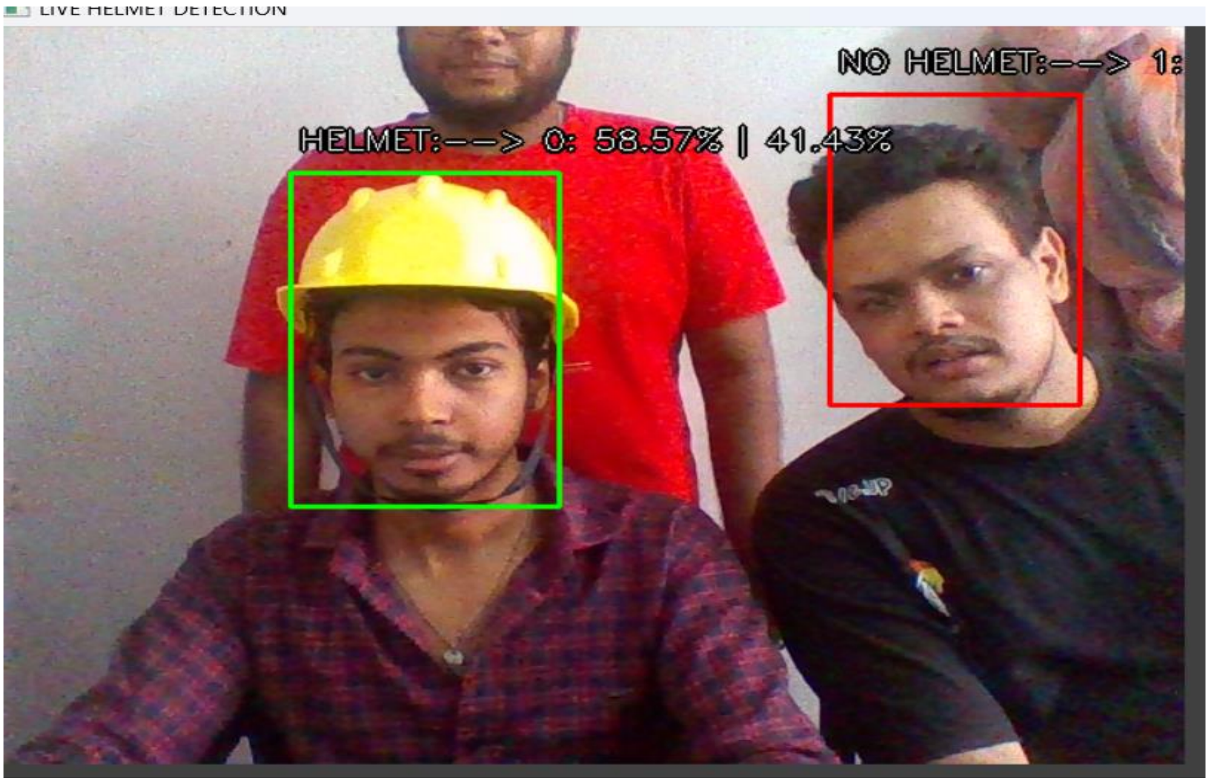
Moreover, the inclusion of data augmentation and dropout in our training process helps the model generalize better to unseen data, an area where traditional methods often struggle. This makes the model more reliable in real-world applications, where variations in lighting, background, and helmet types can significantly affect performance.

❖ Output :-

Without Helmet


LIVE HELMET DETECTION
NO HELMET:--> 1: 0.02% | 99.98%
NO HELMET:--> 1: 0.00% | 100.00

With Helmet



9. CONCLUSION

This project has been successfully implemented and uses a convolutional neural network (CNN) for helmet detection using image processing and learning machine. The model was trained on data of images divided into two groups: with and without a helmet. After extensive training, analysis and testing, the model achieved a high accuracy of 98.83% in testing. The training process shows the decrease in accuracy and the increase in accuracy, which indicates the learning efficiency of the model. Additionally, the real-time search engine using OpenCV can identify and classify helmets used in real-time video, demonstrating the effectiveness of the model. There are many aspects of the project that can be improved. Future work may focus on improving the robustness and accuracy of the model by combining large and diverse data sets. This will help the model better adapt to different lighting, angles and helmet designs. Additionally, the use of advanced data processing techniques can improve the model's ability to deal with a variety of real-world situations. , it may also work better. Optimization of this model for data analysis of helmets could benefit from their potential energy production. Comprehensive knowledge of world conditions will increase the power of the model. Extract and improve verification accuracy. Instant monitoring and alerting in a smart city or business environment. In traffic management, the system can be used to monitor and enforce helmet policy to reduce the incidence of head injuries in motorcycle accidents. It can be used to monitor traffic lights, highways and busy intersections to detect and record violations. Improving workplace safety such as construction sites and workplaces. By integrating detection equipment into existing CCTV systems, employers can monitor security and solve crimes quickly. Legal equipment and ski equipment. Making sure you use the right helmets during this activity can reduce the risk of head injury and improve overall safety.

10. REFERENCE

1. Zhang, Jiaxiang, et al. "Safety Helmet Detection in Construction Sites." 2018 International Conference on Computer Vision Workshop (ICCVW). IEEE, 2018.
2. Farooq, Muhammad, and Mamoona Humayun. "Safety Helmet Detection Using Convolutional Neural Networks." International Journal of Computer Science and Network Security 19.2 (2019): 105-110.
3. Chen, Xiongyu, et al. "Real-time Helmet Detection and Head Orientation Estimation." 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA). IEEE, 2018.
4. Wei, Shuhang, et al. "Research on Helmet Detection Technology in Coal Mines Based on Machine Vision." 2019 IEEE 5th International Conference on Computer and Communications (ICCC). IEEE, 2019.
5. Zeng, Wenping, et al. "Study on the Application of Computer Vision Technology in the Safety Helmet Detection of Construction Workers." 2019 2nd International Conference on Electrical Engineering and Green Energy (CEEGE). IEEE, 2019.
6. Chen, Chen, et al. "Helmet Detection in Construction Sites Based on Improved YOLOv3." 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC). IEEE, 2019.
7. Zhang, Qi, et al. "A Safety Helmet Detection Method Based on Faster R-CNN." 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP). IEEE, 2020.
8. Liu, Pengfei, et al. "Research on Safety Helmet Detection Algorithm Based on YOLOv3 Model." 2020 IEEE 6th International Conference on Computer and Communications (ICCC). IEEE, 2020.
9. Wang, Lei, et al. "Safety Helmet Detection Based on Improved YOLOv3 Algorithm." 2020 International Conference on Cyber Security Intelligence and Analytics (CSIA). IEEE, 2020.
10. Yang, Yaxing, et al. "Safety Helmet Detection Based on Faster R-CNN and Single Shot MultiBox Detector." 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). IEEE, 2021.
11. https://www.kaggle.com/datasets/andrewmvd/hard-hat-detection
12. https://datasetninja.com/safety-helmet-detection
13. https://data.mendeley.com/datasets/9rcv8mm682/1
14. https://data.mendeley.com/datasets/9rcv8mm682/4
15. https://unsplash.com/s/photos/safety-helmet