Problem Statement

Business Context

Workplace safety in hazardous environments like construction sites and industrial plants is crucial to prevent accidents and injuries. One of the most important safety measures is ensuring workers wear safety helmets, which protect against head injuries from falling objects and machinery. Non-compliance with helmet regulations increases the risk of serious injuries or fatalities, making effective monitoring essential, especially in large-scale operations where manual oversight is prone to errors and inefficiency.

To overcome these challenges, SafeGuard Corp plans to develop an automated image analysis system to detect whether workers are wearing safety helmets. This system will improve safety enforcement, ensuring compliance and reducing the risk of head injuries. By automating helmet monitoring, SafeGuard aims to enhance efficiency, scalability, and accuracy, ultimately fostering a safer work environment while minimizing human error in safety oversight.

Objective

As a data scientist at SafeGuard Corp, you are tasked with developing an image classification model that classifies images into one of two categories:

- **With Helmet:** Workers wearing safety helmets.
- **Without Helmet:** Workers not wearing safety helmets.

The ultimate objective is to prepare the model for deployment as part of an automated monitoring system. This system will enable real-time analysis of images and assist in ensuring compliance with workplace safety regulations.

Data Description

The dataset consists of **631 images**, equally divided into two categories:

- **With Helmet:** 311 images showing workers wearing helmets.
- **Without Helmet:** 320 images showing workers not wearing helmets.

Dataset Characteristics:

- **Variations in Conditions:** Images include diverse environments such as construction sites, factories, and industrial settings, with variations in lighting, angles, and worker postures to simulate real-world conditions.
- **Worker Activities:** Workers are depicted in different actions such as standing, using tools, or moving, ensuring robust model learning for various scenarios.

Installing and Importing the Necessary Libraries

```
!pip install tensorflow -q
```

```
import tensorflow as tf
print("Num GPUs Available:",
len(tf.config.list_physical_devices('GPU')))
print(tf.__version__)
```

```
Num GPUs Available: 1
2.18.0
```

```
import os
import random
import numpy as np
# Importing numpy for Matrix Operations
import pandas as pd
import seaborn as sns
import matplotlib.image as mpimg
# Importing pandas to read CSV files
import matplotlib.pyplot as plt
# Importting matplotlib for Plotting and visualizing images
import math
# Importing math module to perform mathematical operations
import cv2


# Tensorflow modules
import keras
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Importing the ImageDataGenerator for data augmentation
from tensorflow.keras.models import Sequential
# Importing the sequential module to define a sequential model
from tensorflow.keras.layers import
Dense,Dropout,Flatten,Conv2D,MaxPooling2D,BatchNormalization #
Defining all the layers to build our CNN Model
from tensorflow.keras.optimizers import Adam,SGD
# Importing the optimizers which can be used in our model
from sklearn import preprocessing
# Importing the preprocessing module to preprocess the data
from sklearn.model_selection import train_test_split
# Importing train_test_split function to split the data into train and
test
from sklearn.metrics import confusion_matrix
# Importing confusion_matrix to plot the confusion matrix
from tensorflow.keras.models import Model
from keras.applications.vgg16 import VGG16
```

```
from tensorflow.keras.layers import SpatialDropout2D,
GlobalAveragePooling2D
from tensorflow.keras.utils  import clear_session as cls

# Display images using OpenCV
from google.colab.patches import cv2_imshow

#Imports functions for evaluating the performance of machine learning
models
from sklearn.metrics import confusion_matrix, f1_score,accuracy_score,
recall_score, precision_score, classification_report
from sklearn.metrics import mean_squared_error as mse
# Importing cv2_imshow from google.patches to display images

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Set the seed using keras.utils.set_random_seed. This will set:
# 1) `numpy` seed
# 2) backend random seed
# 3) `python` random seed
tf.keras.utils.set_random_seed(812)
```

Data Overview

##Loading the data

```
from google.colab import drive
drive.mount('/content/drive')
pathl='/content/drive/MyDrive/Dataset/Labels_proj.csv'
pathim='/content/drive/MyDrive/Dataset/images_proj.npy'
```

```
Mounted at /content/drive
```

```
labels=pd.read_csv(pathl)
labels.head()
```


```
{"summary":"{\n  \"name\": \"labels\",\n  \"rows\": 631,\n
\"fields\": [\n    {\n      \"column\": \"Label\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"labels"}
```

```
images=np.load(pathim)
images.shape
```

```
(631, 200, 200, 3)
```

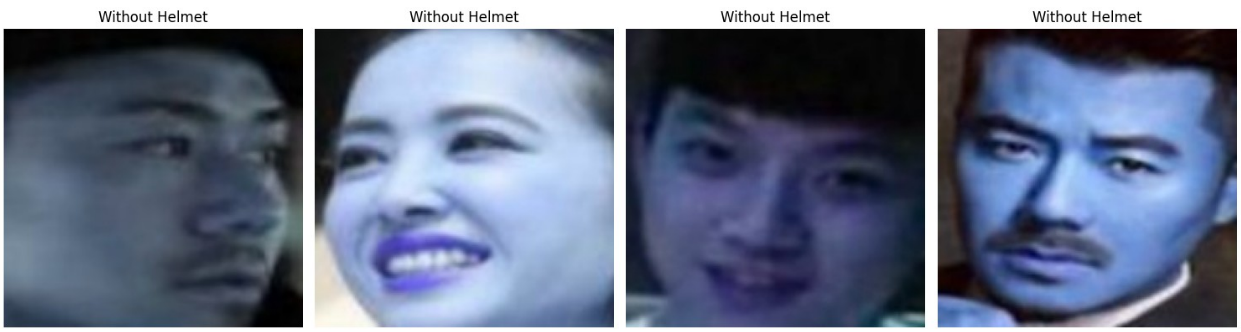
Exploratory Data Analysis

###Plotting random images from each of the classes and printing their corresponding labels.

```
la1=labels[labels['Label']==1].index.tolist()
la0=labels[labels['Label']==0].index.tolist()
lam=la1[:4]+la0[:4]
lam
```

```
[0, 1, 2, 3, 271, 272, 273, 274]
```

```
fig,ax=plt.subplots(2,4,figsize=(15,15))

ax=ax.flatten()
for i,j in zip(ax,lam):
  i.imshow(images[j])
  i.axis('off')
  if j in la1:
    i.set_title('With Helmet')
  else:
    i.set_title('Without Helmet')
plt.tight_layout()
plt.show()
```

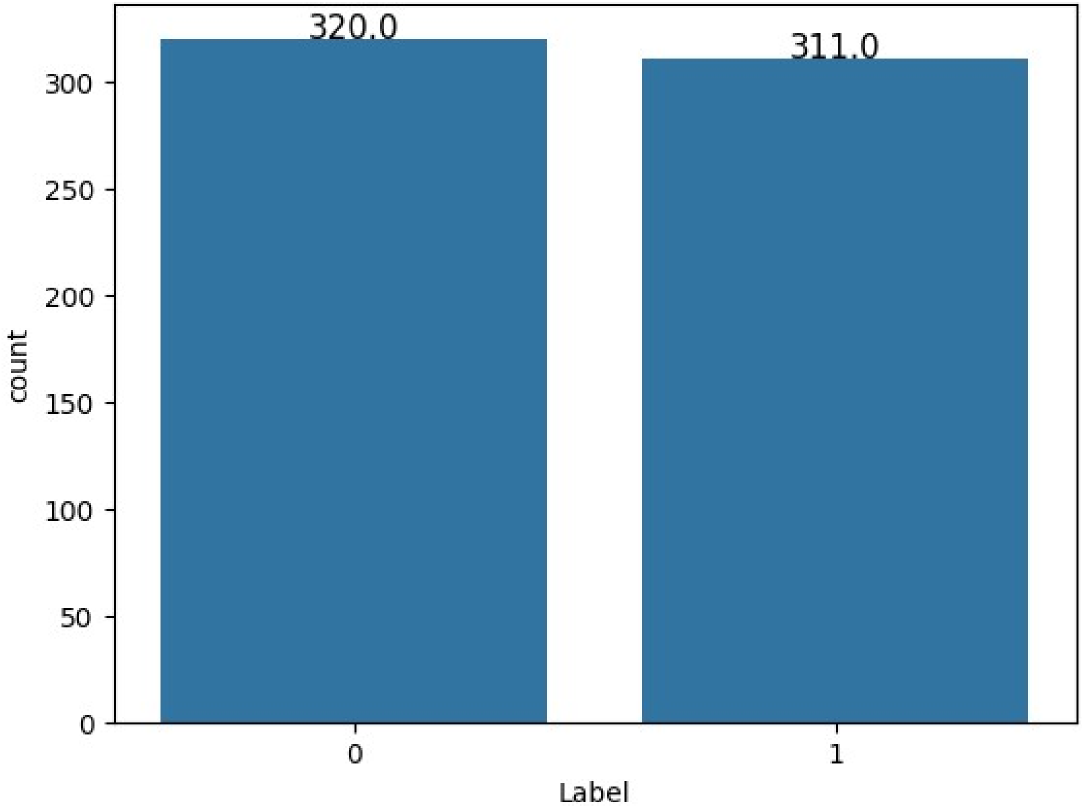


Observations

- From the images, we observe a blue tint, which occurs because matplotlib prints in RGB format, while the image might be in BGR format.

Checking for class imbalance

```
a=sns.countplot(data=labels,x='Label')
for i in a.patches:

    a.text(
        x=(i.get_x() + i.get_width() / 2),
        y=i.get_height() + 5,
        s=f"{i.get_height()}",
        ha='center',
        va='center',
        size=12
    )

# Show the plot
plt.show()
```



Observations:

- The Class imbalance is not that very big to affect the performance of the model.

Data Preprocessing

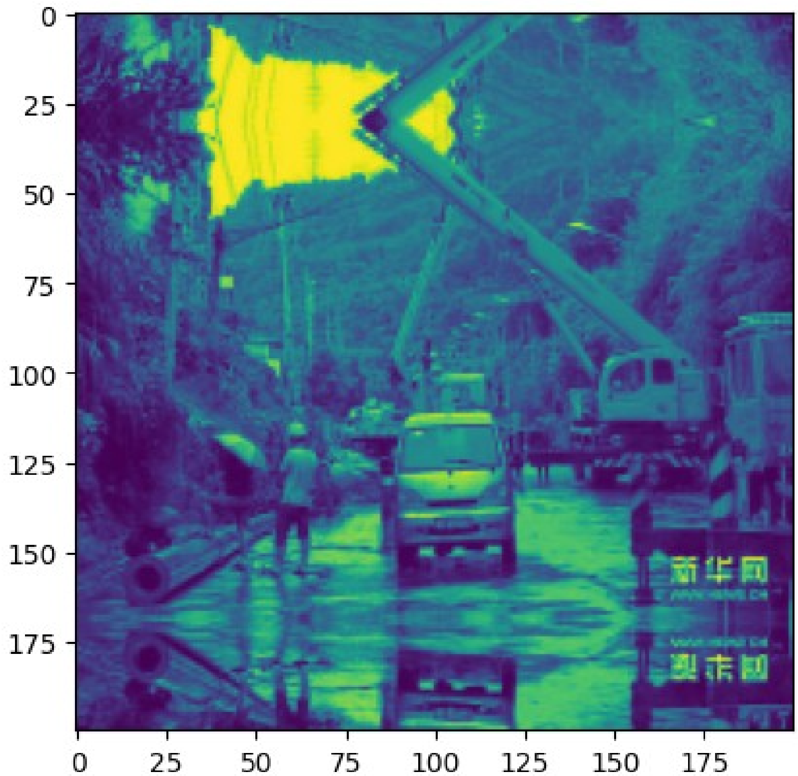
Converting images to grayscale

```
imagegs=np.array([cv2.cvtColor(image,cv2.COLOR_BGR2GRAY) for image in
images])
imagegs=imagegs.reshape(631,200,200,1)
imagegs.shape
```

```
(631, 200, 200, 1)
```

```
plt.imshow(imagegs[0])
```

```
<matplotlib.image.AxesImage at 0x7c9699903950>
```



Splitting the dataset

```
x=imagegs
y=labels['Label']

x_train,x_temp,y_train,y_temp=train_test_split(x,y,test_size=0.4,strat
ify=y,random_state=812)

x_val,x_test,y_val,y_test=train_test_split(x_temp,y_temp,test_size=0.5
,stratify=y_temp,random_state=812)

x_val.shape,x_test.shape,x_train.shape

((126, 200, 200, 1), (127, 200, 200, 1), (378, 200, 200, 1))

y_val.value_counts(),y_test.value_counts(),y_train.value_counts()

(Label
 0    64
 1    62
 Name: count, dtype: int64,
 Label
 0    64
 1    63
 Name: count, dtype: int64,
 Label
 0    192
```

```
1    186
Name: count, dtype: int64)
```

Data Normalization

```
x_train_normalized = x_train.astype('float32')/255.0
x_val_normalized = x_val.astype('float32')/255.0
x_test_normalized = x_test.astype('float32')/255.0

y_train[0]
```

```
np.int64(1)
```

Data splitting & Normalisation for color data:

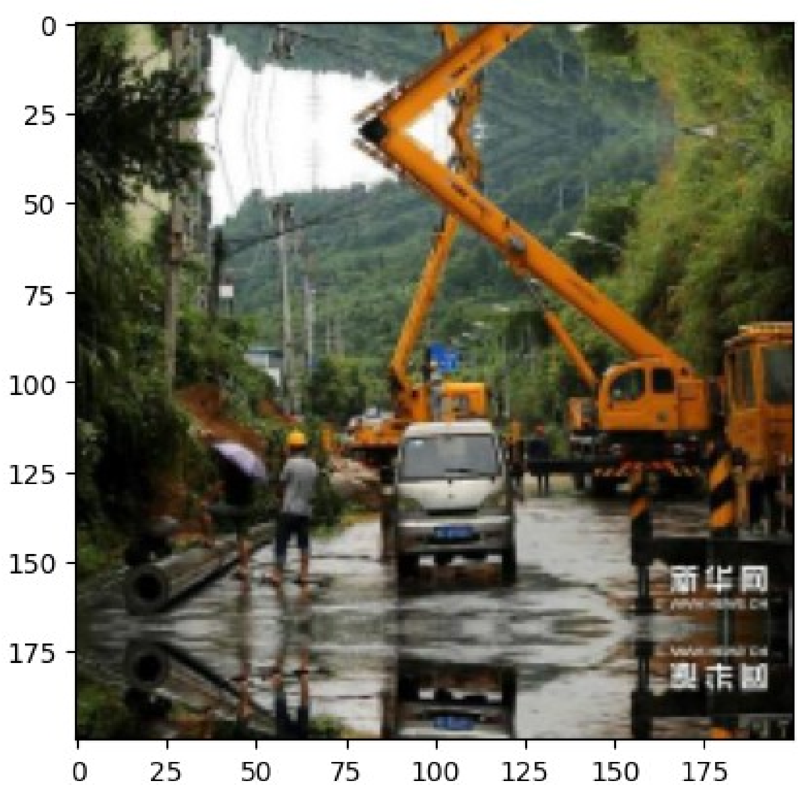
- Since VGG16 model works with three colour channels.

```
imagcol=np.array([cv2.cvtColor(image,cv2.COLOR_BGR2RGB) for image in images])
imagcol.shape
```

```
(631, 200, 200, 3)
```

```
plt.imshow(imagcol[0])
```

```
<matplotlib.image.AxesImage at 0x7c96999df110>
```



```
xc=imagcol
yc=labels['Label']

xc_train,xc_temp,yc_train,yc_temp=train_test_split(xc,yc,test_size=0.4
,stratify=y,random_state=812)

xc_val,xc_test,yc_val,yc_test=train_test_split(xc_temp,yc_temp,test_si
ze=0.5,stratify=yc_temp,random_state=812)

xc_val.shape,xc_test.shape,xc_train.shape
```

```
((126, 200, 200, 3), (127, 200, 200, 3), (378, 200, 200, 3))
```

```
yc_val.value_counts(),yc_test.value_counts(),yc_train.value_counts()
```

```
(Label
 0    64
 1    62
 Name: count, dtype: int64,
 Label
 0    64
 1    63
 Name: count, dtype: int64,
 Label
 0    192
 1    186
 Name: count, dtype: int64)
```

```
xc_train_normalized = xc_train.astype('float32')/255.0
xc_val_normalized = xc_val.astype('float32')/255.0
xc_test_normalized = xc_test.astype('float32')/255.0
```

Model Building

##Model Evaluation Criterion

The primary metric for evaluating the model will be **Recall** for the "Without Helmet" category.

Reason for Choosing Recall:

- **Critical Safety Focus:** Missing instances of workers not wearing helmets can result in severe safety risks. Prioritizing recall ensures that most non-compliant cases are detected, minimizing the risk of the workers getting injured.
- **Error Consequences:** False negatives (failing to detect a worker without a helmet) are more critical than false positives (incorrectly flagging a compliant worker), as undetected non-compliance could lead to accidents which could be even life threatening and also lead to legal issues.

Secondary Metrics:

- **Precision for "Without Helmet":** To avoid excessive false positives that could undermine system reliability.
- **F1 Score:** To balance recall and precision, ensuring an overall robust performance.
- **Accuracy:** For general performance evaluation.

Utility Functions

```
# defining a function to compute different metrics to check
performance of a classification model built using statsmodels
def model_performance_classification(model, predictors, target):
    """
    Function to compute different metrics to check classification
model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # checking which probabilities are greater than threshold
    pred = model.predict(predictors).reshape(-1)>0.5

    target = target.to_numpy().reshape(-1)


    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred, average='weighted')  # to
compute Recall
    precision = precision_score(target, pred, average='weighted')  #
to compute Precision
    f1 = f1_score(target, pred, average='weighted')  # to compute F1-
score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame({"Accuracy": acc, "Recall": recall,
"Precision": precision, "F1 Score": f1,},index=[0],)

    return df_perf

def plot_confusion_matrix(model,predictors,target,ml=False):
    """
    Function to plot the confusion matrix

    model: classifier
    predictors: independent variables
    target: dependent variable
    ml: To specify if the model used is an sklearn ML model or not
(True means ML model)
    """
```

```
    # checking which probabilities are greater than threshold
    pred = model.predict(predictors).reshape(-1)>0.5

    target = target.to_numpy().reshape(-1)

    # Plotting the Confusion Matrix using confusion matrix() function
which is also predefined tensorflow module
    confusion_matrix = tf.math.confusion_matrix(target,pred)
    f, ax = plt.subplots(figsize=(10, 8))
    sns.heatmap(
        confusion_matrix,
        annot=True,
        linewidths=.4,
        fmt="d",
        square=True,
        xticklabels=["Without Helmet", "With Helmet"],
        yticklabels=["Without Helmet", "With Helmet"],
        ax=ax
    )
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

# model for plotting loss
def plot_history(history):
  plt.figure(figsize=(15,5))
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Validation'], loc='upper right')
  plt.show()
```

##Model 1: Simple Convolutional Neural Network (CNN)

```
cls()

m1=Sequential()
m1.add(Conv2D(64,
(3,3),activation='relu',input_shape=(200,200,1),padding = 'same'))
m1.add(BatchNormalization())
m1.add(MaxPooling2D(2,2))

m1.add(Conv2D(32,(3,3),activation='relu',padding='same'))
m1.add(BatchNormalization())
m1.add(MaxPooling2D(2,2))
```

```
m1.add(Conv2D(32,(3,3),activation='relu',padding='same'))
m1.add(MaxPooling2D(2,2))
m1.add(Flatten())
m1.add(Dense(128,activation='relu'))
m1.add(BatchNormalization())
m1.add(Dropout(0.5))
m1.add(Dense(64,activation='relu'))
m1.add(BatchNormalization())
m1.add(Dropout(0.5))
m1.add(Dense(1,activation='sigmoid'))

m1.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d (Conv2D)                 │ (None, 200, 200, 64)   │
640 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization             │ (None, 200, 200, 64)   │
256 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 100, 100, 64)   │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_1 (Conv2D)               │ (None, 100, 100, 32)   │
18,464 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_1           │ (None, 100, 100, 32)   │
128 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 50, 50, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_2 (Conv2D)               │ (None, 50, 50, 32)     │
```

```
9,248 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 25, 25, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ flatten (Flatten)               │ (None, 20000)          │
0 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ dense (Dense)                   │ (None, 128)            │
2,560,128 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ batch_normalization_2           │ (None, 128)            │
512 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ dropout (Dropout)               │ (None, 128)            │
0 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ dense_1 (Dense)                 │ (None, 64)             │
8,256 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ batch_normalization_3           │ (None, 64)             │
256 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ dropout_1 (Dropout)             │ (None, 64)             │
0 │
├─────────────────────────────────┼────────────────────────┼───────
──────┤
│ dense_2 (Dense)                 │ (None, 1)              │
65 │
└─────────────────────────────────┴────────────────────────┴───────
──────┘

 Total params: 2,597,953 (9.91 MB)

 Trainable params: 2,597,377 (9.91 MB)

 Non-trainable params: 576 (2.25 KB)
```

```
opti1=keras.optimizers.Adam(learning_rate=0.001)
m1.compile(optimizer=opti1,loss='binary_crossentropy',metrics=['accuracy'])

h1=m1.fit(x_train_normalized,y_train,epochs=32,validation_data=(x_val_normalized,y_val),batch_size=32)
```

```
Epoch 1/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 18s 671ms/step - accuracy: 0.7948 - loss: 0.4489 - val_accuracy: 0.5317 - val_loss: 0.6626
Epoch 2/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 54ms/step - accuracy: 0.9554 - loss: 0.1190 - val_accuracy: 0.5079 - val_loss: 0.9911
Epoch 3/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9581 - loss: 0.0884 - val_accuracy: 0.5079 - val_loss: 1.3227
Epoch 4/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9749 - loss: 0.0517 - val_accuracy: 0.5079 - val_loss: 1.5227
Epoch 5/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9913 - loss: 0.0314 - val_accuracy: 0.5079 - val_loss: 1.8202
Epoch 6/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9923 - loss: 0.0288 - val_accuracy: 0.5079 - val_loss: 1.8100
Epoch 7/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0158 - val_accuracy: 0.5079 - val_loss: 2.1666
Epoch 8/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0165 - val_accuracy: 0.5079 - val_loss: 2.4214
Epoch 9/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0114 - val_accuracy: 0.5079 - val_loss: 2.6409
Epoch 10/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0103 - val_accuracy: 0.5079 - val_loss: 2.7557
Epoch 11/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0087 - val_accuracy: 0.5079 - val_loss: 2.9510
Epoch 12/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss: 0.0091 - val_accuracy: 0.5079 - val_loss: 2.9800
Epoch 13/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0054 - val_accuracy: 0.5079 - val_loss: 3.0975
Epoch 14/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 1.0000 - loss: 0.0069 - val_accuracy: 0.5079 - val_loss: 3.2048
Epoch 15/32
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 1.0000 - loss:
0.0053 - val_accuracy: 0.5079 - val_loss: 3.2756
Epoch 16/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step - accuracy: 1.0000 - loss:
0.0049 - val_accuracy: 0.5079 - val_loss: 3.2095
Epoch 17/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 1.0000 - loss:
0.0049 - val_accuracy: 0.5079 - val_loss: 3.3308
Epoch 18/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9967 - loss:
0.0062 - val_accuracy: 0.5079 - val_loss: 3.2886
Epoch 19/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss:
0.0055 - val_accuracy: 0.5079 - val_loss: 3.0630
Epoch 20/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0040 - val_accuracy: 0.5079 - val_loss: 3.0490
Epoch 21/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0027 - val_accuracy: 0.5079 - val_loss: 3.2925
Epoch 22/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 1.0000 - loss:
0.0029 - val_accuracy: 0.5079 - val_loss: 3.3412
Epoch 23/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.9891 - loss:
0.0125 - val_accuracy: 0.5079 - val_loss: 3.5700
Epoch 24/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss:
0.0021 - val_accuracy: 0.5079 - val_loss: 3.5227
Epoch 25/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0029 - val_accuracy: 0.5079 - val_loss: 3.3244
Epoch 26/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0030 - val_accuracy: 0.5079 - val_loss: 3.0013
Epoch 27/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 1.0000 - loss:
0.0018 - val_accuracy: 0.5159 - val_loss: 2.8879
Epoch 28/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0023 - val_accuracy: 0.5238 - val_loss: 2.9027
Epoch 29/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0021 - val_accuracy: 0.5317 - val_loss: 2.7379
Epoch 30/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
0.0020 - val_accuracy: 0.5635 - val_loss: 2.3283
Epoch 31/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss:
```

```
0.0016 - val_accuracy: 0.6032 - val_loss: 1.9654
Epoch 32/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 1.0000 - loss:
0.0013 - val_accuracy: 0.6111 - val_loss: 1.8073
```

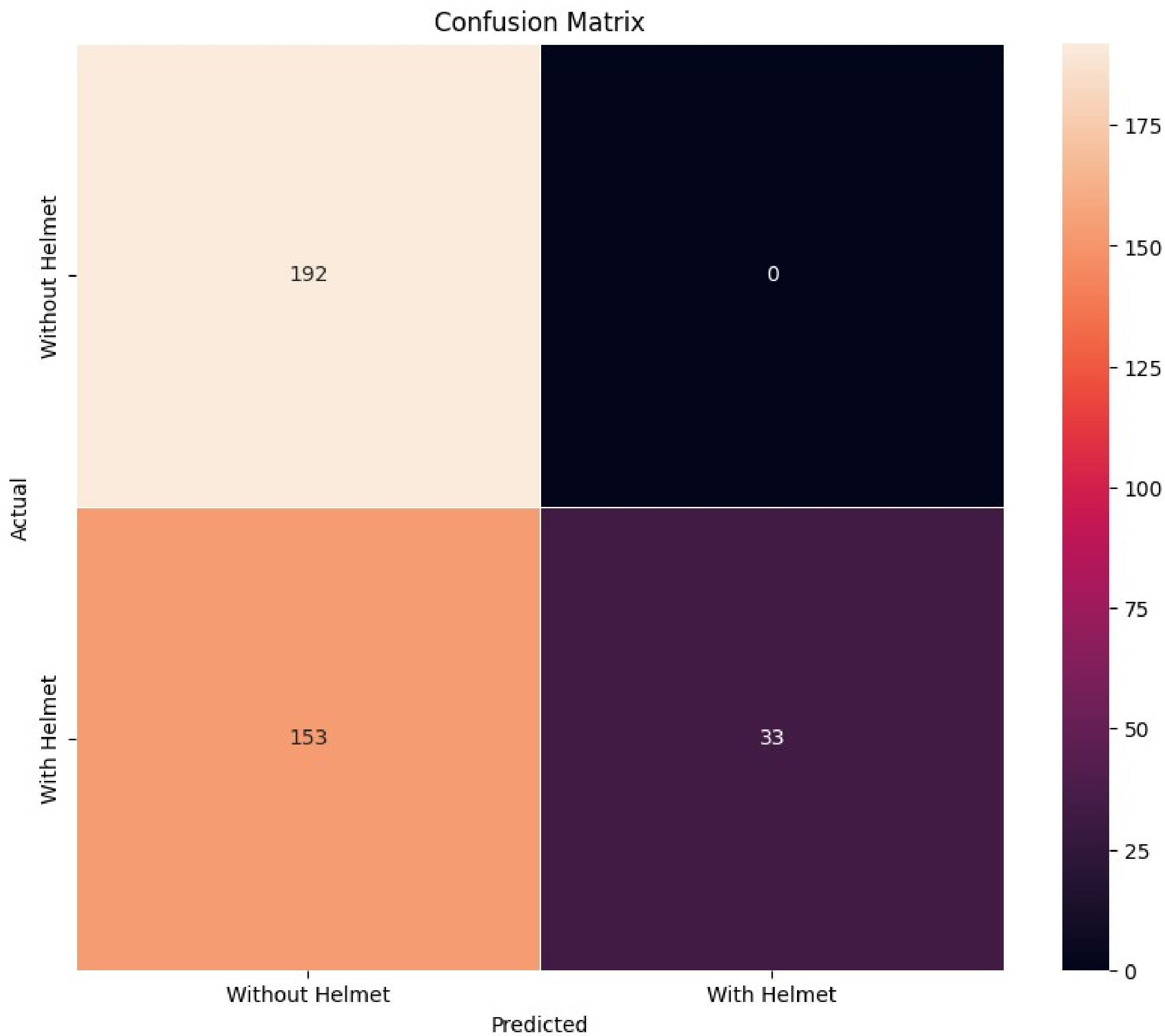
```
m1tr=model_performance_classification(m1,x_train_normalized,y_train)
m1tr
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 111ms/step
```

{"summary":"{\n \"name\": \"m1tr\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.5952380952380952,\n   \"max\": 0.5952380952380952,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.5952380952380952\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.5952380952380952,\n   \"max\": 0.5952380952380952,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.5952380952380952\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.7747412008281573,\n   \"max\": 0.7747412008281573,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.7747412008281573\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.5115102748836582,\n   \"max\": 0.5115102748836582,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.5115102748836582\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m1tr"}

```
plot_confusion_matrix(m1,x_train_normalized,y_train)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```




```
m1vl=model_performance_classification(m1,x_val_normalized,y_val)
m1vl
```

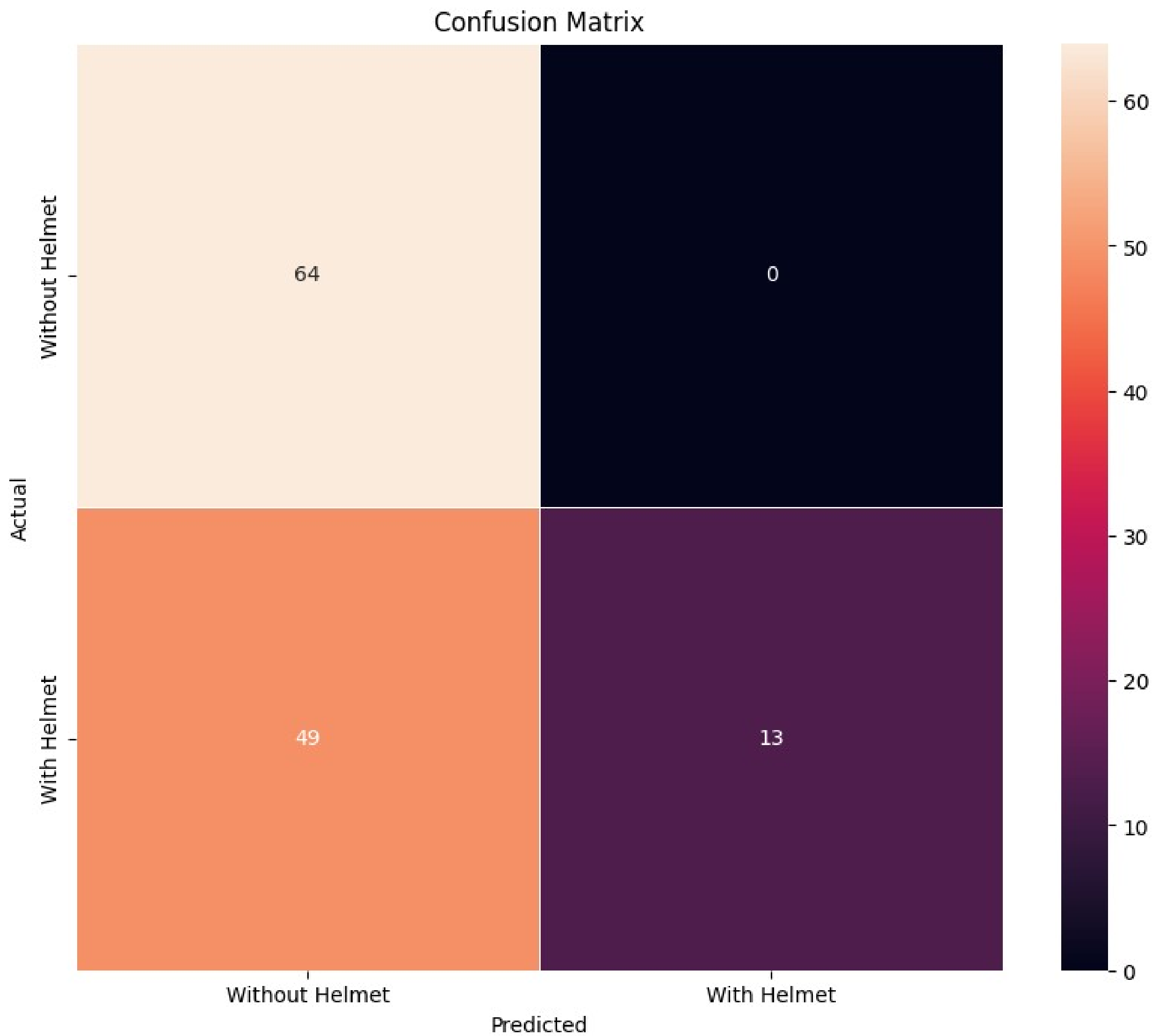
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step

{"summary":"{\n \"name\": \"m1vl\",\n \"rows\": 1,\n \"fields\": [\
n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
0.6111111111111112,\n   \"max\": 0.6111111111111112,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
0.6111111111111112\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": null,\n   \"min\": 0.6111111111111112,\n   \"max\":
0.6111111111111112,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   0.6111111111111112\n   ],\n

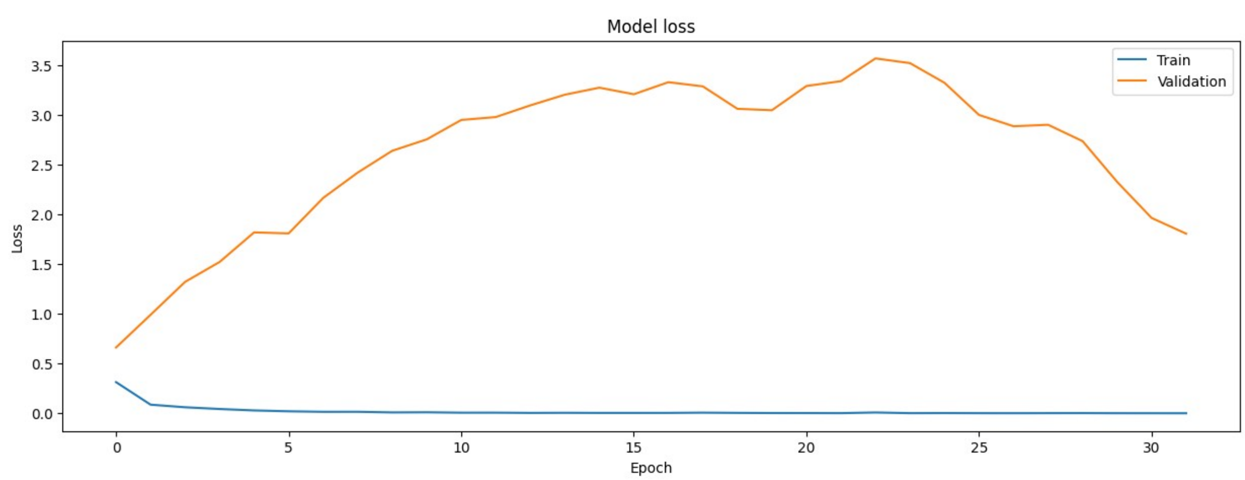
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Precision\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 0.7797443461160275,\n   \"max\":
0.7797443461160275,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   0.7797443461160275\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": null,\n
\"min\": 0.5379033270558695,\n   \"max\": 0.5379033270558695,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
0.5379033270558695\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"m1vl"}

```
plot_confusion_matrix(m1,x_val_normalized,y_val)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```




```
plot_history(h1)
```



Vizualizing the predictions

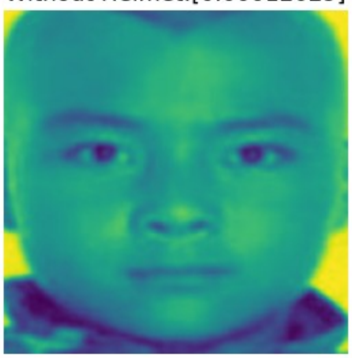
```
pre = m1.predict(x_train_normalized)
f,ax=plt.subplots(4,2,figsize=(15,15))
for i,j in zip(ax.flatten(),range(8)):
  i.imshow(x_train[j])
  i.axis('off')
  if pre[j]>0.5:
    i.set_title(f'With Helmet:{pre[j]}')
  else:
    i.set_title(f'Without Helmet:{pre[j]}')
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```

Without Helmet:[0.01579482]

Without Helmet:[0.00044471]

Without Helmet:[0.00012029]



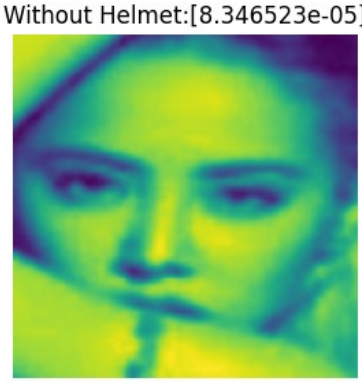
Without Helmet:[0.00234477]

Without Helmet:[0.00321866]

Without Helmet:[0.04541063]

Without Helmet:[0.00116103]

Without Helmet:[8.346523e-05]



Observations on model 1:

- As we observe the metrics evaluations and confusion matrix we can clearly see that recall is very low and metrics are not upto mark.
- The FN are fairly high with low recalls which means the model is not able to distinguish peolple `With Helmets` as FP are zero.
- The model is not able to classify even with grayscale image which has reduced dimensions.
- Heavy validation losses also shows that the model struggles to distinguish between people `With Helmets` and those Without Helmets, as this indicates the model had not learned the distinguishing features effectively.

Model 2: (VGG-16 (Base))

```
vgg16_m=VGG16(weights='imagenet',include_top=False,input_shape=(200,200,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
for layers in vgg16_m.layers:
  layers.trainable=False
```

```
for layers in vgg16_m.layers:
  print(layers.name,layers.trainable)
```

```
input_layer_1 False
block1_conv1 False
block1_conv2 False
block1_pool False
block2_conv1 False
block2_conv2 False
block2_pool False
block3_conv1 False
block3_conv2 False
block3_conv3 False
block3_pool False
block4_conv1 False
block4_conv2 False
block4_conv3 False
block4_pool False
block5_conv1 False
block5_conv2 False
block5_conv3 False
block5_pool False
```

```
vgg16_m.summary()
```

```
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 200, 200, 3)	0
block1_conv1 (Conv2D)	(None, 200, 200, 64)	1,792
block1_conv2 (Conv2D)	(None, 200, 200, 64)	36,928
block1_pool (MaxPooling2D)	(None, 100, 100, 64)	0
block2_conv1 (Conv2D)	(None, 100, 100, 128)	73,856
block2_conv2 (Conv2D)	(None, 100, 100, 128)	147,584
block2_pool (MaxPooling2D)	(None, 50, 50, 128)	0
block3_conv1 (Conv2D)	(None, 50, 50, 256)	295,168
block3_conv2 (Conv2D)	(None, 50, 50, 256)	590,080
block3_conv3 (Conv2D)	(None, 50, 50, 256)	590,080
block3_pool (MaxPooling2D)	(None, 25, 25, 256)	0

```
│ block4_conv1 (Conv2D)             │ (None, 25, 25, 512)       │
1,180,160 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block4_conv2 (Conv2D)             │ (None, 25, 25, 512)       │
2,359,808 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block4_conv3 (Conv2D)             │ (None, 25, 25, 512)       │
2,359,808 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block4_pool (MaxPooling2D)        │ (None, 12, 12, 512)       │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block5_conv1 (Conv2D)             │ (None, 12, 12, 512)       │
2,359,808 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block5_conv2 (Conv2D)             │ (None, 12, 12, 512)       │
2,359,808 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block5_conv3 (Conv2D)             │ (None, 12, 12, 512)       │
2,359,808 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ block5_pool (MaxPooling2D)        │ (None, 6, 6, 512)         │
0 │
└───────────────────────────────────┴───────────────────────────┴──────────
───────┘

 Total params: 14,714,688 (56.13 MB)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 14,714,688 (56.13 MB)

cls()

m2=Sequential()
m2.add(vgg16_m)
m2.add(Flatten())

m2.add(Dense(1, activation='sigmoid'))

opti2=keras.optimizers.Adam(learning_rate=0.001)
m2.compile(optimizer=opti2,loss='binary_crossentropy',metrics=['accura
cy'])
```

```
h2=m2.fit(xc_train_normalized,yc_train,epochs=32,validation_data=(xc_v
al_normalized,yc_val),batch_size=32)
```

```
Epoch 1/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 43s 3s/step - accuracy: 0.7551 - loss:
0.4722 - val_accuracy: 0.9841 - val_loss: 0.0369
Epoch 2/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 42s 211ms/step - accuracy: 1.0000 - loss:
0.0167 - val_accuracy: 0.9841 - val_loss: 0.0183
Epoch 3/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 212ms/step - accuracy: 1.0000 - loss:
0.0043 - val_accuracy: 1.0000 - val_loss: 0.0104
Epoch 4/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 273ms/step - accuracy: 1.0000 - loss:
0.0034 - val_accuracy: 1.0000 - val_loss: 0.0097
Epoch 5/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 276ms/step - accuracy: 1.0000 - loss:
0.0026 - val_accuracy: 1.0000 - val_loss: 0.0104
Epoch 6/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 275ms/step - accuracy: 1.0000 - loss:
0.0021 - val_accuracy: 1.0000 - val_loss: 0.0106
Epoch 7/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 220ms/step - accuracy: 1.0000 - loss:
0.0019 - val_accuracy: 1.0000 - val_loss: 0.0101
Epoch 8/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 222ms/step - accuracy: 1.0000 - loss:
0.0017 - val_accuracy: 1.0000 - val_loss: 0.0095
Epoch 9/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 278ms/step - accuracy: 1.0000 - loss:
0.0016 - val_accuracy: 1.0000 - val_loss: 0.0092
Epoch 10/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 282ms/step - accuracy: 1.0000 - loss:
0.0015 - val_accuracy: 1.0000 - val_loss: 0.0089
Epoch 11/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 225ms/step - accuracy: 1.0000 - loss:
0.0014 - val_accuracy: 1.0000 - val_loss: 0.0087
Epoch 12/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 226ms/step - accuracy: 1.0000 - loss:
0.0013 - val_accuracy: 1.0000 - val_loss: 0.0085
Epoch 13/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 223ms/step - accuracy: 1.0000 - loss:
0.0012 - val_accuracy: 1.0000 - val_loss: 0.0083
Epoch 14/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 278ms/step - accuracy: 1.0000 - loss:
0.0011 - val_accuracy: 1.0000 - val_loss: 0.0082
Epoch 15/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 223ms/step - accuracy: 1.0000 - loss:
0.0010 - val_accuracy: 1.0000 - val_loss: 0.0080
Epoch 16/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 221ms/step - accuracy: 1.0000 - loss:
```

```
9.4375e-04 - val_accuracy: 1.0000 - val_loss: 0.0078
Epoch 17/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 277ms/step - accuracy: 1.0000 - loss:
8.8498e-04 - val_accuracy: 1.0000 - val_loss: 0.0077
Epoch 18/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 219ms/step - accuracy: 1.0000 - loss:
8.3164e-04 - val_accuracy: 1.0000 - val_loss: 0.0075
Epoch 19/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 1.0000 - loss:
7.8314e-04 - val_accuracy: 1.0000 - val_loss: 0.0074
Epoch 20/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 216ms/step - accuracy: 1.0000 - loss:
7.3893e-04 - val_accuracy: 1.0000 - val_loss: 0.0073
Epoch 21/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 218ms/step - accuracy: 1.0000 - loss:
6.9851e-04 - val_accuracy: 1.0000 - val_loss: 0.0072
Epoch 22/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 214ms/step - accuracy: 1.0000 - loss:
6.6145e-04 - val_accuracy: 1.0000 - val_loss: 0.0070
Epoch 23/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 219ms/step - accuracy: 1.0000 - loss:
6.2740e-04 - val_accuracy: 1.0000 - val_loss: 0.0069
Epoch 24/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 213ms/step - accuracy: 1.0000 - loss:
5.9603e-04 - val_accuracy: 1.0000 - val_loss: 0.0068
Epoch 25/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 1.0000 - loss:
5.6707e-04 - val_accuracy: 1.0000 - val_loss: 0.0068
Epoch 26/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 1.0000 - loss:
5.4026e-04 - val_accuracy: 1.0000 - val_loss: 0.0067
Epoch 27/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 275ms/step - accuracy: 1.0000 - loss:
5.1540e-04 - val_accuracy: 1.0000 - val_loss: 0.0066
Epoch 28/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 275ms/step - accuracy: 1.0000 - loss:
4.9231e-04 - val_accuracy: 1.0000 - val_loss: 0.0065
Epoch 29/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 279ms/step - accuracy: 1.0000 - loss:
4.7080e-04 - val_accuracy: 1.0000 - val_loss: 0.0064
Epoch 30/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 219ms/step - accuracy: 1.0000 - loss:
4.5075e-04 - val_accuracy: 1.0000 - val_loss: 0.0063
Epoch 31/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 219ms/step - accuracy: 1.0000 - loss:
4.3202e-04 - val_accuracy: 1.0000 - val_loss: 0.0063
Epoch 32/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 221ms/step - accuracy: 1.0000 - loss:
4.1448e-04 - val_accuracy: 1.0000 - val_loss: 0.0062
```

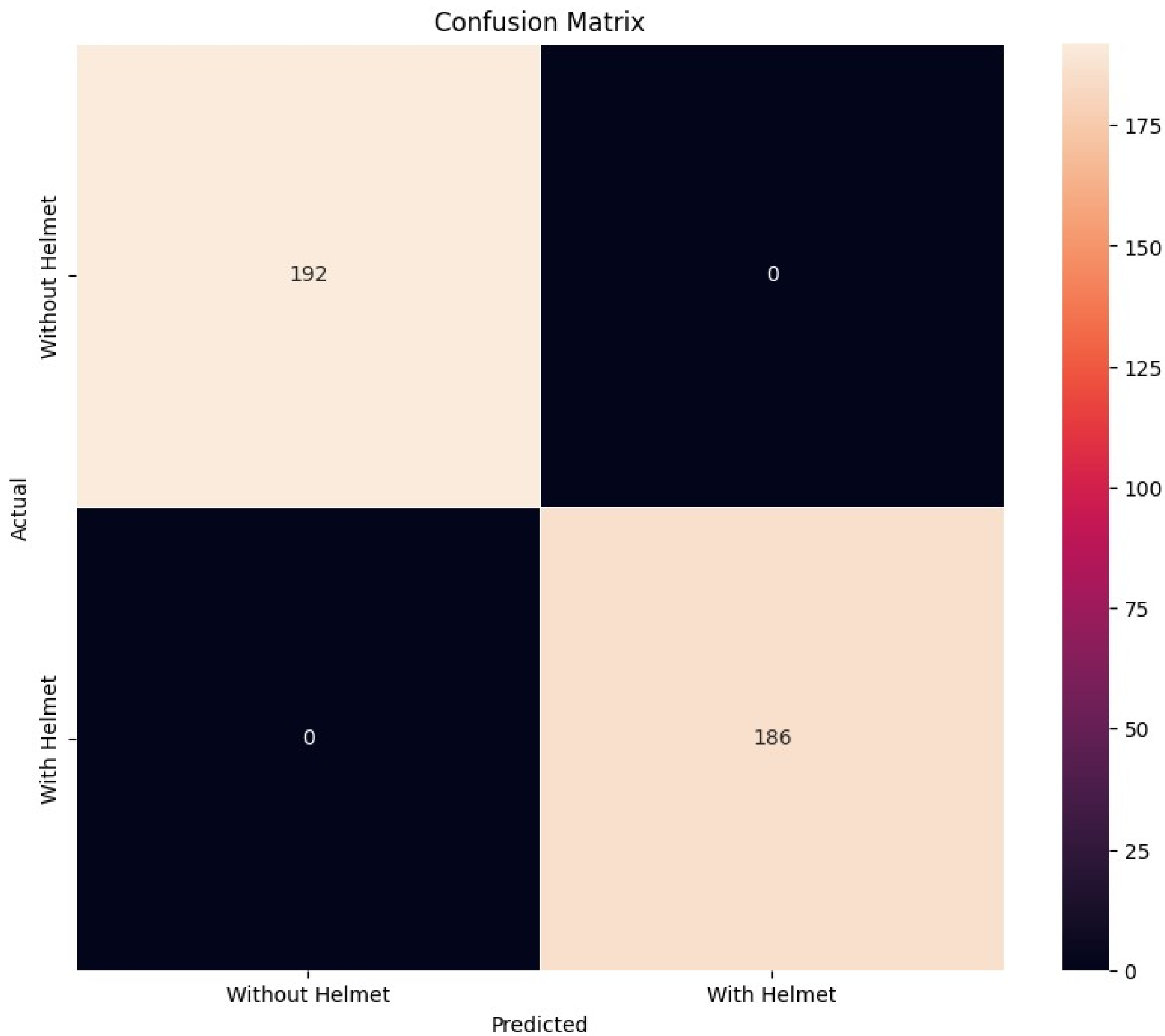
```
m2tr=model_performance_classification(m2,xc_train_normalized,yc_train)
m2tr
```

12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 204ms/step

{"summary":"{\n \"name\": \"m2tr\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m2tr"}

```
plot_confusion_matrix(m2,xc_train_normalized,yc_train)
```

12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 161ms/step



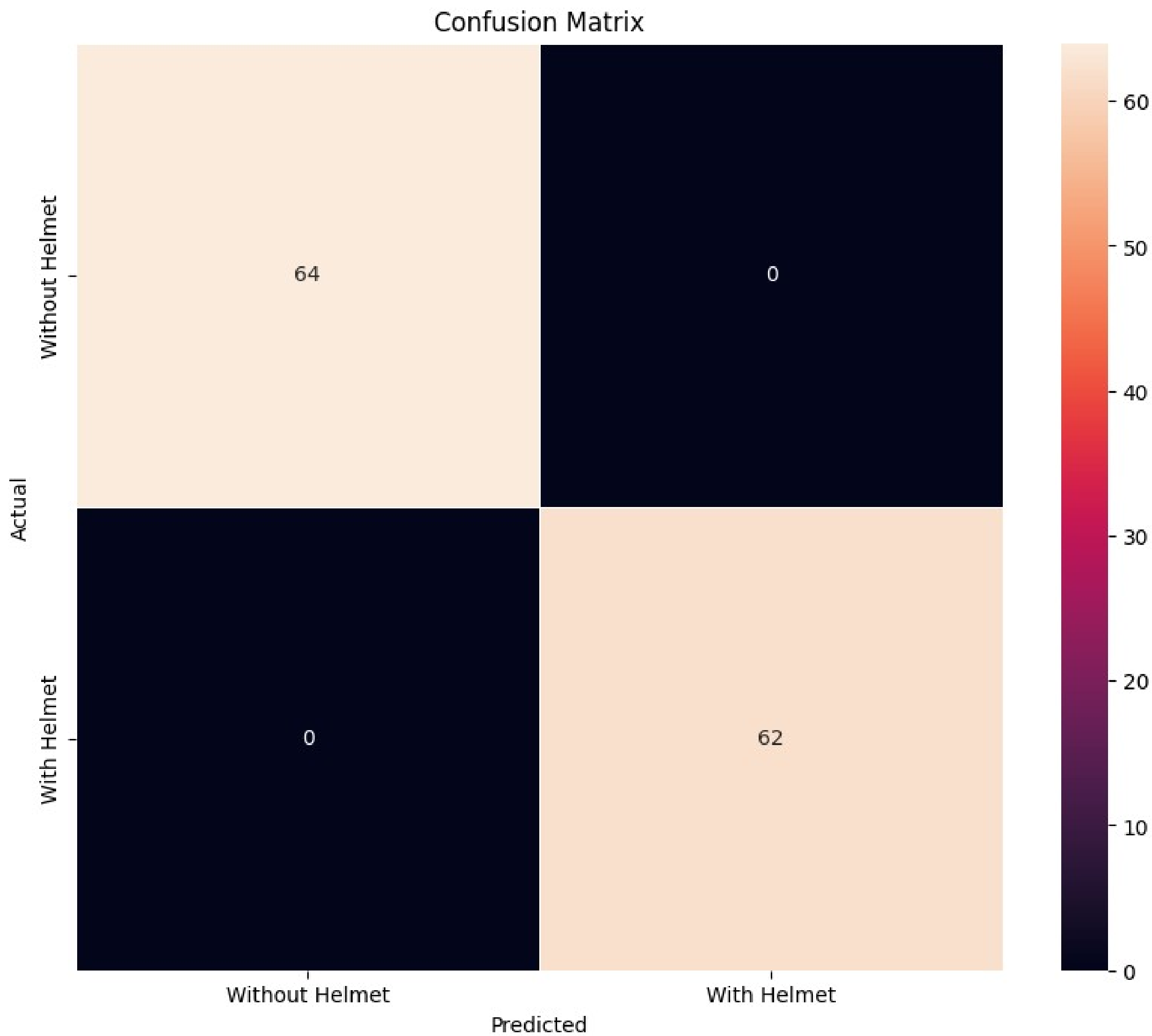

```
m2vl=model_performance_classification(m2,xc_val_normalized,yc_val)
m2vl

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 267ms/step
```

{"summary":"{\n \"name\": \"m2vl\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\":

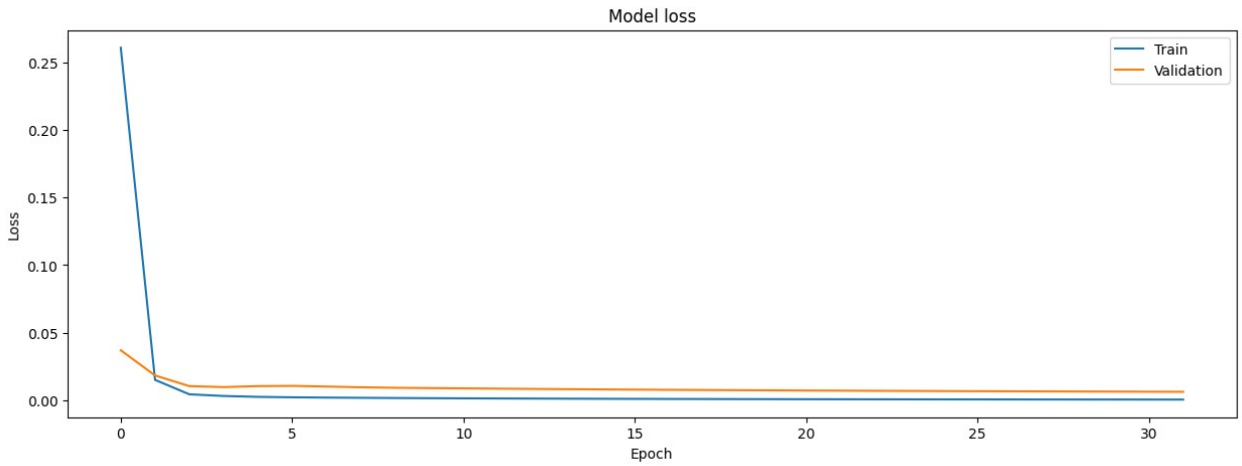
\"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m2vl"}

```
plot_confusion_matrix(m2,xc_val_normalized,yc_val)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 160ms/step
```



```
plot_history(h2)
```



Visualizing the prediction:

```
prev=m2.predict(xc_train_normalized)
f,ax=plt.subplots(4,2,figsize=(15,15))
for i,j in zip(ax.flatten(),range(8)):
  i.imshow(xc_train_normalized[j])
  i.axis('off')
  if prev[j]>0.5:
    i.set_title(f'With Helmet:{prev[j]}')
  else:
    i.set_title(f'Without Helmet:{prev[j]}')
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 162ms/step
```

With Helmet:[0.99999464]

With Helmet:[0.997755]

Without Helmet:[2.100612e-05]

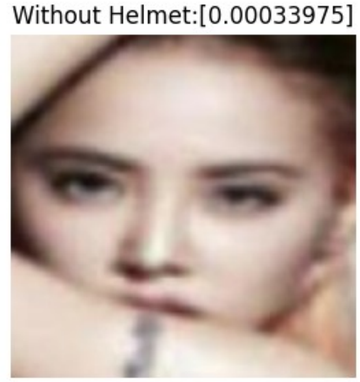
With Helmet:[0.99984515]

With Helmet:[0.9999515]

With Helmet:[0.99999917]

With Helmet:[0.99998057]

Without Helmet:[0.00033975]



Observations on model 2:

- As we see the metrics(recall) and confusion matrix the score are too good which can also be overfitting.
- while The validation loss is slightly higher than training it is within limits as the curves are smooth and uniform.
- But, the Prediction visualisations are also correct maybe due to the data being small & clean.

Model 3: (VGG-16 (Base + FFNN))

```
cls()

m3=Sequential()
m3.add(vgg16_m)
m3.add(Flatten())
m3.add(Dense(32, activation='relu'))
m3.add(BatchNormalization())
m3.add(Dropout(0.5))
m3.add(Dense(32, activation='relu'))
m3.add(BatchNormalization())
m3.add(Dropout(0.5))
m3.add(Dense(1, activation='sigmoid'))



opti=keras.optimizers.Adam(learning_rate=0.001)
m3.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accurac
y'])

h3=m3.fit(xc_train_normalized,yc_train,epochs=32,validation_data=(xc_v
al_normalized,yc_val),batch_size=32)

Epoch 1/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 11s 605ms/step - accuracy: 0.7833 - loss:
0.5235 - val_accuracy: 0.7222 - val_loss: 0.4812
Epoch 2/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 276ms/step - accuracy: 0.9462 - loss:
0.1645 - val_accuracy: 0.8492 - val_loss: 0.2329
Epoch 3/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 276ms/step - accuracy: 0.9771 - loss:
0.1006 - val_accuracy: 0.9524 - val_loss: 0.1292
Epoch 4/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 279ms/step - accuracy: 0.9630 - loss:
0.0867 - val_accuracy: 1.0000 - val_loss: 0.0666
Epoch 5/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 276ms/step - accuracy: 0.9935 - loss:
0.0552 - val_accuracy: 1.0000 - val_loss: 0.0408
Epoch 6/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 221ms/step - accuracy: 0.9755 - loss:
0.0696 - val_accuracy: 1.0000 - val_loss: 0.0267
```

```
Epoch 7/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 220ms/step - accuracy: 0.9967 - loss:
0.0378 - val_accuracy: 1.0000 - val_loss: 0.0167
Epoch 8/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 275ms/step - accuracy: 0.9996 - loss:
0.0352 - val_accuracy: 1.0000 - val_loss: 0.0132
Epoch 9/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 281ms/step - accuracy: 0.9917 - loss:
0.0371 - val_accuracy: 1.0000 - val_loss: 0.0098
Epoch 10/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 220ms/step - accuracy: 0.9991 - loss:
0.0277 - val_accuracy: 1.0000 - val_loss: 0.0083
Epoch 11/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 219ms/step - accuracy: 0.9909 - loss:
0.0320 - val_accuracy: 1.0000 - val_loss: 0.0082
Epoch 12/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 224ms/step - accuracy: 1.0000 - loss:
0.0246 - val_accuracy: 0.9921 - val_loss: 0.0092
Epoch 13/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 222ms/step - accuracy: 1.0000 - loss:
0.0171 - val_accuracy: 0.9921 - val_loss: 0.0093
Epoch 14/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 220ms/step - accuracy: 1.0000 - loss:
0.0183 - val_accuracy: 0.9921 - val_loss: 0.0091
Epoch 15/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 223ms/step - accuracy: 1.0000 - loss:
0.0196 - val_accuracy: 0.9921 - val_loss: 0.0095
Epoch 16/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 221ms/step - accuracy: 0.9923 - loss:
0.0234 - val_accuracy: 0.9921 - val_loss: 0.0097
Epoch 17/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 278ms/step - accuracy: 1.0000 - loss:
0.0119 - val_accuracy: 0.9921 - val_loss: 0.0115
Epoch 18/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 222ms/step - accuracy: 1.0000 - loss:
0.0144 - val_accuracy: 0.9921 - val_loss: 0.0123
Epoch 19/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 225ms/step - accuracy: 1.0000 - loss:
0.0123 - val_accuracy: 0.9921 - val_loss: 0.0120
Epoch 20/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 220ms/step - accuracy: 1.0000 - loss:
0.0087 - val_accuracy: 0.9921 - val_loss: 0.0112
Epoch 21/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 278ms/step - accuracy: 1.0000 - loss:
0.0079 - val_accuracy: 0.9921 - val_loss: 0.0112
Epoch 22/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 278ms/step - accuracy: 0.9989 - loss:
0.0151 - val_accuracy: 0.9921 - val_loss: 0.0109
Epoch 23/32
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 218ms/step - accuracy: 1.0000 - loss: 0.0099 - val_accuracy: 0.9921 - val_loss: 0.0117
Epoch 24/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 220ms/step - accuracy: 1.0000 - loss: 0.0083 - val_accuracy: 0.9921 - val_loss: 0.0128
Epoch 25/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 221ms/step - accuracy: 1.0000 - loss: 0.0087 - val_accuracy: 0.9921 - val_loss: 0.0130
Epoch 26/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 218ms/step - accuracy: 1.0000 - loss: 0.0093 - val_accuracy: 0.9921 - val_loss: 0.0129
Epoch 27/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 220ms/step - accuracy: 0.9994 - loss: 0.0077 - val_accuracy: 0.9921 - val_loss: 0.0126
Epoch 28/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 279ms/step - accuracy: 0.9994 - loss: 0.0110 - val_accuracy: 0.9921 - val_loss: 0.0138
Epoch 29/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 219ms/step - accuracy: 1.0000 - loss: 0.0115 - val_accuracy: 0.9921 - val_loss: 0.0141
Epoch 30/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 218ms/step - accuracy: 1.0000 - loss: 0.0074 - val_accuracy: 0.9921 - val_loss: 0.0157
Epoch 31/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 220ms/step - accuracy: 1.0000 - loss: 0.0142 - val_accuracy: 0.9921 - val_loss: 0.0175
Epoch 32/32
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 219ms/step - accuracy: 1.0000 - loss: 0.0124 - val_accuracy: 0.9921 - val_loss: 0.0175
```

```
m3tr=model_performance_classification(m3,xc_train_normalized,yc_train)
m3tr
```

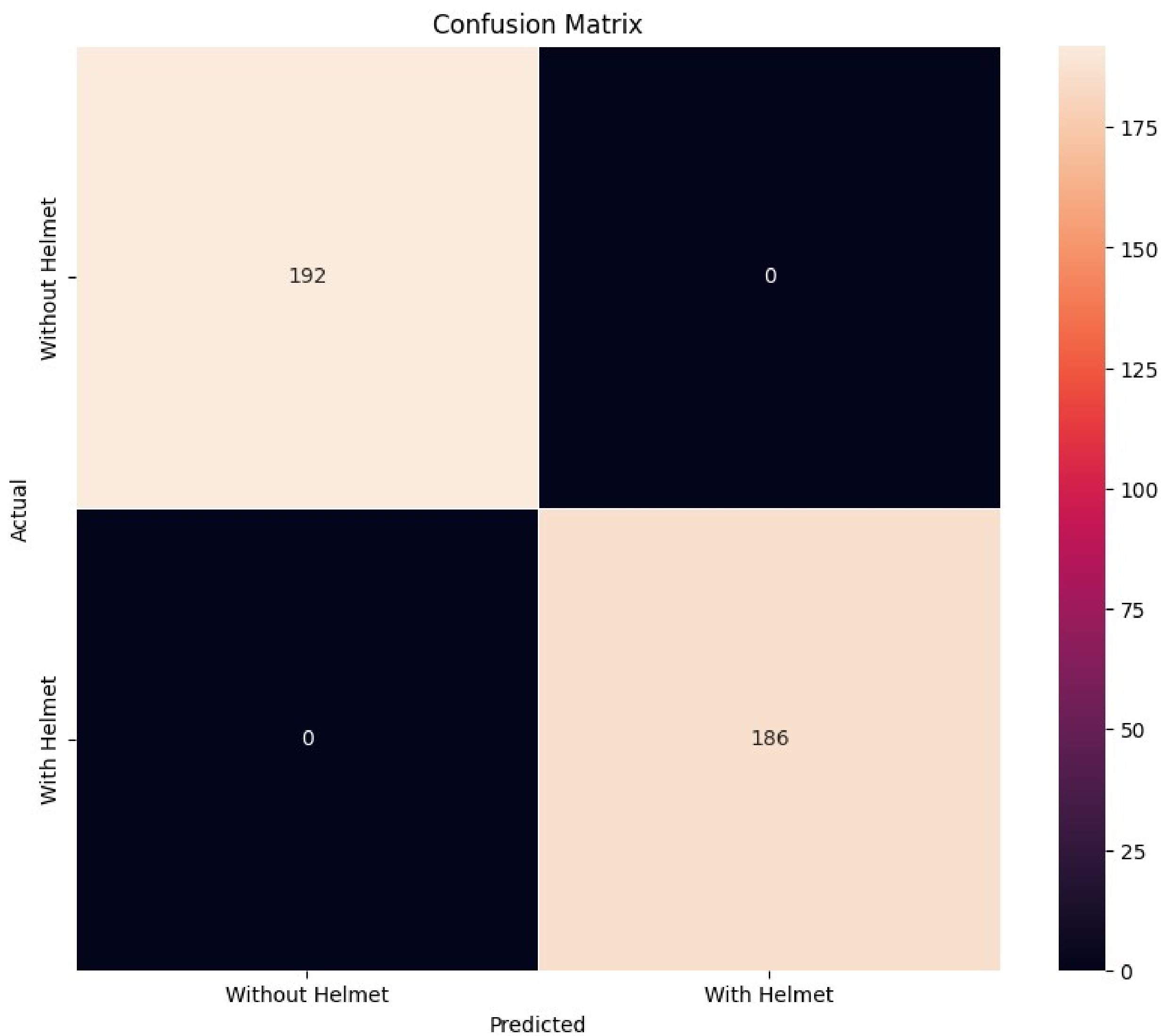
```
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 213ms/step
```

{"summary":"{\n \"name\": \"m3tr\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m3tr"}

```
plot_confusion_matrix(m3,xc_train_normalized,yc_train)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 157ms/step
```



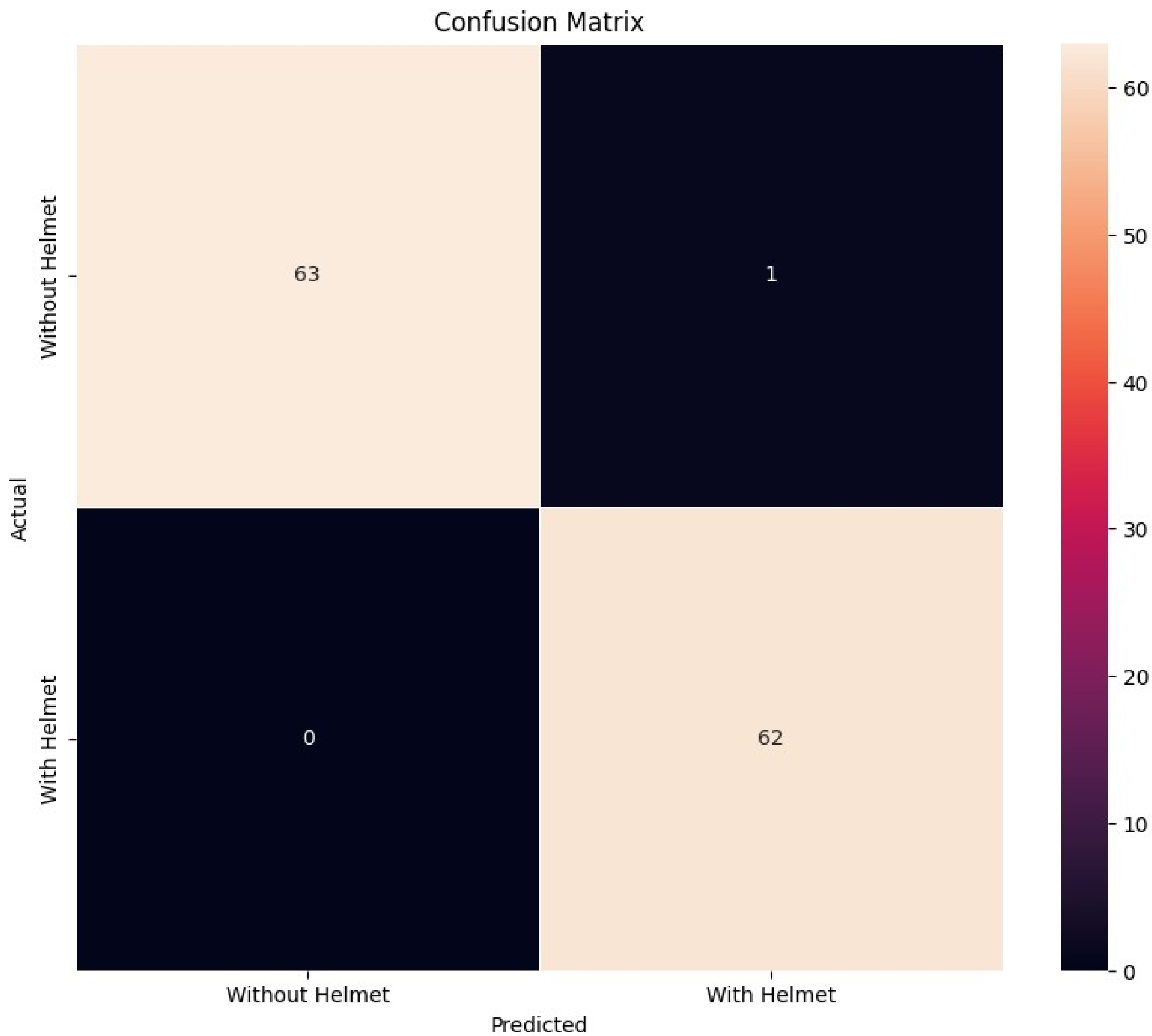

```
m3vl=model_performance_classification(m3,xc_val_normalized,yc_val)
m3vl
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 346ms/step
```

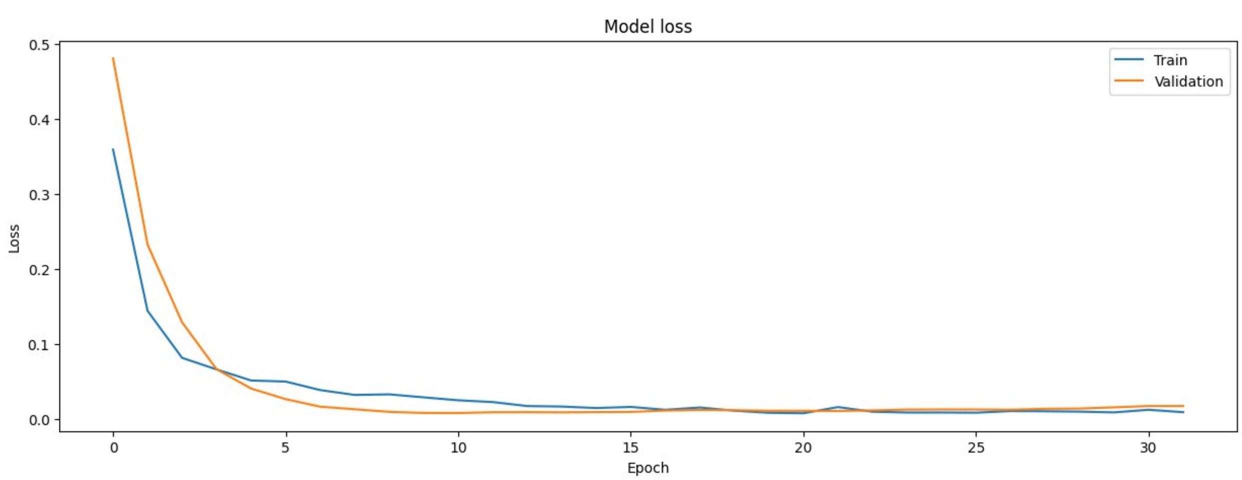
{"summary":"{\n \"name\": \"m3vl\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9920634920634921,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9920634920634921,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9921894683799446,\n   \"max\": 0.9921894683799446,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9921894683799446\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9920639920009998,\n   \"max\": 0.9920639920009998,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9920639920009998\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m3vl"}

```
plot_confusion_matrix(m3,xc_val_normalized,yc_val)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 162ms/step
```




```
plot_history(h3)
```



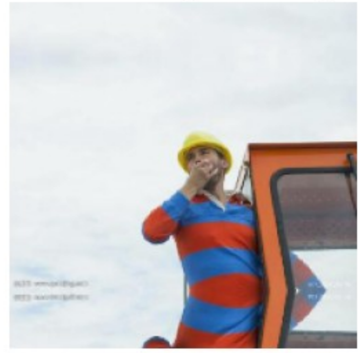
Visualizing the predictions

```
prev=m3.predict(xc_train_normalized)
f,ax=plt.subplots(4,2,figsize=(15,15))
for i,j in zip(ax.flatten(),range(8)):
  i.imshow(xc_train_normalized[j])
  i.axis('off')
  if prev[j]>0.5:
    i.set_title(f'With Helmet:{prev[j]}')
  else:
    i.set_title(f'Without Helmet:{prev[j]}')
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step
```

With Helmet:[0.99993825]

With Helmet:[0.9992337]



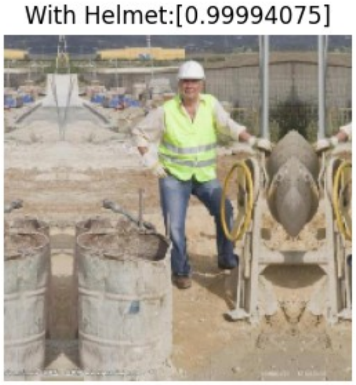
Without Helmet:[0.00010013]

With Helmet:[0.99967813]

With Helmet:[0.9998204]

With Helmet:[0.9999492]

With Helmet:[0.99994075]



Without Helmet:[0.00035484]

Observations of model 3::

- The model also shows slight overfitting.
- The loss curves show slight oscillations when compared to **Model 2**, but there is convergence and the validation losses are smaller in this model.
- The Predicted values are becoming more concise in certain cases with metric evaluation being same as previous model.
- As the metrics(recall) and confusion matrix scores show model is doing really well may not be robust since the data being clean and small.

Model 4: (VGG-16 (Base + FFNN + Data Augmentation)

- In most of the real-world case studies, it is challenging to acquire a large number of images and then train CNNs.
- To overcome this problem, one approach we might consider is **Data Augmentation**.
- CNNs have the property of **translational invariance**, which means they can recognise an object even if its appearance shifts translationally in some way. - Taking this attribute into account, we can augment the images using the techniques listed below
  - Horizontal Flip (should be set to True/False)
  - Vertical Flip (should be set to True/False)
  - Height Shift (should be between 0 and 1)
  - Width Shift (should be between 0 and 1)
  - Rotation (should be between 0 and 180)
  - Shear (should be between 0 and 1)
  - Zoom (should be between 0 and 1) etc.

Remember, **data augmentation should not be used in the validation/test data set**.

```
train_datagen = ImageDataGenerator(
                              horizontal_flip = True,
                              vertical_flip = False,
                              height_shift_range= 0.1,
                              width_shift_range=0.1,
                              rotation_range=20,
                              shear_range = 0.1,
                              zoom_range=0.1)


# Flowing training images in batches of 20 using train_datagen generator
train_generator = 
train_datagen.flow(xc_train_normalized,yc_train,batch_size=21,shuffle=True)

cls()
```

```
m4=Sequential()
m4.add(vgg16_m)
m4.add(Flatten())
m4.add(Dense(32, activation='relu'))
m4.add(BatchNormalization())
m4.add(Dropout(0.5))
m4.add(Dense(32, activation='relu'))
m4.add(BatchNormalization())
m4.add(Dropout(0.5))
m4.add(Dense(1, activation='sigmoid'))

opti=keras.optimizers.Adam(learning_rate=0.001)
m4.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accurac
y'])

h4=m4.fit(train_generator,epochs=32,validation_data=(xc_val_normalized
,yc_val),batch_size=32)
```

```
Epoch 1/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 19s 387ms/step - accuracy: 0.8195 - loss:
0.3973 - val_accuracy: 1.0000 - val_loss: 0.0460
Epoch 2/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 288ms/step - accuracy: 0.9513 - loss:
0.1421 - val_accuracy: 1.0000 - val_loss: 0.0367
Epoch 3/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 248ms/step - accuracy: 0.9645 - loss:
0.1034 - val_accuracy: 0.9921 - val_loss: 0.0302
Epoch 4/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 320ms/step - accuracy: 0.9726 - loss:
0.0863 - val_accuracy: 0.9921 - val_loss: 0.0200
Epoch 5/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 251ms/step - accuracy: 0.9960 - loss:
0.0553 - val_accuracy: 0.9921 - val_loss: 0.0168
Epoch 6/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 290ms/step - accuracy: 0.9753 - loss:
0.0517 - val_accuracy: 0.9921 - val_loss: 0.0203
Epoch 7/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 315ms/step - accuracy: 0.9937 - loss:
0.0350 - val_accuracy: 0.9841 - val_loss: 0.0285
Epoch 8/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 287ms/step - accuracy: 0.9948 - loss:
0.0327 - val_accuracy: 0.9921 - val_loss: 0.0224
Epoch 9/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 279ms/step - accuracy: 0.9928 - loss:
0.0441 - val_accuracy: 0.9921 - val_loss: 0.0211
Epoch 10/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 258ms/step - accuracy: 0.9993 - loss:
0.0189 - val_accuracy: 0.9921 - val_loss: 0.0160
Epoch 11/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 250ms/step - accuracy: 0.9879 - loss:
```

```
0.0284 - val_accuracy: 0.9921 - val_loss: 0.0096
Epoch 12/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 307ms/step - accuracy: 0.9936 - loss:
0.0318 - val_accuracy: 0.9921 - val_loss: 0.0123
Epoch 13/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 250ms/step - accuracy: 0.9946 - loss:
0.0231 - val_accuracy: 0.9921 - val_loss: 0.0138
Epoch 14/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 285ms/step - accuracy: 1.0000 - loss:
0.0116 - val_accuracy: 0.9921 - val_loss: 0.0122
Epoch 15/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 10s 248ms/step - accuracy: 0.9937 - loss:
0.0388 - val_accuracy: 0.9921 - val_loss: 0.0110
Epoch 16/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 332ms/step - accuracy: 1.0000 - loss:
0.0148 - val_accuracy: 0.9921 - val_loss: 0.0117
Epoch 17/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 248ms/step - accuracy: 0.9979 - loss:
0.0221 - val_accuracy: 0.9921 - val_loss: 0.0132
Epoch 18/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 251ms/step - accuracy: 0.9917 - loss:
0.0255 - val_accuracy: 0.9921 - val_loss: 0.0109
Epoch 19/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 312ms/step - accuracy: 0.9976 - loss:
0.0119 - val_accuracy: 0.9921 - val_loss: 0.0091
Epoch 20/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 247ms/step - accuracy: 0.9984 - loss:
0.0122 - val_accuracy: 0.9921 - val_loss: 0.0099
Epoch 21/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 287ms/step - accuracy: 0.9979 - loss:
0.0178 - val_accuracy: 0.9921 - val_loss: 0.0113
Epoch 22/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 271ms/step - accuracy: 0.9887 - loss:
0.0341 - val_accuracy: 0.9921 - val_loss: 0.0191
Epoch 23/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 250ms/step - accuracy: 1.0000 - loss:
0.0113 - val_accuracy: 0.9921 - val_loss: 0.0199
Epoch 24/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 304ms/step - accuracy: 1.0000 - loss:
0.0123 - val_accuracy: 0.9921 - val_loss: 0.0134
Epoch 25/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 251ms/step - accuracy: 0.9957 - loss:
0.0289 - val_accuracy: 0.9921 - val_loss: 0.0122
Epoch 26/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 287ms/step - accuracy: 0.9972 - loss:
0.0244 - val_accuracy: 1.0000 - val_loss: 0.0025
Epoch 27/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 308ms/step - accuracy: 0.9920 - loss:
0.0244 - val_accuracy: 1.0000 - val_loss: 0.0018
```

```
Epoch 28/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 250ms/step - accuracy: 0.9997 - loss:
0.0184 - val_accuracy: 1.0000 - val_loss: 0.0019
Epoch 29/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 257ms/step - accuracy: 1.0000 - loss:
0.0089 - val_accuracy: 1.0000 - val_loss: 0.0030
Epoch 30/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 281ms/step - accuracy: 0.9993 - loss:
0.0083 - val_accuracy: 1.0000 - val_loss: 0.0053
Epoch 31/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 287ms/step - accuracy: 0.9997 - loss:
0.0135 - val_accuracy: 1.0000 - val_loss: 0.0046
Epoch 32/32
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 307ms/step - accuracy: 1.0000 - loss:
0.0048 - val_accuracy: 0.9921 - val_loss: 0.0066
```

```
# Initialize empty lists to collect data
all_images = []
all_labels = []

# Iterate through the generator until all data is retrieved
for _ in range(len(train_generator)):
    imag, labela = next(train_generator)
    all_images.append(imag)
    all_labels.append(labela)

# Concatenate all batches into single arrays
all_images = np.concatenate(all_images, axis=0)
all_labels = np.concatenate(all_labels, axis=0)

print(all_images.shape)  # Shape: (total_samples, height, width,
channels)
print(all_labels.shape)  #
```

```
(378, 200, 200, 3)
(378,)
```

```
all_labels=pd.Series(all_labels)
all_labels
```

```
0      1
1      0
2      1
3      0
4      1
      ..
373    0
374    1
375    1
376    1
```

```
377    0
Length: 378, dtype: int64
```

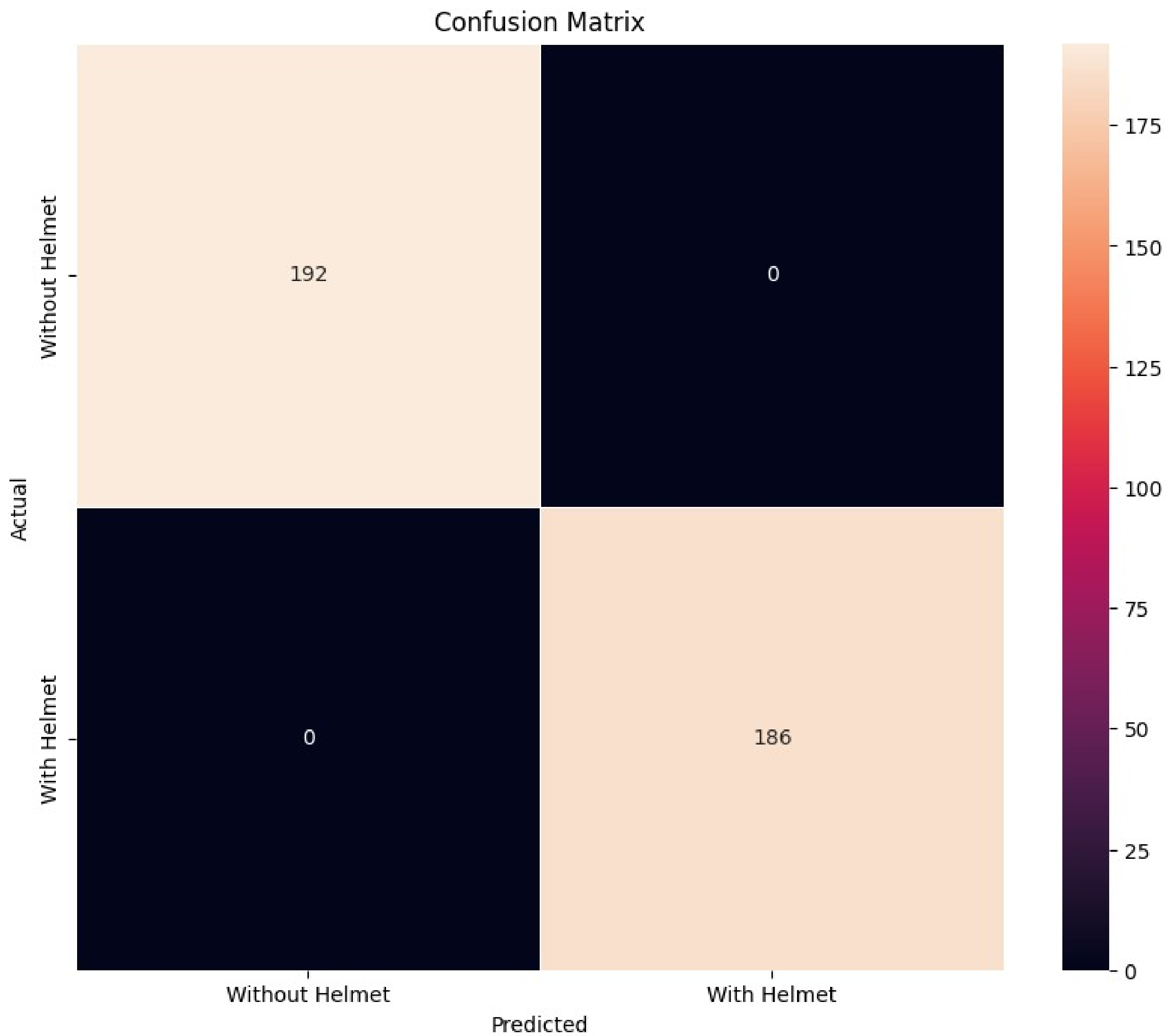
```
m4tr=model_performance_classification(m4,all_images,all_labels)
m4tr
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 205ms/step
```

{"summary":"{\n \"name\": \"m4tr\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m4tr"}

```
plot_confusion_matrix(m4,all_images,all_labels)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 157ms/step
```




```
m4vl=model_performance_classification(m4,xc_val_normalized,yc_val)
m4vl
```

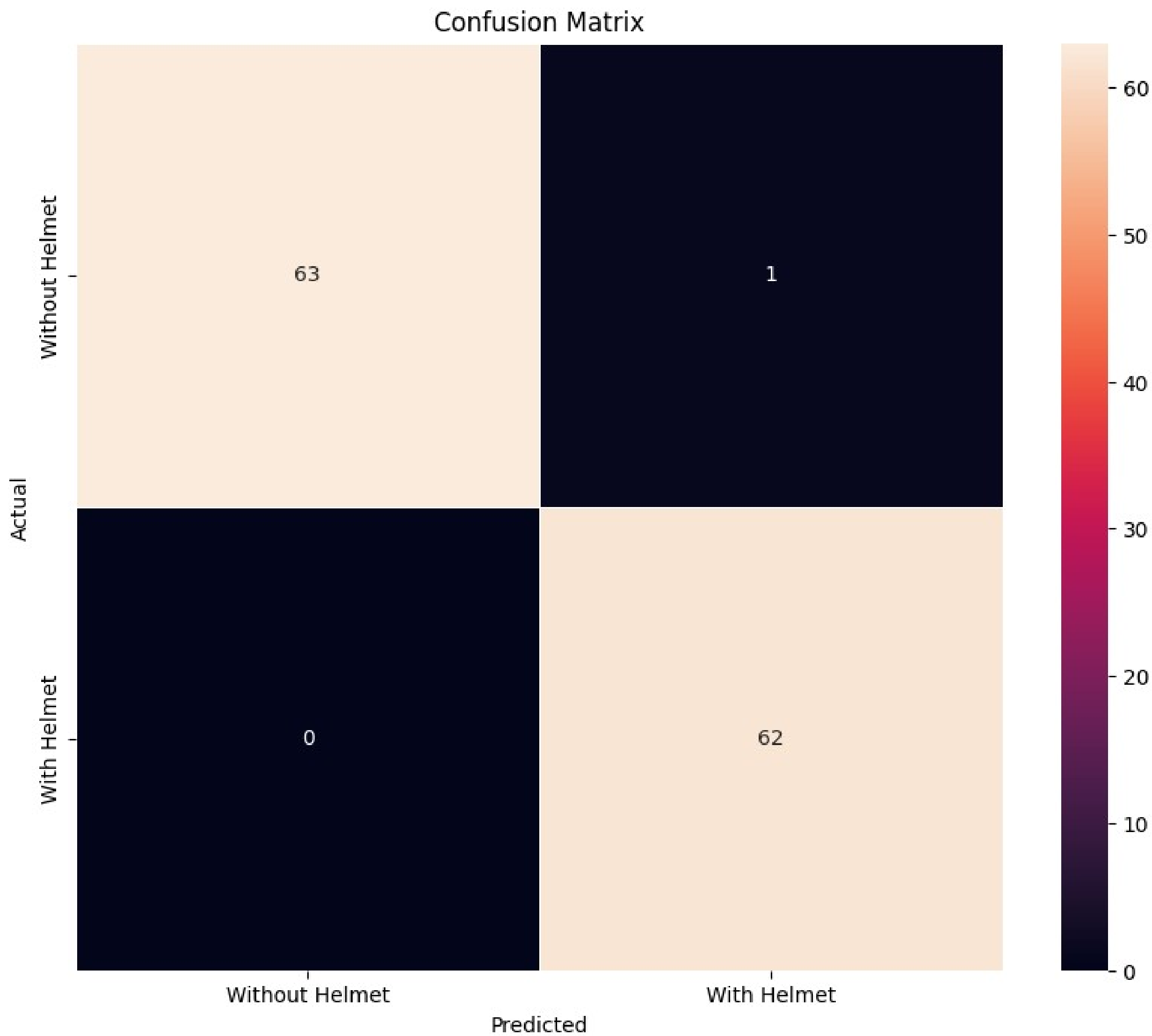
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 307ms/step
```

{"summary":"{\n \"name\": \"m4vl\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9920634920634921,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9920634920634921,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9920634920634921\n   ],\n

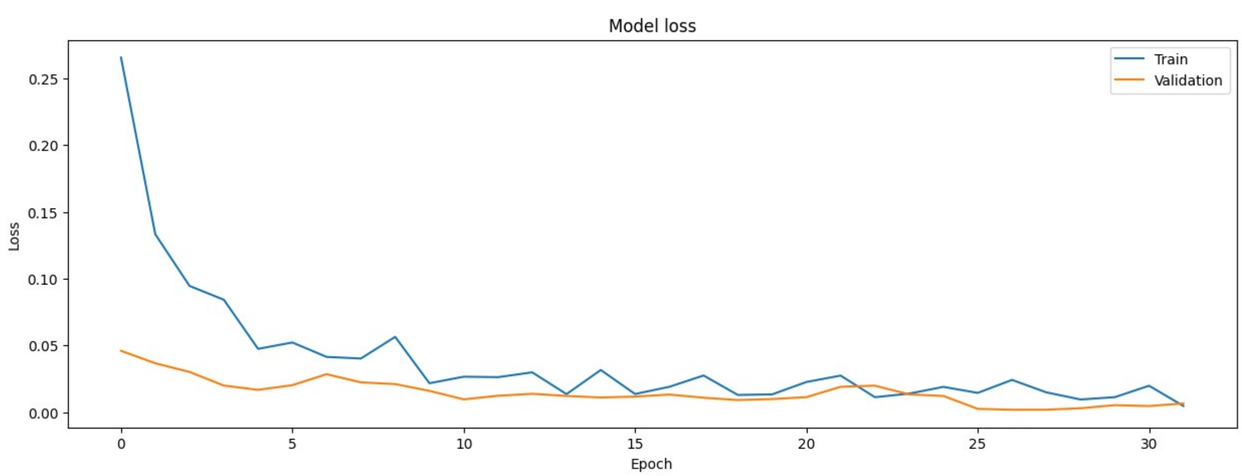
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Precision\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 0.9921894683799446,\n   \"max\":
0.9921894683799446,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   0.9921894683799446\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": null,\n
\"min\": 0.9920639920009998,\n   \"max\": 0.9920639920009998,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
0.9920639920009998\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"m4vl"}

```
plot_confusion_matrix(m4,xc_val_normalized,yc_val)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 166ms/step
```




```
plot_history(h4)
```



Visualizing the predictions

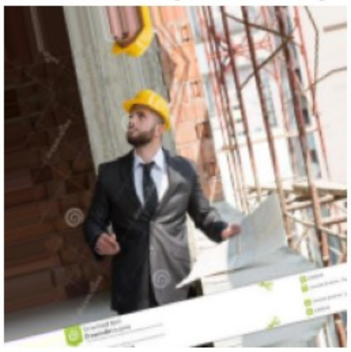
```
prev=m4.predict(imag)

f,ax=plt.subplots(4,2,figsize=(15,15))
for i,j in zip(ax.flatten(),range(8)):
  i.imshow(imag[j])
  i.axis('off')
  if prev[j]>0.5:
    i.set_title(f'With Helmet:{prev[j]}')
  else:
    i.set_title(f'Without Helmet:{prev[j]}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 516ms/step
```

With Helmet:[0.99971765]

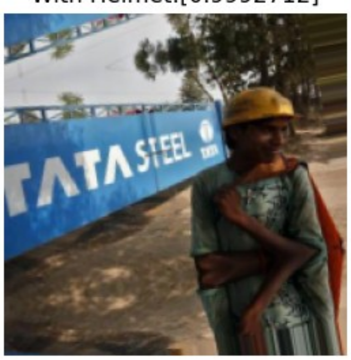
With Helmet:[0.9997824]

With Helmet:[0.99994946]

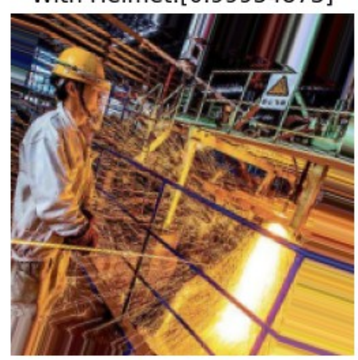


Without Helmet:[0.00016988]

With Helmet:[0.9992712]



With Helmet:[0.99954873]



Without Helmet:[5.3697964e-05]

With Helmet:[0.99994206]

Observations on model 4:

- Although model may shows slight overfitting The metrics(recall) and matix shows the model is doing very well even with the augmented data which shows its robustness when compared to the previous models.
- The loss curves suggest that validation losses are more smaller than previous models.

Model Performance Comparison and Final Model Selection

```
train_eval_result=pd.concat([m1tr.T,m2tr.T,m3tr.T,m4tr.T],axis=1)
train_eval_result.columns=['Model 1:Basic CNN','Model 2:Base
VGG16','Model 3:VGG16+FNN','Model 4:VGG16+FNN+Data augmentation']
print("Models Results on Training data")
print("**"*50)
train_eval_result
```

```
Models Results on Training data
****************************************************************************************************
```

{"summary":"{\n \"name\": \"train_eval_result\",\n \"rows\": 4,\n \"fields\": [\n   {\n   \"column\": \"Model 1:Basic CNN\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.11096318684183888,\n   \"min\": 0.5115102748836582,\n   \"max\": 0.7747412008281573,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.5952380952380952,\n   0.7747412008281573,\n   0.5115102748836582\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 2:Base VGG16\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 3:VGG16+FNN\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 4:VGG16+FNN+Data augmentation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"train_eval_result"}

```
val_eval_result=pd.concat([m1vl.T,m2vl.T,m3vl.T,m4vl.T],axis=1)
val_eval_result.columns=train_eval_result.columns
print("Models Results on validation data")
print("**"*50)
val_eval_result
```

```
Models Results on validation data
****************************************************************************************************
```

{"summary":"{\n \"name\": \"val_eval_result\",\n \"rows\": 4,\n \"fields\": [\n   {\n   \"column\": \"Model 1:Basic CNN\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.10250210310405837,\n   \"min\": 0.5379033270558695,\n   \"max\": 0.7797443461160275,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.6111111111111112,\n   0.7797443461160275,\n   0.5379033270558695\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 2:Base VGG16\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 3:VGG16+FNN\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 6.290527678051543e-05,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9921894683799446,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Model 4:VGG16+FNN+Data augmentation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 6.290527678051543e-05,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9921894683799446,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"val_eval_result"}

Final Model Selection

Based on the observations:

- As we observe model 1 metrics are not good and model is not able to classify people with helmets.
- The results of **models 2, 3, and 4** are identical in terms of performance metrics(recall).
- However, **Model 4** was trained using augmented data, which enhances its robustness to variations in the images.

Conclusion

Model 4 is selected as the final model for deployment due to its ability to generalize better to unseen data.

Test Performance

```
m4ts=model_performance_classification(m4,xc_test_normalized,yc_test)
m4ts
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step
```

{"summary":"{\n \"name\": \"m4ts\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m4ts"}

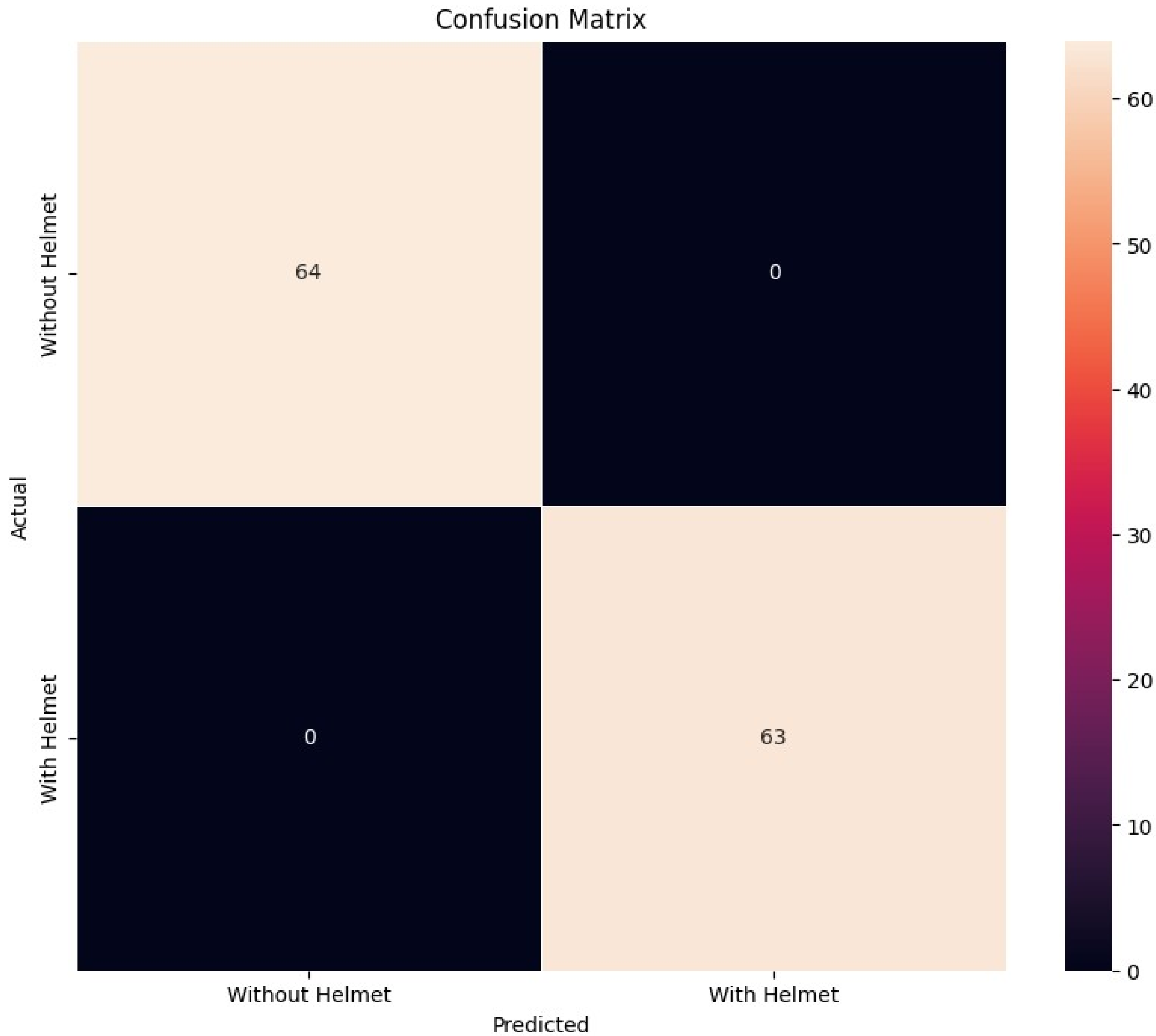
```
m4eval=pd.concat([m4tr.T,m4vl.T,m4ts.T],axis=1)
m4eval.columns=['Train','Validation','Test']
m4eval
```

{"summary":"{\n \"name\": \"m4eval\",\n \"rows\": 4,\n \"fields\": [\n   {\n   \"column\": \"Train\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Validation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 6.290527678051543e-05,\n   \"min\": 0.9920634920634921,\n   \"max\": 0.9921894683799446,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.9920634920634921\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Test\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

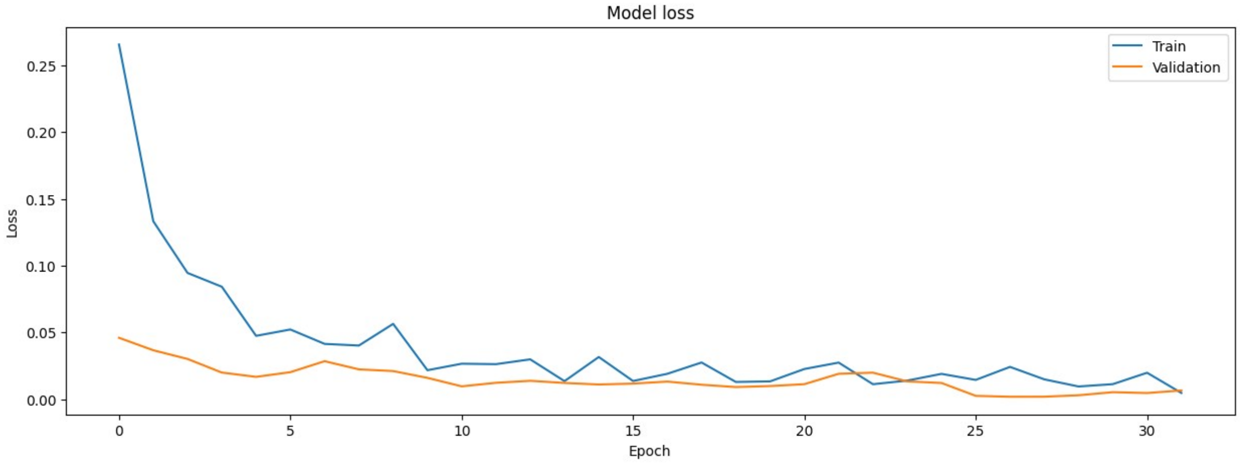
\"std\": 0.0,\n   \"min\": 1.0,\n   \"max\": 1.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   1.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"m4eval"}


```
plot_confusion_matrix(m4,xc_test_normalized,yc_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 163ms/step
```




```
plot_history(h4)
```




```
print("Classification Report - Train data",end="\n\n")
crtr =
classification_report(yc_train,m4.predict(xc_train_normalized)>0.5)
print(crtr)
```

```
Classification Report - Train data

12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 160ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       192
           1       1.00      1.00      1.00       186

    accuracy                           1.00       378
   macro avg       1.00      1.00      1.00       378
weighted avg       1.00      1.00      1.00       378
```

```
print("Classification Report - Validation data",end="\n\n")
crvl = classification_report(yc_val,m4.predict(xc_val_normalized)>0.5)
print(crvl)
```

```
Classification Report - Validation data

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 159ms/step
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        64
           1       0.98      1.00      0.99        62

    accuracy                           0.99       126
   macro avg       0.99      0.99      0.99       126
weighted avg       0.99      0.99      0.99       126
```

```
print("Classification Report - Test data",end="\n\n")
crts =
classification_report(yc_test,m4.predict(xc_test_normalized)>0.5)
print(crts)
```

```
Classification Report - Test data

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 159ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        63

    accuracy                           1.00       127
   macro avg       1.00      1.00      1.00       127
weighted avg       1.00      1.00      1.00       127
```

Observations on best model(model 4):

- The reports, metrics(recall) and matrix as well loss curves all suggest very good performance.
- Since the model is trained on augmented data it is robust and able learn the right elements excluding the noise.

Actionable Insights & Recommendations

1. Model Evaluation and Selection

- Even though the primary evaluation criterion is recall due to the safety-critical nature of the task, all metrics (accuracy, precision, and F1-score) are essential for a holistic assessment. Model 4 achieves very good performance across these metrics with slight overfitting, making it the most suitable candidate.

- Validation results indicate that Model 4 is more robust than other models, showing convergence in loss curves and better generalization even with data augmentation which confirms robustness.

2. Data Considerations

- The dataset size is a significant limitation. Even with augmentation, it is insufficient for robust generalization. Increasing the dataset size with real-world samples is critical.

- A larger and more diverse dataset will improve the model’s ability to handle varied conditions such as different lighting, backgrounds, and helmet types.

- Focus on collecting edge-case scenarios (e.g., partially visible helmets, different helmet styles, and obstructions) to ensure comprehensive training and evaluation.

3. Leveraging Advanced Architectures and Strategies

- Utilize advanced architectures, such as transformers, which excel at capturing global context and intricate patterns in data.
- Leverage models designed for improved feature extraction and adaptability, particularly beneficial for datasets with limited diversity.
- Integrate these architectures with effective preprocessing and augmentation techniques to maximize performance and reduce overfitting.