Course:

Machine Learning & Content Analytics

MSc Business Analytics, part-time 2021-2022

Professor:

Haris Papageorgiou

Assignment:

Helmet Detection on Motorcyclists using Image Classifiers

Students:

Alamanis Konstantinos (p2822103)

Gkini Eftychia (p2822108)

Skoli Maria (p2822131)

Vasileiou Giorgos (p2822106)

August 2022

Contents

Abstract

This paper aims to improve the monitoring of traffic violation of not wearing a helmet and consequently to reduce road accidents. To achieve this, a deep learning framework comprised of three models will be exploited. With the use of Machine Learning algorithms, we have performed Image Classification implementing not only a CNN model but also pretrained models such as InceptionV3 and MobileNet2 using as weights the “Imagenet” weights which are proposed for image classification. For all those models, we set up the same Hyperparameters (batch size, epochs, learning rate, callbacks) to be comparable. The best accuracy score (89%) was achieved from both InceptionV3 and MobileNet2 pre-trained models. Considering the specificities of our solution, we decided to proceed with MobileNet2 for reasons that will be explained in the main body of this paper.

Introduction

Nowadays, road accidents are one of the major problems societies must deal with. Motorbike accidents can cause severe injuries and sometimes can lead to human death. Greece has the third highest share of motorcycle fatalities among the EU Member States, according to the latest report based on 2018 figures published by Eurostat. According to the data, in 2018, 27.1% of all road accident fatalities in Greece occurred among motorcyclists, with Cyprus in second place at 28.6% and Malta first at 44.4%.

Bicycle lanes and helmets may reduce the risk of death. Almost, three-quarters of fatal crashes (74%) involved a head injury. Nearly all bicyclists who died (97%) were not wearing a helmet. Helmet use among those bicyclists with serious injuries was low (13%), but it was even lower among bicyclists killed (3%). Although the helmet use is mandatory in many countries, there are motorcyclists that do not use it or use it incorrectly.

The challenge faced when researching motorcycle accidents is the absence of data available for the number of accidents occurring throughout the country over a year. The failure to report a motorcycle accident to the police or state motor vehicle departments may be the reason for the lack of data.

Nowadays, Machine learning (ML) comes to fill the gap. Machine Learning is the field of Artificial Intelligence in which a trained model works on its own using the inputs given during training period.

Machine learning algorithms build a mathematical model of sample data, known as "training data", to make predictions or decisions and are also used in the applications of object detection and image classification. Therefore, by training with a specific dataset, a Helmet detection model can be implemented.

In that context, intelligent traffic systems were implemented using computer vision algorithms, such as: background and foreground image detection to segment the moving objects in scene and image descriptors to extract features. Computational intelligence algorithms are used too, like machine learning algorithms to classify the objects.

Our project

This report noted that a complete action plan must be set up to save lives. We believe that the police force should act to monitor the people not wearing a helmet and prevent them from accidents. We thought that by cooperating with the government to place cameras at crucial traffic lights to each region, we can monitor the helmet violations.

To the regions we receive the most signals of violation, a message will be sent to the nearest policy station to reinforce checks on the road.

Finally, we can use those data for statistical reasons. For example, in regions with high percentages of violations we can organize driving behavioral lessons, or collision simulations to build a “drive with safety” mindset.

Our team believe that this view is more efficient than detecting the license plate and automatically send a fine to the motorcyclist. That’s why by the time a policy officer observes someone not wearing a helmet, gives order to stop the vehicle. This action can prevent an upcoming accident.

In the future and during scaling-up our solution, combining it with micro-computers such as Raspberry Pi 4 would give us the ability to construct a great tool for authorities to that can be placed in traffic lights to enhance the drivers and motorcyclists to abide the law.

Our Vision

The main goal of this project is to help government to increase safety in the roads and reduce the road accidents.

Methodology

Taking as input image footages from the road, we want to see if there is at least one person who does not wear a helmet in the corresponding image.

The first methodology we have examined was to crop the images based on the bounding boxes denoting the head of the motorcyclist. We thought that it would be beneficial for image classification to reduce the environment of search. However, the fact that we had the coordinates of motorcyclists' heads doesn't imply that these coordinates would be available in the production and then they should be determined by an object detection model. So, we decided to keep the images uncropped to train the algorithm classifying the photos despite the noisy environment. A fully explained model based on cropped images can be found in Appendix for reference.

For evaluating the performance of each model, the dataset was split to train, validation, and test sets.

The first step was to implement a CNN (Convolutional Neural Network) model in a context of pattern recognition to classify whether there is a person without wearing a helmet. The pattern recognition is achieved by implementing multiple filters in the initial image. Technics of max pooling and dropout are also used to reduce the size of the image and simultaneously to increase the channels. As the dataset consists of colorful images the initial image has 3 channels declaring the RGB colors (red, blue, green) and therefore the filters should be implemented separately to each channel.

We have started with a simple CNN model, with one layer of 32 filters. Then, we created a "Custom" CNN model by adding more layers of 16, 32, 64 and 128 filters accordingly to obtain better results. In both CNN models we used the Adam optimizer.

For further improvements, we utilized pretrained models like MobileNet2 and InceptionV3 models, which were trained on large image datasets. This method is known as transfer learning.

Finally, we have compared all these models to conclude which is the best for our project. Each model has been evaluated based on the accuracy, the validation loss, the precision, the recall, and False Positive Rate.

For our goal, it is also important to consider False Positive Rate along with accuracy because it would be devastating to receive more false alerts of not wearing a helmet. Having that in mind we have created a plot depicting the combination of accuracy and false positive rate for each model. In addition, ROC Curves of models considered, which is a two-dimensional graph with TPR (true positive rate) on the y-axis and FPR (false positive rate) on the x-axis. Thus, it represents the trade-off between the value (true positive rate) of the classifier, and its cost (false positive rate).

A more detailed explanation of the models and the visualization of the evaluation metrics will be discussed in the following chapters.

Data Collection

For this project we used the Helmet-Detection dataset from Kaggle.

Dataset Overview

This dataset contains 764 images capturing 2 distinct classes as well as their corresponding annotations (*.xml files). All images are in “png” file format.

The classes are:

- With helmet (1)
- Without helmet (0)

Regarding the annotations each of them contains valuable information in which class the object belongs. The object name is noted as *“With helmet”* if the person in the photo wears a helmet and *“Without Helmet”* if not. Also, the annotations play a major role as they indicate the position of the head (Bounding Boxes). We should note here that there are images depicting more than one person. In that case, the corresponding “xml” file contains specialized information for all people. Below we can have a brief view of the dataset. The green box indicates the existence of the helmet. Otherwise, there is a red box.

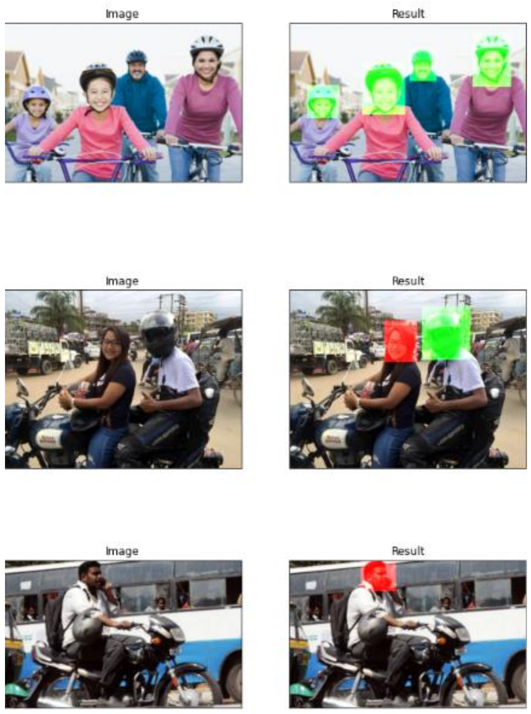


Figure 1:Dataset Overview

Data Processing/Annotation/Normalization

Next step in our analysis was the data preprocessing. Not all images had the same size and therefore, it is needed to resize all images to 224x224. Secondly, the approach of extracting the label information from the image's annotations is described by the following rule:

"In case of having an image with multiple objects (people) and at least one does not wear a helmet then this image is labeled as "Without Helmet" (class 0), otherwise it is label "With Helmet" (class 1)."

In this way an array with the information of the image and its label-class was extracted.

To increase the chances of a robust model, it is needed to use a significant number of images. For that purpose, we have performed augmentation techniques such as rotation, brightness, zoom on the existing images. Data augmentation techniques were utilized to enhance the generalization ability of the model. Specifically, we have selected to rotate the images to prevent inaccuracies based on each camera's angle. The brightness augmentation technique is also useful to deal with the different lighting conditions, such as day and night or different weather conditions.

Finally, the zoom technique is important when the motorcyclist is too far from the camera. Examples of augmented images are shown below.

Figure 2: Zoom

Figure 3: Rotation

Figure 4: Brightness

The final dataset consists of 3044 images, of which 2283 are augmented images and 67% of them are labeled as "With Helmet" (1).

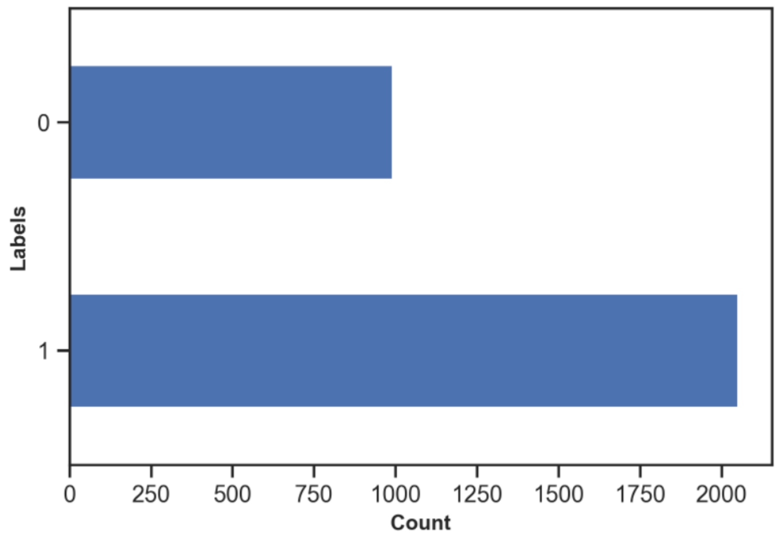


Figure 5: Dataset Balance

The last step in images preprocessing was to normalize the images. Because the images consist of RGB coefficients in range of [0,255] interval, each image was normalized by dividing with 255 to be used as input in the models.

Concerning the training and the evaluation of the models, the dataset was split to a training, a validation, and a testing set. The 75% of the data was used for training the model and the rest 25% was used to validate and test the model. We have resolved the issue of the unbalanced data by using the "stratify" field during splitting.

Experiments – Setup, Configuration

During the training of the neural networks, it was necessary to do experiments and take some decisions about the way that we will train our models.

The biggest challenge was how long to train the model as too little training will mean that the model underfits the train set and too much training will mean that the model will overfit the train set. So, we input some optimization techniques as hyperparameters to reduce overfitting without compromising on model accuracy.

One technique for reducing the size of each parameter dimension is *“Early Stopping”*, which checks at end of every epoch whether the loss is no longer decreasing. The use of early stopping requires the selection of a performance measure to monitor and define the patience (the number of epochs to wait). So, if the validation loss is not decreased in as many epochs as the "patience" we set then the training process will be stopped.

Another technique named as *“ReduceLROnPlateau”,* defines the learning rate and controls how quickly the model is adapted to the problem. A learning rate that is too high can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too low can cause the process to get stuck. The use of ReduceLROnPlateau, requires the selection to specify the metric to monitor, define the patience and the value that the learning rate will be multiplied.

Another critical part was to specify the *“activation function”* in the hidden and output layers as it controls how well the network model learns from the training dataset. Without an activation function, a neural network is a simple linear regression model. The activation functions we used in our CNN models are ReLU and SoftMax.

In hidden layers the Rectified Linear Unit, or ReLU activation function was used. It is the most common activation for hidden layers as avoid vanishing gradient problem, it is less computationally expensive and often achieves better performance. ReLU is a piecewise linear function that in comparison with other activation function the input values are not in a specific range. It is calculated as follows, if the input value is positive the value is returned, otherwise, it will output zero.

In the output layer, which is the last layer, we use the SoftMax function. This function calculates the probabilities distribution and directly outputs a prediction.

"SoftMax" is a form of logistic regression that normalizes an input value into a vector of the outputs values which are interrelated and the total sums up to one. It is calculated using the following function $\frac{e^x}{sum(e^x)}$.

After the completion of the layers, an optimizer must be determined. To do so, we did many experiments using different optimizers to find the optimal one for our models. We used mainly the Adam optimizer, which is a stochastic gradient descent method that is based on adaptive estimation of first order and second-order moments. Also, in our CNN model we tried the Root Mean Squared Propagation (RMSProp), which is an extension of the gradient descent optimization algorithm. This algorithm does well on online and non-stationary problems such as noisy problems. However, it seems that Adam produces slightly better results than RMSProp. At the end, we tried different combinations of layers with multiple filters to increase the predictive ability of the model.

Algorithms, NLP architectures/systems

As the manipulation of the dataset was finished, we fitted some classification models to the data. For the implementation of this project, it was necessary to import libraries in Python used for machine learning and specifically in neural networks. Some of the main libraries we have used are: TensorFlow, Keras, NumPy, Pandas, Matplotlib and Seaborn.

For each training example there will be a set of input values (vectors) and one or more associated specified output values. The aim of this work is to design a neural network and the selection of the architecture as well as its parameters that will achieve the highest accuracy, after a series of experiments. The possible combinations that can occur are many.

In the next paragraphs we are going to discuss the architectures of a two custom CNN models and two pretrained models we developed for our project.

A. CNN (Convolutional Neural Networks)

Convolutional Neural Networks (CNNs) are primarily used for image classification and recognition. CNNs are structured differently as compared to a regular neural network. In a regular neural network, each layer consists of a set of neurons. Each layer is connected to all neurons in the previous layer.

The way CNNs work, is that they have 3-dimensional layers in a width, height, and depth manner. A brief view of the CNN's architecture is shown in the picture below:

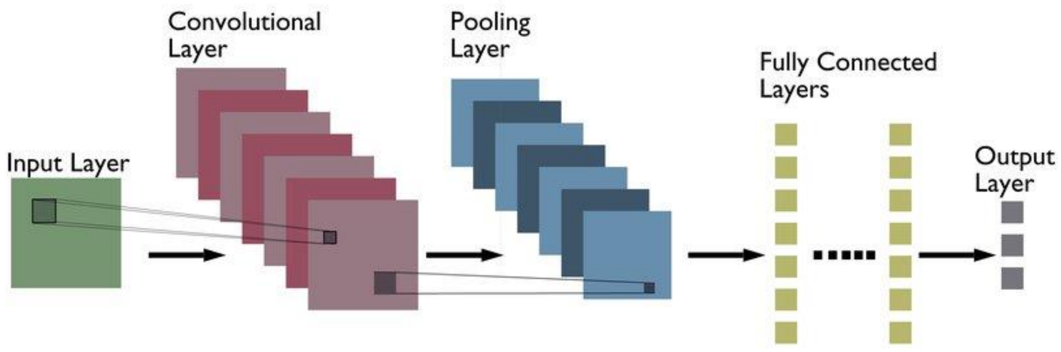
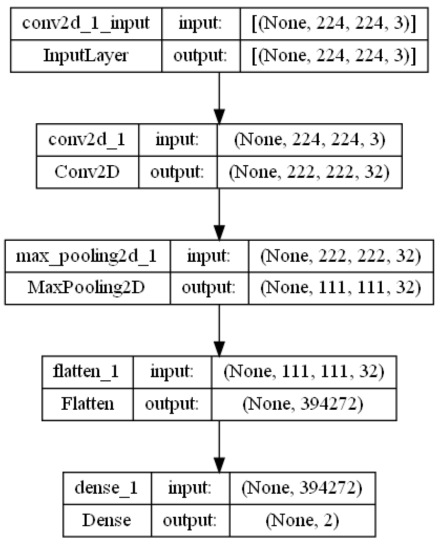


Figure 6: CNN Architecture

A1. CNN v1 – Few layers

Firstly, we used a Convolutional neural network (CNN), in which the input pictures are processed through a succession of layers, including convolutional, pooling and then the output of CNN is formed, which classifies pictures.



```
conv2d_1_input    input:  [(None, 224, 224, 3)]
InputLayer        output: [(None, 224, 224, 3)]

conv2d_1          input:  (None, 224, 224, 3)
Conv2D            output: (None, 222, 222, 32)

max_pooling2d_1   input:  (None, 222, 222, 32)
MaxPooling2D      output: (None, 111, 111, 32)

flatten_1         input:  (None, 111, 111, 32)
Flatten           output: (None, 394272)

dense_1           input:  (None, 394272)
Dense             output: (None, 2)
```

Figure 7: CNN v1-Few layers Architecture

The precise structure of the CNN v1 shown in Figure 7 is as follows:

- A convolutional layer with 32 filters, kernel size equals to 3 and activation function "ReLU". Kernel size defines the size of the filter we implement (3x3).
- A max-pooling layer with pool size 2x2.
- And a dense layer with 2 neurons and "SoftMax" as activation function.

- The last layer calculates the loss function related to the target classes. The goal of this form of training is to reduce the overall classification error of the models by correctly calculating the output value of the training example (backpropagation), which is the overall goal of the model.

Concerning about the training and the evaluation of the model (CNN v1) in our first experiment:

- As an optimization algorithm, Adam solver with a learning rate equal to 1^{E-03} was selected.
- The model was trained for 50 epochs with batch size of 32 and stopped early on 13th epoch, restoring best weights from epoch 3.

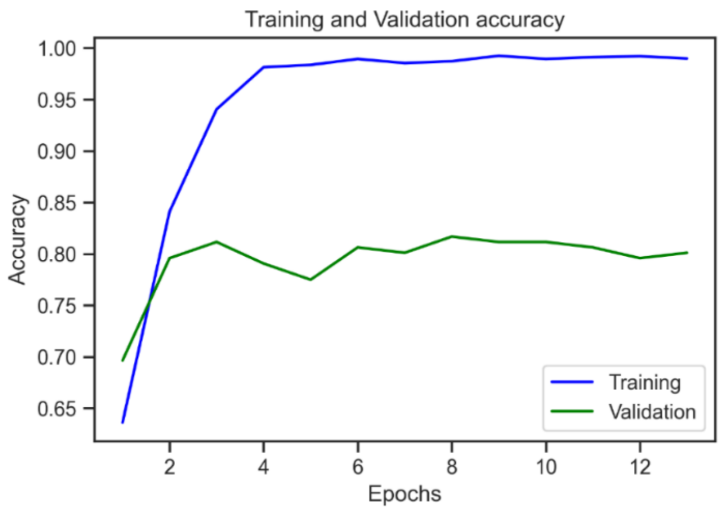


Figure 8: Training and Validation accuracy of CNN v1 Few Layers

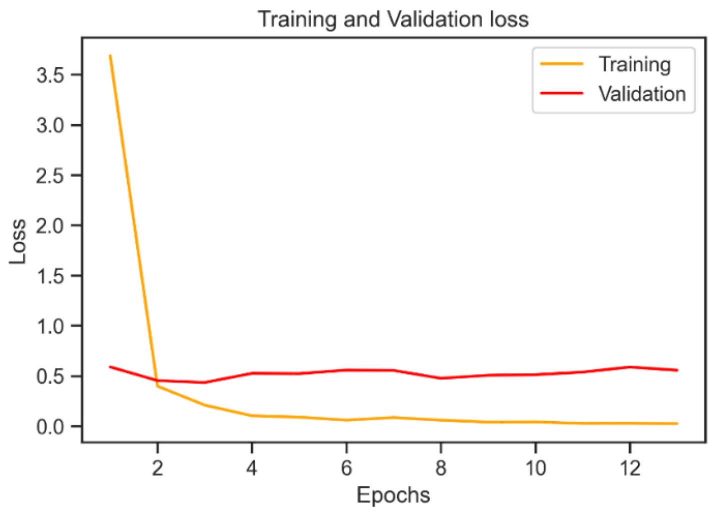


Figure 9: Training and Validation Loss of CNN v1 Few Layers

A2. CNN v2 - adjusted

The next step was to add more layers to the previous CNN model to increase the performance of the model. Therefore, the CNN v2- adjusted model was developed.

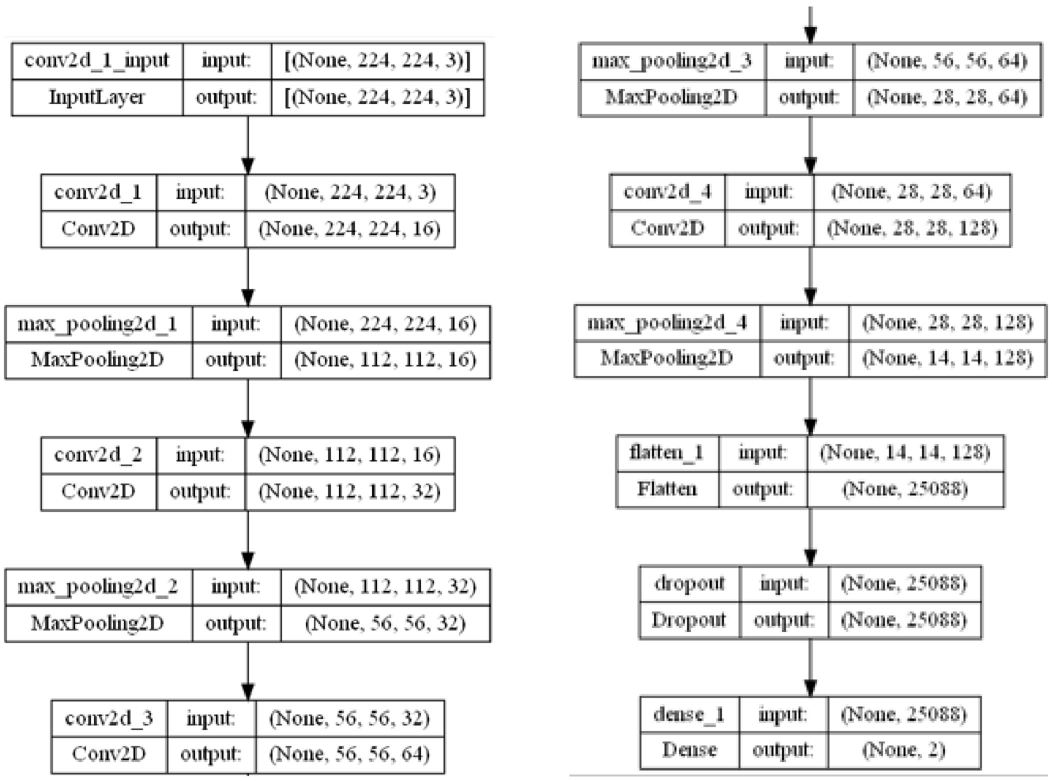

conv2d_1_input | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
conv2d_1 | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 224, 224, 16)
max_pooling2d_1 | input: | (None, 224, 224, 16)
MaxPooling2D | output: | (None, 112, 112, 16)
conv2d_2 | input: | (None, 112, 112, 16)
Conv2D | output: | (None, 112, 112, 32)
max_pooling2d_2 | input: | (None, 112, 112, 32)
MaxPooling2D | output: | (None, 56, 56, 32)
conv2d_3 | input: | (None, 56, 56, 32)
Conv2D | output: | (None, 56, 56, 64)
max_pooling2d_3 | input: | (None, 56, 56, 64)
MaxPooling2D | output: | (None, 28, 28, 64)
conv2d_4 | input: | (None, 28, 28, 64)
Conv2D | output: | (None, 28, 28, 128)
max_pooling2d_4 | input: | (None, 28, 28, 128)
MaxPooling2D | output: | (None, 14, 14, 128)
flatten_1 | input: | (None, 14, 14, 128)
Flatten | output: | (None, 25088)
dropout | input: | (None, 25088)
Dropout | output: | (None, 25088)
dense_1 | input: | (None, 25088)
Dense | output: | (None, 2)

Figure 10: CNN v2 Adjusted Architecture

The precise structure of the (CNN v2) shown in Figure 10 is as follows:

- A convolutional layer with 16 filters, kernel size equals to 3 and "ReLU" as activation function.
- A max-pooling layer with pool size 2x2
- A convolutional layer with 32 filters, kernel size equals to 3 and "ReLU" as activation function.
- A max-pooling layer with pool size 2x2
- And a dense layer with 2 neurons and SoftMax as activation function.
- A convolutional layer with 64 filters, kernel size equals to 3 and "ReLU" as activation function and a max-pooling layer with pool size 2x2
- And a dense layer with 2 neurons and SoftMax as activation function.
- A convolutional layer with 128 filters, kernel size equals to 3 and "ReLU" as activation function.
- A max-pooling layer with pool size 2x2

- A dense layer with 2 neurons and "SoftMax" as activation function.

Concerning about the training and the evaluation of the CNN v2 model:

- As an optimization algorithm, “Adam” solver with a learning rate equal to 1^{E-03} was selected.
- The model was trained for 50 epochs with batch size of 32, stopped earlier on 23rd epoch and restore best weights from 13th epoch.

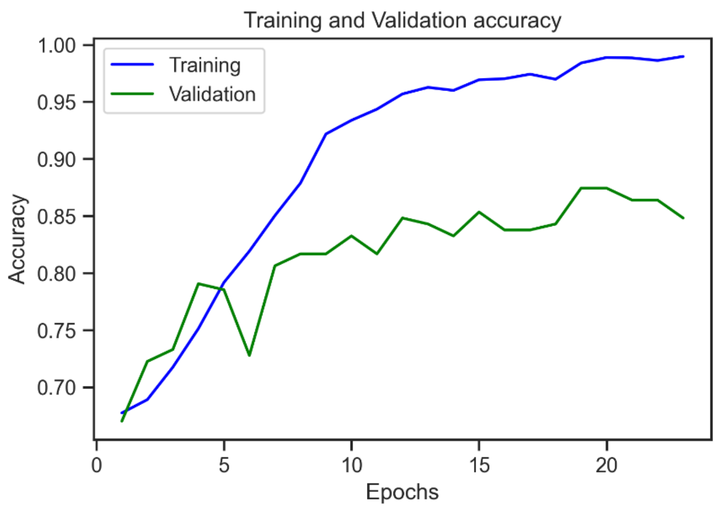


Figure 11: Training and Validation Accuracy of CNN v2 Adjusted

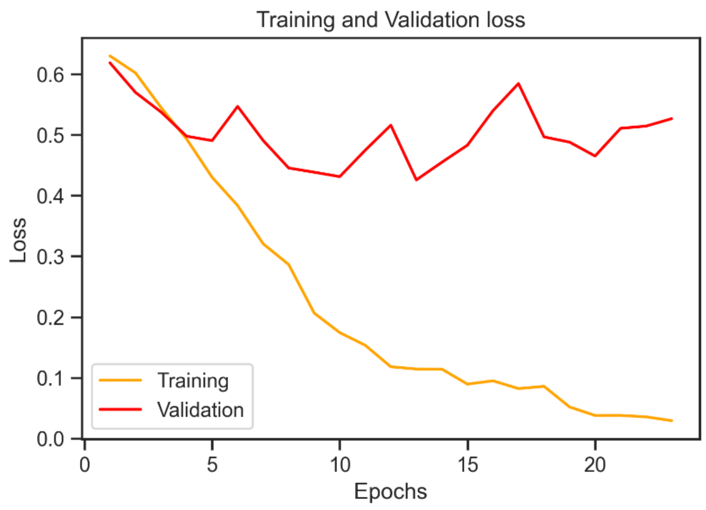


Figure 12: Training and Validation Loss of CNN v2 Adjusted

B. Pre-trained models

B1. InceptionV3 architecture

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification. The inception V3 is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014. As the name suggests it was developed by a team at Google.

When multiple deep layers of convolutions were used in a model it resulted in the overfitting of the data. To avoid this from happening the inception V1 model uses the idea of using multiple filters of different sizes on the same level. Thus, in the inception models instead of having deep layers, we have parallel layers thus making our model wider rather than making it deeper.

The inception V3 is just the advanced and optimized version of the inception V1 model. The Inception V3 model used several techniques for optimizing the network for better model adaptation.

- It has higher efficiency.
- It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised.
- It is computationally less expensive.
- It uses auxiliary Classifiers as regularizes.

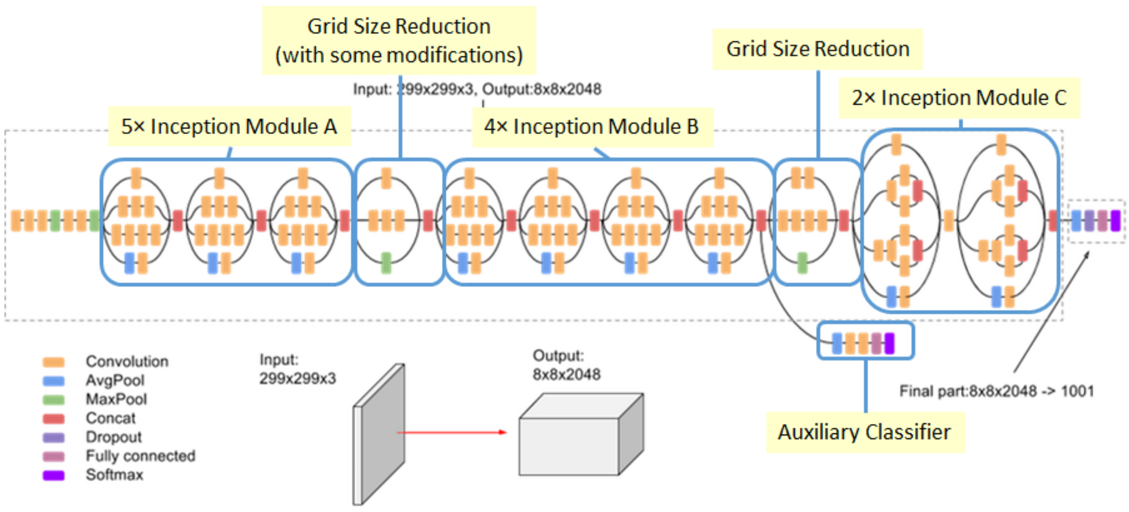


Figure 13 - Inception V3 model Architecture.

B2. MobileNet2 architecture

MobileNetV2, constructed by Google, is also a very capable, light-weighted (in terms of computations) model for image classification. Starting with V1 (MobileNetV1) a Depthwise Separable Convolution is introduced which dramatically reduce the complexity cost and model size of the network, which is suitable to Mobile devices, or any devices with low computational power. In MobileNetV2, a better module is introduced with inverted residual structure.

- In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the Depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.
- And there is an expansion factor t. And t=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64×t=64×6=384 channels.

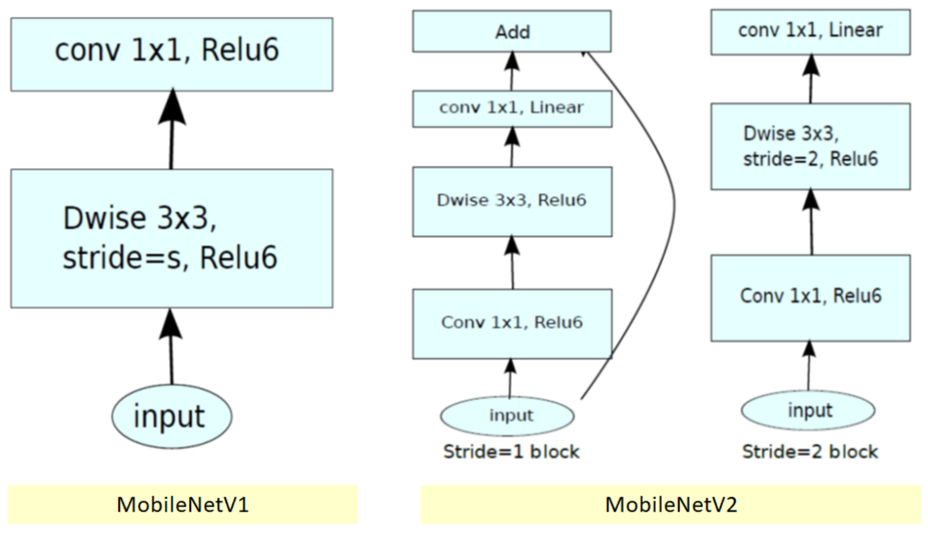


Figure 14 - MobileNetV1 vs MobileNetV2 architecture

B3. Transfer Learning

For our solution, we utilized the architectures of InceptionV3 and MobileNet2 models, along with their weights that have been computed after training on “imagenet” (>1.2M images with ~1000 classes) dataset. This method is known as Transfer Learning, where a model trained in millions of images and, mostly for a long time, is capitalized from other users to build on it by adding more layers or classifiers. In our case, the approach we followed for Transfer Learning was:

- Instantiate models and load weights from TensorFlow, trained on “imagenet” dataset and not including last layer.
- “Freezing” rest layers as non-trainable to avoid destroying any of the information they contain during future training rounds.
- Adding a new trainable layer with “SoftMax” activation function.
- Fitting the model with a higher learning rate compared to rest CNNs (1e-02 vs 1e-03) to tune that last added layer

Once your model has converged on the new data, unfreezing all or part of the pre-trained model and retrain the whole model end-to-end with a very low learning rate. This step is known as fine-tuning and can help you achieve incremental gains on accuracy. For that purpose, a very low learning rate (1e-04) and a small number of epochs (10) used, because the training of a much larger model than in the first round of training, on a dataset that is typically very small. This step help to decrease the risk of overfitting very quickly if you apply large weight updates.

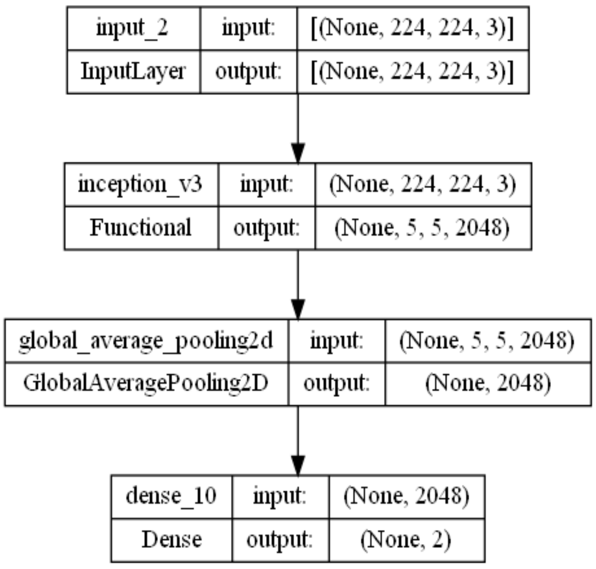

input_2 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
inception_v3 | input: | (None, 224, 224, 3)
Functional | output: | (None, 5, 5, 2048)
global_average_pooling2d | input: | (None, 5, 5, 2048)
GlobalAveragePooling2D | output: | (None, 2048)
dense_10 | input: | (None, 2048)
Dense | output: | (None, 2)

Figure 15 - Custom InceptionV3 in Transfer Learning

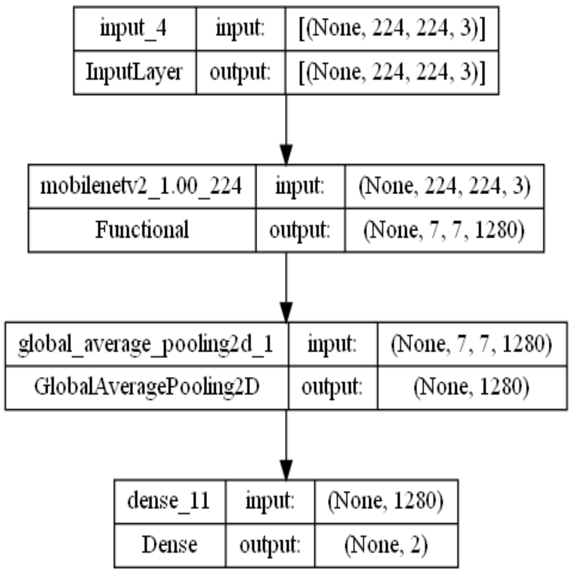

input_4 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
mobilenetv2_1.00_224 | input: | (None, 224, 224, 3)
Functional | output: | (None, 7, 7, 1280)
global_average_pooling2d_1 | input: | (None, 7, 7, 1280)
GlobalAveragePooling2D | output: | (None, 1280)
dense_11 | input: | (None, 1280)
Dense | output: | (None, 2)

Figure 16 - Custom MobileNet2 in Transfer Learning

For InceptionV3 these are the results of training. As we can observe from the plots below, the validation loss decreased from 6th epoch and onwards, where early stopping triggered at 23rd epoch, restoring best weights from 13th epoch. These are the results of initial training and not fine-tuning.

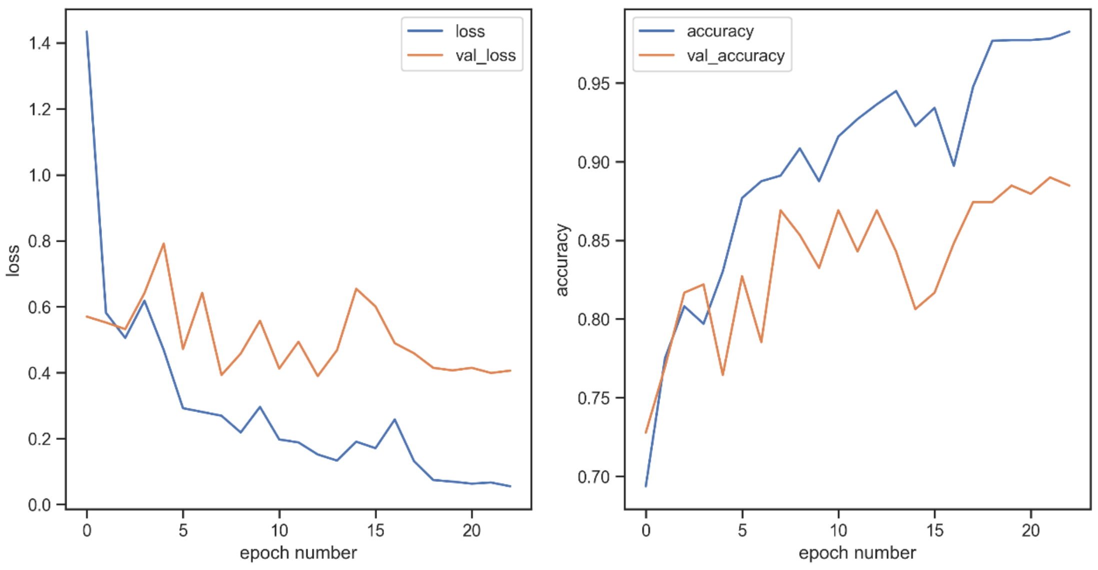


Figure 17: Training and Validation Loss and Accuracy for InceptionV3 model

Regarding MobileNet2, below are presented the results of initial training. The model was trained for 15 epochs, before early stopping triggered. Best weights retrieved from 5th epoch. These are the results of initial training and not fine-tuning.

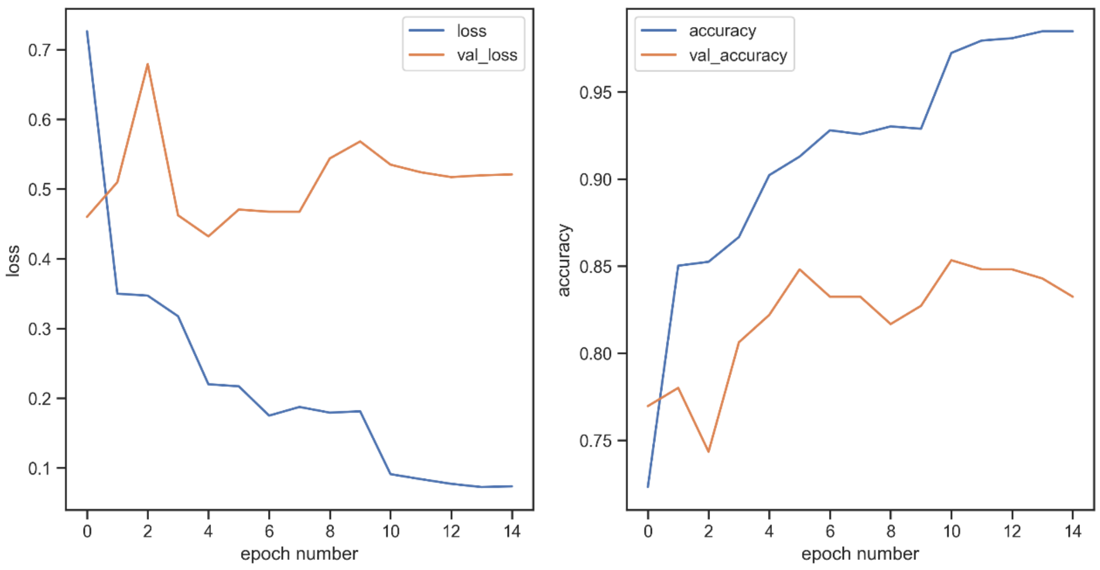


Figure 18: Training and Validation Loss and Accuracy for MobileNet2 model

Results & Quantitative Analysis (incl. Visualizations)

In this section, we are going to compare the mentioned models based on their performance and we will present the results of the best model (MobileNet2) in detail.

To measure the effectiveness of the algorithms, five major metrics are used: accuracy, precision, recall, F1 score and Area Under Curve (AUC). All performance metrics can be calculated using the four parameters of the confusion matrix. However, we mainly considered the accuracy score, the ROC curve, and the False Positive Rate (FPR) as the more critical metrics for our problem.

These metrics will be assessed by comparing the outputs of the models with the ground truth. The classifier will predict the most likely class for the test data based on what it has learned during training. Since the data is fully labeled, the predicted value can be validated against the actual label to measure the accuracy of the model.

Table 19 shows the comparison of the models. As we can see all the accuracy scores were particularly high. Accuracy is how close or far is a given set of measurements from their true value. Specifically, the model mob_net2 has the best accuracy score of 89,7% with no significant difference from the inception_v3 model that has an accuracy of 89,0%.

Classifier	Accuracy	Precision	Recall	F1	AUC
Mob_net2	89.7	91.7	92.9	92.3	88.6
Inception_v3	89.0	95.1	89.2	92.1	85.7
CNN_v2	83.1	93.8	83.3	88.2	77.5
CNN_few layers	80.2	91.4	81.2	86.2	73.9

Figure 19: Model's Results in percentages (%)

Figure 20 shows the ROC curves for each model. If the curve is somewhere near the 50% diagonal line, it suggests that the model randomly predicts the output variable and therefore, the larger the area, the better the classifier. We can observe that the best Area Under Curve (AUC) is 88.6% of the "Mobile Net" model.

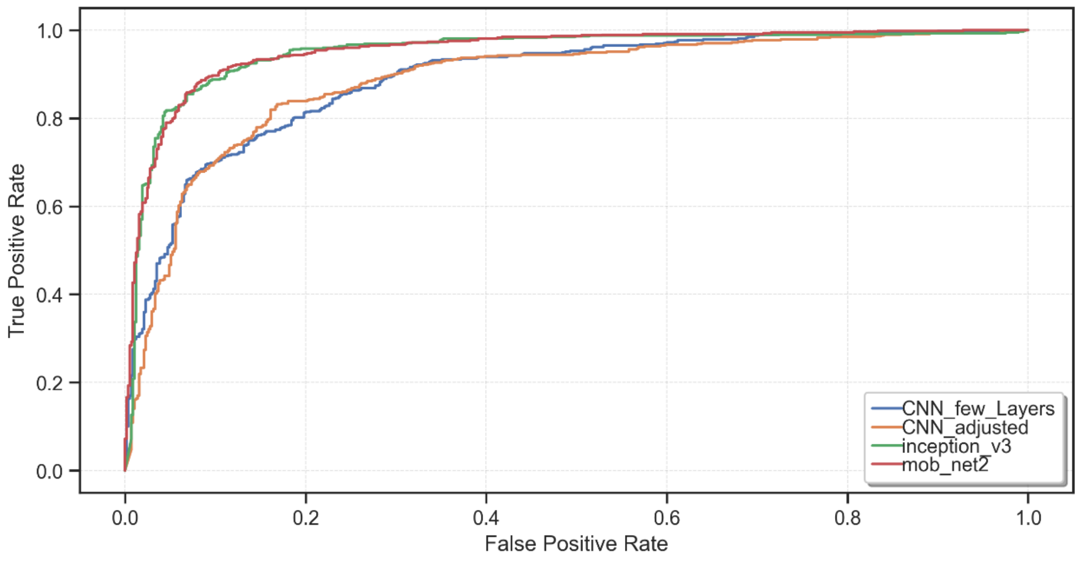


Figure 20: ROC Curve Comparison Plot

However, the accuracy & AUC curve should not be the only determinant metrics in the selection of the best model. The False Positive Rate is also a parameter that must be considered.

False Positive Rate (FPR) is the rate of the truth is negative, but the model predicts a positive, in our case when the motorcyclist is not wearing a helmet, but the prediction inaccurately reports that he is. In technical terms, the false positive rate is defined as the probability of falsely rejecting the null hypothesis, a Type I error in statistics.

We believe that the selected model to fulfill our case should has a lower score in False Positive Rate instead of False Negative Rate. Misclassifying pictures and just putting police in a place where most of motorcyclists wear helmets is less important than not sending police in a delinquency area.

From the figure below, we can see the model with the best combination of accuracy and False Positive Rate is the MobileNet2 as it has the highest accuracy 89,7% and the lowest False Positive Rate 7%. Therefore, this is the reason that we conclude with MobileNet2 as our best model.

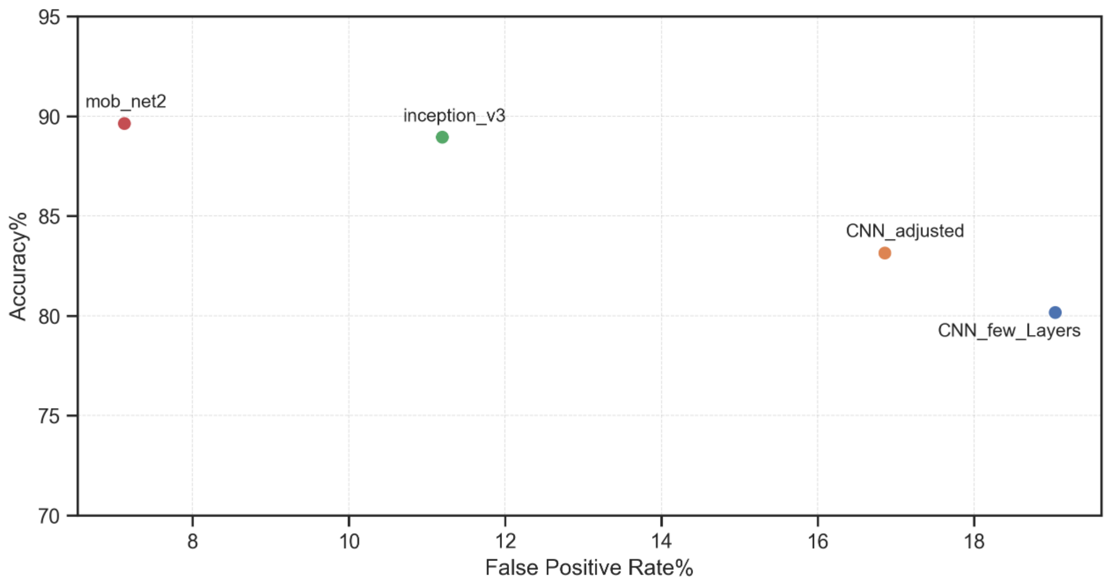


Figure 21: Model's results based on Accuracy and False Positive Rate in percentages (%)

In the Appendix Section we have reported some extra visualizations.

Below, we display some extra results of the InceptionV3 and the MobileNet2 models.

The confusion matrices of MobileNet2 and InceptionV3 are presented where we can see that most of images are classified correctly. For the MobileNet2 only 27 images are classified as wearing a helmet while there is a violation of the law.

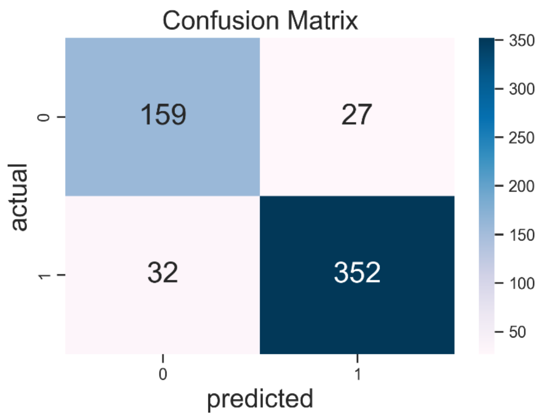


Figure 22: Confusion Matrix of MobileNet2 model

On the other side, from confusion matrix of InceptionV3, the respective number is 46.

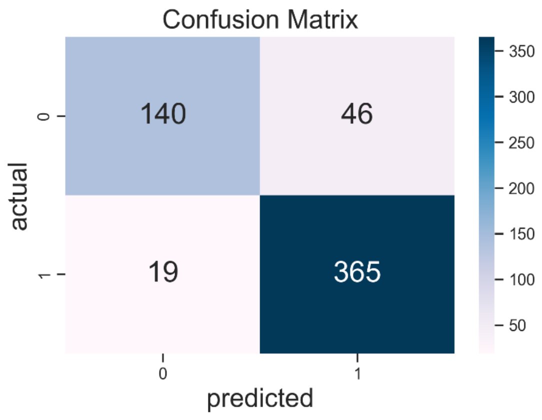


Figure 23: Confusion Matrix of InceptionV3 model

Qualitative & Error Analysis

In this section, we will verify the effectiveness of the proposed model through experiments.

Specifically, we have selected random images from web that are not included in the initial dataset to test the predictive ability of the model. Moreover, we revised the preprocessing steps to transform the image as a model input. As we can see below, our model predicts correctly even in cases where the motorcyclist holds the helmet in his hand or wears a headscarf, which can be easily misinterpreted as a helmet.

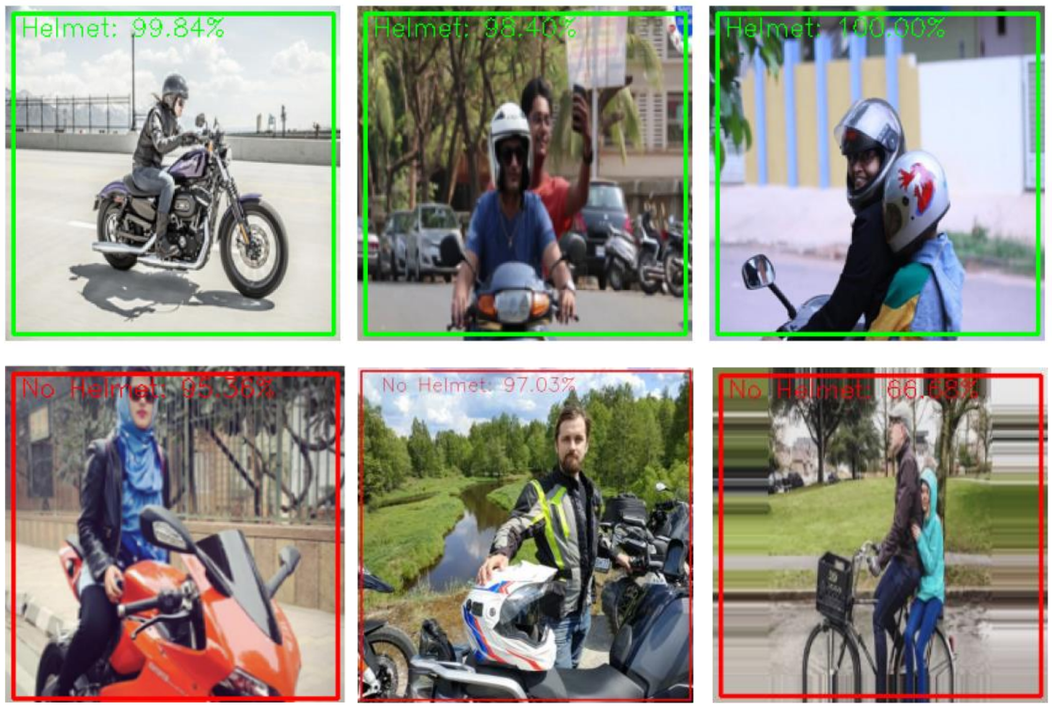


Figure 24: Correctly Classified Images

However, there are also cases where the model misclassifies the images. For example, black hair or black skin sometimes can be predicted wrongly as helmet. To minimize this error in the future, we can enrich the dataset with images from different nationalities.

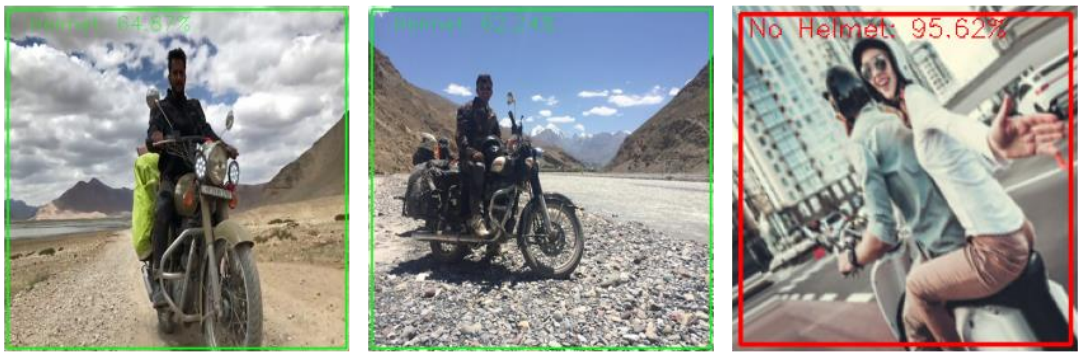


Figure 25: Wrongly Classified Images

Discussion, Comments/Notes and Future Work

In future study we aim to improve our model's efficiency by minimizing the research area for the algorithm. We will try to focus into models that detect first the motorbike and consequently the upper body of the motorcyclist. Then, the image classifier we have created will predict the label. In this view, we will eliminate the false alarms when a pedestrian is detected by the camera. In further analysis, a text recognition should follow to detect the license plate of the motorcyclist and automatically convert it into text. Finally, the police can identify the personal information of the offender and send him a fine.

Apart from that, we will increase the amount of training images and we will use a more diverse dataset. For example, we can add footages of people with different nationalities, races, and gender.

Following all this processes we can enhance our model and display a new approach of an end-to-end automated helmet detection model.

Members/Roles

The team consists of 4 members. Eftychia and Maria have postgraduate studies in mathematics, George in computer science and Konstantinos in business administration. All the members of the team contributed to all parts of this project. The main tasks that had been executed were:

- Data Collection (Konstantinos and Maria)
- Data Pre-Processing (George and Eftychia)
- Augmentation Techniques (Eftychia)
- Models Development (All)
- Report delivery (All)

Time Plan

Regarding the time management of this project, we spent almost a month in data collection and data processing in order to reach out the suitable form of the images in order to be used as an input in our models. During this month, our attention was also concentrated into deep dive on keras. During August, we focused on model developments. The last week was dedicated to write down all the methodology and processes used in the Report delivery. Overall, the estimated duration of the project was 2.5 months with 5 hours per week on average.

Bibliography

1. Bicycle Helmet Statistics
2. Greece Has Third Highest Motorcycle Deaths In EU – Greek City Times
3. The Ultimate Resource for Motorcycle Accident Statistics 2022 (motorcyclelegalfoundation.com)
4. How to Configure Image Data Augmentation in Keras (machinelearningmastery.com)
5. What is Data Augmentation? Techniques & Examples in 2022 (aimultiple.com)
6. ReLU (Rectified Linear Unit) Activation Function (opengenus.org)
7. A Complete Guide to Adam and RMSprop Optimizer | by Sanghvirajit | Analytics Vidhya | Medium
8. Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog
9. Different Types of CNN Architectures Explained: Examples - Data Analytics (vitalflux.com)
10. Building a Convolutional Neural Network for Image Classification with Tensorflow | by Sidath Asiri | Medium
11. Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015 | by Sik-Ho Tsang | Medium
12. MobileNetV2 Explained | Papers With Code
13. MobileNet Convolutional neural network Machine Learning Algorithms | Analytics Vidhya (medium.com)

Appendices

Model's Results

Below we can find some extra visualization of the performance of the models based on different metrics.

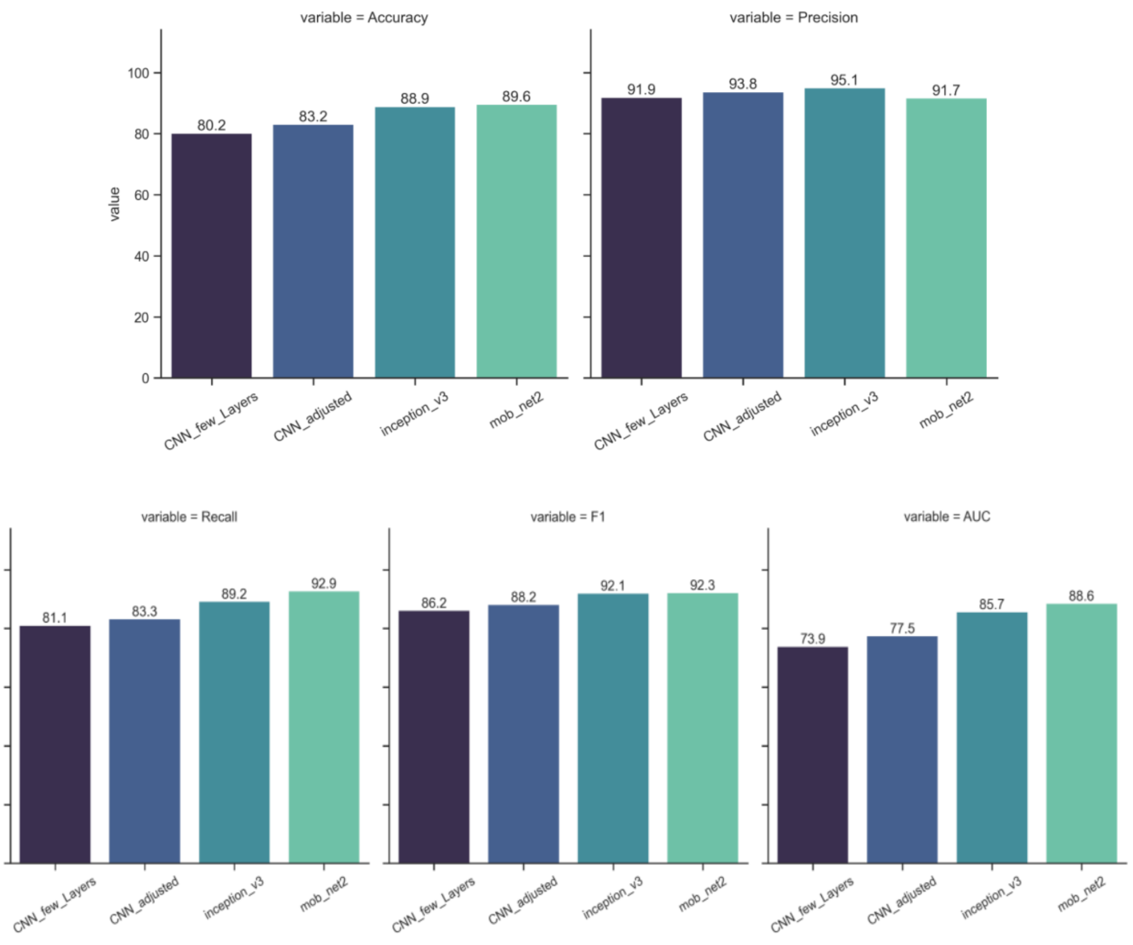


Figure 26: Model's results

Cropped images methodology:

Here we are going to execute a brief presentation of a model implemented on the cropped images as we discussed in the methodology section. First, we cropped the full images based on the annotations. So, the new dataset consists of photos depicting only the head of the motorcyclist and the label (“With Helmet”, “Without Helmet”) was extracted from the corresponding annotation object. As far the augmentation part, we have executed only rotation and brightness augmentation because the zoom augmentation will reduce the image resolution. Then we performed the same models we discussed above.

Below, we can see the goodness of fit of the InceptionV3 model to the cropped images data.

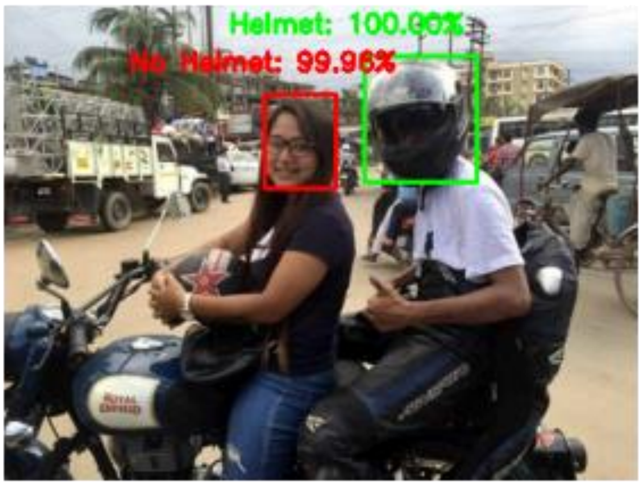


Figure 27: Cropped Image's model performance

However, to fulfill the project with the cropped images, a pretrained model to retrieve the bounding boxes of the upper body of the motorcyclist was needed. Although we tried to implement the pretrained model “haarcascade_upperbody”, we found out that it does not perform well in our data.