UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Artificial Intelligence and Data Engineering

PPE Detection

Project report

Luana Bussu

Academic Year: 2022/2023

Contents

1 Introduction

In recent years, the importance of personal protective equipment (PPE) has been underlined by its crucial role in ensuring the safety of workers in various industries, significantly reducing the risk of occupational accidents and illnesses. The ability to accurately detect and identify the presence of PPE in real time is therefore critical to maintaining a safe working environment.
This project report explores the application of Convolutional Neural Networks (CNNs) in the detection of PPE. An illustration is presented, comparing a custom-built CNN to a pre-trained CNN, evaluating their effectiveness in identifying different types of PPE items, as helmets, and jackets.

CNNs are a class of deep learning models that have shown considerable success in various computer vision tasks, including image classification and object detection. Taking advantage of their ability to extract complex features from images, the project's goal is to develop a PPE detection system capable of locating these devices on an image.
The goals of the project are twofold: first, to design and train a CNN from scratch to detect the presence of PPE in images. This approach allows to gain a deeper understanding of the underlying architecture of a CNN. Second, to use pre-trained CNN to leverage its existing knowledge for PPE detection. By comparing the performance of both approaches, we can evaluate the advantages and limitations of each.

The results of this project, when scaled up, can have significant implications for industries where PPE is crucial, such as construction, healthcare, manufacturing, and emergency services. A reliable PPE detection system can assist in monitoring, improve safety protocols, and potentially reduce the number of accidents and injuries.
The following sections of this report outline the methodology implemented for data collection and preprocessing, explain the architectural design of the CNN models, the training process and hyperparameter optimization, and conclude with a summary of results and recommendations for future enhancements.

2 Dataset

Roboflow is a powerful platform that simplifies the management and manipulation of image and video datasets, making it an invaluable resource for researchers, developers, and data scientists. With its extensive features and tools, Roboflow simplifies the complexities associated with handling large-scale datasets by providing functionalities such as data annotation, preprocessing, augmentation, and export capabilities.
One of the notable strengths of Roboflow is its support for various popular data formats commonly used in computer vision, ensuring compatibility with a wide range of tools and frameworks. For instance, Roboflow perfectly integrates with the widely adopted COCO format, which offers a standardized structure for representing object detection, segmentation, and keypoints annotations. This compatibility facilitates effortless integration with deep learning frameworks like TensorFlow and PyTorch.

The project dataset is structured in a main directory that includes three sub-directories: train, test, and validation. Each directory contains images of the related set and a CSV file containing annotations specific to that set.

	Worker	Visibility Vest	Helmet	**Total**
Training Set	1950	890	1791	**4631**
Validation Set	298	153	264	**715**
Test Set	641	333	577	**1551**
Total	**2889**	**1376**	**2632**	**6897**

Table 1: Dataset volumes

The annotation file provides information about the position and label of the various devices present in each image. Specifically, the position is expressed in coordinates using the PascalVOC format, which describes the bounding box coordinates as xmin, ymin, xmax, and ymax.

Figure 1: Example of dataset images

3 Data preprocessing

In this section, the data preprocessing steps are elaborated to prepare the dataset for training the PPE detection model. These steps include the selection of relevant images, cropping to isolate individual PPE instances, and the application of various transformations to improve data diversity and quality. All the cited image transformations were performed using Albumentations library.

3.1 Image Selection and Cropping

The initial dataset, obtained from Roboflow, contained images belonging to several classes so, the initial preprocessing step involved the selection of images that included at least one instance of the specified classes, helmets and vests. To maintain simplicity, a decision was made to focus on images featuring a single PPE device in each. Consequently, each original image was cropped to generate multiple images, each highlighting a single PPE item while preserving as much background context as possible.
To facilitate this process, a "generate_new_regions" function was implemented. For each image, this function identified non-overlapping regions suitable for a single bounding box (representing a PPE item). It determined the adjusted bounding box coordinates for these new regions, ensuring the highlighting of different PPE instances.
After acquiring the adjusted coordinates, the Albumentations library was employed to crop the original images to fit the new regions. The Albumentations library was chosen for its capability to easily apply transformations to both the images and the corresponding box coordinates simply specifying boxes format, as PascalVOC in our case. Subsequently, the cropped images were resized to a standardized dimension of 224x224 pixels.

Figure 2: Example of image cropping

To maintain data quality, images suffering from significant degradation due to cropping and resizing were manual removal. This step was crucial to prevent the inclusion of low-quality samples that might damage the training process.

Figure 3: Bad quality images

3.2 Data Augmentation

The resulting dataset consists of a number of images belonging to the two selected classes namely 'Vest' and 'Helmet'. These classes are highly imbalanced, with the 'vest' class having significantly fewer samples than the 'helmet' one. To mitigate this class imbalance, an oversampling technique is employed for the minority class.

The rebalancing process involved applying data augmentation transformations to the original images, leveraging the capabilities of the Albumentations library. These transformations included:
Geometric Transformations: this category included horizontal flips, vertical flips, and random 90-degree rotations. They help diversify the orientation of objects within the images, making the dataset more robust.
Color and Brightness Adjustments: random changes in brightness and contrast, color jittering (adjusting saturation and brightness), and converting images to grayscale are applied. These adjustments introduce variations in color and lighting conditions, further enhancing the dataset's diversity.

Figure 4: Data augmentation example

Figure 5: Data augmentation example

By selectively augmenting the minority class samples with these transformations, we aim to create a more balanced dataset. This balance is crucial for training robust machine learning models that perform well across all classes, regardless of their initial class distribution."

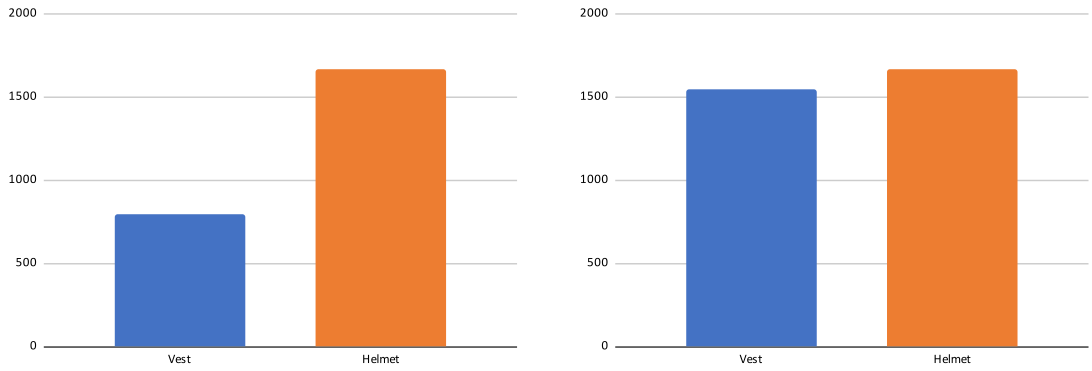

Figure 6: Classes count before and after augmentation

4 CCN from Scratch

The object detection using a CNN architecture built from scratch was built defying a standardCNN class as base and making changes and improvements to the model in order to improve network performances. It provides the basic structure and functionality required for performing image classification and object detection tasks.

Training data is generated in batches, and the model is trained for a given number of epochs using a combination of loss functions for the regression and classification heads, such as mean squared error and cross-entropy, respectively. Evaluation metrics include classification accuracy and Intersection over Union (7), measuring the overlap between predicted and ground truth bounding boxes. Throughout the following sections, various experiments will be presented, exploring changes to the architecture and hyperparameters of the base model trying to improve the results.

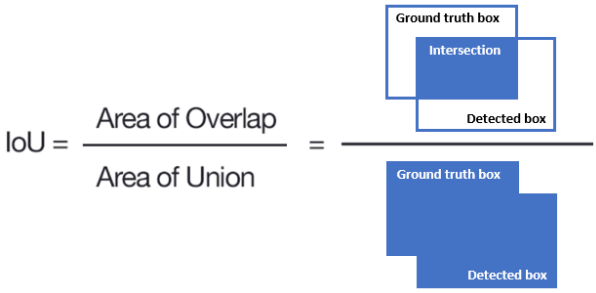

Figure 7: Intersection over Union

4.1 Base model

In the initial experiment, a standard convolutional neural network architecture was employed as the foundational model for the object detection approach. This architecture is represented by the **StandardCNN** class, serving as a starting point for the object detection task.

This base model consists of convolutional layers followed by pooling layers, which play a crucial role in extracting spatial features from images and reducing their dimensions. After these layers, a Flatten layer is employed, which transforms the multi-dimensional feature maps into a one-dimensional vector. Dense layers at the end of the network enable the model to make better decisions.
At the end of the network, there are two key components: a regression head, responsible for predicting bounding box coordinates using a *sigmoid* activation function, and a classification head, which determines class probabilities using a *softmax* activation function. The input to this network is an image with dimensions (SIZE, SIZE, 3), where SIZE represents both the width and height of the input image. In this case, SIZE is set to

224, and the 3 represents the three RGB color channels corresponding to Red, Green, and Blue. To ensure consistency in the input data, the first layer of the network is a Rescaling layer. This layer scales the pixel values of the input image to fall within the range [0, 1] by dividing each pixel value by 255.
The convolutional layers are responsible for learning hierarchical features from input images. The first convolutional layer employs 16 filters with a (3, 3) kernel size, utilizing "same" padding. It applies the $ReLU$ activation function and is followed by the first max-pooling layer featuring a (4, 4) pooling size with a stride of 4. Subsequent convolutional layers employ 32, 64, and 128 filters, respectively, each following the same configuration as the first layer.
Following the convolutional layers, the output is directed to a Flatten layer. This layer transforms the multi-dimensional feature maps into a one-dimensional vector, resulting in the complete loss of spatial information. This transformation potentially affected the model's capacity to identify patterns relying on spatial context. Additionally, the introduction of a Flatten layer may increase the risk of overfitting due to the substantial increase in parameters, which in this model exceeds 250K. As an alternative to the Flatten layer, an experiment involving GlobalAveragePooling is outlined in the following section to address these considerations.

Finally, the network presents two output heads: the classifier_head that outputs class probabilities using a Dense layer with NUM_CLASSES units and *softmax* activation function, and a regressor head that outputs four values representing bounding box coordinates using a Dense layer with 4 units and *softmax* activation.
The network is compiled using the Adam optimizer and a combination of loss functions: categorical cross-entropy for the classifier_head and mean squared error for the regressor_head. During experimentations, it became evident that the model faced challenges in generating bounding box coordinates. To address this issue, different weights were assigned to the loss functions, with a weight of 1 for the classifier and 15 for the regressor attempting to improve the performance of the regressor in predicting bounding box coordinates. The model underwent training for a total of 100 epochs, utilizing batches of training set data and undergoing validation with batches sourced from the validation set. Both datasets were generated through a function called *data_generation*, which extracted data from the dataset directory. During this process, bounding boxes were rescaled to ensure they fell within the range of [0, 1] by dividing by the image dimensions.

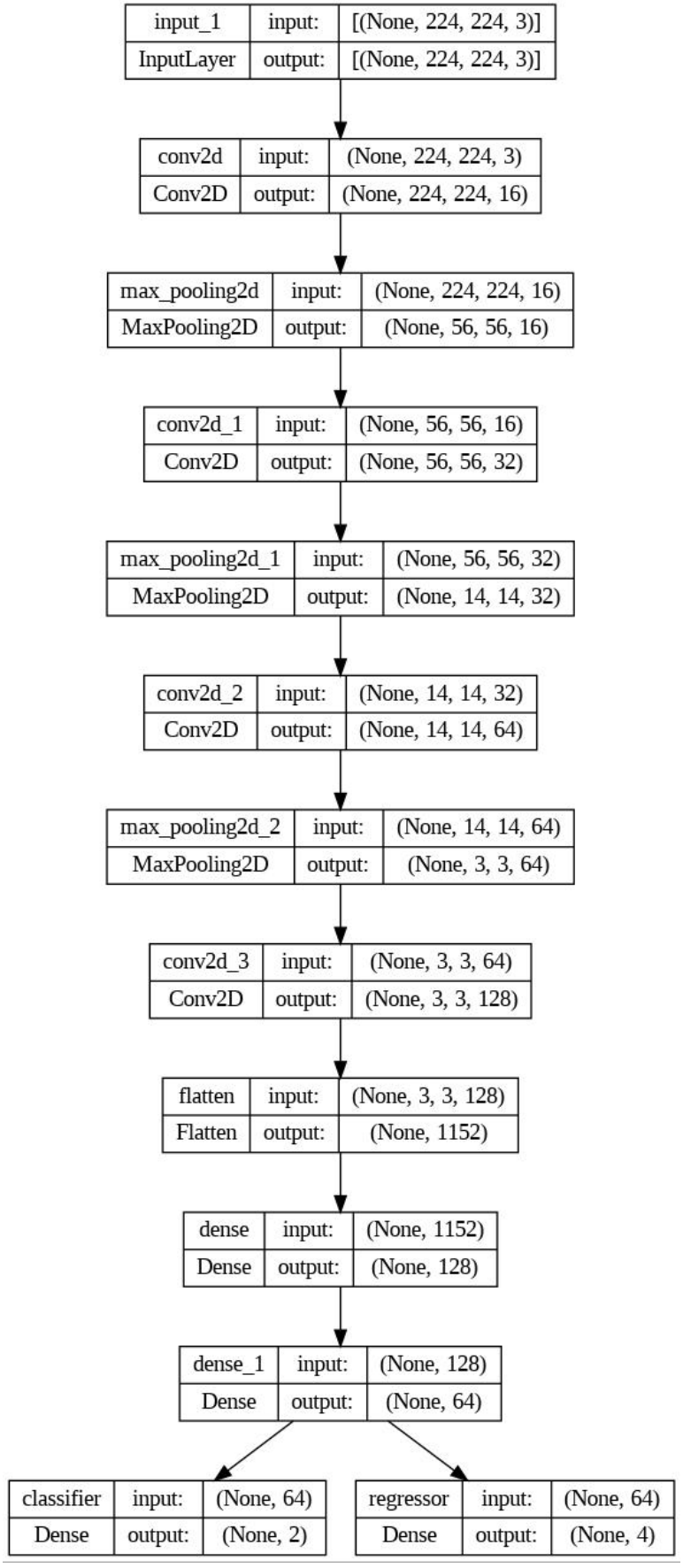

Figure 8: Base model architecture

After the training, the model achieves a training accuracy of 98,4% and a training IoU of 22%. However, when evaluating its performance on the validation dataset, a significant gap is observed, with an accuracy of 73% and an IoU of 22%. This different between the training and validation results indicate that the model overfits the training data and fails to generalize well as can be seen from figures 9 and 10.

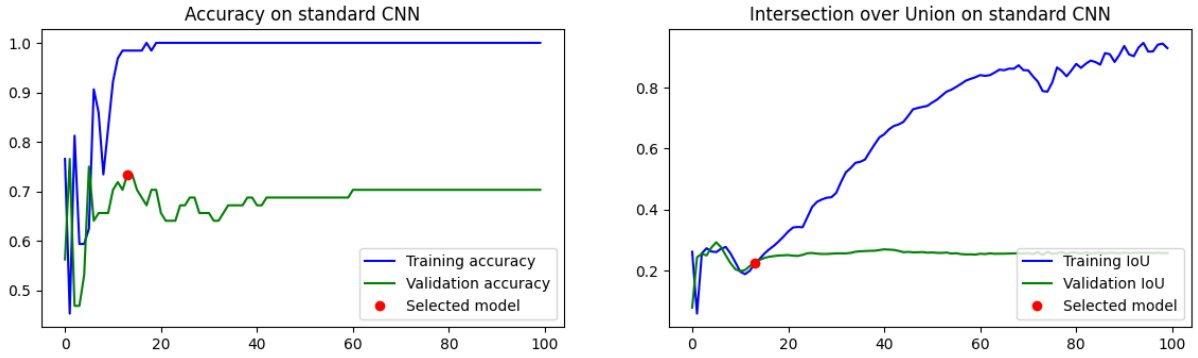

Figure 9: Accuracy and IoU of base model

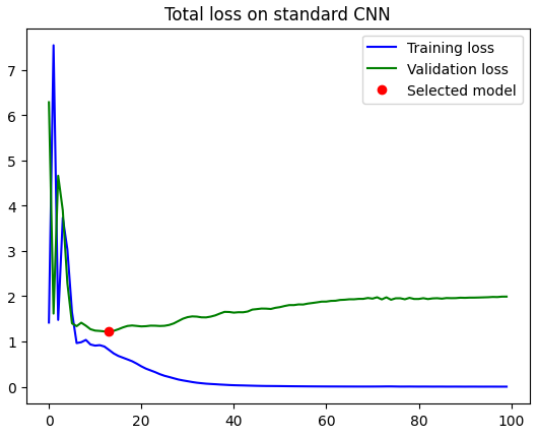

Figure 10: Base model Losses

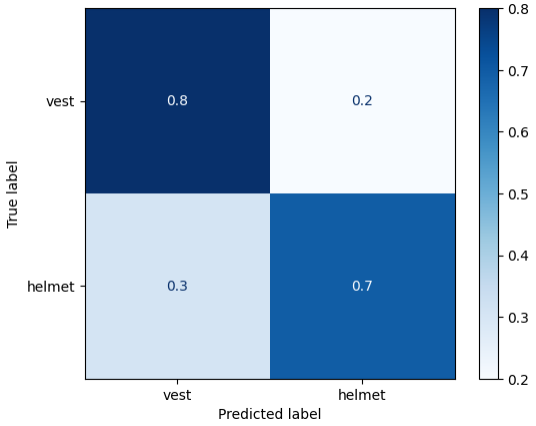

Figure 11: Confusion matrix

	Train	Validation	Test
Accuracy	98,4%	73,4%	70,3%
IoU	22%	22%	19%

Table 2: Base Model Performances

The overfitting problem can be due to several factors. Firstly, the limited size in the training dataset contributes to this issue. To mitigate it, expanding the dataset or acquiring more diverse data sources might be useful. Additionally, the model's complexity itself plays a crucial role in overfitting. The model's architecture, especially the use of a Flatten layer following the convolutional layers, results in an excessive number of parameters, exceeding 250K. This high level of complexity leads the model to memorize complex

details within the training data, thus compromising its ability to generalize well. One potential solution to mitigate this risk is to replace the Flatten layer with Global Average Pooling, which retains some spatial information while reducing parameter counts, making the model less prone to overfitting. Alternatively, employing Dropout layers could also help address overfitting concerns.

4.2 Base model with GlobalAveragingPooling

In this section, the integration of the GlobalAveragePooling2D layer is explored as a replacement for the Flatten layer in the model architecture. Instead of simply flattening the feature maps into a one-dimensional vector, the GlobalAveragePooling2D layer calculates the average value for each feature map across the entire spatial dimension. This leads to a single value per feature, which not only reduces the spatial dimensions but also plays a crucial role in parameters reduction that pass from 250K to just over 100K leading to several benefits. Efficiency is improved, resulting in faster training times. Preserving some spatial information via GlobalAveragePooling2D is useful in tasks such as object detection where the location of features is important for accurate predictions. Most importantly, the reduced model complexity serves as a first defense against overfitting.

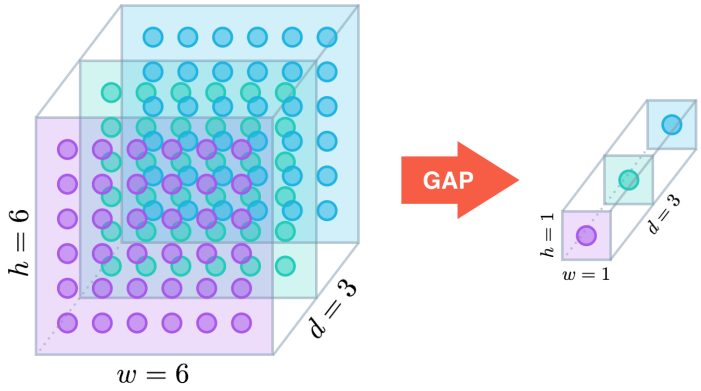

Figure 12: Global Average Pooling

To ensure a fair comparison of results, the model was compiled and trained in a consistent way with the previous experiment. This involved employing the Adam optimizer, training over 100 epochs, and maintaining same loss weights, with a weight of 15 for the regressor and 1 for the classifier.

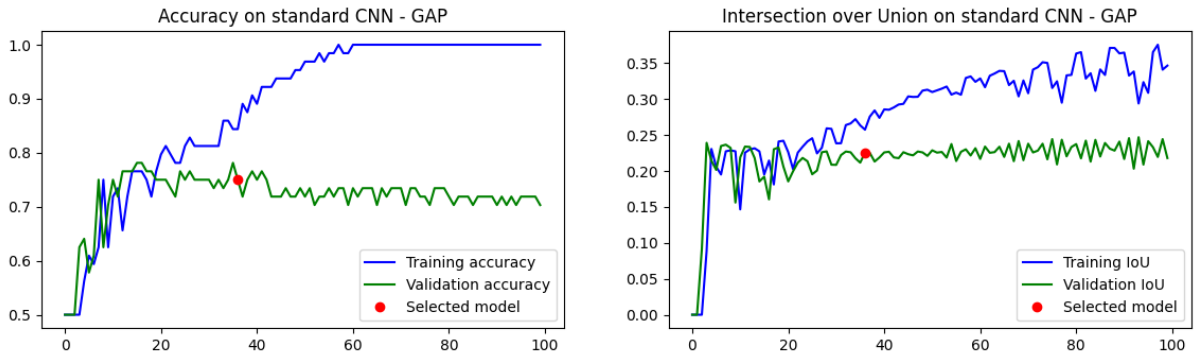

Figure 13: Accuracy and IoU of base model with GAP

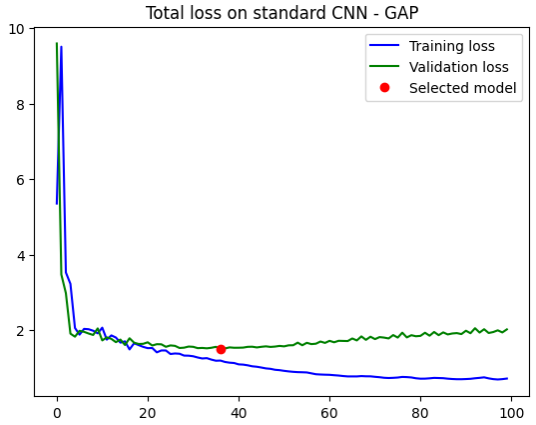

Figure 14: Base Model Losses with GAP

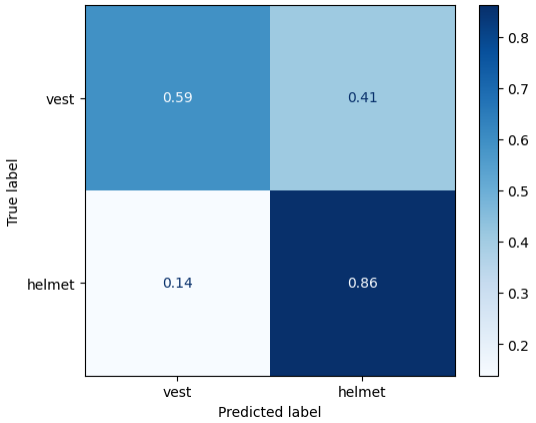

Figure 15: Confusion matrix

	Train	Validation	Test
Accuracy	84,4%	75%	74%
IoU	25,7%	22,5%	20,9%

Table 3: Base Model with GAP Performances

As expected, the model's performances show a small reduction in overfitting and a few increase in results with respect to the model with the Flatten layer, as evidenced by the results on the validation, train and test sets. Test set accuracy has increased from 70,3% to 74% and IoU has increase from 19% to 20,9%. Additionally, it's worth noting that this model reached best results at epoch 36 during training, while the model with Flatten stopped at epoch 13. This conferms the model with GlobalAveragePooling2D has benefited from reduced overfitting, resulting in improved generalization on the validation and test data. In the next section, the addition of dropout layers to this model will be analyzed trying to further reduce overfitting and potentially achieve improved performance.

4.3 Dropout model

In this section, the Dropout Model is explored, representing an extension of the GlobalAveragePooling model with the incorporation of a Dropout Layer. The goal of this experiment is to further mitigate overfitting and try to improve performance. The Dropout Layer are placed after the Dense layers with a Dropout rate of 0.5, which means that during training, approximately half of the neurons in these layers are temporarily deactivated, randomly chosen at each training step.

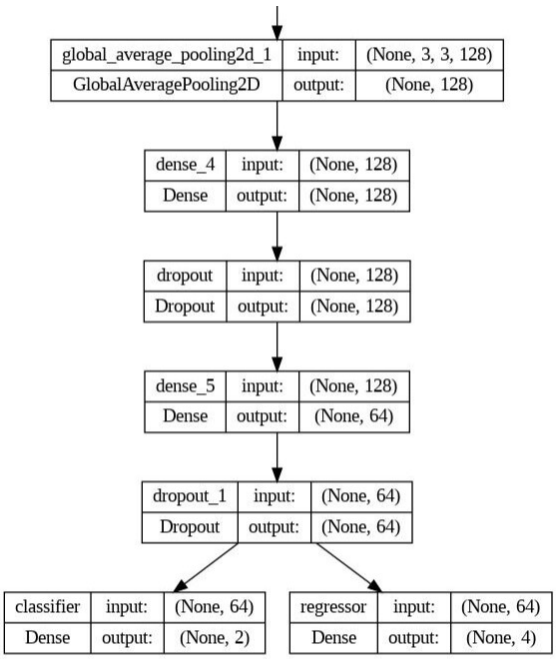

Figure 16: Dropout model architecture

Regarding the training process, initially, the model was trained for 100 epochs as before. However, it became evident that the performance curves continued to rise. As a result, the training epochs were extended to 400, while keeping the rest of the configuration unchanged

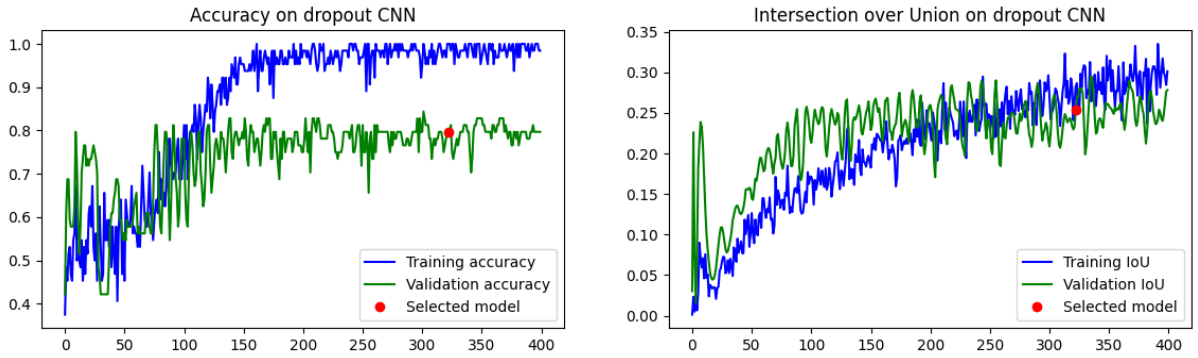

Figure 17: Accuracy and IoU of Dropout model

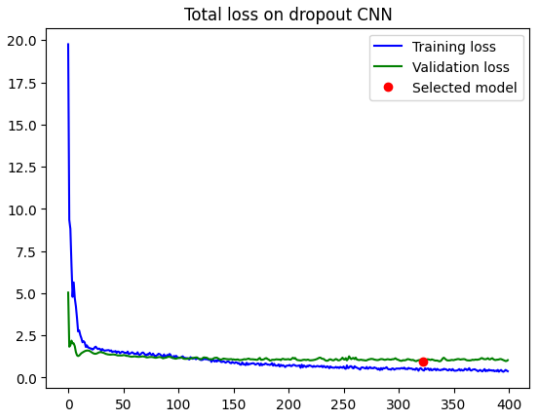

Figure 18: Dropout model Losses

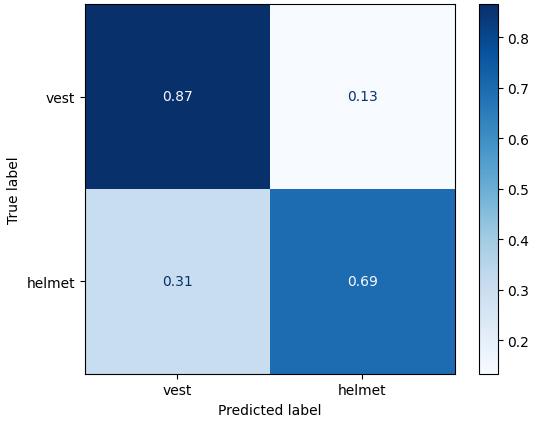

Figure 19: Confusion matrix

	Train	Validation	Test
Accuracy	100%	79,7%	78,6%
IoU	28,2%	25,4%	22,3%

Table 4: Dropout Model Performances

The inclusion of Dropout Layers within the model architecture has notably influenced its performance. These layers helped diminishing overfitting, as indicated by the narrower disparities between training and validation/test accuracy, as well as IoU scores. This demonstrates an improved ability to generalize the model to previously unseen data, meaning greater robustness. Furthermore, the test accuracy and IoU have demonstrated improvements compared to prior experiments. Starting from the Dropout model which given better results so far, an hyperparameters tuning was made in the following section trying to further improve model scores.

4.4 Hyperparameters Tuning

After identifying the best model through prior experiments, the Keras Tuner was employed for further improve performance. The tuner was used to fine-tune several critical hyperparameters:

- **Learning Rate:** the learning rate for the optimizer was adjusted to fine-tune the rate at which the model updated its parameters during training. Options included 1e-2, 1e-3, 1e-4, and 1e-5.
- **Dropout Rate:** it was tuned, allowing values between 0.3 and 0.8 with a step size of 0.1.
- **Loss Weights for Classifier and Regressor:** to balance the contribution of the classifier and regressor components of the model, loss weight ranged from 1.0 to

10.0 with a step size of 1.0, and the regressor loss weight ranged from 10.0 to 20.0 with a step size of 1.0.

The Keras Tuner used the Hyperband algorithm for the tuning process, aiming to minimize validation loss. Each trial was limited to a maximum of 10 epochs.
Upon completing the hyperparameter search, the best set of hyperparameters was extracted from the tuner's results, inluding:

- **Learning Rate:** 0,0001
- **Dropout Rate:** 0,4
- **Loss Weight for Classifier:** 1.0
- **Loss Weight for Regressor:** 10.0

This comprehensive hyperparameter tuning process was conducted to fine-tune the model's architecture and training configuration, ultimately achieving the best possible performance for the given task.

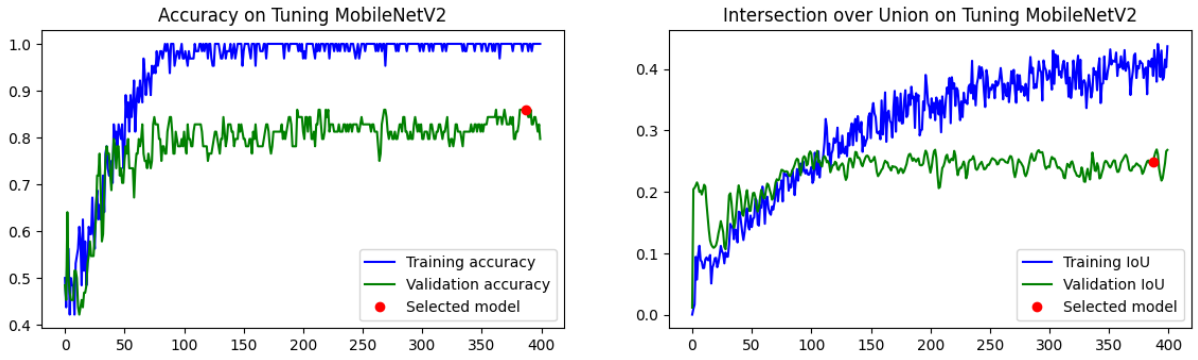

Figure 20: Accuracy and IoU of Hyperparameters tuning

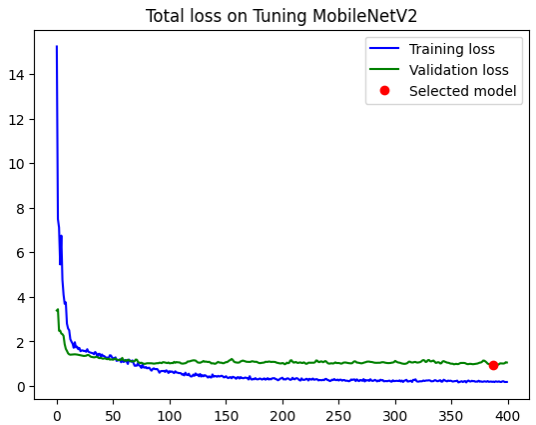

Figure 21: Hyperparameters tuning Losses

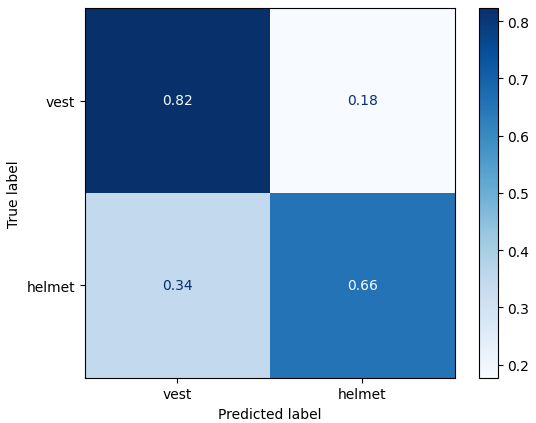

Figure 22: Confusion matrix

	Train	Validation	Test
Accuracy	100%	85,9%	78,1%
IoU	38,9%	24,9%	25,9%

Table 5: Hyperparameters tuning Performances

The performance on the test dataset, with an accuracy of 78.1% and an IoU of 25.9%, represents the best results achieved so far. This signifies the model's enhanced ability to accurately classify objects and predict bounding box coordinates.

5 MobileNetV2

MobileNet is a convolutional neural network architecture that is designed to be lightweight and efficient for mobile and embedded devices. It was developed by Google and introduced in 2017 as a solution to enable deep learning applications on resource-constrained devices with limited computational power and memory. MobileNet achieves efficiency by splitting the usual convolution into two steps: one called *Depthwise Convolution* phase that processes each input channel separately with separate 2D filters applied to each input channel and maintaining the same number of channels but reducing spatial dimensions, and another called *Pointwise Convolution* phase that combines the output of each channel using a 1x1 convolution to create the final output. This approach reduces computation and model size while keeping accuracy reasonable.

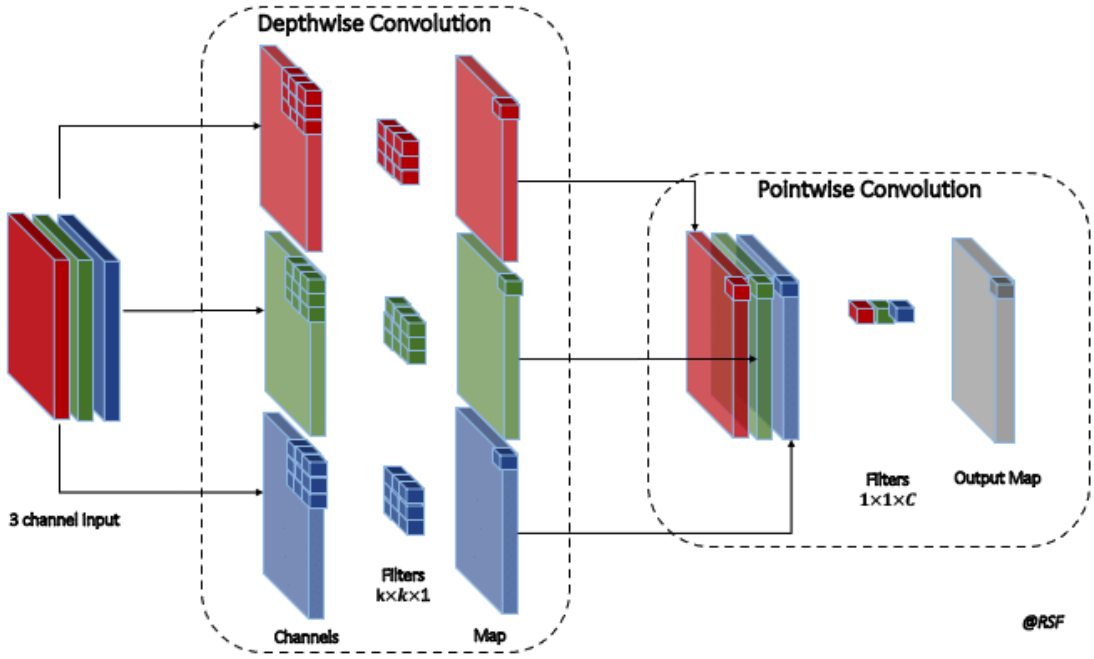

Figure 23: MobileNetV2 Convolution

MobileNet has proven to be effective for various computer vision tasks, including object detection, for this reason various experiment were exploited using this network. To use MobileNet as a base model for an object detection problem, it can be used the "one-shot detection" approach where, starting from the pre-trained MobileNet model, the final classification layer is removed, and new layers for both classification and regression are added. By adding these additional layers to the pre-trained MobileNet model, the network can be fine-tuned on the specific object detection task. The pre-trained weights of MobileNet provide a good initialization for the network, allowing to converge faster and achieve accurate results with less training data.

5.1 Experiment 1

In this experiment, the MobileNetV2 architecture was employed as the base model with pre-trained weights obtained from the "Imagenet" dataset. The top classifier layer was excluded and a GlobalAveragePooling layer was added to capture spatial information and reduce dimensionality. A dense layer with 128 units and *relu* activation was introduced and, to mitigate overfitting, also a Dropout layer with a rate of 0,6. As before, the model included two output heads for classification and regression. The model was compiled using Adam optimizer and using as loss weights 1 for the classifier and 12 for the regressor and it was trained for 400 epochs.

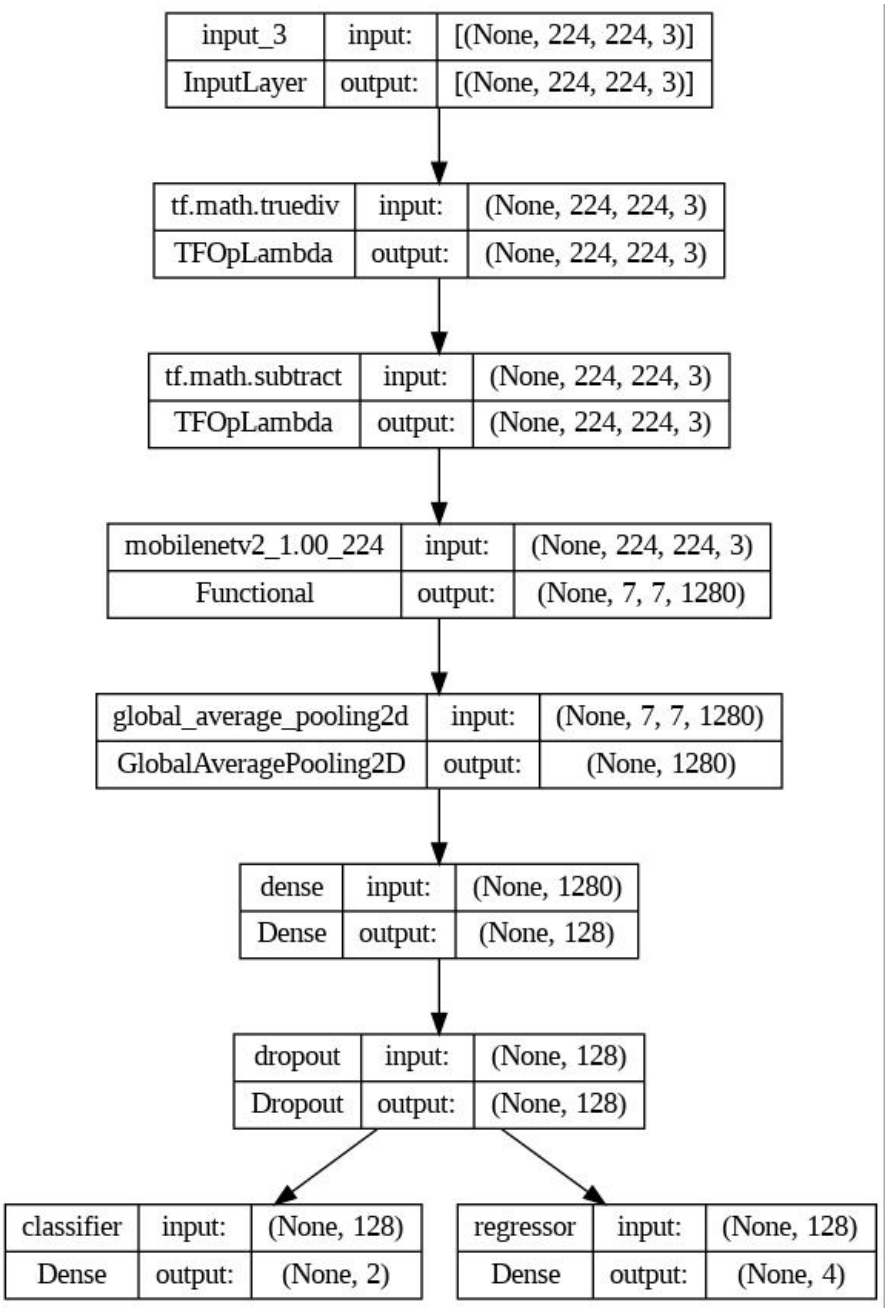

Figure 24: Base model architecture

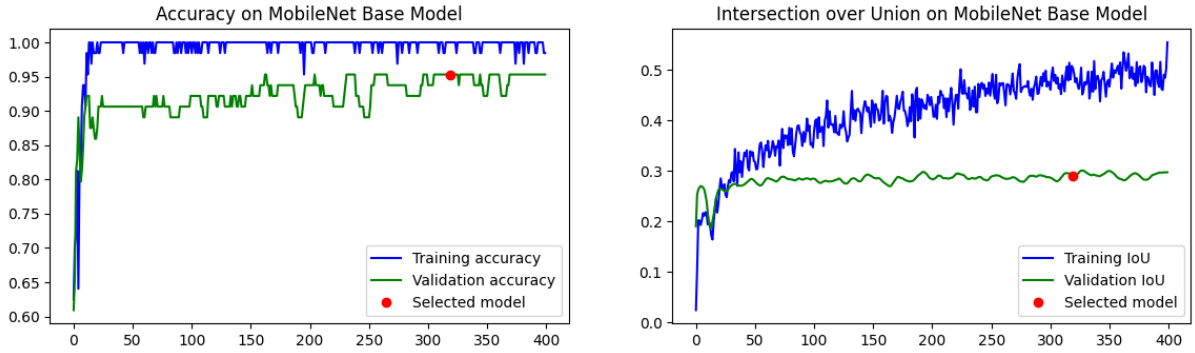

Figure 25: Accuracy and IoU of MobileNetV2 - First Experiment

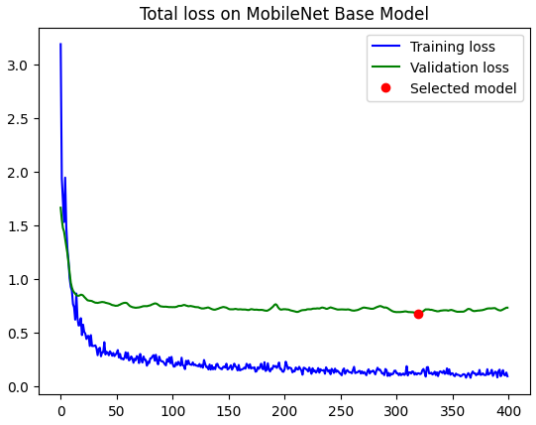

Figure 26: Losses of MobileNetV2 - First Experiment

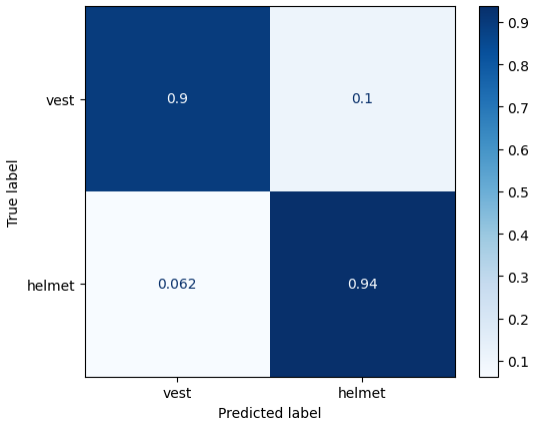

Figure 27: Confusion matrix

	Train	Validation	Test
Accuracy	100%	95,3%	93,2%
IoU	50%	29%	24,5%

Table 6: MobileNetV2 - Performance Experiment 1

The results reveal a substantial improvement in both classification and bounding box prediction tasks, the validation and test accuracy metrics, reaching 95.3% and 93.2%, respectively, highlight the model's improved generalization capabilities, same for the test IoU scores, reaching 24.5%.

5.2 Experiment 2

In this section, a modified version of the base MobileNetV2 model was explored adding an extra Dense layer followed by another Dropout layer. This experiment aimed to test whether the increased model complexity achieved by introducing an additional layer could lead to performance improvements in the object detection task.

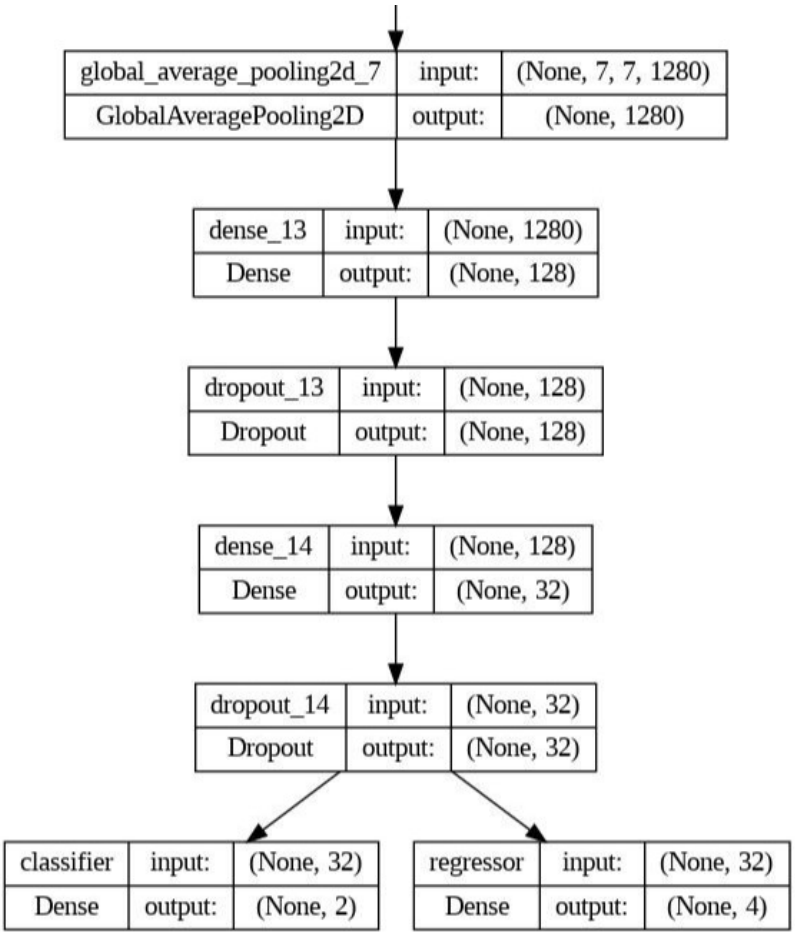

Figure 28: Bigger model architecture

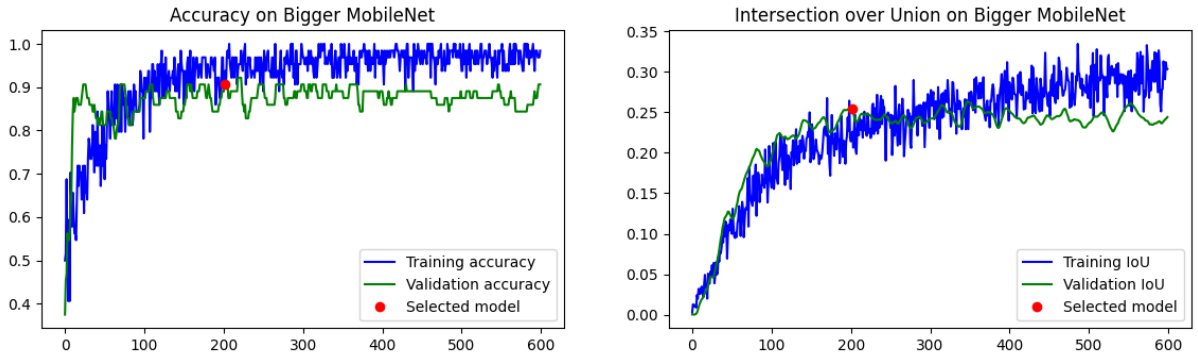

Figure 29: Accuracy and IoU of MobileNetV2 - Second Experiment

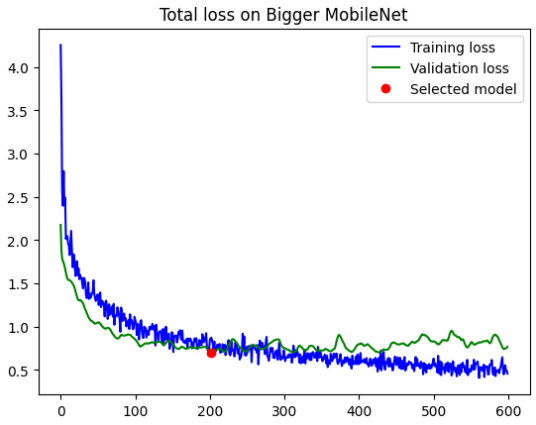

Figure 30: Losses of MobileNetV2 - Second Experiment

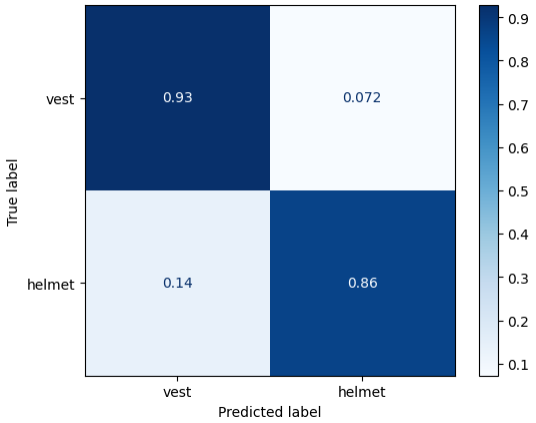

Figure 31: Confusion matrix

	Train	Validation	Test
Accuracy	93,8%	90,6%	91,1%
IoU	22,6%	25,4%	21,5%

Table 7: MobileNetV2 - Performance Experiment 2

Although the model demonstrated respectable accuracy and IoU scores, it failed to surpass the results obtained in the previous section. The findings suggest that, in this context, increasing model complexity by adding an extra Dense layer did not yield substantial improvements.

5.3 Hyperparameters Tuning

Similarly to what has been done for the scratch CNN, the hyperparameters tuning was employed also for the MobileNetV2 model of 5.1 trying to find the best value for *learning rate*, *dropout rate*, *classifier loss weight* and *regressor loss weight*. The tuning process was carried out as before and founded:

- **Learning Rate:** 0,0001
- **Dropout Rate:** 0,5
- **Loss Weight for Classifier:** 1.0
- **Loss Weight for Regressor:** 11.0

This comprehensive hyperparameter tuning process was conducted to fine-tune the model's architecture and training configuration, ultimately achieving the best possible performance for the given task.

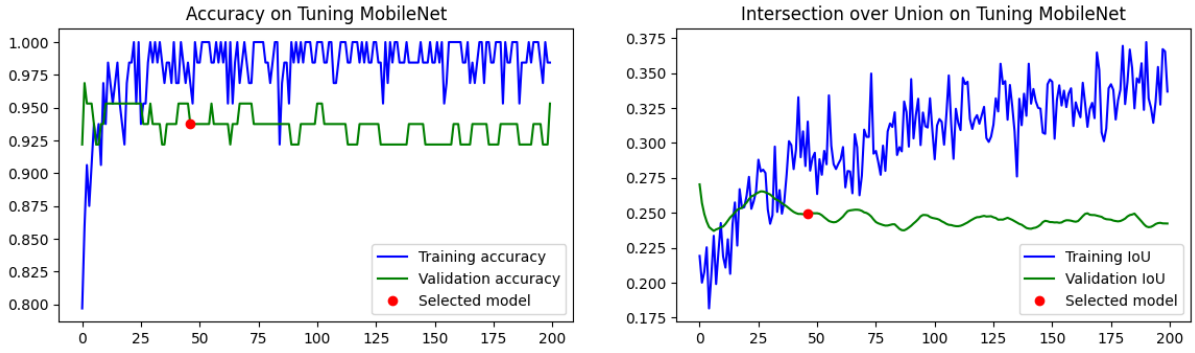

Figure 32: Accuracy and IoU of Tuning model

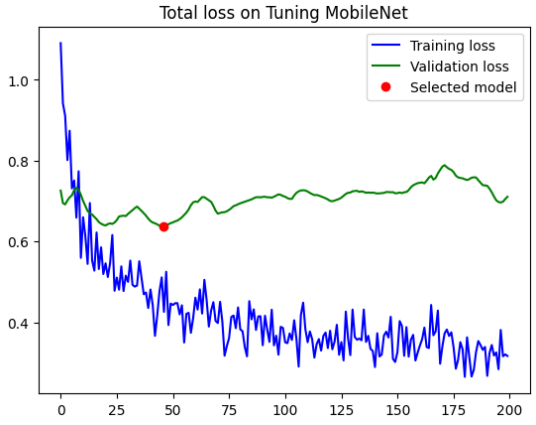

Figure 33: Losses of MobileNetV2 - Hyper-parameters tuning

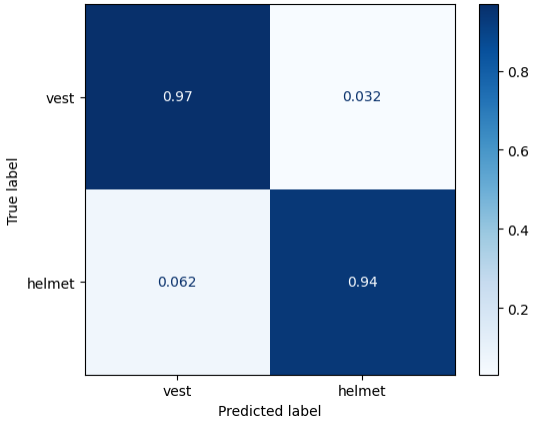

Figure 34: Confusion matrix

	Train	Validation	Test
Accuracy	96,9%	93,8%	96,4%
IoU	31,5%	24,9%	24%

Table 8: MobileNetV2 - Performance Hyperparameter tuning

Reaching a test accuracy of 96,4% and a test IoU of 24% this is the model with best performances obtained so far as shown in 35.

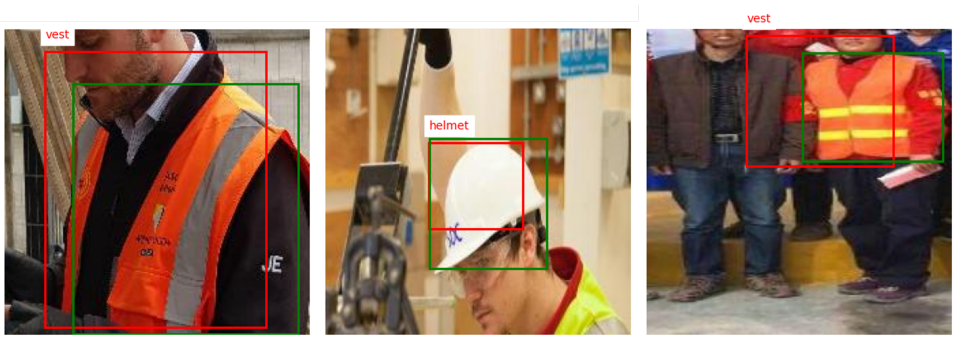

Figure 35: Model results

6 Conclusion

The document presents a series of experiments carried out to address the problem of object detection and classification of PPE. The study involved two steps: the first was exploring a scratch implementation, involving various architectural designs and techniques. Subsequently, the focus shifted to using a pre-trained MobileNetV2 model for the same task.

The results showed a notable performance advantage when using pretrained networks compared to starting from scratch. This difference is due to the ability of pre-trained networks to leverage a large number of parameters trained on large datasets. In contrast, the scratch network, being trained on a relatively smaller dataset, faces inherent limitations in feature extraction and generalization. Despite the better performance, the model using the pre-trained network also failed to achieve excellent results in bounding box detection. To imorove the study's applicability in real systems, several improvements can be considered:

- Two-Step Detection and Classification: one approach involves trying improve performances first detecting a PPE target within an image and subsequently, the identified region can be subjected to a classification task. This two-step approach can potentially boost accuracy and localization precision.
- Multi-Target Detection: another area for improvement is extending the project's capabilities to detect images containing multiple PPE targets. This improvement would be particularly valuable in real-world scenarios where multiple workers or objects of interest coexist in a single image. This can be achieved by adopting object detection models that are well-suited for multi-object scenarios as Faster R-CNN and YOLO.

References

[1] Wadii Boulila, Ayyub Alzahem, Aseel Almoudi, Muhanad Afifi, Ibrahim Alturki, and Maha Driss. A deep learning-based approach for real-time facemask detection. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1478–1481, 2021.

[2] Alexis B. Cook. Global average pooling layers for object localization, 2017.

[3] Hyperband tuner. https://keras.io/api/applications/mobilenet/.

[4] Mobilenet, mobilenetv2, and mobilenetv3. https://keras.io/api/applications/mobilenet/.