UNIVERSITÀ DI PISA

COMPUTATIONAL INTELLIGENCE AND DEEP LEARNING PROJECT

ACADEMIC YEAR 2022-23

Massagli Lorenzo

Patimo Antonio

Buchignani Fabio

TABLE OF CONTENTS

1. INTRODUCTION

The project is an attempt to build a neural network able to perform face detection on images containing single faces and classify each face in mask/no-mask. We implemented the object detector as a CNN with two heads, a classification head that outputs the class of the object and a regression head that outputs the predicted coordinates of the bounding box enclosing the face, the architecture was made from scratch and its results are compared with pretrained networks, using the technique of transfer learning to adapt the network to our problem. This report collects the analysis performed and considerations made for each experiment.

2. OBJECT DETECTION OVERVIEW

Object detection is an image-processing related task where the main objective is to detect which parts of the image contain salient objects and classify those objects. Each object is defined in terms of a class, that indicates the category to which the object belongs, and the information related to where the object is positioned inside the image. Most object detectors associate each object with a so-called bounding box, which is ideally simply the smallest square that covers the object completely. The coordinates of each bounding box are given in terms of pixel in the image, following two possible conventions:

- The x,y coordinates of the pixel which is the center of the box, and width and height in pixels.
- Xmin, Xmax, Ymin and Ymax, which are the x-axis coordinates and y-axis coordinates that correspond to edges of the box.

2.1 Architectures

Object detectors architectures can be divided in two main frameworks:

- Two step architectures, also called region proposal-based frameworks.
- One step architecture, also called regression/classification based frameworks.

The difference with these two classes of algorithms, is that in the region proposal-based framework interesting regions of an image (regions that could contain an object) are found at first, and then a CNN extract features from each region and classifies the region (if an object is detected inside the region). Examples for this approach are R-CNN and FPN architectures.

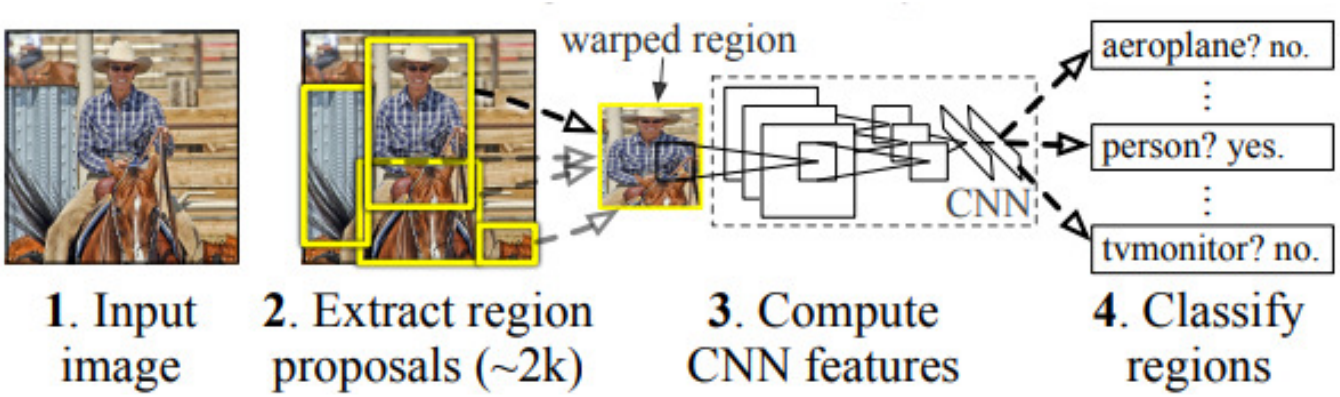


Figure 1: R-CNN architecture

In the second approach a CNN is used at first for feature extraction, and the extracted features are used to predict bounding boxes and classes at the same time. These algorithms include YOLO architecture (You Only Look Once) and SSD (Single-Shot MultiBox Detector).

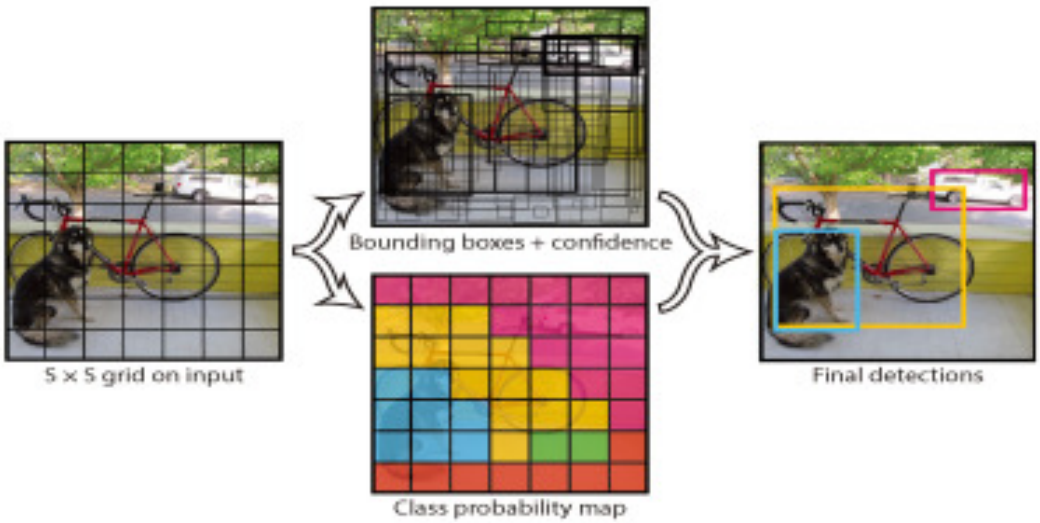


Figure 2: YOLO architecture

2.2 Training method and performance

In the object detection task, the ground-truth is composed of manually annotated bounding boxes and categories for each object in the image. Given that, there's the need to define a loss function and a way to measure the performance of the network which are tuned specifically for this task.

The loss function is a combination of two losses, the classification loss, and the localization loss. The baseline for the classification loss is the classical cross-entropy based loss function. For the localization loss, a common approach is the Smooth-L1 loss function (or a variant called Huber Loss).

The Smooth-L1 loss function acts independently on each of the four dimensions defining a bounding box: given Vtrue and Vpredicted (they could be for example the position of the center of the box on the x-axis) this function is computed as follows:

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

Or in the Huber variant as:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$

The typical value for the threshold is one, and x is the difference between Vtrue and Vpredicted. The overall localization loss function can be the sum of the smooth-L1 loss for all the four values representing a bounding box. This loss function combines advantages coming from L1 loss (the gradient is steady for large values of x, which means large differences between Vtrue and Vpredicted) and from L2 loss (limits oscillations around 0, when we have small differences between Vtrue and Vpredicted). On the other side, the problem with this loss function is that the four values are accounted separately, as if there was no correlation among them.

For what regards the performances of the object detector, accuracy is not a direct option because object detection is not only a classification task, but a more complex one, that also includes regression on bounding boxes.
Intersection over Union can be used as a metric to understand the closeness of the prediction to the ground truth. The IoU is computed as the intersection area of the two bounding boxes, the ground truth and the predicted one, over the union area.

2.3 Face-mask detection problem

Because of the limitations of Google Colab, creating an architecture from scratch capable of performing multiple object detection was not feasible, because it would have taken too many resources in terms of training time and capacity of the network. The idea was instead to perform single object detection, where the assumption is that the images fed to the network contain a single face, and each face must be located inside the image and classified in “with mask” or “without mask”. To perform this task, instead of relying on a complex object detection architecture, we could adopt a simple multi-headed CNN to perform feature extraction, with a head in charge of locating the face inside the image (outputting the coordinates of the bounding boxes) and a standard classification head to perform classification.

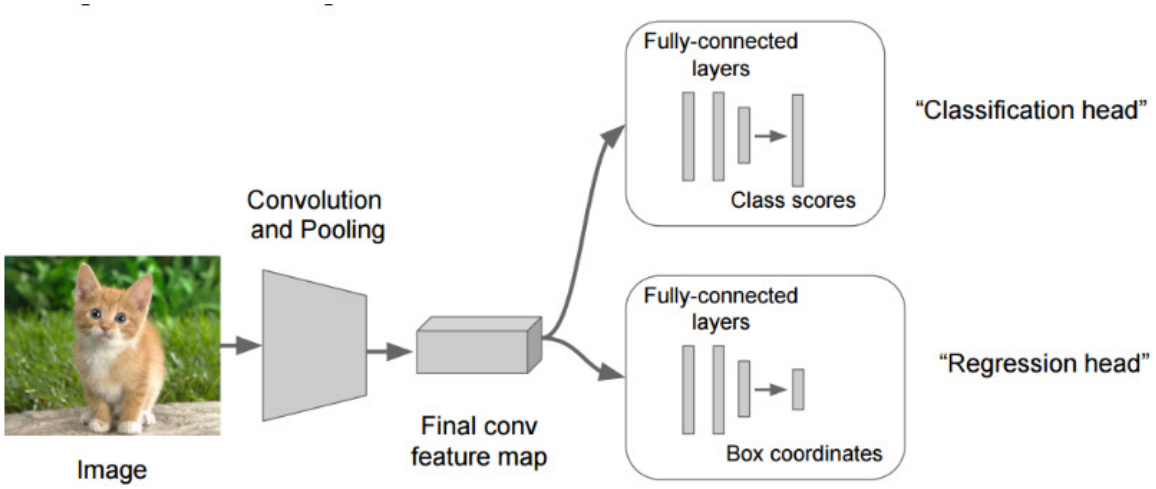


Figure 3: CNN architecture

The CNN used for feature extraction could be also replaced by a pretrained network. As regards the performances, it makes sense to use accuracy as a measure for the classification head and the previously presented Intersection over Union metric for the regression head. The best architecture will be the one that has the best trade-off between these two measures.

3. DATASETS

The dataset used is available at this link:
https://www.kaggle.com/datasets/andrewmvd/face-mask-detection
It consisted of 853 images containing multiple faces. Each face was assigned a bounding box and was categorized in three classes:

- With mask
- With mask, incorrectly worn
- Without mask

Because the second class, “With mask, incorrectly worn”, contained very few examples, we decided to simplify and assign those faces to the class “With mask”.
This dataset was thought for multiple object detection, thus a preprocessing step was needed to crop each single face out of the entire image and reassign bounding boxes accordingly.

3.1 Dataset preprocessing

Sample images in our dataset are like the following:

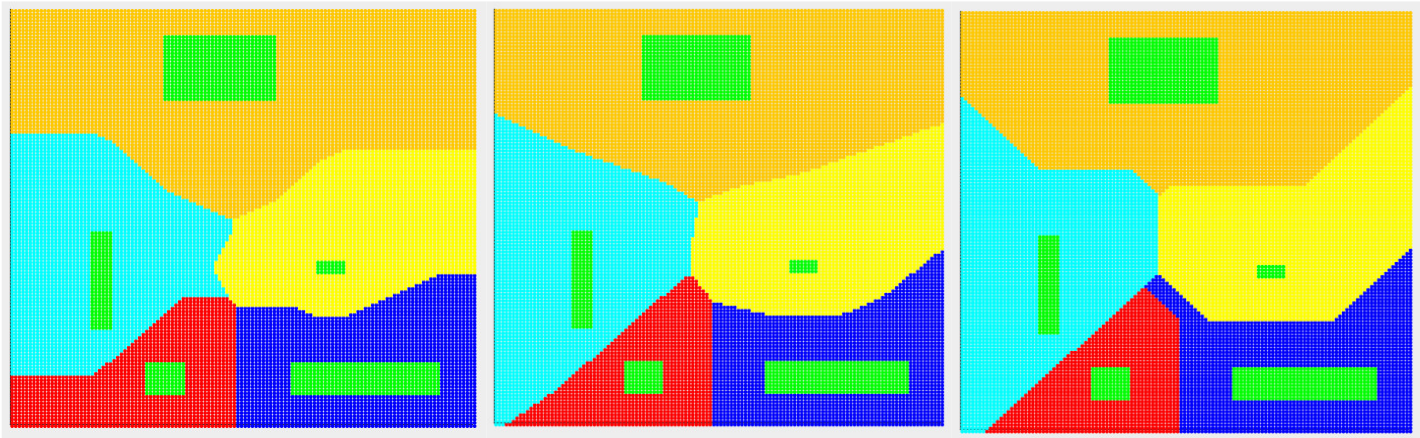
Figure 4: image of the dataset with represented ground truth bounding boxes

Figure 5: image of the dataset with represented ground truth bounding boxes

We have multiple bounding boxes for each image, in particular a bounding box (a set of coordinates) for each face in the image and an associated label. A possible approach was to manually crop images and reassign bounding boxes from scratch. Instead, we decided to exploit the information contained in the ground truth to perform an automatic cropping technique. The assumptions we made are the following:

- We want each cropped image to be rectangular
- We want each cropped image to contain just a face and as much background as possible
- We want to exploit as much as possible the original image

With these assumptions what we would like to do, theoretically, is to partition the image in n partitions, given n the number of faces of the image and assign each pixel of the image to the nearest bounding box according to some distance metric. However, using this method directly isn't of any help because the partitions obtained could be of any shape and not only rectangular.



In the pictures above we considered a trial image, with some bounding boxes represented in green. We partitioned the image pixels with respect to the nearest bounding box, and we changed the definition of distance between each pixel and each bounding box:

- In the leftmost figure, we used the Manhattan distance between the pixel and the nearest pixel belonging to the bounding box as distance measure.
- In the center figure, we used the Euclidean distance between the pixel and the nearest pixel belonging to the bounding box as distance measure.
- In the rightmost figure, we used the Chebyshev distance between the pixel and the nearest pixel belonging to the bounding box as distance measure.

As we can see, none of the distance metrics used were able to give us rectangular partitions. Nevertheless, we observed that, when using the Manhattan distance, the partitions obtained had a property: considering the rectangular shape with borders equal to the coordinates of the leftmost, rightmost, topmost and bottommost pixel of the partition, the obtained rectangle enclosed the bounding box assigned to the partition and never enclosed entirely other bounding boxes. In most cases, the rectangles obtained contained only the bounding box assigned to that partition and no other bounding boxes, or just small portions of them. This is really what we were searching for, so we used this method to crop the images of the dataset and obtain a new dataset of cropped images with a single face for each image.

We implemented this method with two variations:

- Instead of computing the distance for each single pixel, which is very computing intensive for hundreds of images with hundreds of pixels in width and height, we computed it for a grid of selected pixels, where a pixel was selected if the sum of the coordinates divided by 5 had a remainder of two. This still worked heuristically and reduced a lot the computation effort.
- There were cases, as the second image presented above, in which there were many faces sticked together and a crowd was represented in the image. In this situation it could often happen that the rectangle for each face was too small, or contained several portions of other faces. For this reason, we decided to cut those rectangles that had an area under a certain threshold.

The last thing to do was to appropriately modify the ground truth. Being the coordinates of the bounding box in the format (xmin, ymin, xmax, ymax) that was simply done getting the cropped image starting x-offset and y-offset with respect to the original image, and reducing xmin and xmax by x-offset, and reducing ymin and ymax by y-offset.

Using this technique, we were able to get around 1850 images of a single face out of the 853 images of the dataset. This set of images was the dataset used for the implementation of the architectures of the project. The results of this technique, for the two images of the original dataset presented, are below.

Figure 6: Cropped images, with just one face per image

As we can see from the last two images, where the people were distant enough the algorithm cropped each face separately. When a crowd was represented, instead, the algorithm discarded those faces that were too near to the others and kept only those faces that were more in foreground. As we can see from the last two images, there is still some noise in the form of other faces in the background of the images, but we thought the network could deal with that.

3.2 Dataset augmentation

One of the commonly known issues in classification problems in general is class imbalancement. When a class is not well represented in a dataset, any classifier trained on that dataset won't perform well on that class, because the generalization capability of the classifier depends not only on the quality but also on the number of samples the classifier is fed with during the training. In our dataset, the classes "with mask" and "without mask" are highly imbalanced, out of the 1861 images of our dataset:

- 1581 belong to class "with mask"
- 280 belong to class "without mask"

For this reason, we decided to apply data augmentation. This was done in particular flipping horizontally images belonging to the class "without mask" and obtaining double the number of images for this class. Because the features of the image and the flipped image are different, horizontal flipping can provide a good way for augmenting the infrequent class of our dataset. Of course, ground truth bounding boxes had to be flipped accordingly. After this process, the dataset contained a total of 2131 images. Roughly 75% of the images belong to the class "with mask" and 25% to the class "without mask".

4. CNN FROM SCRATCH

The CNN from scratch was built using a stack of convolutional layers and pooling layers with the purpose of creating the architecture for the feature extraction. At the end of the stack, a regression head and a classification head were put, to obtain the predictions for each image. The regression head is a dense layer of 4 neurons, each one in charge of predicting one of the coordinates of the bounding box. For these neurons, we decided to use the Sigmoid activation function, because the values of the coordinates of the ground truth bounding boxes were normalized between 0 and 1. The classification head is another dense layer of 2 neurons, equipped with a softmax activation function, to predict the probability of the sample to belong to each output class.
About the metrics, we kept track of the classification accuracy for the classification head and we implemented Intersection over Union as a custom metric in Keras, to monitor the performances of the regression head. The loss function used during training was a linear combination of the loss function for the classification head (we always used the cross entropy here) and the loss function for the regression head.

4.1 Experiment 1 – Base model

The first experiment we made consisted of a simple convolutional neural network with three convolutional layers and two pooling layers, a flatten layer and the two output heads. The losses used are the categorical cross entropy for the classification head and the mean squared error on the regression head. The loss weights were set at 0.15 for the classification head and 1 for the regression head.

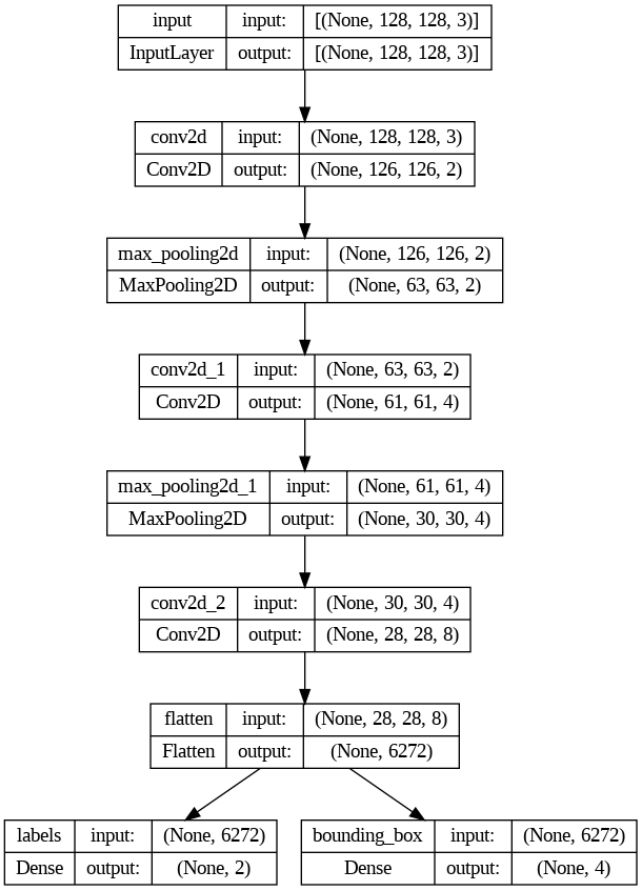


Figure 7: Experiment 1, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 126, 126, 2)	56	['input[0][0]']
max_pooling2d (MaxPooling2D)	(None, 63, 63, 2)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 4)	76	['max_pooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 4)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 8)	296	['max_pooling2d_1[0][0]']
flatten (Flatten)	(None, 6272)	0	['conv2d_2[0][0]']
labels (Dense)	(None, 2)	12546	['flatten[0][0]']
bounding_box (Dense)	(None, 4)	25092	['flatten[0][0]']

Total params: 38,066
Trainable params: 38,066
Non-trainable params: 0

Figure 8: Experiment 1, parameters

The network was trained for 10 epochs and showed very good results in labeling correctly the images of the training set, reaching an almost perfect accuracy. Actually, the same results were not produced on the validation set, probably due to overfitting, as shown in graphs below. The regression head instead was not very performant, especially when generalizing. The mean IoU over the training set continued to increase and reached 25%, while the same metric on the validation set got stuck under 10%. The possible reasons could be a model too simple, not able to extract the correct features for predicting the correct values, overfitting again, a wrong loss function or a mixture of these reasons.

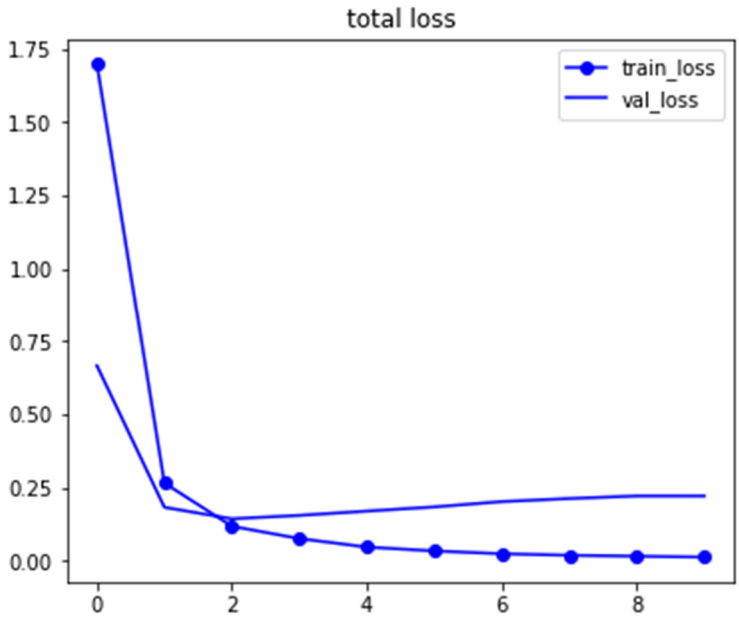


Figure 9: Experiment 1, loss

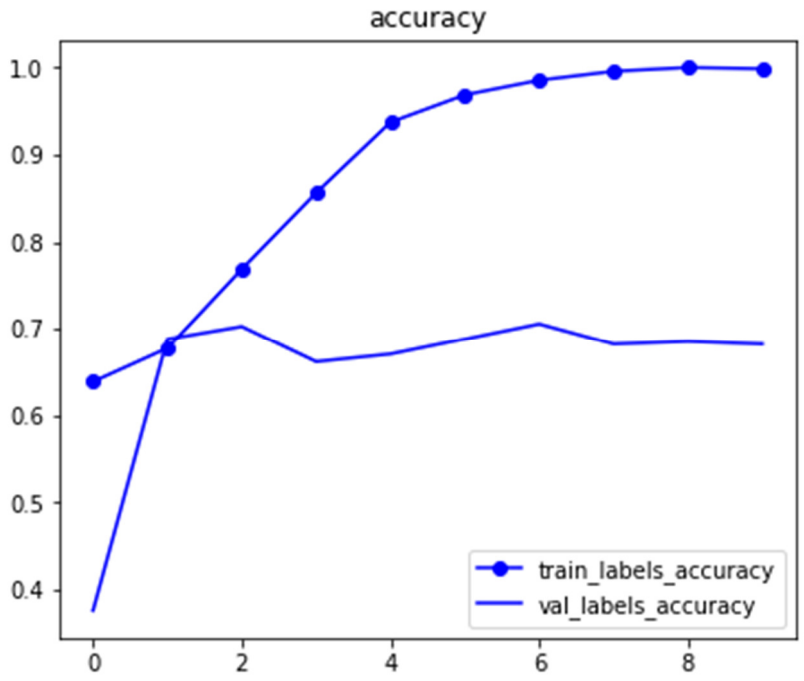


Figure 10: Experiment 1, accuracy

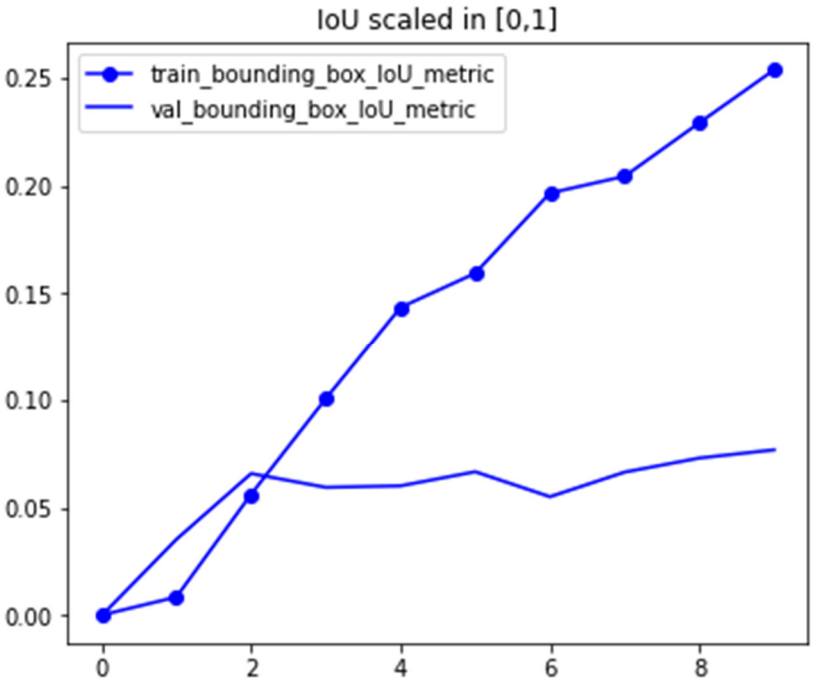


Figure 11: Experiment 1, IoU

In any case, early stopping was used to save the model at the point in which the validation loss was the least value ever reached by the network during training. The model was then restored and tested on the test set, and these were the result.

Experiment 1 – Test set performance	
Accuracy	66.82 %
Intersection over Union	7.27%

4.2 Experiment 2 - Dropout

In the second experiment we modified the model adding the dropout layer. This was done to combat overfitting, that in the previous model seemed to happen quite quickly and to prevent the model from reaching a good performance on non-seen samples. The dropout rate was decided as 0.5 and the layer was put right before the two heads of the network. The resulting architecture was the following:

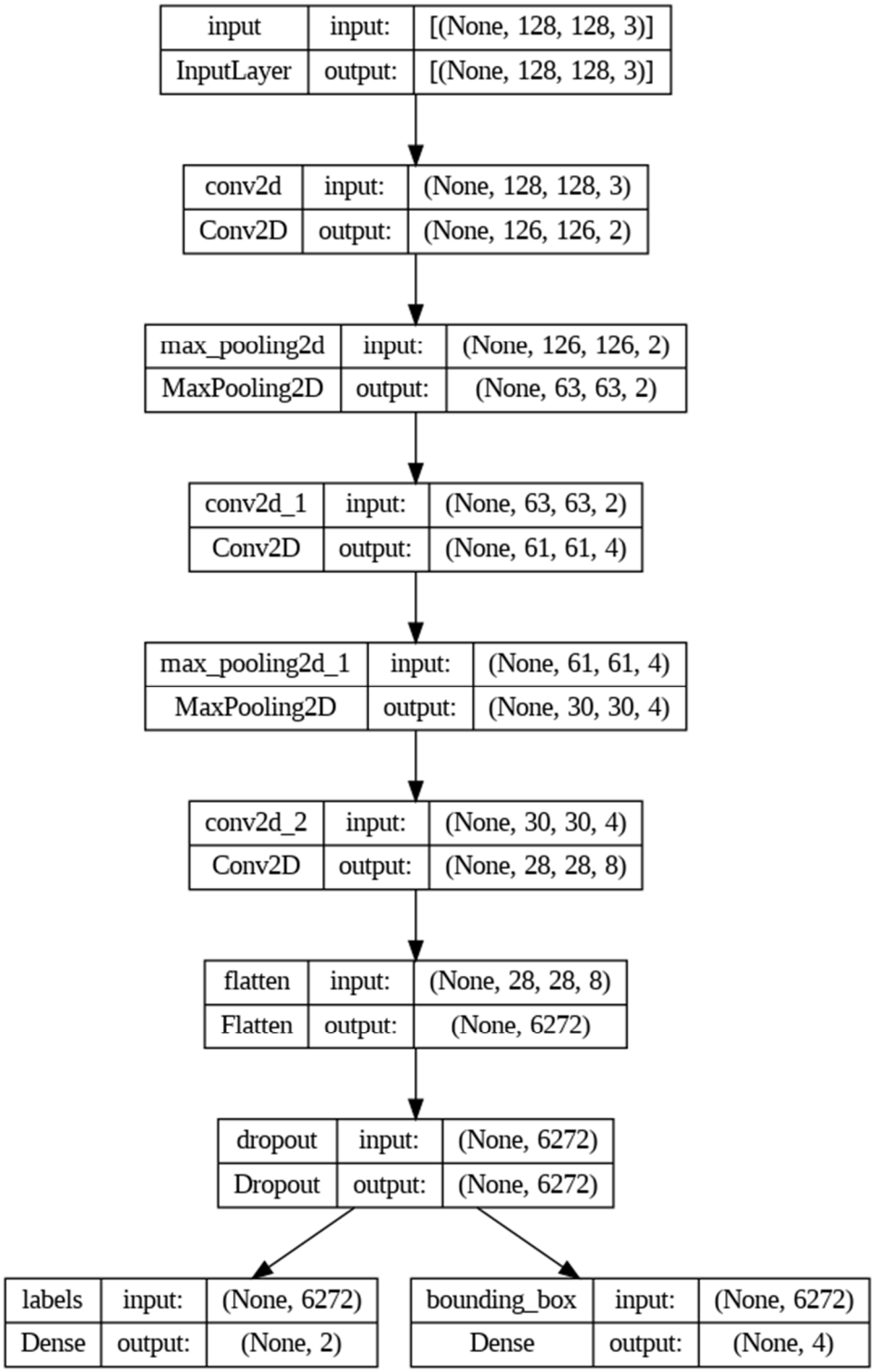


Figure 12: Experiment 2, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 126, 126, 2)	56	['input[0][0]']
max_pooling2d (MaxPooling2D)	(None, 63, 63, 2)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 4)	76	['max_pooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 4)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 8)	296	['max_pooling2d_1[0][0]']
flatten (Flatten)	(None, 6272)	0	['conv2d_2[0][0]']
dropout (Dropout)	(None, 6272)	0	['flatten[0][0]']
labels (Dense)	(None, 2)	12546	['dropout[0][0]']
bounding_box (Dense)	(None, 4)	25092	['dropout[0][0]']

Total params: 38,066
Trainable params: 38,066
Non-trainable params: 0

Figure 13: Experiment 2, parameters

The same training procedure was adopted. In this case, the Intersection over Union metric reduced a lot on the training set, but we were able to increase it by a couple of points in percentage on the validation set with respect to the previous experiment and to brought it over 10%, while we couldn't increase the classification accuracy by a lot, it remained under 70%.

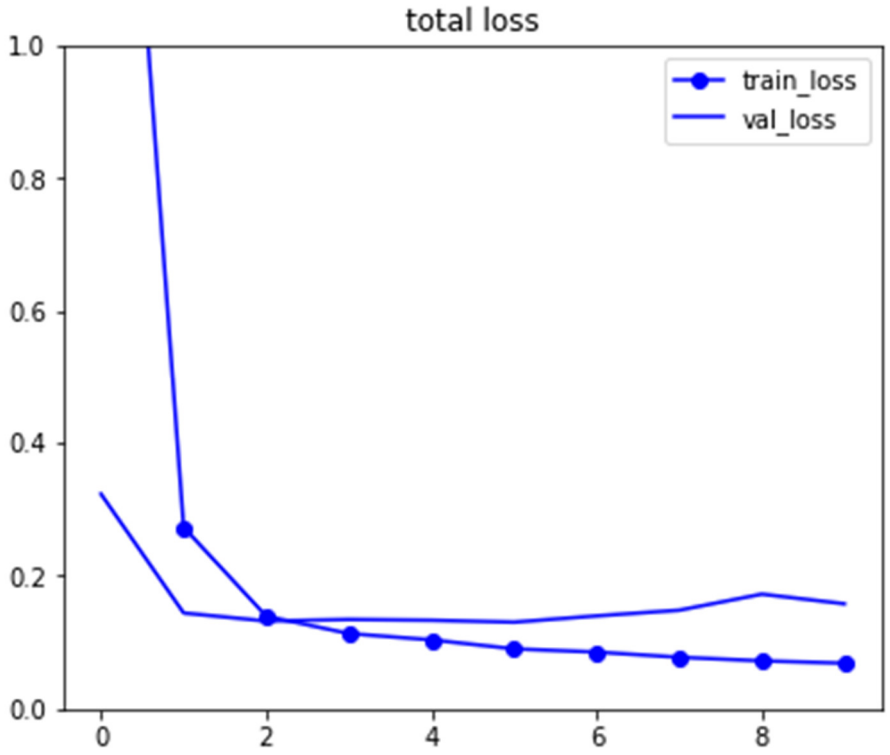


Figure 14: Experiment 2, loss

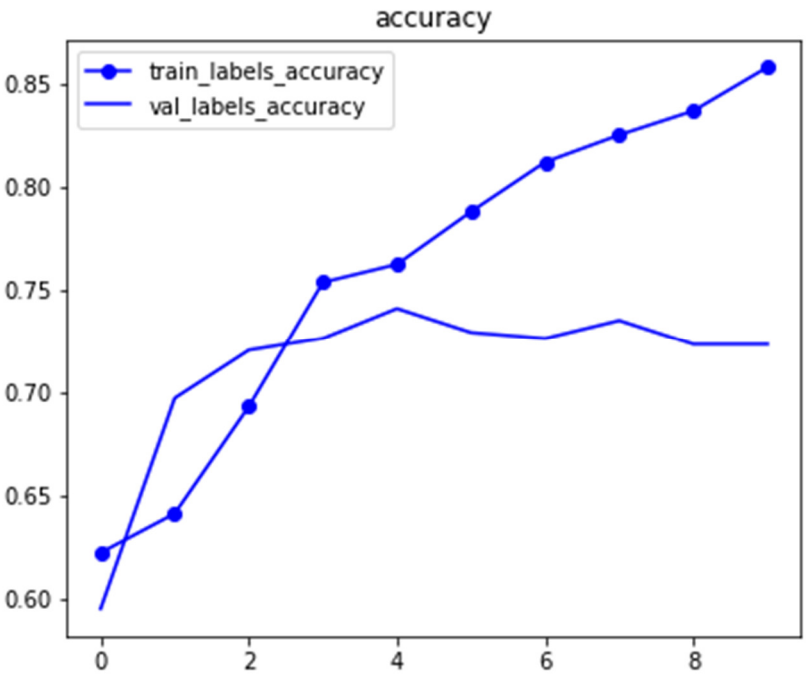


Figure 15: Experiment 2, accuracy

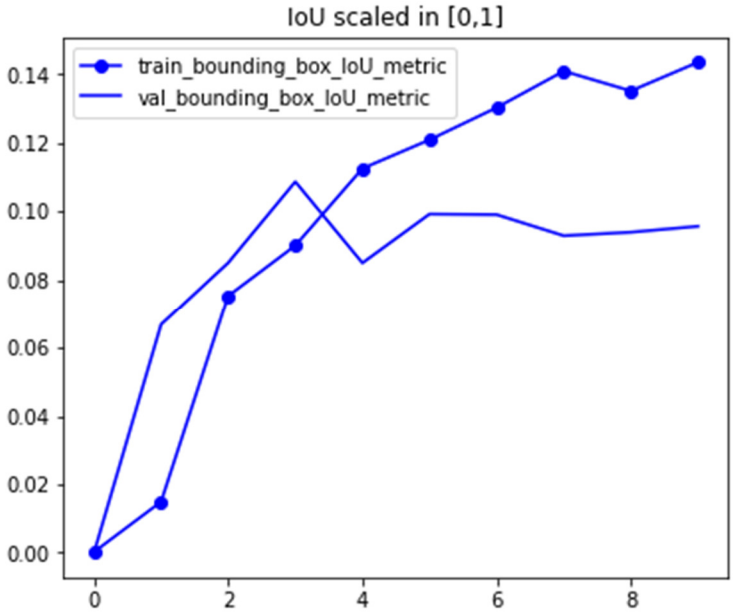


Figure 16: Experiment 2, IoU

Experiment 2 – Test set performance	
Accuracy	69.39%
Intersection over Union	10.54%

The results in the table above regard the performance of the network on the test set. Given these results, dropout helped in reducing the overfitting, but it didn't help much in solving the problems we had with our first experiment. Even if the validation loss started to increase later than before the accuracy on the test set didn't increase and the increase in IoU is not satisfying. At this point, we still had different possible things to experiment to try to increase both metrics.

4.3 Experiment 3 – More complex network

In the third experiment we focused more on trying to adjust the accuracy of the classification head. Briefly analyzing the confusion matrices produced by the first two experiments on the test set, which are represented below, we have seen that the problems the network had were in correctly classifying the class “without mask”. Actually, both networks performed really bad on this class, with an accuracy well below 50% and the overall accuracy was quite high just because the majority of the samples belonged to the class “with mask” where the networks had higher performances.

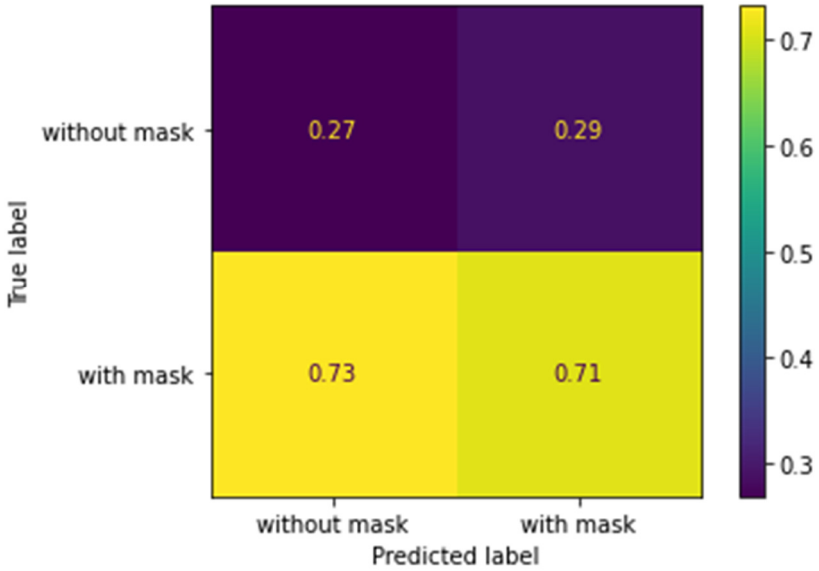


Figure 17: Experiment 1, confusion matrix

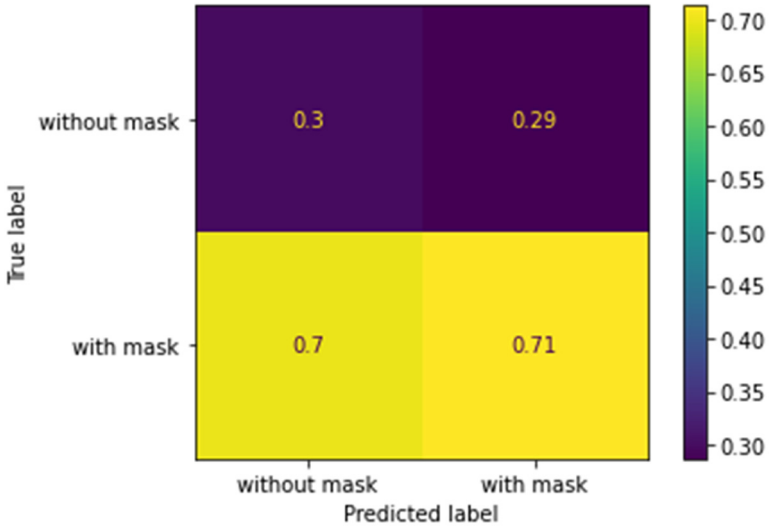


Figure 18: Experiment 2, confusion matrix

In the third experiment, we tried to come up with a network which was more robust at extracting the features from the images and obtaining good results on both classes. To do this, various levels of complexity have been added, and the resulting network is presented below. In particular, we added an additional convolutional block and for each convolutional layer we increased the number of filters.

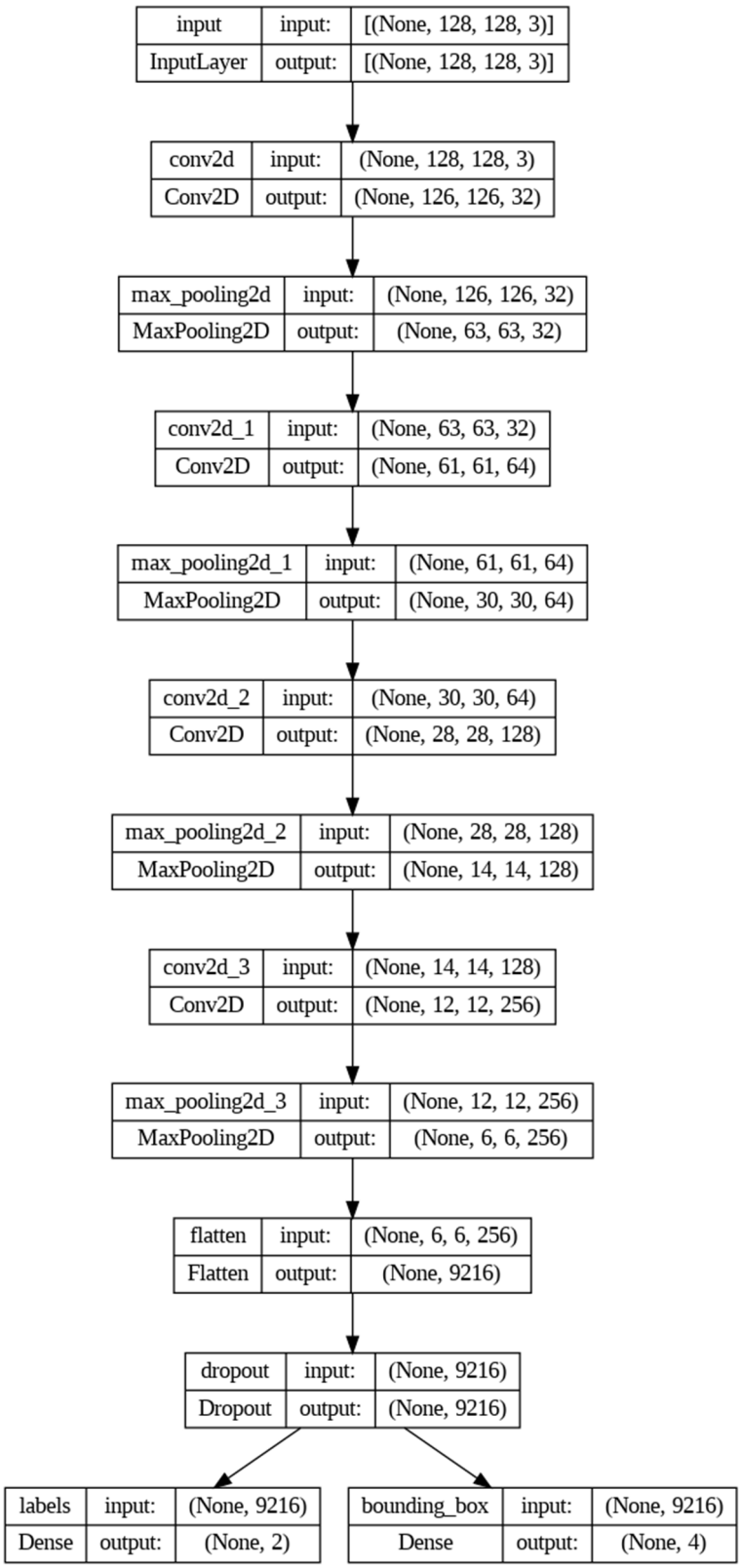


Figure 19: Experiment 3, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 126, 126, 32)	896	['input[0][0]']
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496	['max_pooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856	['max_pooling2d_1[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0	['conv2d_2[0][0]']
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168	['max_pooling2d_2[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0	['conv2d_3[0][0]']
flatten (Flatten)	(None, 9216)	0	['max_pooling2d_3[0][0]']
dropout (Dropout)	(None, 9216)	0	['flatten[0][0]']
labels (Dense)	(None, 2)	18434	['dropout[0][0]']
bounding_box (Dense)	(None, 4)	36868	['dropout[0][0]']

Total params: 443,718
Trainable params: 443,718
Non-trainable params: 0

Figure 20: Experiment 3, parameters

Moreover, we changed the weights of the loss to 1 for the classification head and 10 for the regression head. As a result, the training of the network became slower, but as we can see from the test set, the results were way better than before, especially watching the confusion matrix of the network.

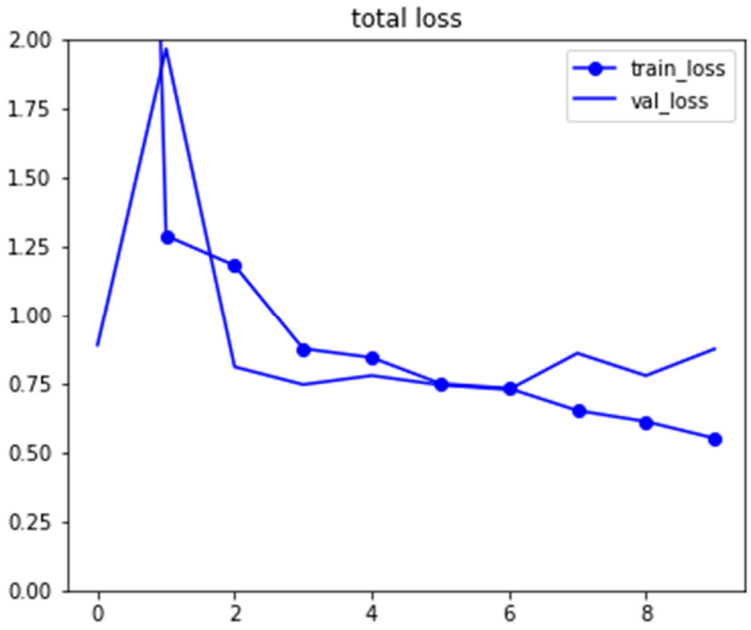


Figure 21: Experiment 3, loss

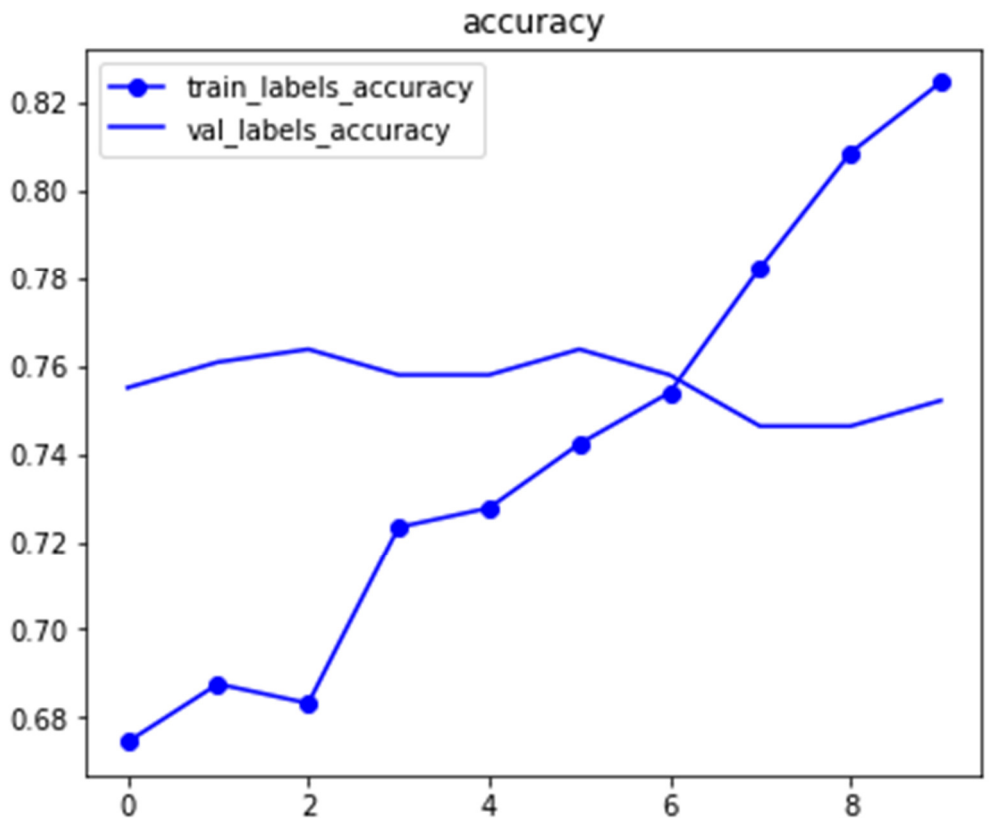


Figure 22: Experiment 3, accuracy

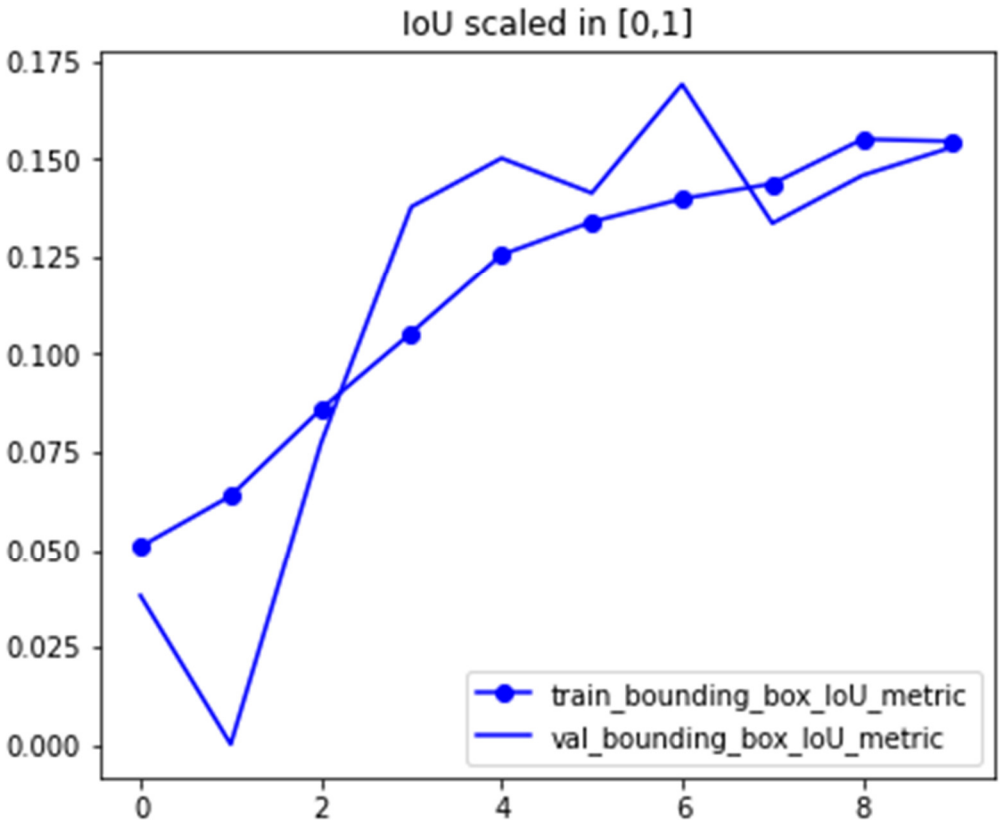


Figure 23: Experiment 3, IoU

As we can see from the graphs, the raw accuracy on the validation set was not much different from the previous network but analyzing the network on the test set and comparing the confusion matrices, we can spot that this network was far better at recognizing the class “without mask” and has also increased performances for the class “with mask”.

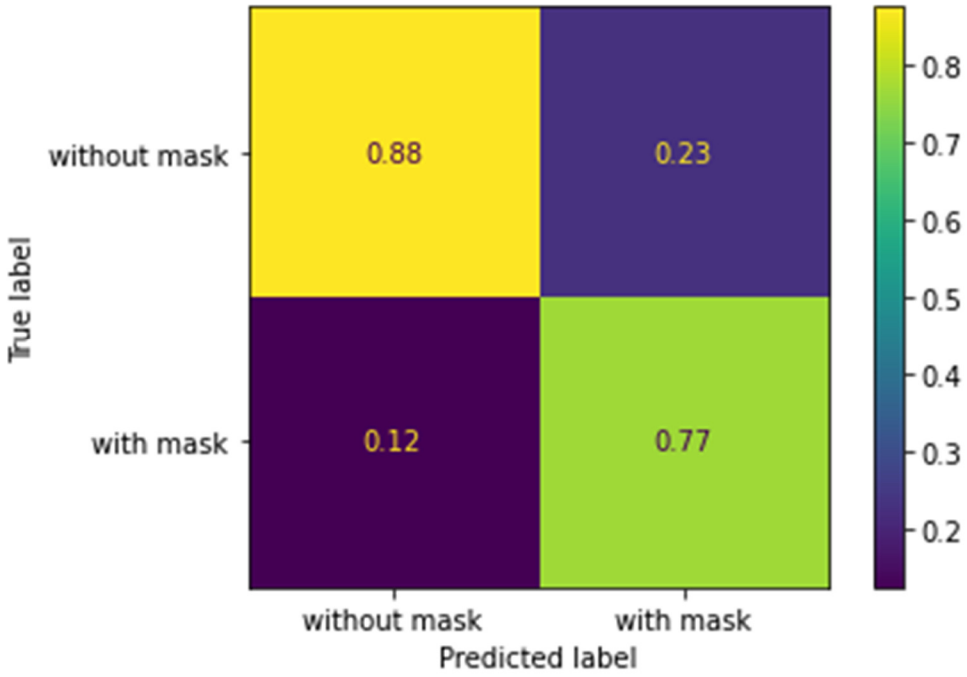


Figure 24: Experiment 3, confusion matrix

Overall, the results on the test set are the following:

Experiment 3 – Test set performance	
Accuracy	77.10%
Intersection over Union	17.12%

Moreover, with this network we were also able to increase the IoU metric, as reported in the table. This showed us that complicating the model and increasing the capacity of the network had a good influence on the performances of the network.

4.4 Experiment 4 – Denormalized bounding boxes

In the fourth model, we tried to denormalize the values of the bounding boxes. The images in the dataset were already resized to the same dimension (128x128 pixels) so this could be an option to obtain better results. The resulting network is very similar to the previous one, but we added separate dropout layers for each head, with different dropout rates, because we have seen that the classification head was overfitting earlier than the regression head. We already knew that having targets (bounding box coordinates) in the range [0,128] instead of [0,1] could make the mean squared error very high and maybe not suitable for being used as a loss function, so we also implemented the smooth-L1 loss function as a custom Keras loss and compared the results we obtained on both networks.

As we mentioned in the introduction, the smooth-L1 loss function merges the L2-loss function (the mean squared error) with the L1-loss function (the absolute error) in a continuous and derivable way. A threshold value is used to determine where the merge happens. The shape of the function is presented below, for different values of the threshold.

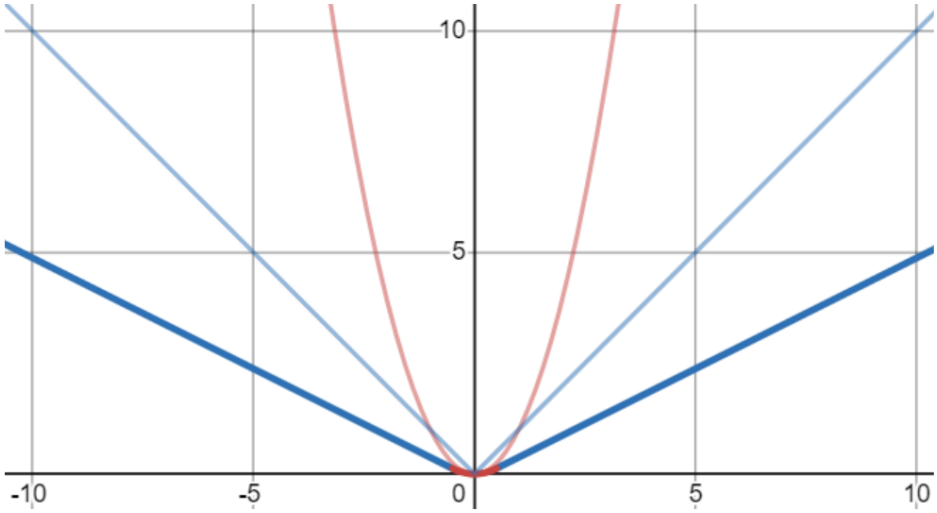


Figure 25: Smooth-L1 function, threshold = 0.5

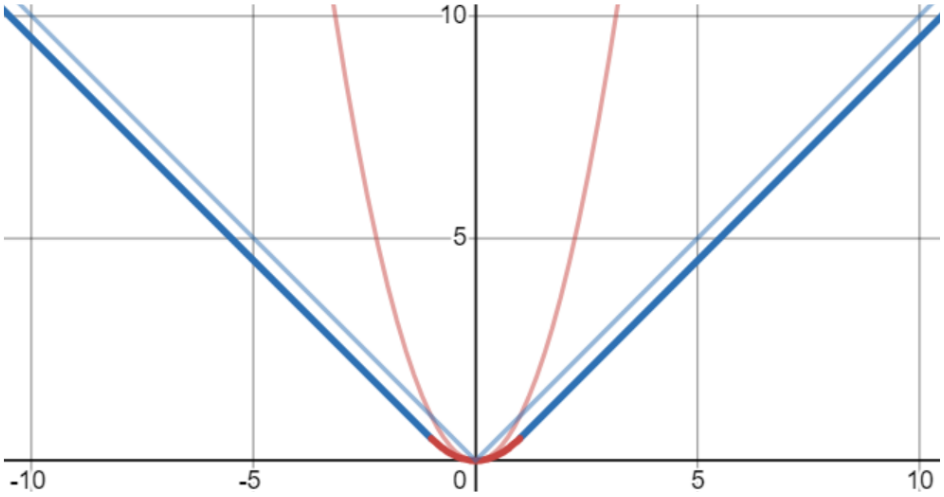


Figure 26: Smooth-L1 function, threshold = 1

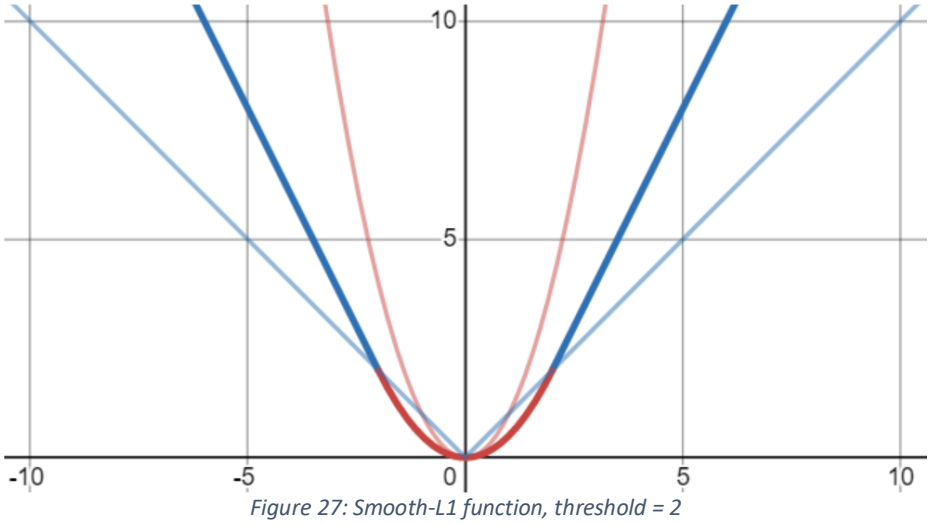


Figure 27: Smooth-L1 function, threshold = 2

In any case we tested both the losses, and the architecture of the network was the same, we just introduced separated dropout layers for both the heads, using different dropout rates, as we mentioned before.

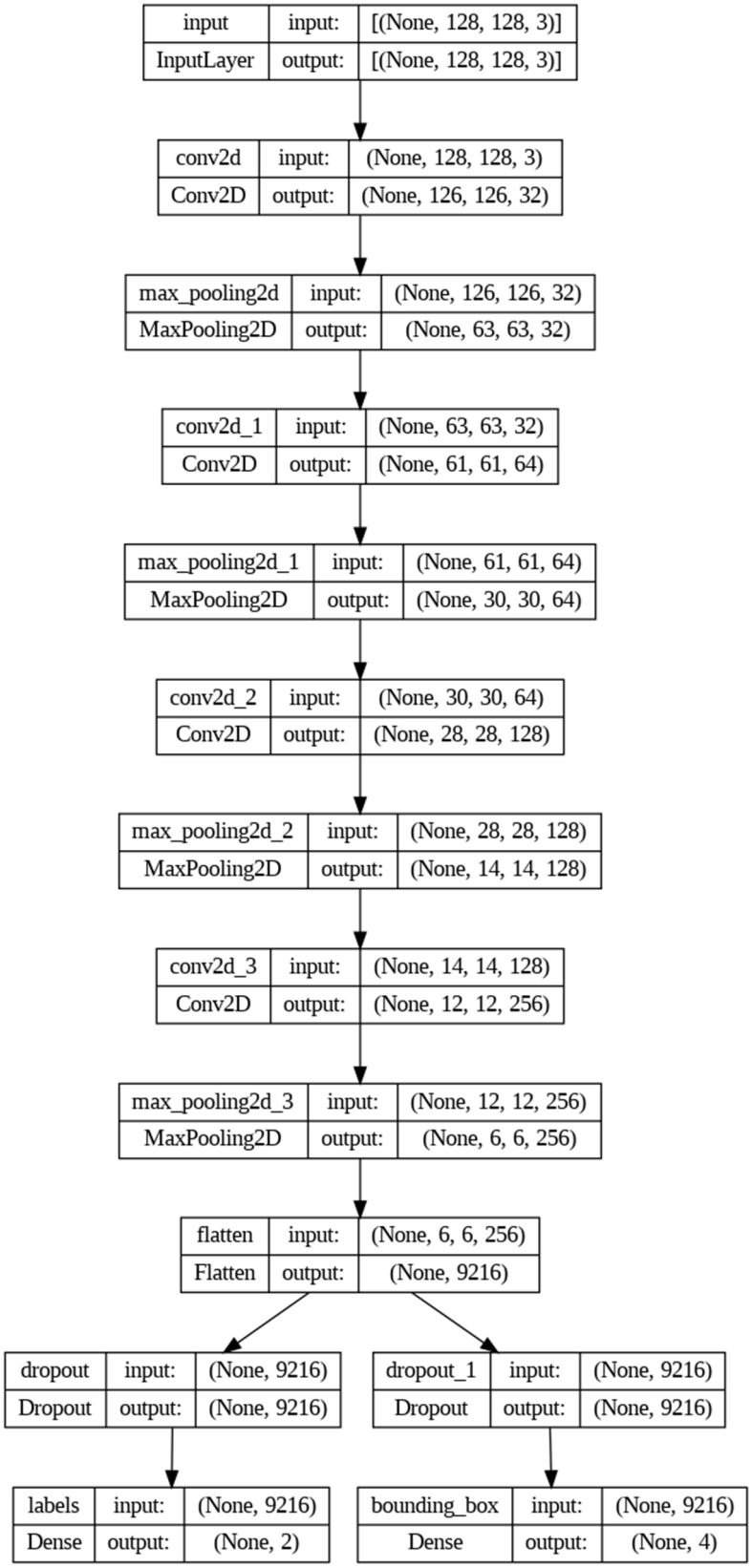


Figure 28: Experiment 4, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 126, 126, 32)	896	['input[0][0]']
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496	['max_pooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856	['max_pooling2d_1[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0	['conv2d_2[0][0]']
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168	['max_pooling2d_2[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0	['conv2d_3[0][0]']
flatten (Flatten)	(None, 9216)	0	['max_pooling2d_3[0][0]']
dropout (Dropout)	(None, 9216)	0	['flatten[0][0]']
dropout_1 (Dropout)	(None, 9216)	0	['flatten[0][0]']
labels (Dense)	(None, 2)	18434	['dropout[0][0]']
bounding_box (Dense)	(None, 4)	36868	['dropout_1[0][0]']

Total params: 443,718
Trainable params: 443,718
Non-trainable params: 0

Figure 29: Experiment 4, parameters

Both the networks, the one with the mean squared error and the one with the Smooth-L1 loss, were tested on 20 epochs, because at the beginning, due to higher values of the loss function, the loss was oscillating and the training seemed to be slower.
For the first network, which used the mean squared error as loss function for the regression head, the results were worse than the previous experiment. As we can see from below, the network performance was more unstable than before on the validation set during training, we changed the weights of the loss function to try to give a less weight to the mean squared error which as expected was high and was oscillating a lot, but the result was still not satisfying.

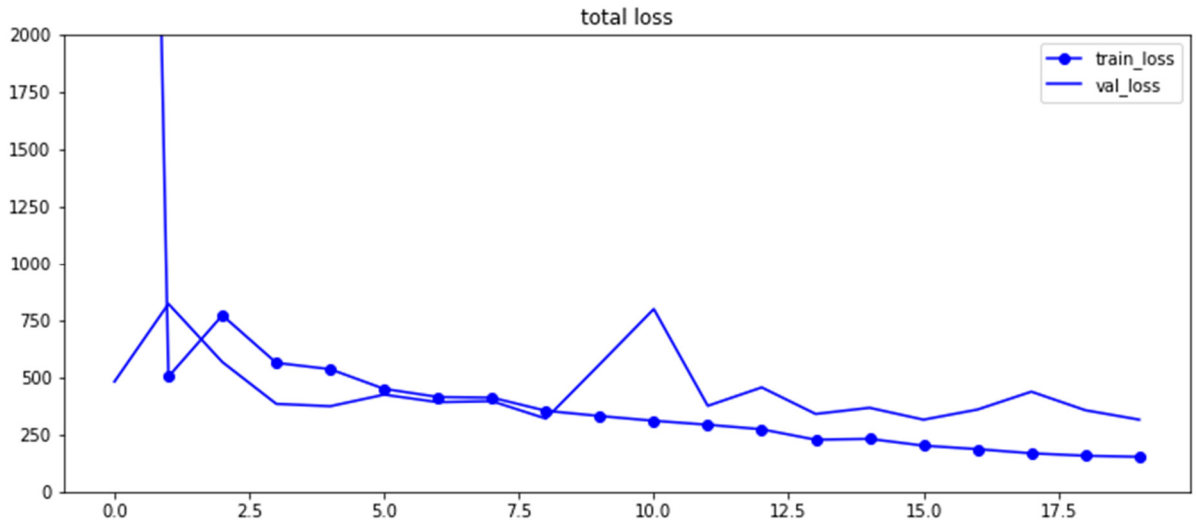


Figure 30: Experiment 4_mse, loss

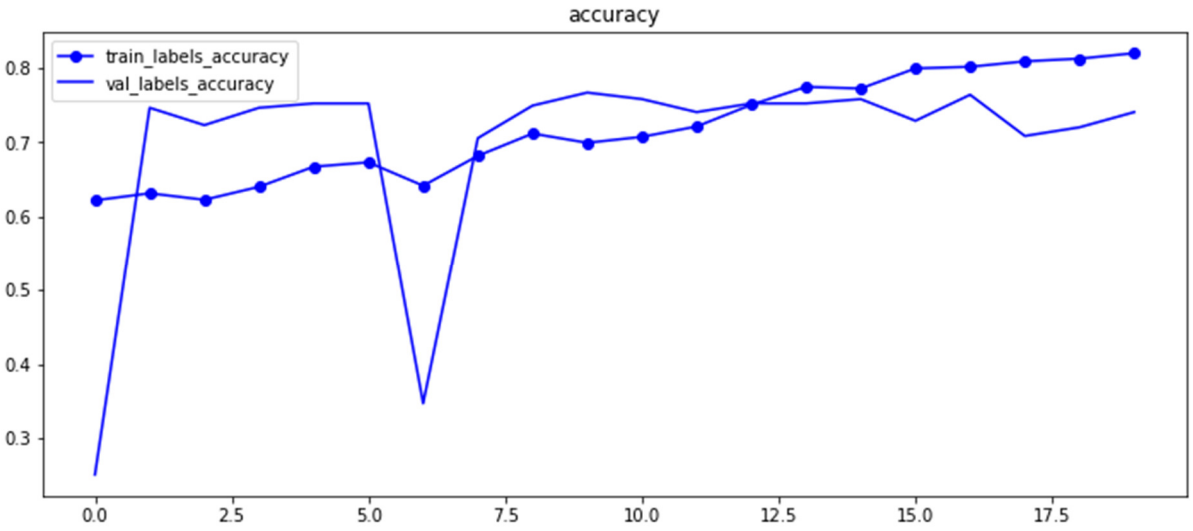


Figure 31: Experiment 4_mse, accuracy

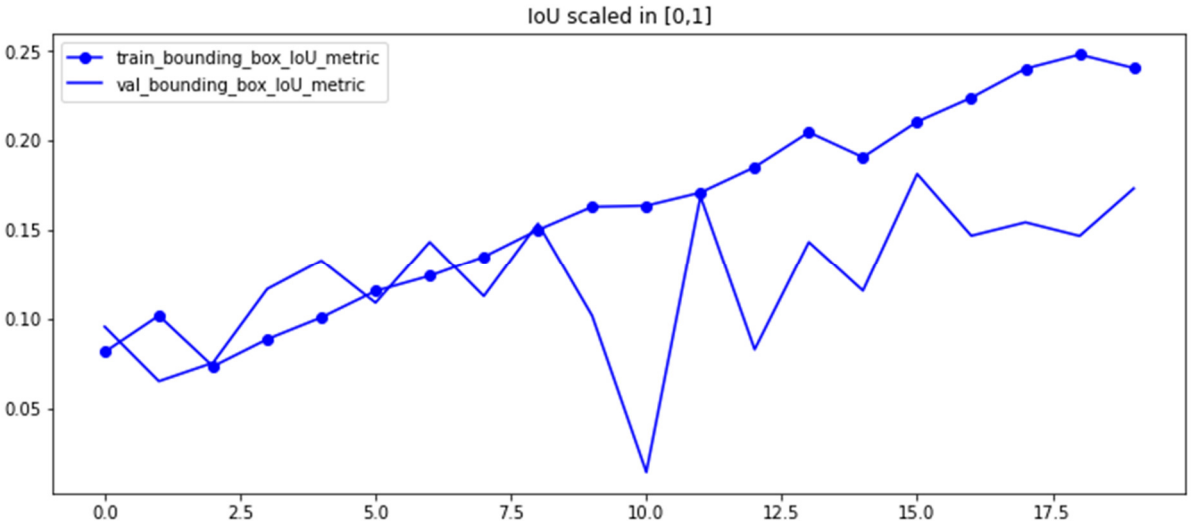


Figure 32: Experiment 4_mse, IoU

Replacing the mean squared error loss with the Smooth-L1 loss reduced this behavior:

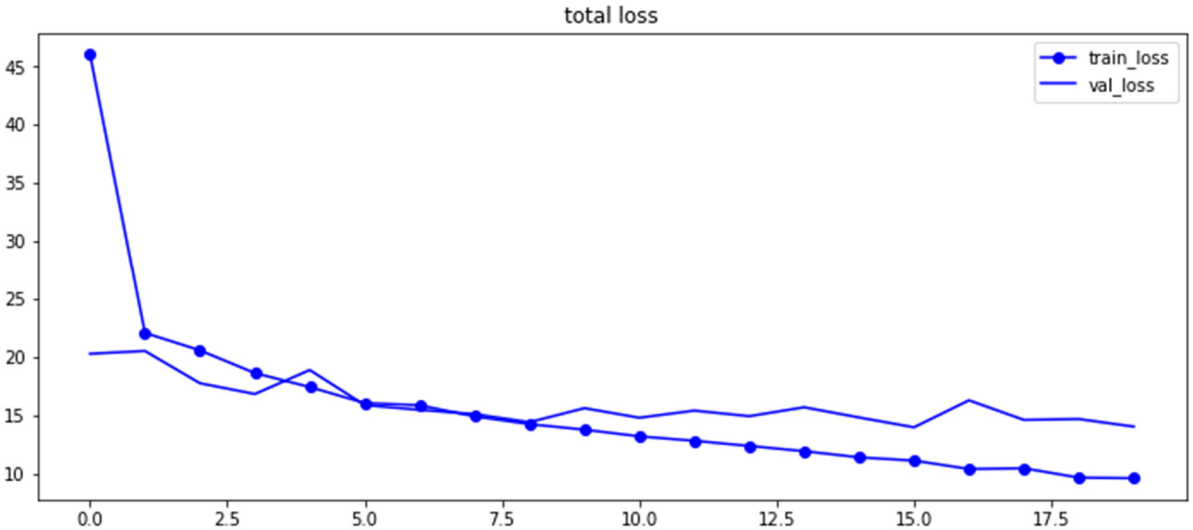


Figure 33: Experiment 4_smooth-L1, loss

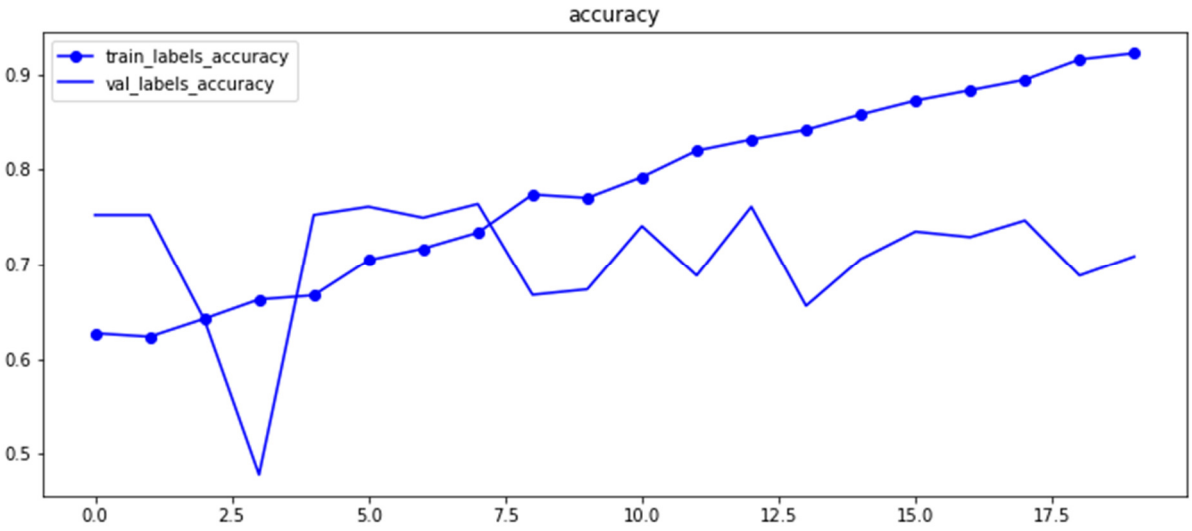


Figure 34: Experiment 4_smooth-L1, accuracy

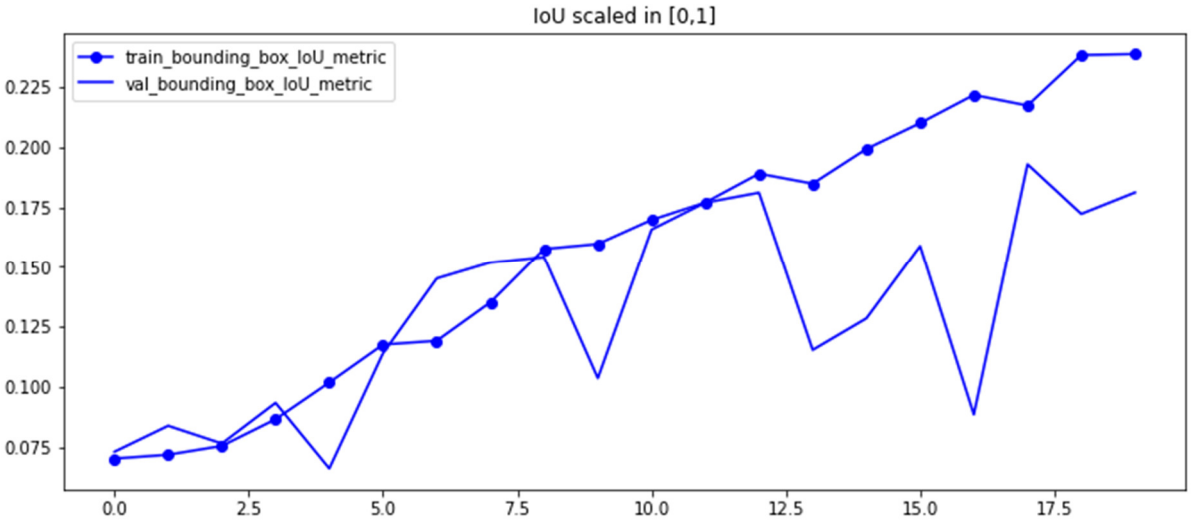


Figure 35: Experiment 4_smooth-L1, IoU

In any case, both the networks failed to reach the performances of the experiment with normalized bounding boxes, especially in terms of classification accuracy. The reason could be that the total loss, in this case, is dominated by the bounding box loss, and even changing the weights and trying to reduce the weight of the bounding box loss was not helpful.

Experiment 4_mse – Test set performance	
Accuracy	74.77%
Intersection over Union	18.37%

Experiment 4_smooth-L1 – Test set performance	
Accuracy	70.79%
Intersection over Union	15.50%

Testing the networks on the test set we can see that even if using the mean square error, the network was oscillating more, the performances were better with respect to the smooth-L1 loss. In both cases the classification accuracy is below the previous experiment

and if we analyze the confusion matrices (reported in the notebooks) we can see that the problem is again the class “without mask” which is again badly recognized by the classifier. This was probably caused by the fact that the total loss is dominated by the bounding box loss, and even changing the weights, as said before, was not helpful. For this reason, we thought working with normalized bounding boxes was a better idea.

4.5 Experiment 5 – Adaptive learning rate

In the next experiment we come back to normalized bounding boxes, since the experiment with denormalized ones didn’t produce good results. As a minor change, in this experiment we realized that for binary classification a softmax-equipped layer of two neurons was not needed. Actually, a single neuron equipped with a sigmoid function is always enough to perform binary classification, so we implemented this change in our network also changing the ground truth for the labels (from a one-hot vector to a single integer, 0 or 1). Moreover, as we started to do in the previous experiment, we worked on the dropout rate: in the third experiment, a single dropout layer was placed after the flattening operation and was in common for the two heads of the network. The graphs below represent, for the third experiment, the separate loss functions behavior:

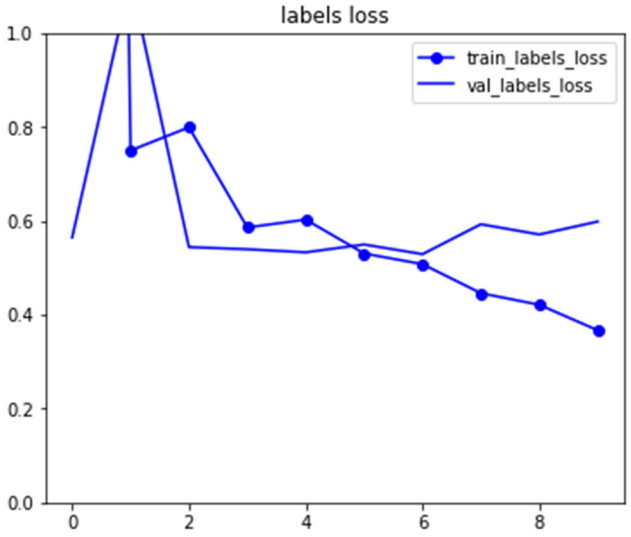


Figure 36: Experiment 3, labels loss

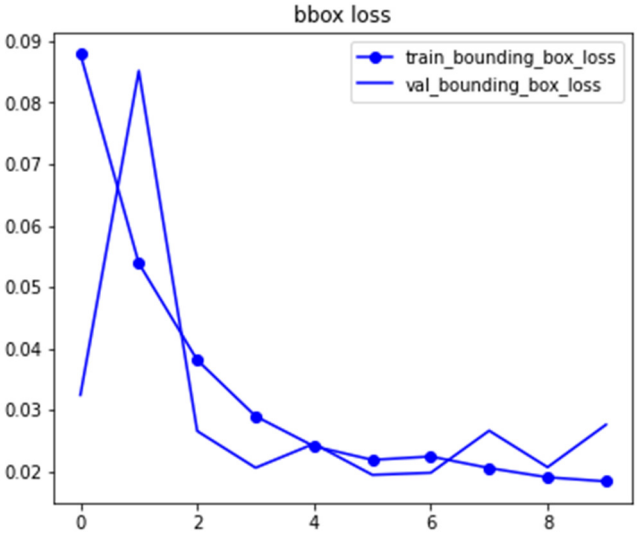


Figure 37: Experiment 3, bounding box loss

As we can see, it seemed that the classification head still overfitted very rapidly, around the third or fourth epoch, while the regression head was slower in overfitting. For this reason, it seemed reasonable to split the dropout layer and select different dropout rates in such a way that the dropout rate of the classification head could be higher with respect to the regression head. This was the only change to the architecture of the network, which is represented below.

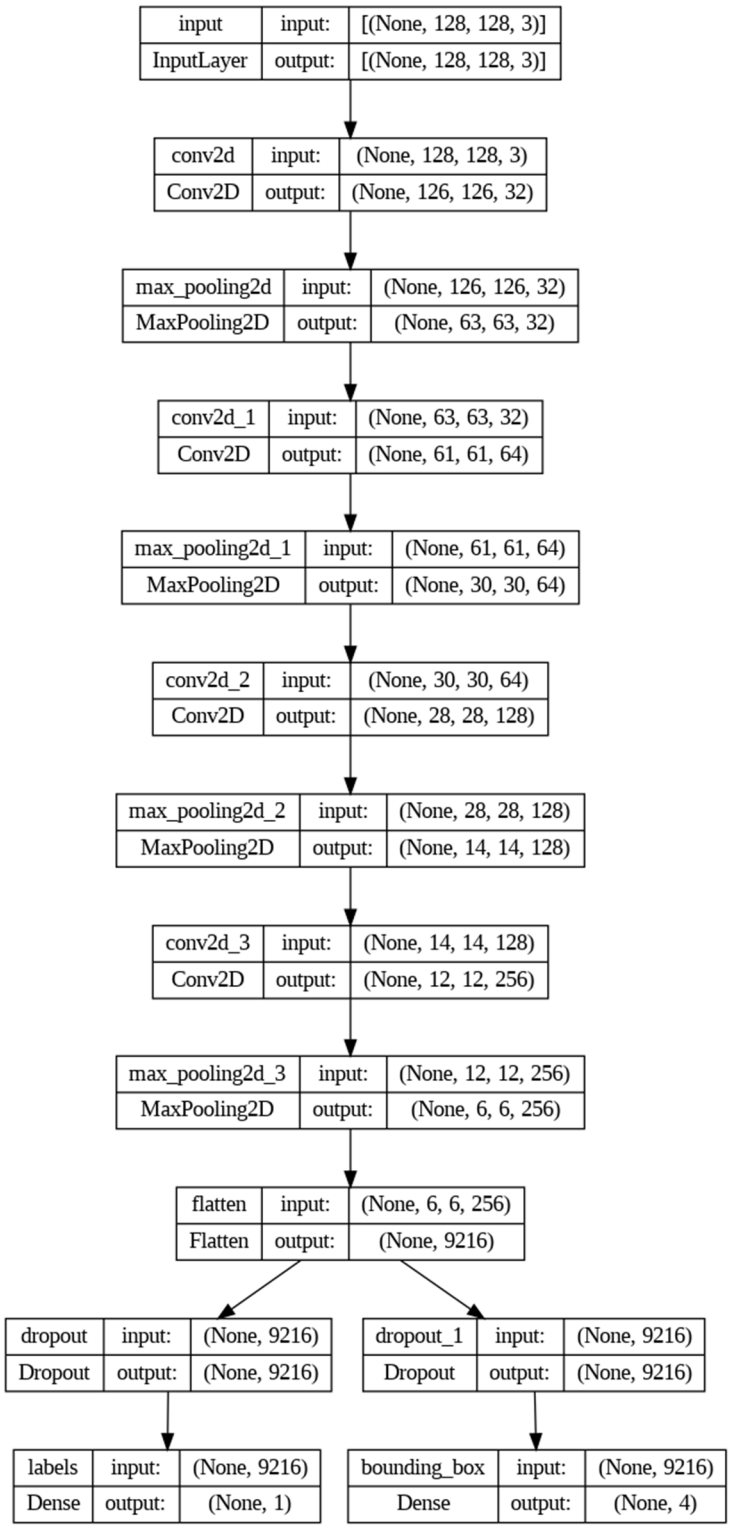


Figure 38: Experiment 5, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 126, 126, 32)	896	['input[0][0]']
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496	['max_pooling2d[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856	['max_pooling2d_1[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0	['conv2d_2[0][0]']
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168	['max_pooling2d_2[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0	['conv2d_3[0][0]']
flatten (Flatten)	(None, 9216)	0	['max_pooling2d_3[0][0]']
dropout (Dropout)	(None, 9216)	0	['flatten[0][0]']
dropout_1 (Dropout)	(None, 9216)	0	['flatten[0][0]']
labels (Dense)	(None, 1)	9217	['dropout[0][0]']
bounding_box (Dense)	(None, 4)	36868	['dropout_1[0][0]']

Total params: 434,501
Trainable params: 434,501
Non-trainable params: 0

Figure 39: Experiment 5, parameters

In addition to this change, an attempt was made to make the learning rate adaptive. In particular, we used the Keras callback ReduceLROnPlateau. This callback was configured to monitor the validation loss. When the validation loss didn't improve consecutively for 3 epochs, the learning rate was reduced to the 20% of the previous learning rate for the next epochs.

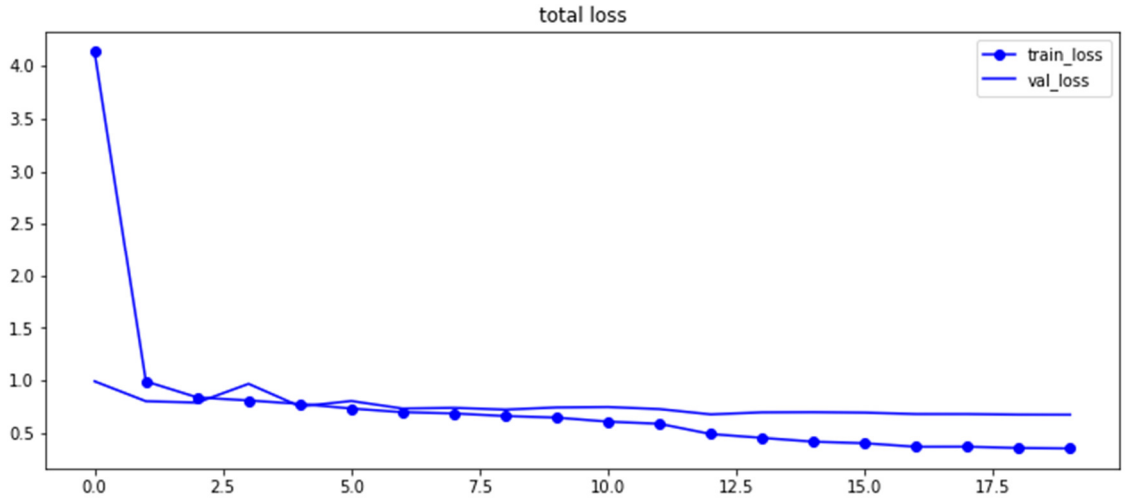


Figure 40: Experiment 5, loss

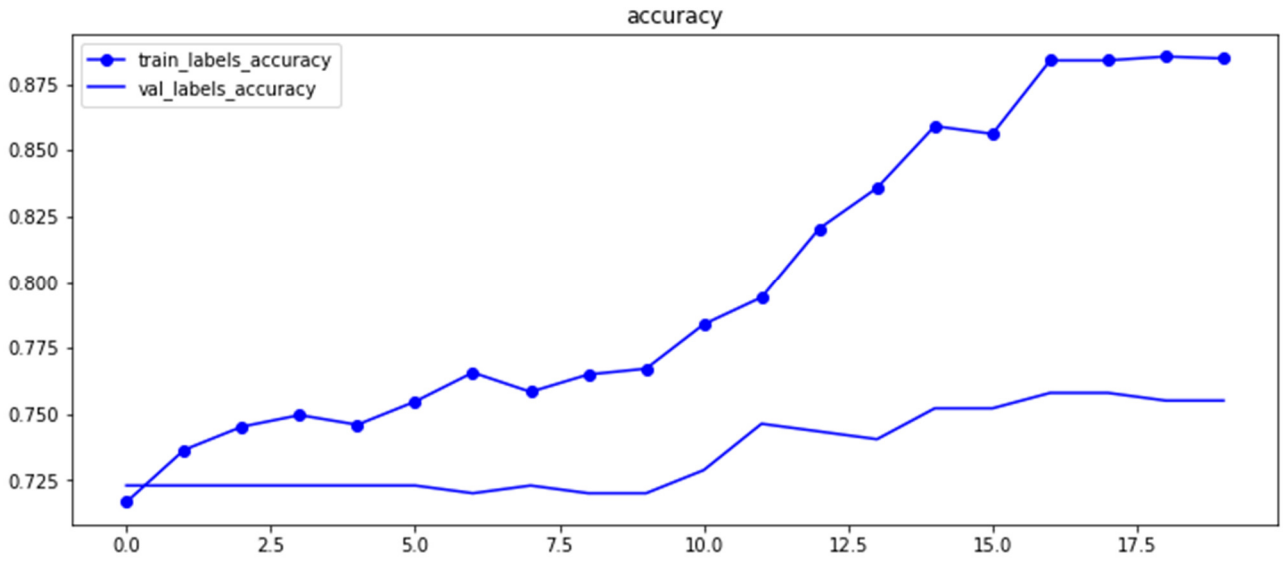


Figure 41: Experiment 5, accuracy

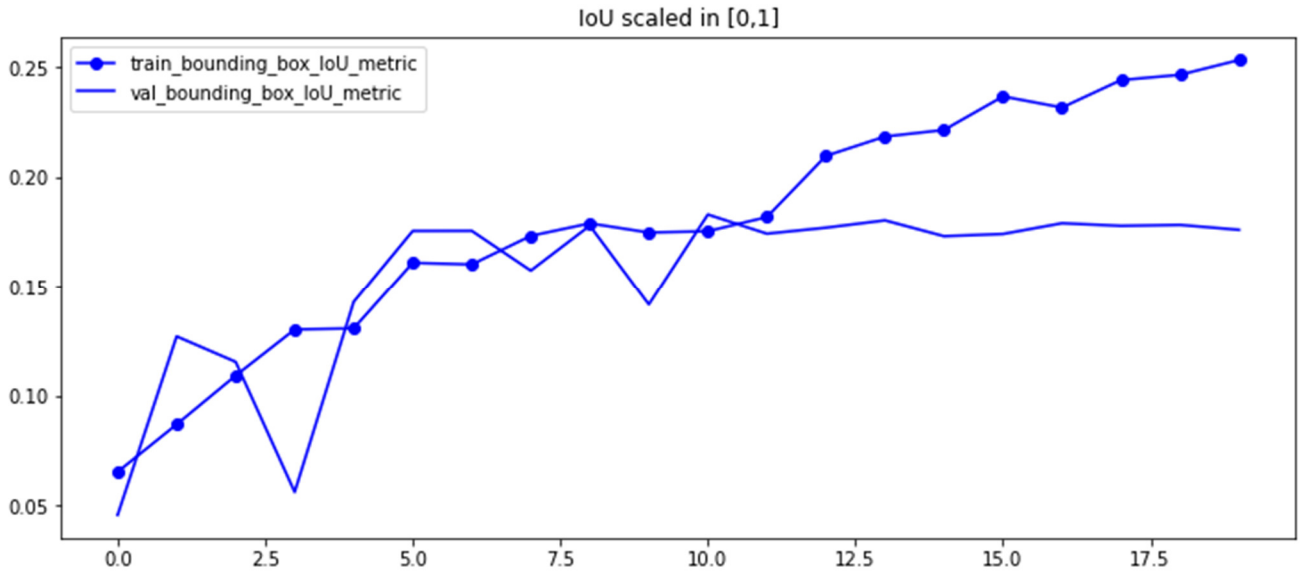


Figure 42: Experiment 5, IoU

In the middle of the training, adjusting the learning rate seemed to help the network reach better performance, but maybe an 80% of reduction of learning rate was too much and the network state, in the last epochs, seemed to remain more or less the same. Also, despite we increased the dropout rate of the classification head, the classification accuracy was still on the same level as before. The IoU got better results because we increased the number of epochs, and almost reached 20% on the test set.

Experiment 5 – Test set performance	
Accuracy	76.16%
Intersection over Union	19.36%

4.6 Experiment 6 – Improvements to hyperparameters

In the sixth experiment we tried different new things:

- First of all, as we have reported before, we tried to modify the setting of the callback to reduce the learning rate adaptively, because the previous parameters seemed too drastic. In this experiment, we reduced the patience to 2 (number of epochs to wait before applying the reduction), but we increased the reduction factor from 20% to 50% of the previous learning rate.
- We made an attempt in increasing the dropout rate before the regression head. The dropout layers remained separate as we still set different rates between the classification head and the regression head.
- To try to increase the accuracy of the classifier, a dense layer with 32 neurons equipped with a REctified Linear Unit function was inserted before the classifier.
- In a final attempt to increase the generalization capability of the network, which still remained the core issue of our networks, we tried to reduce the batch size of each epoch from 32 to 16.

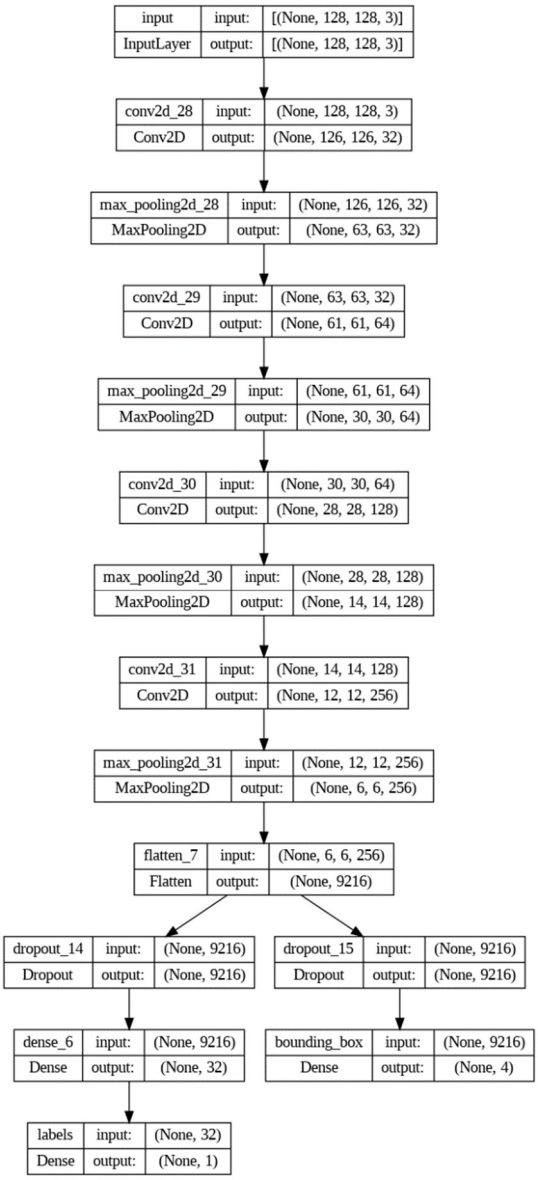


Figure 43: Experiment 6, architecture

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d_28 (Conv2D)	(None, 126, 126, 32)	896	['input[0][0]']
max_pooling2d_28 (MaxPooling2D)	(None, 63, 63, 32)	0	['conv2d_28[0][0]']
conv2d_29 (Conv2D)	(None, 61, 61, 64)	18496	['max_pooling2d_28[0][0]']
max_pooling2d_29 (MaxPooling2D)	(None, 30, 30, 64)	0	['conv2d_29[0][0]']
conv2d_30 (Conv2D)	(None, 28, 28, 128)	73856	['max_pooling2d_29[0][0]']
max_pooling2d_30 (MaxPooling2D)	(None, 14, 14, 128)	0	['conv2d_30[0][0]']
conv2d_31 (Conv2D)	(None, 12, 12, 256)	295168	['max_pooling2d_30[0][0]']
max_pooling2d_31 (MaxPooling2D)	(None, 6, 6, 256)	0	['conv2d_31[0][0]']
flatten_7 (Flatten)	(None, 9216)	0	['max_pooling2d_31[0][0]']
dropout_14 (Dropout)	(None, 9216)	0	['flatten_7[0][0]']
dense_6 (Dense)	(None, 32)	294944	['dropout_14[0][0]']
dropout_15 (Dropout)	(None, 9216)	0	['flatten_7[0][0]']
labels (Dense)	(None, 1)	33	['dense_6[0][0]']
bounding_box (Dense)	(None, 4)	36868	['dropout_15[0][0]']

Total params: 720,261
Trainable params: 720,261
Non-trainable params: 0

Figure 44: Experiment 6, parameters

We also increased further the number of epochs to 30, this was done to be sure to, with the adjustments to the learning rate, arrive at a certain point where the validation loss rarely could descend further.

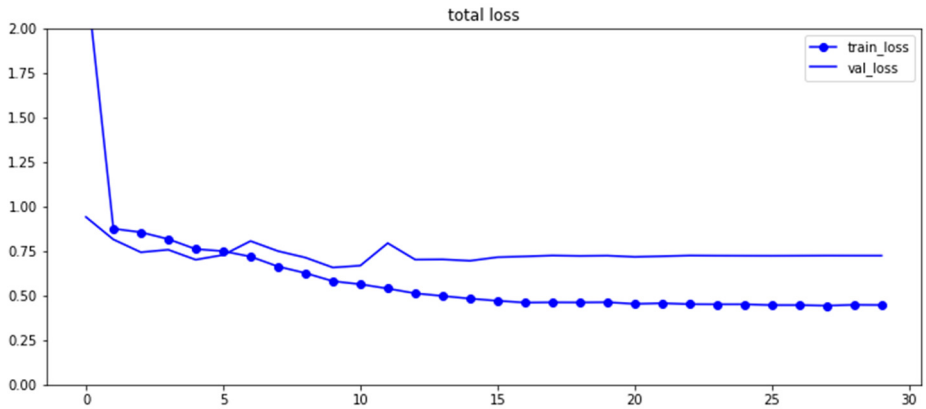


Figure 45: Experiment 6, loss

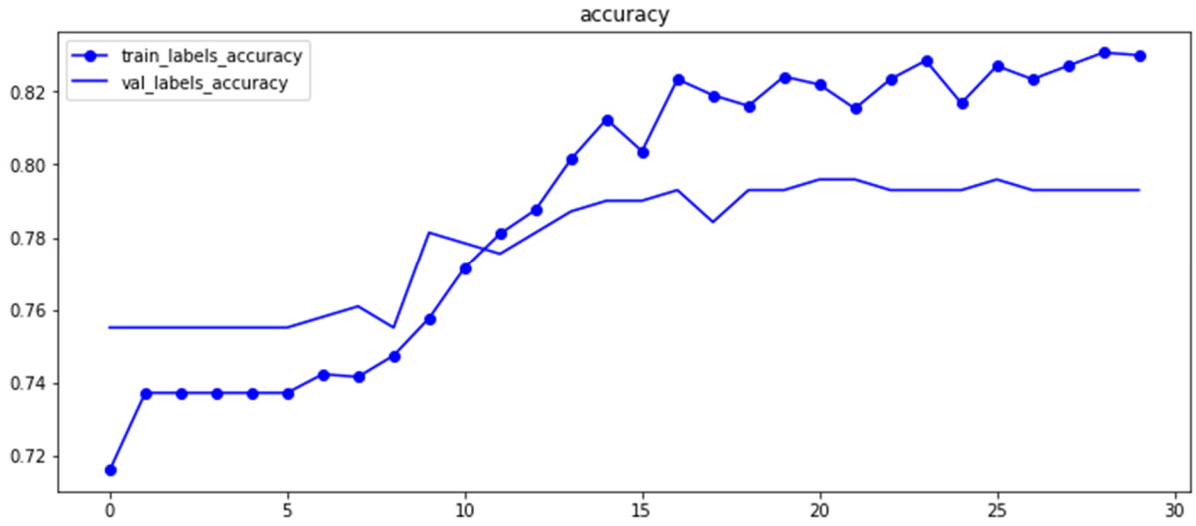


Figure 46: Experiment 6, accuracy

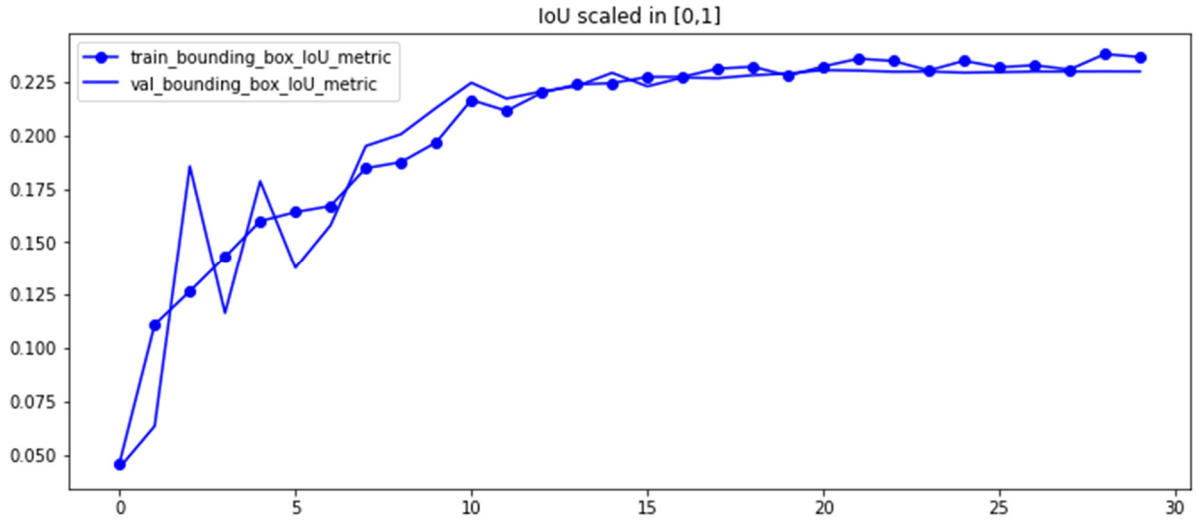


Figure 47: Experiment 6, loss

The results were better than before. The network still arrived at a point in which the learning rate was very low, but during the training, after a couple of reductions of the learning rate, the model with the lowest validation loss performed a bit better on the test set with respect to the previous experiments.

Experiment 6 – Test set performance	
Accuracy	77.57%
Intersection over Union	21.05%

4.7 Experiment 7 – Improvements to hyperparameters

The seventh experiment was similar to the sixth, the architecture is exactly the same as before, so we didn't report it again. In this experiment we focused on some hyperparameters:

- The initial learning rate was reduced from 0.001 to 0.0005
- The reduction factor of the ReduceLRonPlateau was increased from 50% to 80%
- The batch size was reduced from 16 to 8 samples

This experiment was the one with the highest accuracy we were able to obtain with a network from scratch.

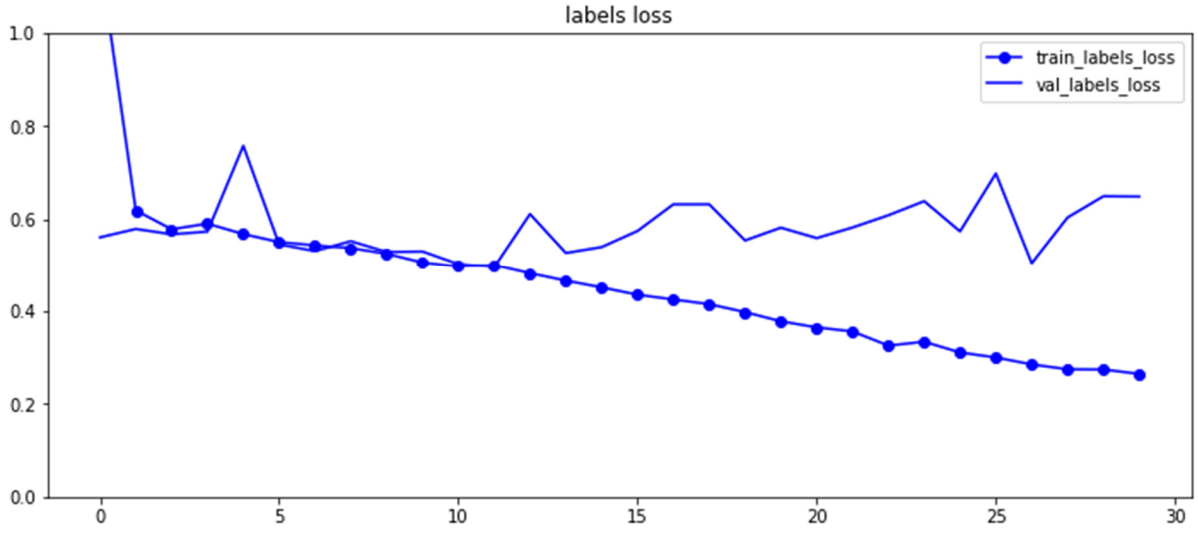


Figure 48: Experiment 7, labels loss

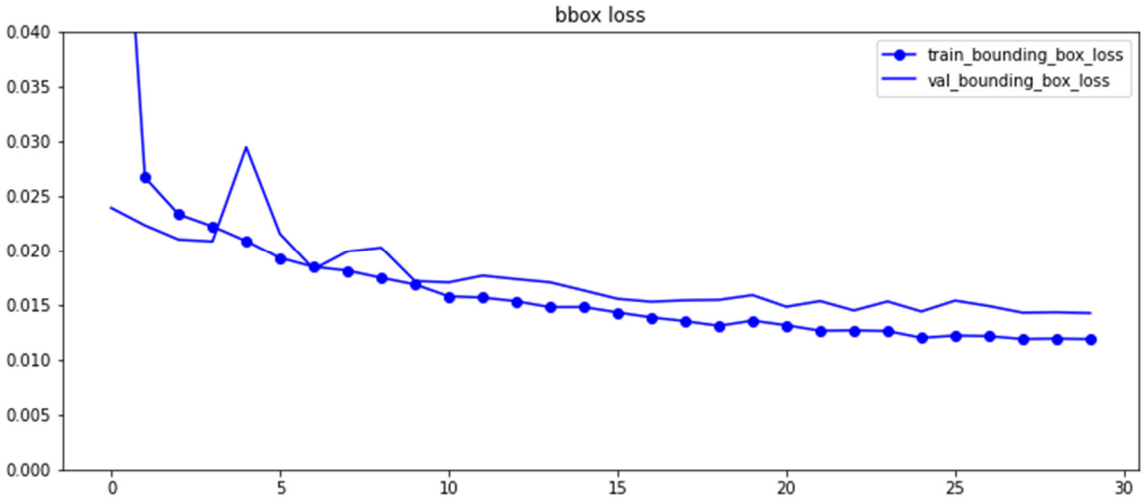


Figure 49: Experiment 7, bounding box loss

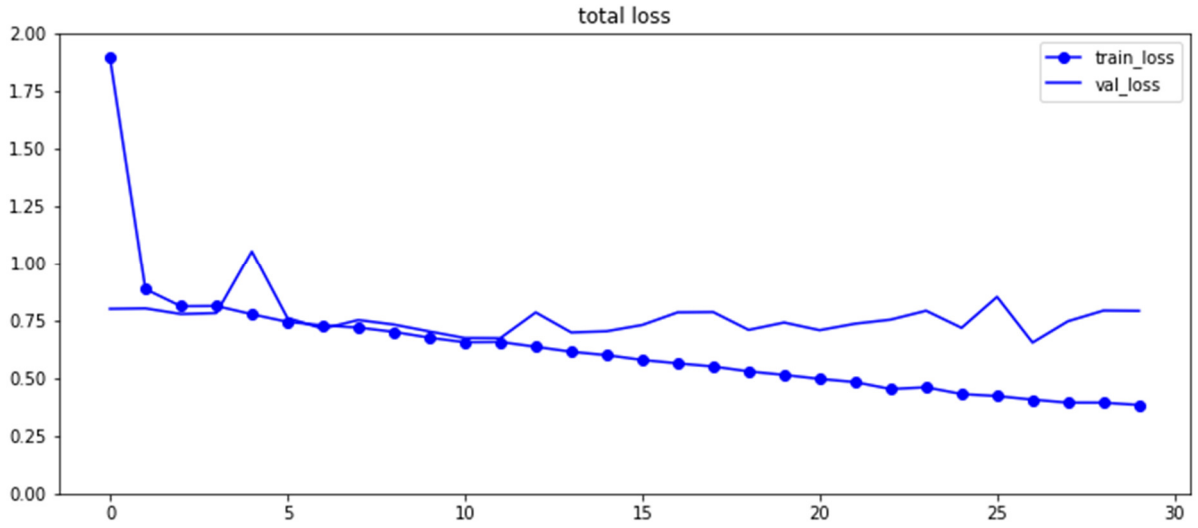


Figure 50: Experiment 7, loss

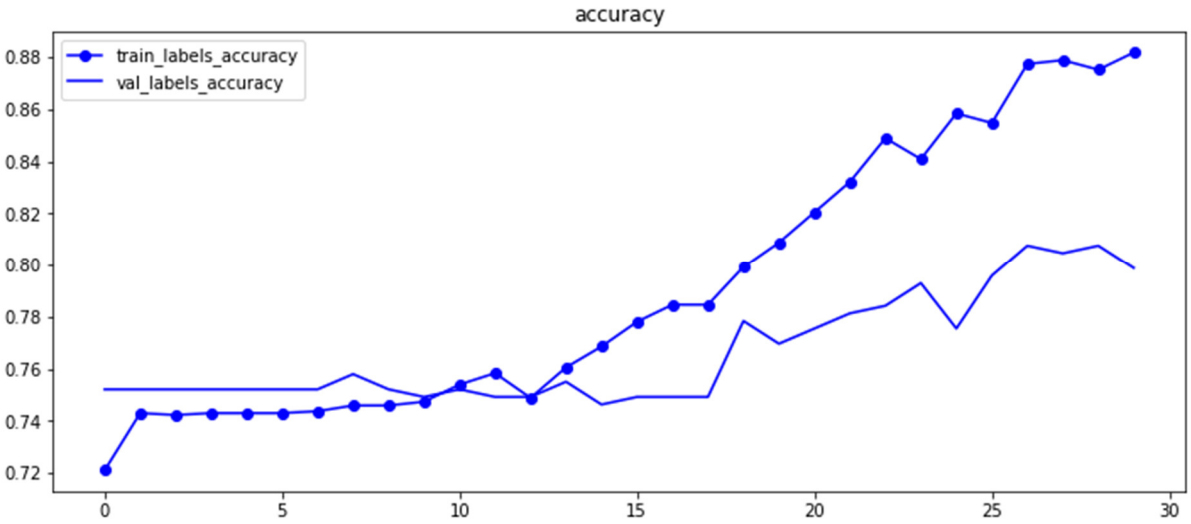


Figure 51: Experiment 7, accuracy

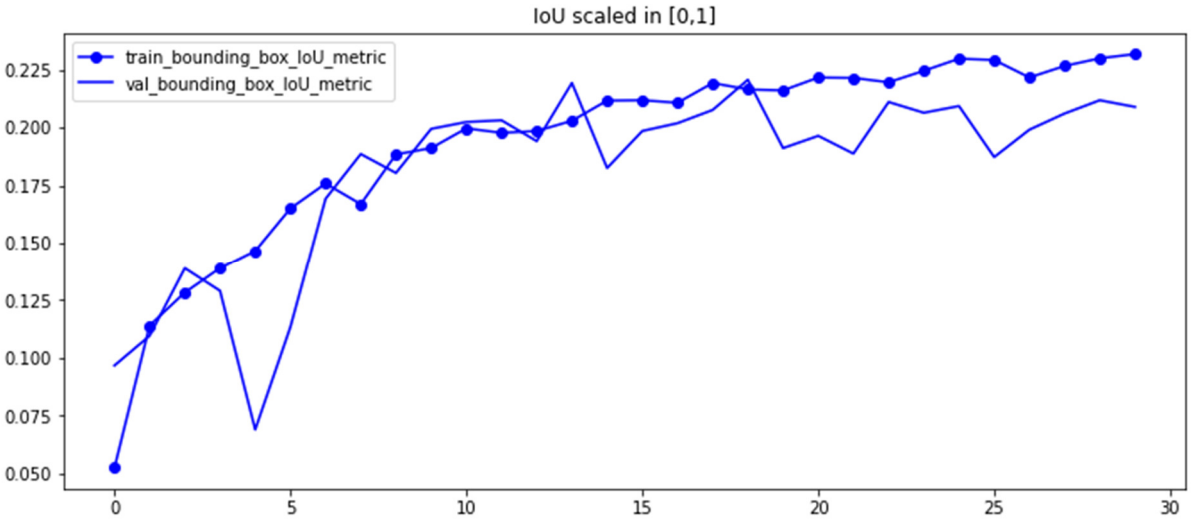
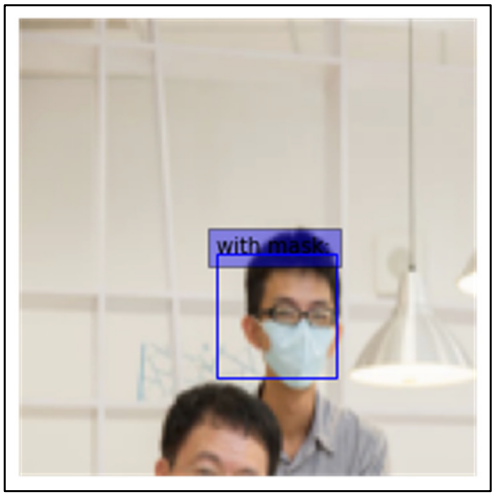


Figure 52: Experiment 7, IoU

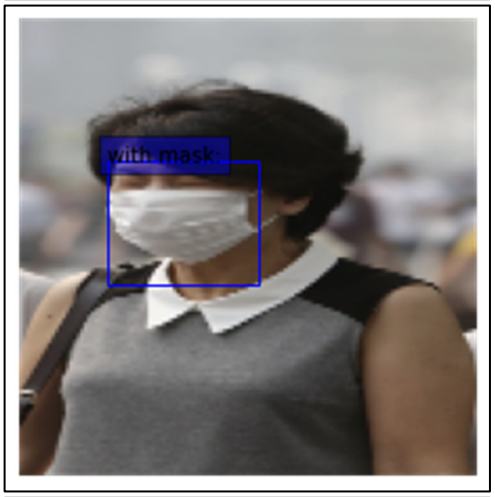
Experiment 7 – Test set performance	
Accuracy	79.44%
Intersection over Union	20.05%

This model has decreased performances for the regression head because in the epoch in which the validation loss was the least, the validation loss on the labels was at his minimum, but the validation loss on the bounding boxes was still decreasing. In general, these two losses should be synchronized as much as possible to obtain a good trade-off between classification accuracy and IoU metric. In any case, with almost 80% of accuracy and an IoU over 20%, we believe this network created from scratch was able to deal with the face mask detection problem. We showed some images of the test set and relative detections below. Some images are noisy and this could also be the reason of why we were not able to get an higher accuracy with this relatively simple network.


with mask


without mask

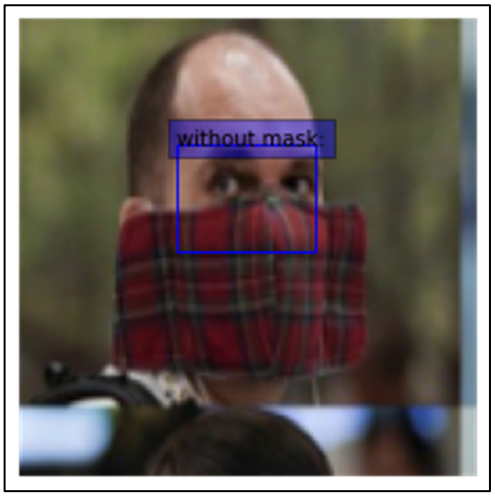

with mask


with mask


without mask


with mask


without mask:


without mask:


with mask:


with mask:

5. PRE-TRAINED NETWORKS

This section describes the results obtained using two different pretrained models (MobileNet and Xception) that were pretrained on other large-scale image datasets.

5.1 MobileNet

MobileNet is a convolutional neural network architecture proposed for mobile and embedded vision applications. We have taken this pretrained model from the tensorflow.keras.application module and we choose the model with the weights obtained during training with the ImageNet dataset and we choose an input of (224,224,3).

5.1.1 Experiment 1 – Base model

The MobileNet has been taken as base model (with frozen weights) and on top of it has been built a GlobalAveragePooling2D layer, a dense layer, a dropout layer and two different output layers (one for mask classification and one for face/mask detection).
MobileNet uses a Global Average Pooling layer to reduce the spatial dimensions of the feature map obtained from the convolutional layers, while retaining the important information. It computes the average of all activation values across all spatial dimension and contains information about the spatial hierarchy. We could use a Flatten layer instead, but it may not be optimal because it simply takes the output of the previous layer and flattens it into a single long vector, losing all spatial information.
As optimizer we use the Root Mean Squared Propagation (RMS) with learning rate 1e-3 and the losses used are the binary cross entropy for the classification head and the mean squared error on the regression head. The loss weights were set at 0.1 for the classification head and 1 for the regression head.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.math.truediv (TFOpLambda)   (None, 224, 224, 3)  0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 224, 224, 3)  0           ['tf.math.truediv[0][0]']

 mobilenetv2_1.00_224 (Function  (None, 7, 7, 1280)  2257984     ['tf.math.subtract[0][0]']
 al)

 global_average_pooling2d (Glob  (None, 1280)        0           ['mobilenetv2_1.00_224[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 256)          327936      ['global_average_pooling2d[0][0]'
                                                                 ]

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 labels (Dense)                 (None, 1)            257         ['dropout[0][0]']

 bounding_box (Dense)           (None, 4)            1028        ['dropout[0][0]']

==================================================================================================
Total params: 2,587,205
Trainable params: 329,221
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

Figure 53: Experiment 1 - MobileNet - Summary

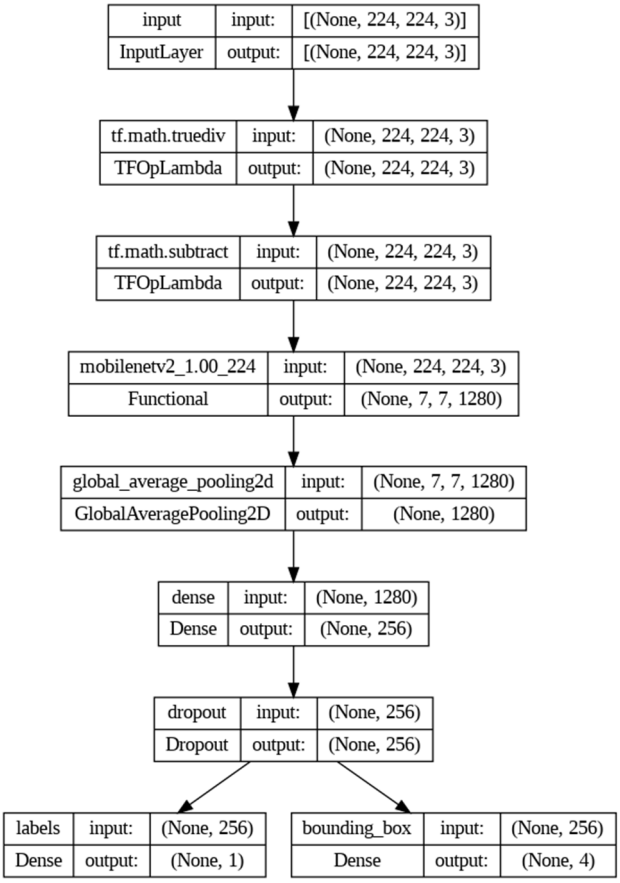


Figure 54: Experiment 1 - MobileNet - Architecture

The model descripted has been trained for 10 epochs and the results obtained from the best model (in terms of lower validation loss) are the following:

	Labels Accuracy	IoU Metric
Train	93.94%	18.16%
Validation	90.09%	19.22%
Test	89.01%	19.09%

Table 1: Experiment 1 - MobileNet - Metrics

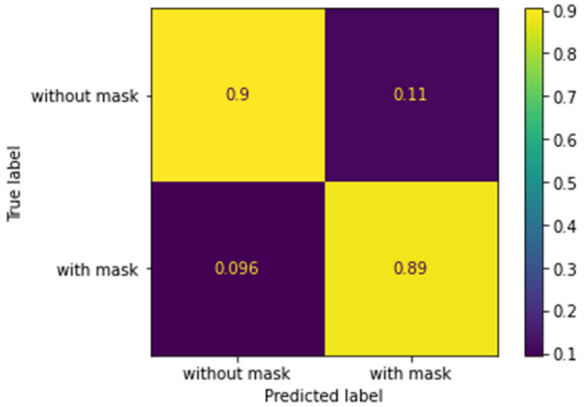


Figure 53: Experiment 1 - MobileNet - Confusion Matrix

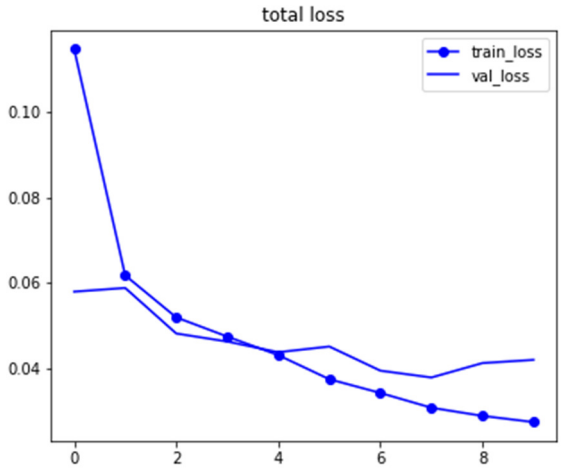


Figure 54: Experiment 1 - MobileNet - Total Loss

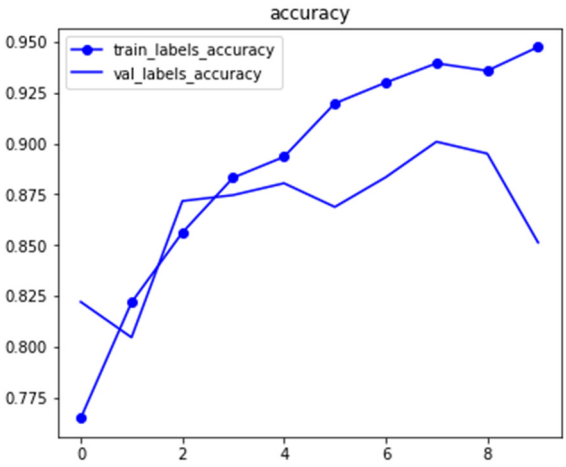


Figure 55: Experiment 1 - MobileNet - Accuracy

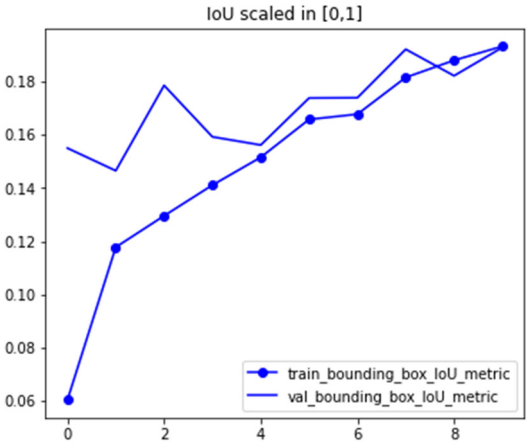


Figure 56: Experiment 1 - MobileNet - IoU scaled

The model shows good performance in terms of accuracy, but it has room for improvement in terms of precision, particularly in detecting objects of interest. Respect to the from scratch network, we can observe that we obtained good results even with a simple model without further improvements.

5.1.2 Experiment 2 - Double Dropout Layer

As second experiment, we tried to add a dropout layer for each output layer that we have with different dropout rates, 0.7 for the dropout related to the classification head, and 0.8 for the dropout layer related to the regression head.

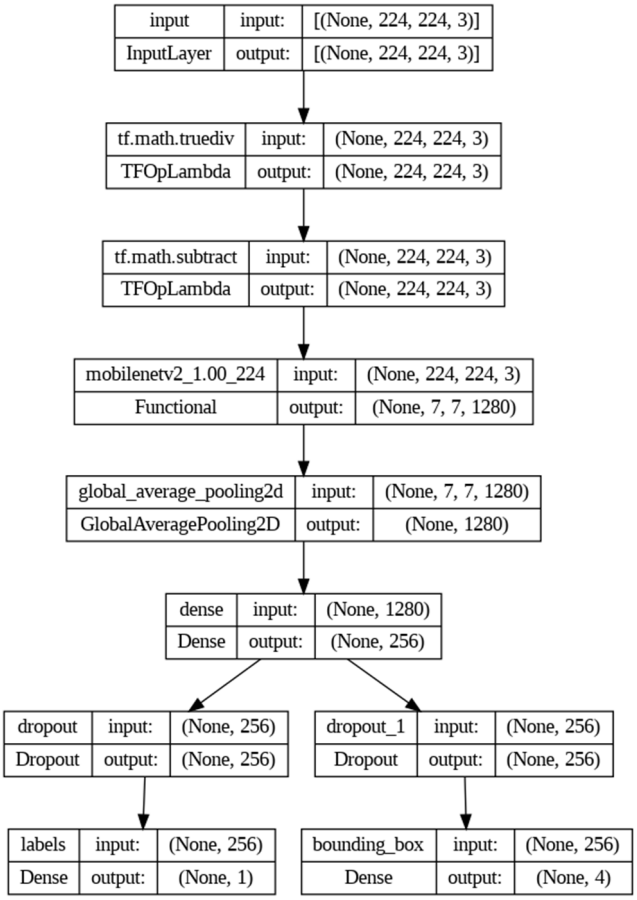


Figure 57: Experiment 2 - MobileNet - Architecture

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.math.truediv (TFOpLambda)   (None, 224, 224, 3)  0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 224, 224, 3)  0           ['tf.math.truediv[0][0]']

 mobilenetv2_1.00_224 (Function  (None, 7, 7, 1280)  2257984     ['tf.math.subtract[0][0]']
 al)

 global_average_pooling2d (Glob  (None, 1280)        0           ['mobilenetv2_1.00_224[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 256)          327936      ['global_average_pooling2d[0][0]'
                                                                 ]

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 dropout_1 (Dropout)            (None, 256)          0           ['dense[0][0]']

 labels (Dense)                 (None, 1)            257         ['dropout[0][0]']

 bounding_box (Dense)           (None, 4)            1028        ['dropout_1[0][0]']

==================================================================================================
Total params: 2,587,205
Trainable params: 329,221
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

Figure 58: Experiment 2 - MobileNet - Summary

The model described has been trained for 10 epochs and the results obtained from the best model (in terms of lower validation loss) are the following:

	Labels Accuracy	IoU Metric
Train	87.96%	14.01%
Validation	87.46%	16.20%
Test	86.68%	16.70%

Table 2: Experiment 2 - MobileNet - Metrics

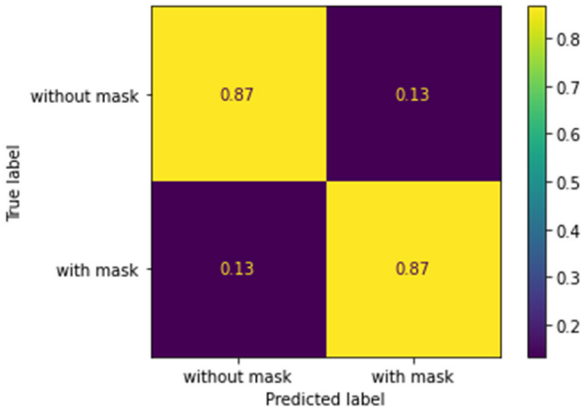


Figure 59: Experiment 2 - MobileNet - Confusion Matrix

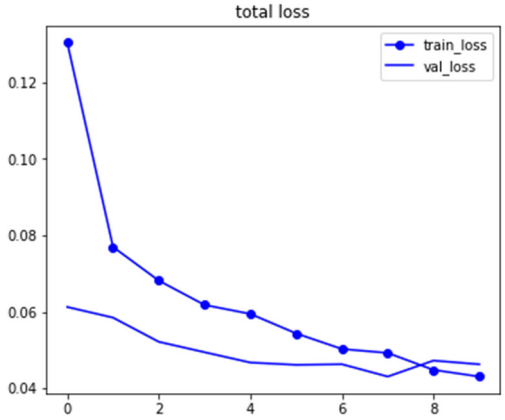


Figure 60: Experiment 2 - MobileNet – Total Loss

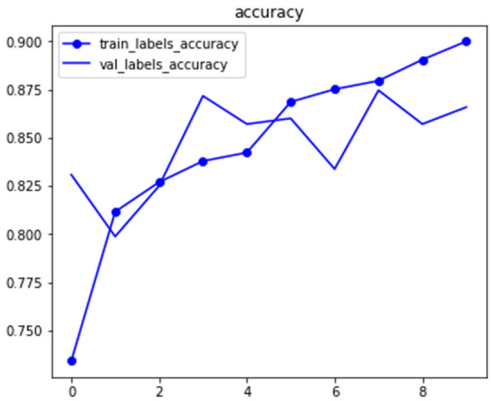


Figure 61: Experiment 2 - MobileNet - Accuracy

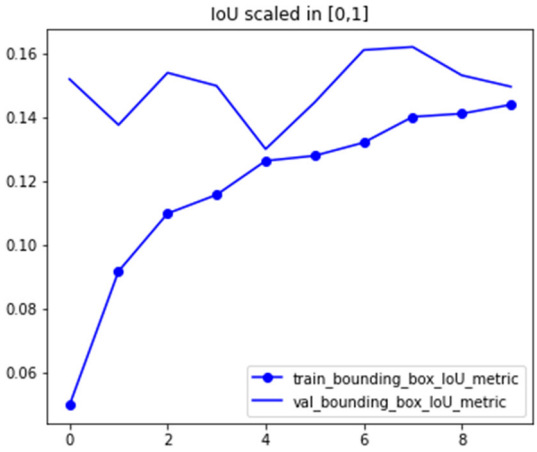


Figure 62: Experiment 2 - MobileNet – IoU scaled

As we can see from the results that we obtained, adding two different dropouts layers for the two outputs doesn't improve the performance of the model as we obtain lower metrics.

5.1.3 Experiment 3 - Smooth L1 loss

As third experiment, we tried to use a different loss function for the IoU metric. We changed the Mean Squared Error loss function with the Smooth L1 loss.
The architecture and the summary are the same as the experiment 1.
The model described has been trained for 10 epochs and the results obtained from the best model (in terms of lower validation loss) are the following:

	Labels Accuracy	IoU Metric
Train	94.60%	16.07%
Validation	89.21%	18.52%
Test	87.38%	19.15%

Table 3: Experiment 3 - MobileNet - Metrics

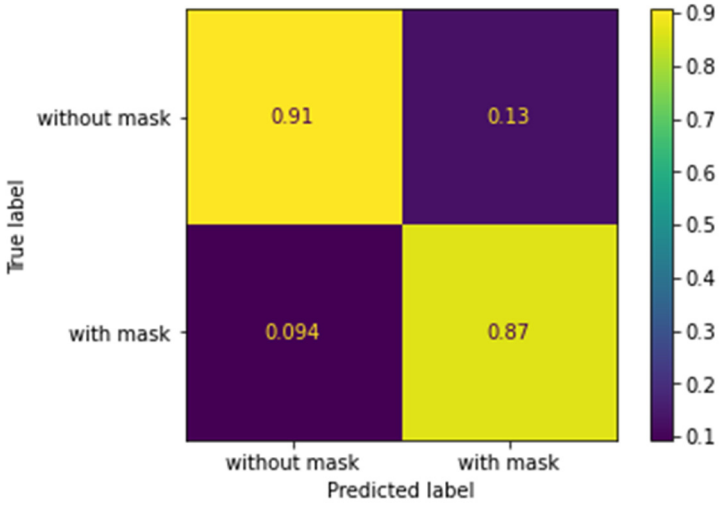


Figure 63: Experiment 3 - MobileNet - Confusion Matrix

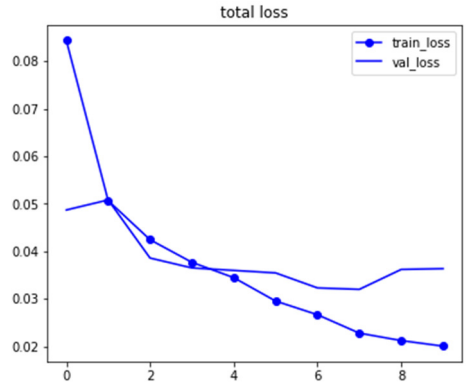


Figure 64: Experiment 3 - MobileNet - Total Loss

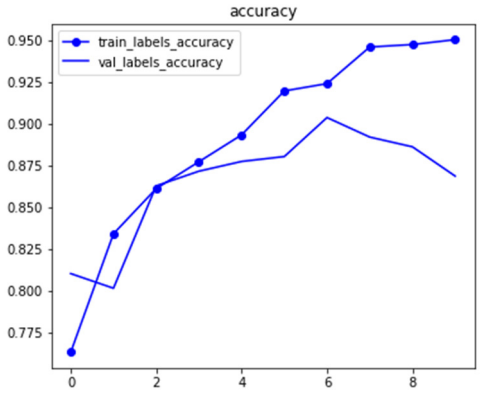


Figure 65: Experiment 3 - MobileNet – Accuracy

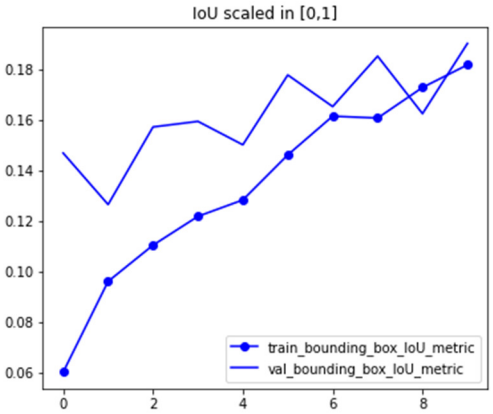


Figure 66: Experiment 3 - MobileNet - IoU scaled

Also in this case, the first experiment still obtains better results in terms accuracy and IoU metrics.

5.1.4 Experiment 4 - Multiple Dense Layers

As fourth experiment, we tried to add multiple hidden dense layers instead of a single one to increase the "complexity" of the data represented by the network. In this experiment we tried with a "constant" of three dense layer composed by 256 neurons each, but as we will see in the Hyperparameters tuning section, we will try with a different amount of hidden dense layers with a different number of hidden neurons.

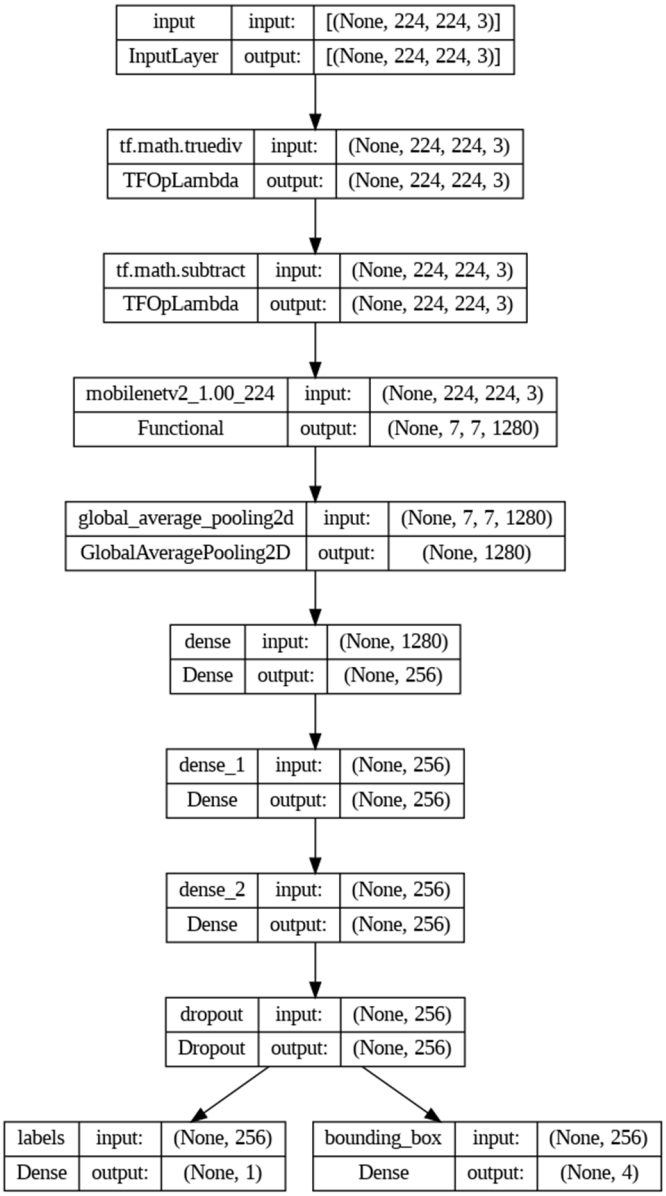


Figure 67: Experiment 4 - MobileNet - Architecture

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input (InputLayer)          [(None, 224, 224, 3       0           []
                             )]

 tf.math.truediv (TFOpLambda)   (None, 224, 224, 3)   0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 224, 224, 3)   0           ['tf.math.truediv[0][0]']

 mobilenetv2_1.00_224 (Function  (None, 7, 7, 1280)   2257984     ['tf.math.subtract[0][0]']
 al)

 global_average_pooling2d (Glob  (None, 1280)         0           ['mobilenetv2_1.00_224[0][0]']
 alAveragePooling2D)

 dense (Dense)               (None, 256)               327936      ['global_average_pooling2d[0][0]'
                                                                  ]

 dense_1 (Dense)             (None, 256)               65792       ['dense[0][0]']

 dense_2 (Dense)             (None, 256)               65792       ['dense_1[0][0]']

 dropout (Dropout)           (None, 256)               0           ['dense_2[0][0]']

 labels (Dense)              (None, 1)                 257         ['dropout[0][0]']

 bounding_box (Dense)        (None, 4)                 1028        ['dropout[0][0]']

=================================================================================================
Total params: 2,718,789
Trainable params: 460,805
Non-trainable params: 2,257,984
_________________________________________________________________________________________________
```

Figure 68: Experiment 4 - MobileNet - Summary

The model descripted has been trained for 10 epochs and the results obtained from the best model (in terms of lower validation loss) are the following:

	Labels Accuracy	IoU Metric
Train	96.64%	18.78%
Validation	89.21%	20.84%
Test	90.65%	20.28%

Table 4: Experiment 4 - MobileNet – Metrics

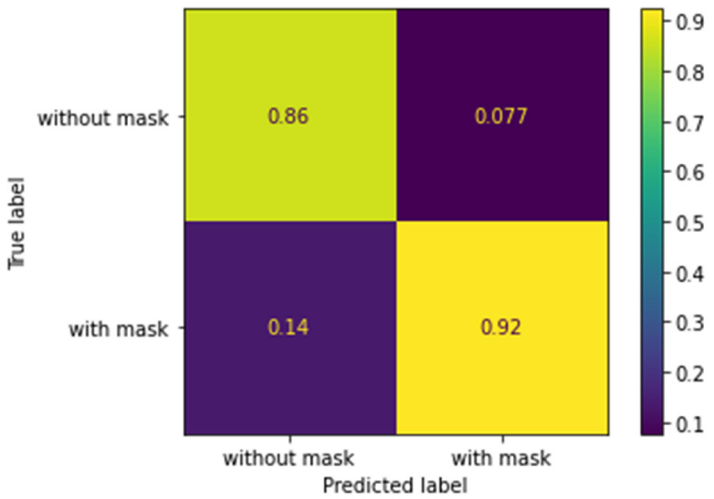


Figure 69: Experiment 4 - MobileNet - Confusion Matrix

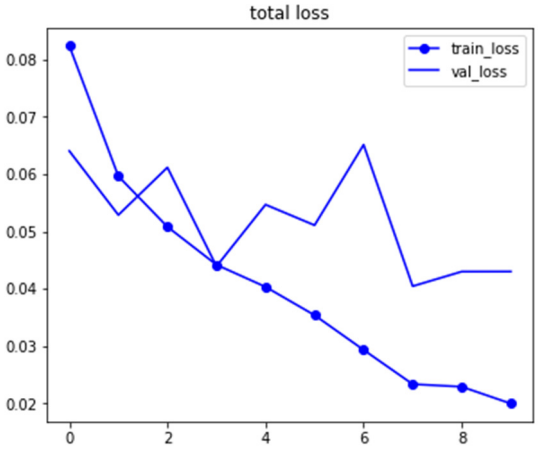


Figure 70: Experiment 4 - MobileNet - Total Loss

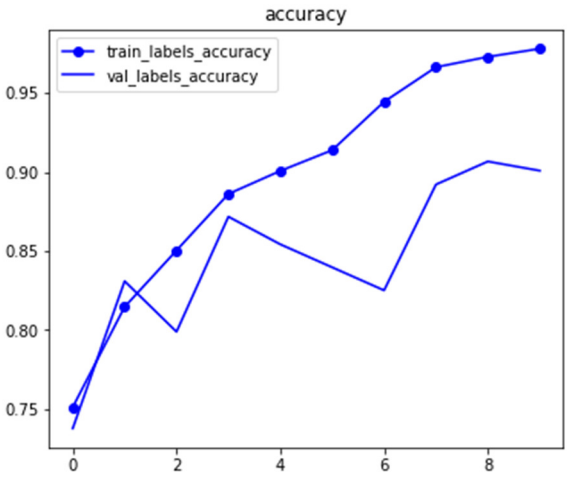


Figure 71: Experiment 4 - MobileNet – Accuracy

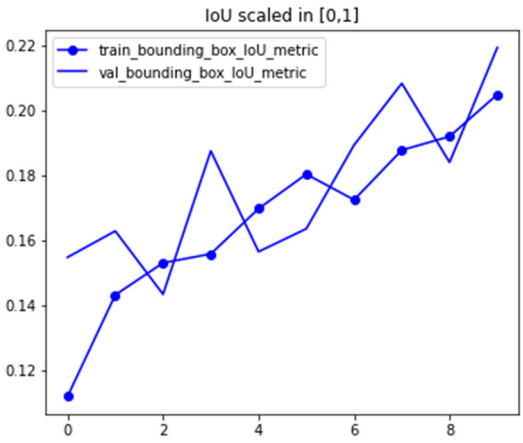


Figure 72: Experiment 4 - MobileNet - IoU scaled

Overall, this model has higher accuracy on the training set, but lower accuracy on the validation and test sets compared to the first experiment model. This model also has slightly higher IoU values, but higher false negative rate compared to the first experiment model. So, depending on the use case, one model may perform better than the other.
For those results, we decided to add this test into the Hyperparameter tuning.

5.1.5 Experiment 5 - Hyperparameters tuning

As fifth experiment, we tried to apply a Hyperparameters tuning to find the best ones and see the results that we can obtain.
We used the Hyperband, from the Keras Tuner library, which trains many models for a few epochs and carries forward only the top-performing half of models to the "next round".
The hyperparameters that we have tried to tune are the following:

	Trial Values
Dense hidden layer number of neurons	Values between 128 and 2048 with a step of 128.
Dropout percentage	Values between 0.5 and 0.8 with a step of 0.1
Number of Dense hidden layers	Values 1, 2 and 3
Learning rate	Values 1e-2, 1e-3, 1e-4 and 1e-5

The Hyperband has found that the best parameters are:

	Best values
Dense hidden layer number of neurons	1536
Dropout percentage	0.5
Number of Dense hidden layers	2
Learning rate	1e-4

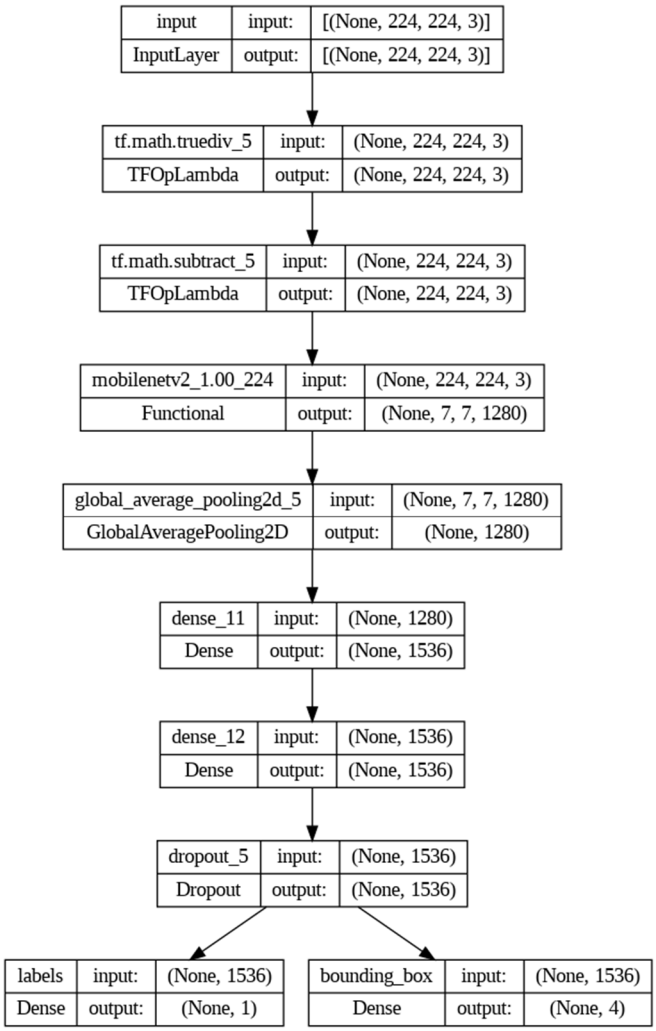


Figure 73: Experiment 5 - MobileNet - Architecture

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 224, 224, 3  0           []
                                )]

 tf.math.truediv_5 (TFOpLambda)  (None, 224, 224, 3)  0          ['input[0][0]']

 tf.math.subtract_5 (TFOpLambda  (None, 224, 224, 3)  0          ['tf.math.truediv_5[0][0]']
 )

 mobilenetv2_1.00_224 (Function  (None, 7, 7, 1280)  2257984     ['tf.math.subtract_5[0][0]']
 al)

 global_average_pooling2d_5 (Gl  (None, 1280)         0          ['mobilenetv2_1.00_224[37][0]']
 obalAveragePooling2D)

 dense_11 (Dense)               (None, 1536)         1967616     ['global_average_pooling2d_5[0][0
                                                                 ]']

 dense_12 (Dense)               (None, 1536)         2360832     ['dense_11[0][0]']

 dropout_5 (Dropout)            (None, 1536)         0           ['dense_12[0][0]']

 labels (Dense)                 (None, 1)            1537        ['dropout_5[0][0]']

 bounding_box (Dense)           (None, 4)            6148        ['dropout_5[0][0]']

==================================================================================================
Total params: 6,594,117
Trainable params: 4,336,133
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

Figure 74: Experiment 5 - MobileNet – Summary

The model descripted has been trained for 13 epochs because has been found that the validation loss is the lower at this number of epoch and the results obtained from the best model (in terms of lower validation loss) are the following:

	Labels Accuracy	IoU Metric
Train	99.64%	25.66%
Validation	92.71%	20.08%
Test	91.12%	20.17%

Table 5: Experiment 5 - MobileNet – Metrics

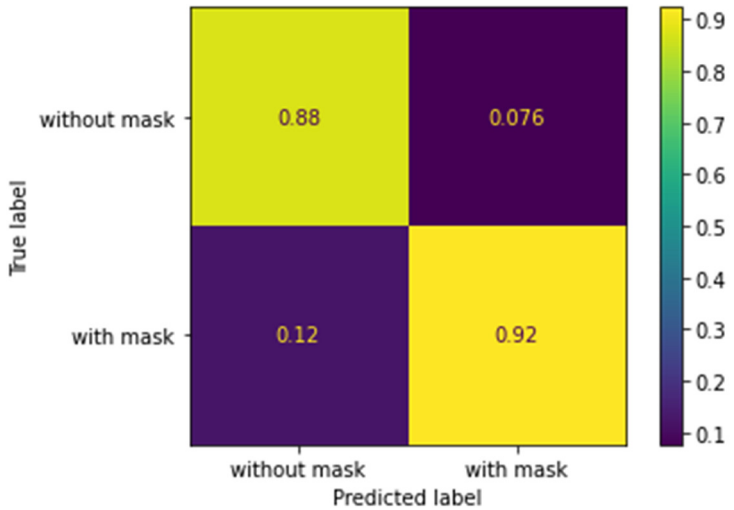


Figure 75: Experiment 5 - MobileNet – Confusion Matrix

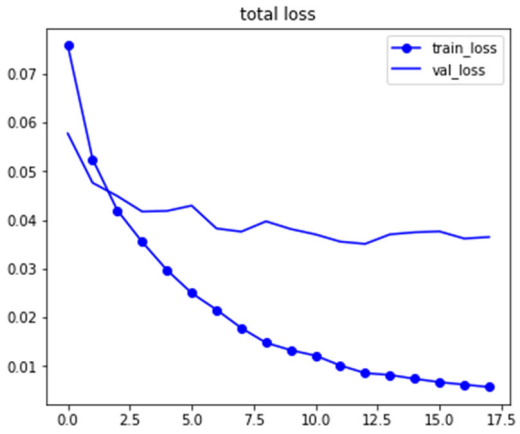


Figure 76: Experiment 5 - MobileNet – Total Loss

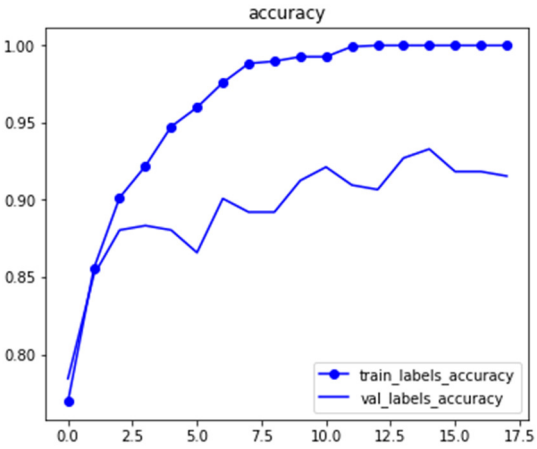


Figure 77: Experiment 5 - MobileNet – Accuracy

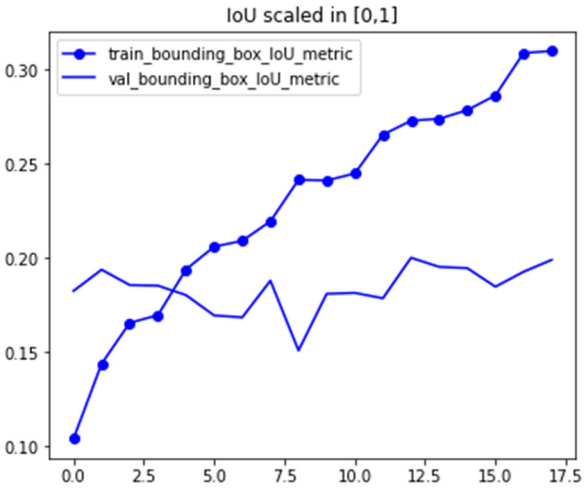


Figure 78: Experiment 5 - MobileNet – IoU scaled

As expected, we improved the metrics. Further experiment and improvements could be going to investigate how to reduce the overfitting, we will do this in this last experiment (experiment 6) adding a dropout layer foreach dense layer.

5.1.6 Experiment 6 - Reduce overfitting

The sixth experiment we will try to add a dropout layer foreach dense layer in the model described in the experiment 5.
This should reduce overfitting and improve the performances.

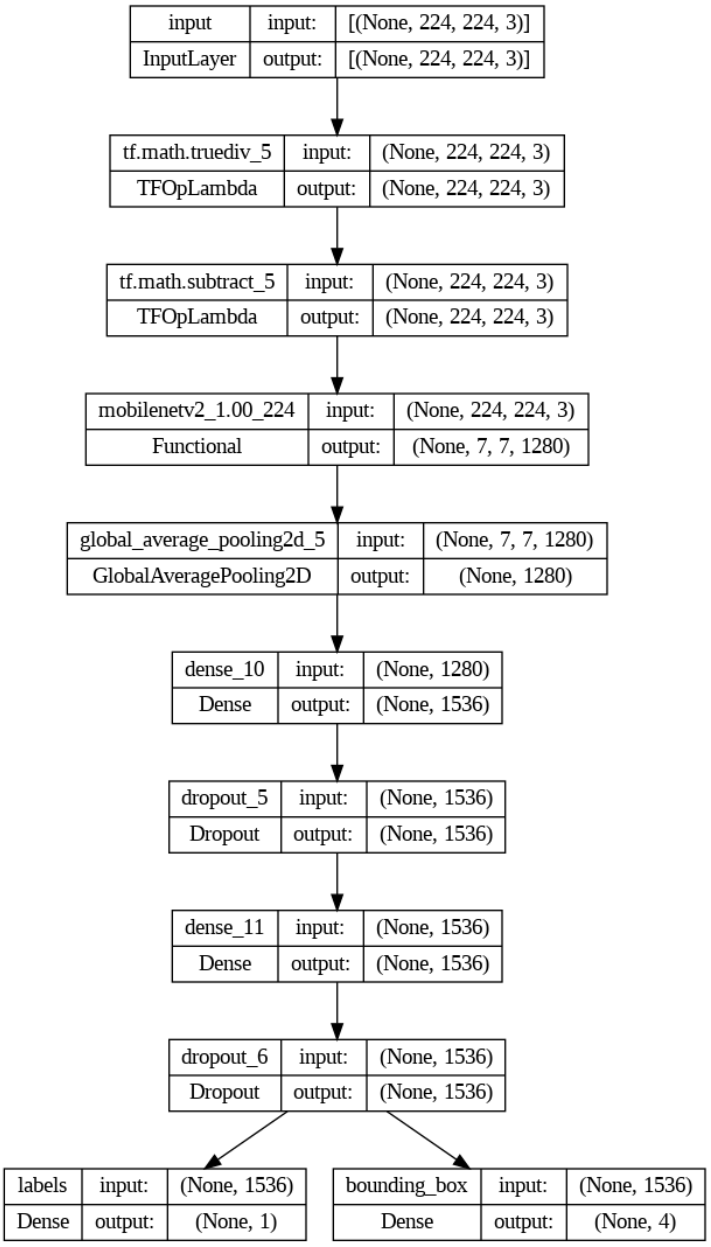


Figure 79: Experiment 6 - MobileNet - Architecture

The summary of the model is the same as experiment 5.
The model descripted has been trained using two different callbacks:

- Early Stopping: It is a regularization technique used in deep learning to prevent overfitting. The idea behind early stopping is to stop training the model once its performance on a validation set stops improving, to prevent the model from memorizing the training data and overfitting to it.
- ReduceLROnPlateau: It is a callback function used to dynamically adjust the learning rate during training. The idea is to reduce the learning rate when the validation loss plateaus, meaning that it stops improving.

The results obtained are the following:

	Labels Accuracy	IoU Metric
Train	99.56%	21.64%
Validation	92.13%	21.73%
Test	92.75%	21.36%

Table 5: Experiment 6 - MobileNet – Metrics

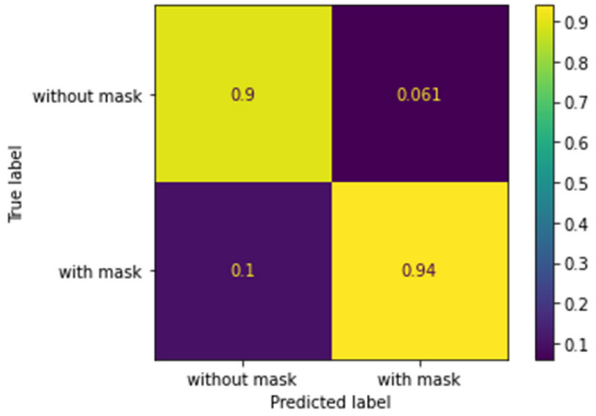


Figure 80: Experiment 6 - MobileNet – Confusion Matrix

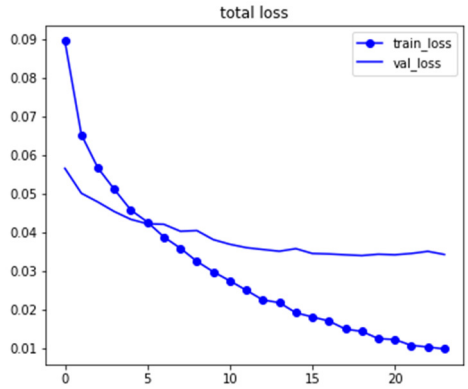


Figure 81: Experiment 6 - MobileNet – Total Loss

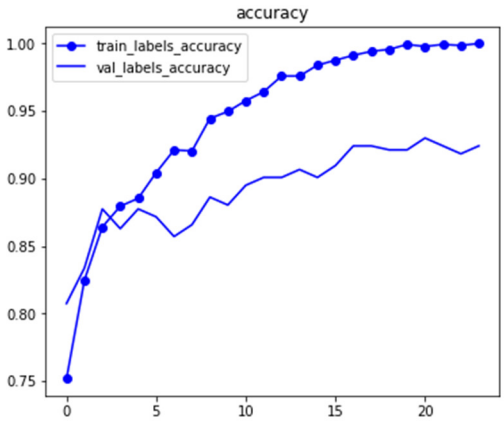


Figure 82: Experiment 6 - MobileNet – Accuracy

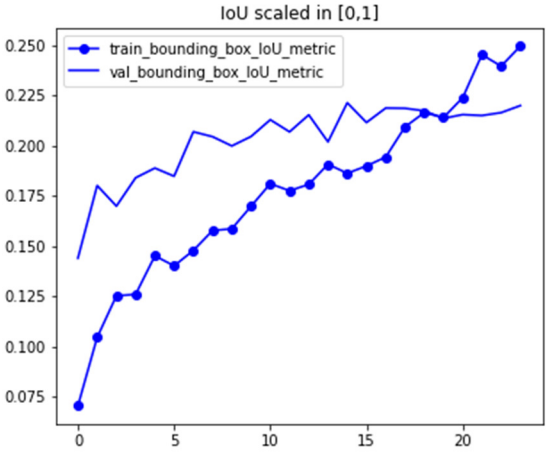


Figure 83: Experiment 6 - MobileNet – IoU

As we can see, the results obtained are overall better than the experiment 5 results.
The accuracy and IoU metric scores are higher on both the validation and test sets, indicating that the model is generalizing better to new, unseen data.
The confusion matrix also provides some additional insights. In both the results (experiment 5 and 6), the model appears to be making relatively few false positive predictions. This is a good sign, as it means that the model is not classifying many samples as positive that should be negative. In terms of the false negative predictions, the results of the experiment 6 appears to have fewer false negatives, than in the results in experiment 5. This indicates that the model is more effectively identifying positive samples.
We show below some of the results of this model:

5.2 Xception

A new model starting from a pretrained network was build using the Xception pretrained architecture. We have taken this pretrained model from the tensorflow.keras.application module and we choose the model with the weights obtained during training with the ImageNet dataset and we choose an input of (128,128,3). For our purpose we used only the convolutional base of the pretrained network to extract the learned general and reusable representations. We decided to use the Xception pretrained network because it is a network that is less prone to overfitting, and this was very important to us because it was one of the major challenges considering the small dimension of our dataset.
On top of the convolutional base, we inserted a GlobalAveragePooling2D layer to flatten the dimensions and a Dense layer of 256 units using the ReLu activation function and a dropout layer with a parameter of 0.5 to fight overfitting.
On top of this we inserted a regression head and a classification head to obtain the correct prediction for each image. As for the final model of the network from scratch, the regression head is a dense layer of 4 neurons using the sigmoid activation function, each one in charge of predicting one of the coordinates of the bounding box. The classification head is a dense layer with only 1 neuron using a sigmoid activation function to perform the binary classification task.

5.2.1 Experiment 1 – Base model

In the first experiment we used the build base pretrained model as described above. The loss function used were the binary cross entropy for the classification head and the mean squared error for the regression head. These 2 losses were weighted using a weight of 0.1 for the label loss and 1.0 for the bounding box loss.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 128, 128, 3  0           []
                                )]

 tf.math.truediv (TFOpLambda)   (None, 128, 128, 3)  0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 128, 128, 3)  0           ['tf.math.truediv[0][0]']

 xception (Functional)          (None, 4, 4, 2048)   20861480    ['tf.math.subtract[0][0]']

 global_average_pooling2d (Glob  (None, 2048)        0           ['xception[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 256)          524544      ['global_average_pooling2d[0][0]'
                                                                 ]

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 labels (Dense)                 (None, 1)            257         ['dropout[0][0]']

 bounding_box (Dense)           (None, 4)            1028        ['dropout[0][0]']

==================================================================================================
Total params: 21,387,309
Trainable params: 525,829
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
```

Figure 86: Experiment 1 – Xception - summary

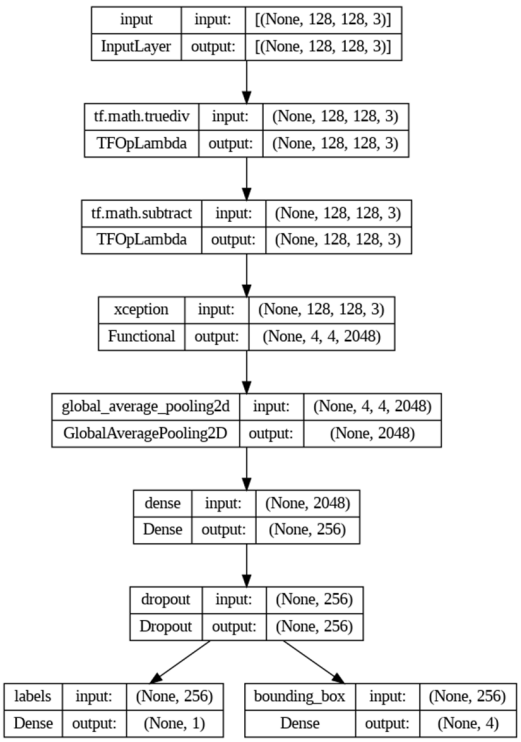


Figure 87: Experiment 1 – Xception - architecture

We trained this model for 10 epochs using a validation split of 0.2 and a batch size of 32 samples. As we did before we used the ModelCheckpoint call-back to monitor the val-loss value and save the result with the minimum value on this parameter.

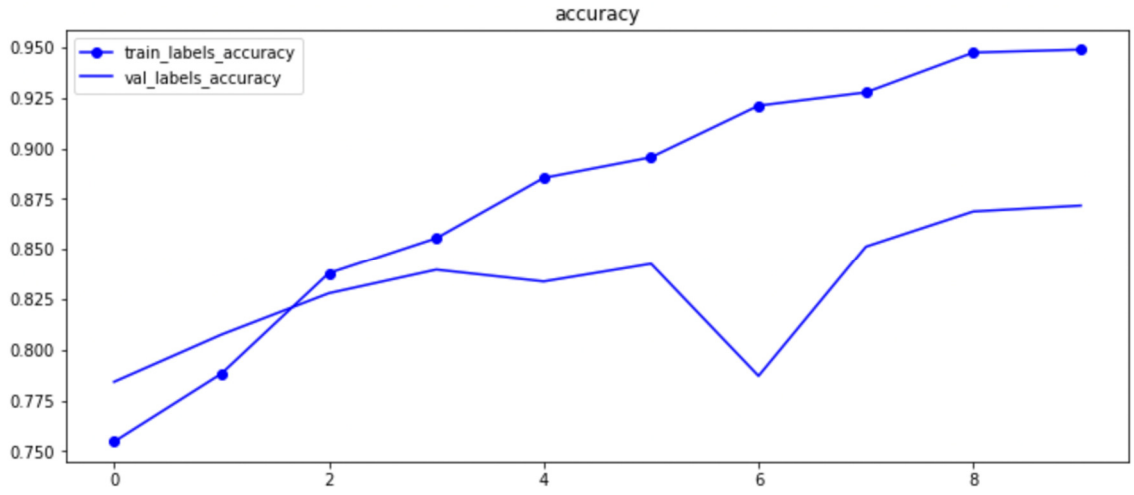


Figure 88: Experiment 1 – Xception - Accuracy

Base model – Test set performance	
Accuracy	85,28%
Intersection over Union	18,33%

Table 6: Base model test set performance

The performance obtained are very good if compared with the base model of the scratch network but as we can see from the accuracy plot the model overfit so in the next experiment, we try to fight this.

5.2.2 Experiment 2 – Adding different dropout layers

To fight overfitting, we tried to insert two different dropout layers before the classification head and regression head using the same parameters that gave good results in the scratch network: 0.7 for the regression head and 0.8 for the classification head. In the figure 89 is shown the new model architecture and in the figures 90 and table 7 are shown the results obtained with this new network.

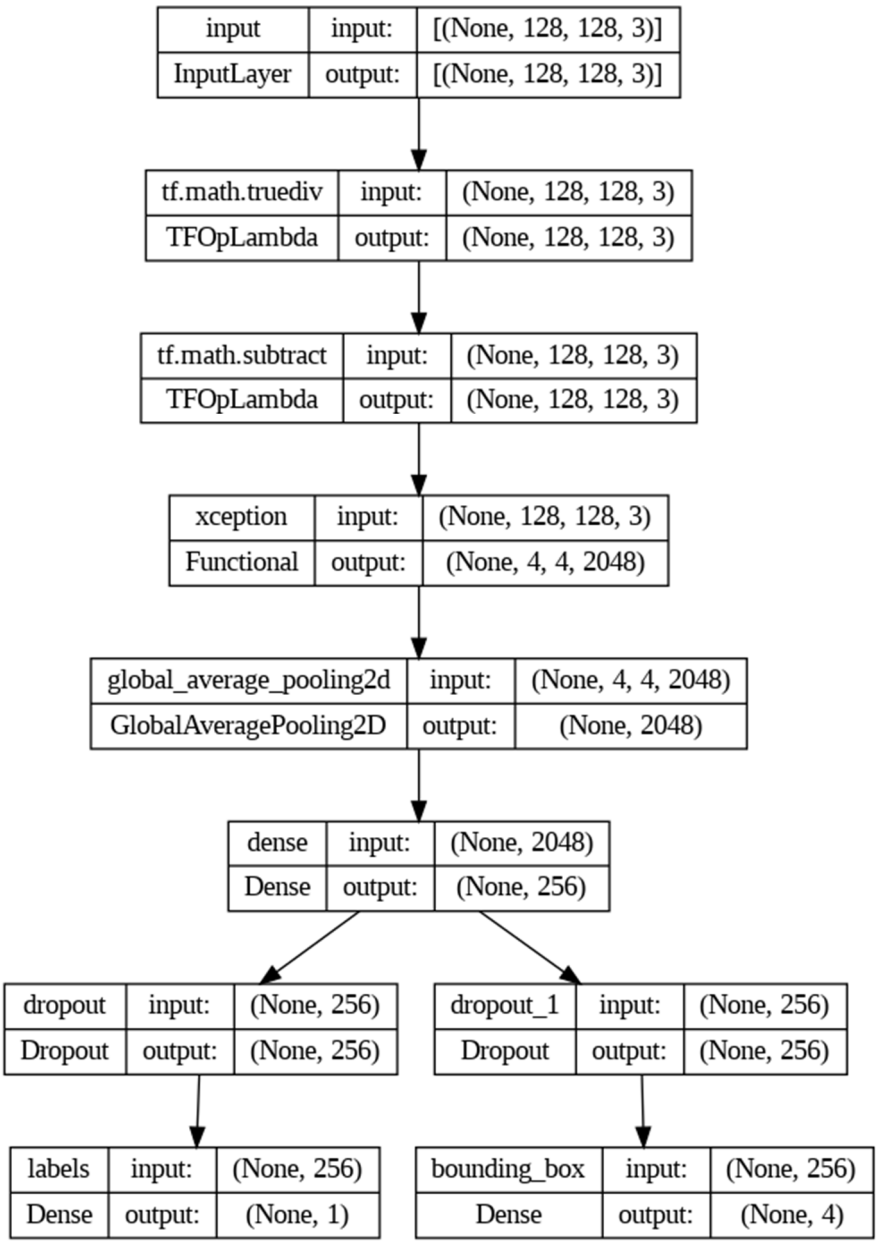


Figure 89: Experiment 2 – Xception - architecture

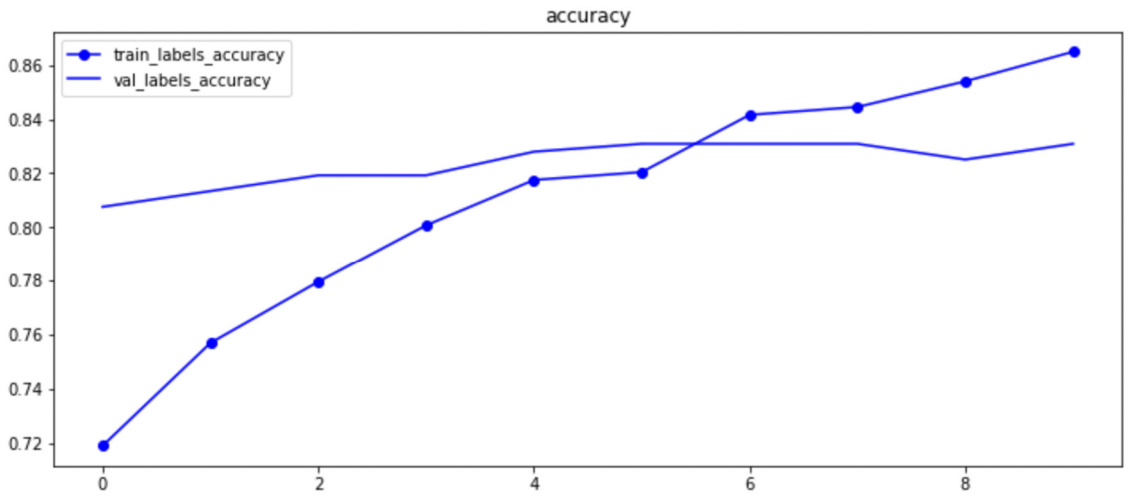


Figure 90: Experiment 2 – Xception - Accuracy

Base model – Test set performance	
Accuracy	82,25%
Intersection over Union	14,97%

Table 7: Experiment 2 test set performance

As we can see from the accuracy plot adding the two dropout layers reduced overfitting even if the results on the test set are slightly worse. Comparing the confusion matrix with the base model we can see that we obtain a good increase in the prediction of the images with the "without mask" label. In the figure 91 is show the comparison between the two confusion matrices.

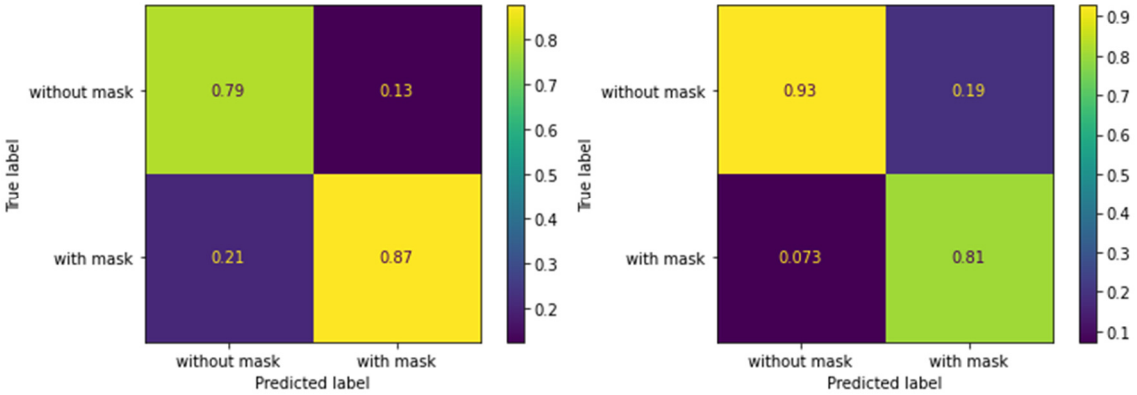


Figure 91: Experiment 1 and 2 – Xception - confusion matrices comparison

5.2.3 Experiment 3 – Smooth L1 loss

In the third experiment we tried to use the Smooth L1 loss function for the IoU metric to see if we could obtain better results in this metric. In the table 8 are shown the results obtained.

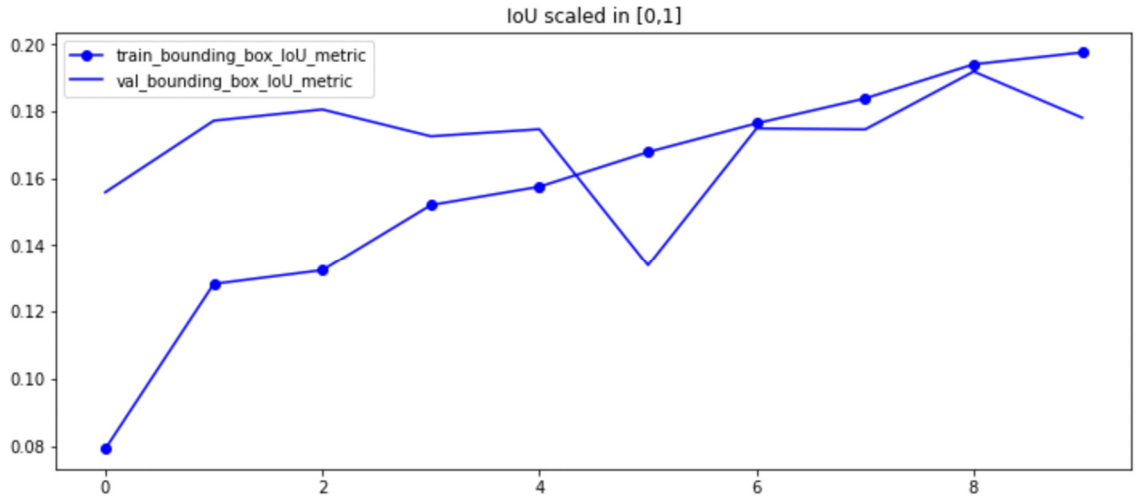


Figure 92: Experiment 1 – Xception - IoU metric results

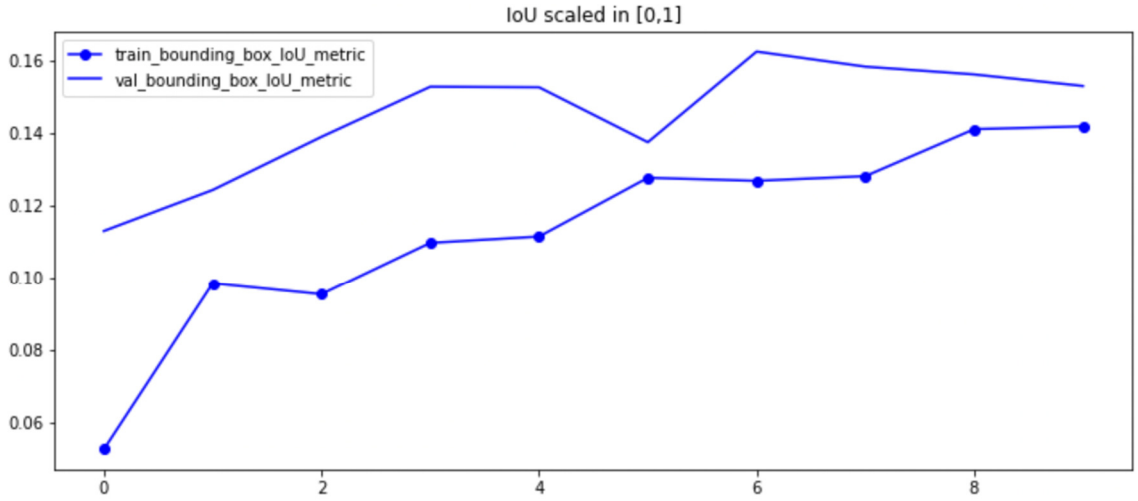


Figure 93: Experiment 3 – Xception - IoU metric results

Base model and experiment 3 IoU metric comparison	
Base model	18,33%
Experiment 3 model	15,37%

Table 8: Base model and experiment 3 model IoU metric results

As we can see the results obtained using the Smooth L1 loss didn't improve so we decided to continue using the Mean square error loss function in the next experiments.

5.2.4 Experiment 4 – Multiple dense layers

In the fourth experiment we tried to increment the model complexity to achieve better performance. We did that increasing the number of dense layers after the convolution base of the pretrained network inserting other two dense layers with 256 units shifting from 1 to 3 dense layers in total. The new model architecture is shown in the figure 94 and the results obtained from training are shown in the figures 95, 96 and table 9.

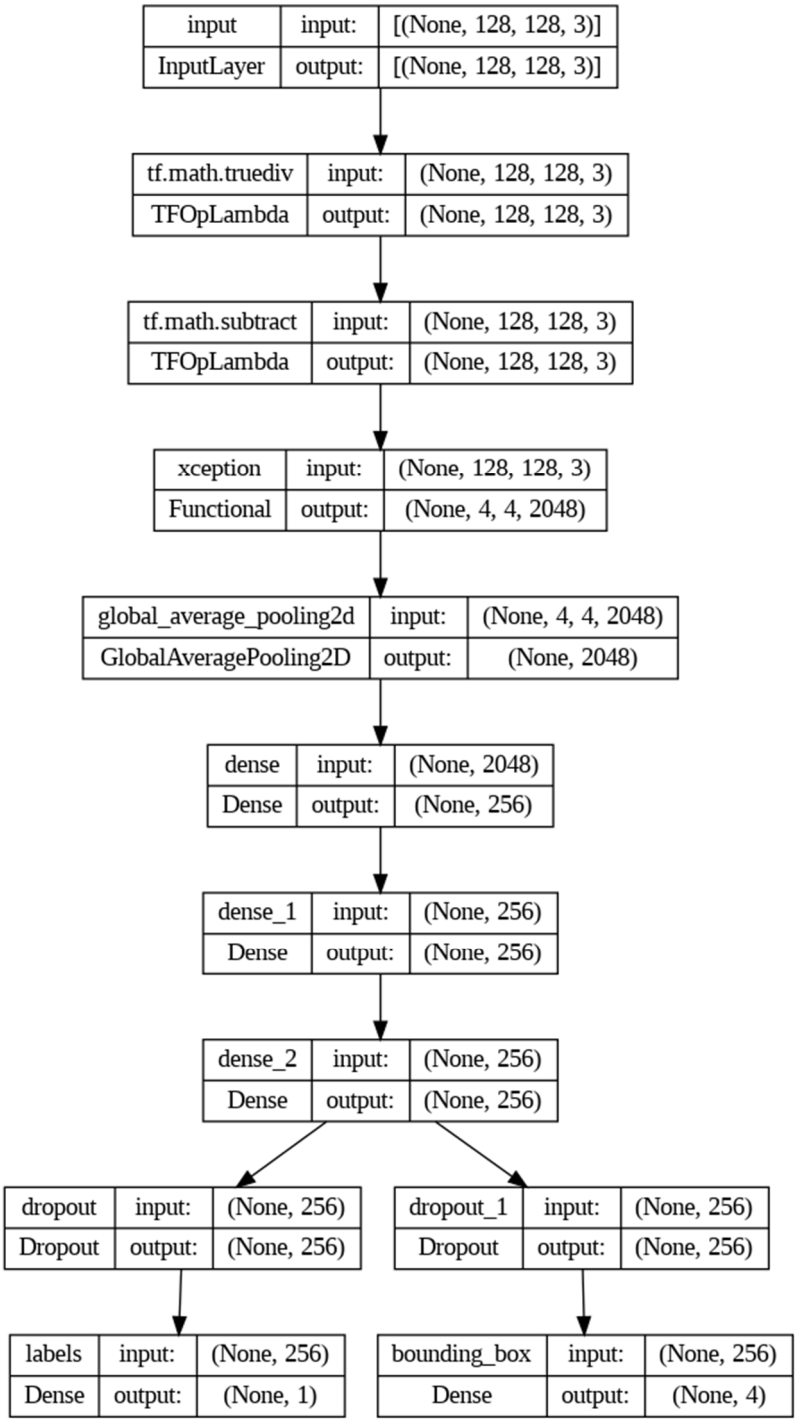


Figure 94: Experiment 4 – Xception - Architecture

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input (InputLayer)             [(None, 128, 128, 3  0           []
                                )]

 tf.math.truediv (TFOpLambda)   (None, 128, 128, 3)  0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 128, 128, 3)  0           ['tf.math.truediv[0][0]']

 xception (Functional)          (None, 4, 4, 2048)   20861480    ['tf.math.subtract[0][0]']

 global_average_pooling2d (Glob  (None, 2048)        0           ['xception[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 256)          524544      ['global_average_pooling2d[0][0]'
                                                                 ]

 dense_1 (Dense)                (None, 256)          65792       ['dense[0][0]']

 dense_2 (Dense)                (None, 256)          65792       ['dense_1[0][0]']

 dropout (Dropout)              (None, 256)          0           ['dense_2[0][0]']

 dropout_1 (Dropout)            (None, 256)          0           ['dense_2[0][0]']

 labels (Dense)                 (None, 1)            257         ['dropout[0][0]']

 bounding_box (Dense)           (None, 4)            1028        ['dropout_1[0][0]']

=================================================================================================
Total params: 21,518,893
Trainable params: 657,413
Non-trainable params: 20,861,480
_________________________________________________________________________________________________
```

Figure 95: Experiment 4 – Xception - Summary

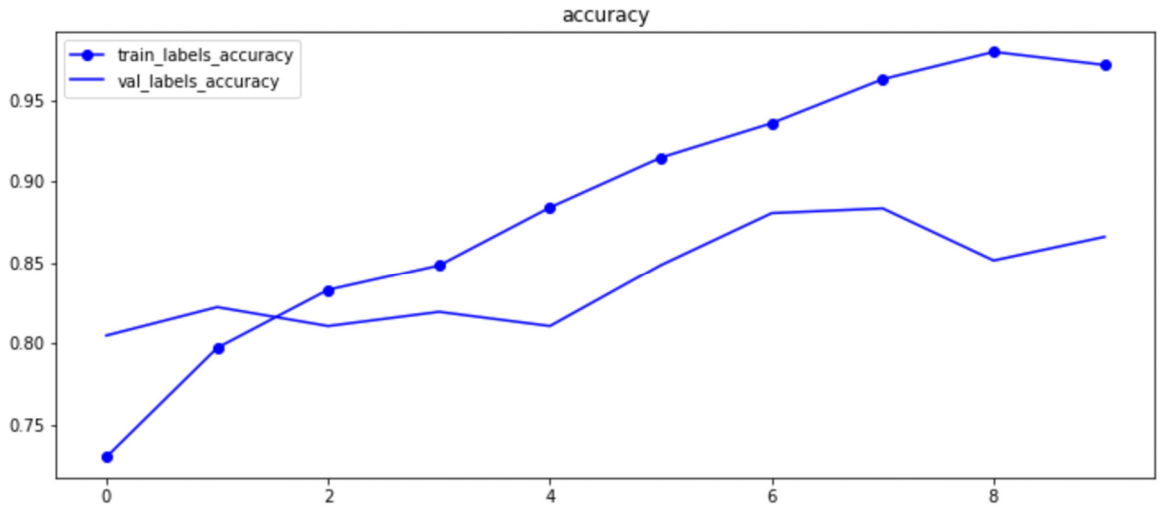


Figure 96: Experiment 4 – Xception - Accuracy

Experiment 4 model – Test set performance	
Accuracy	85,28%
Intersection over Union	15,60%

Table 9: Experiment 4 model test set performance

The increment in the complexity of the architecture didn't increase the performance of the system and increased the overfitting but improved the results in the confusion matrix balancing the performance on both classes as we can see below in the figure 97.

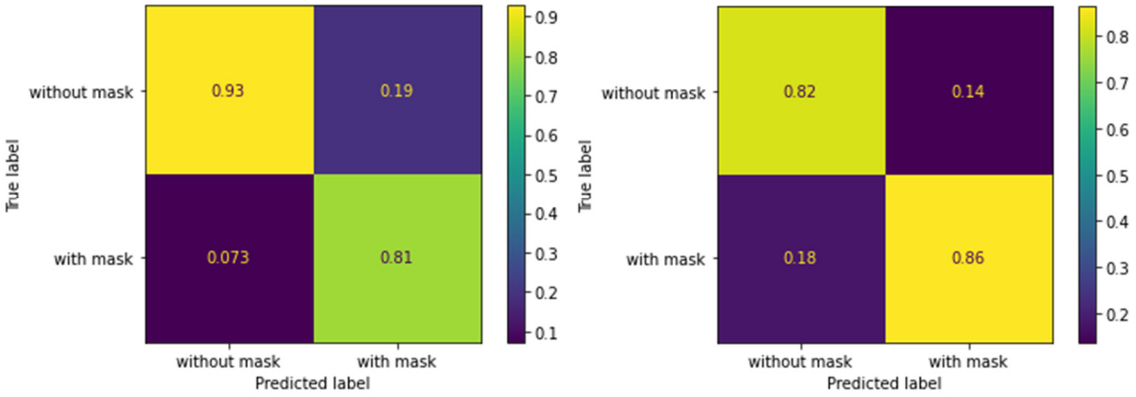


Figure 97: Experiment 2 and 4 – Xception - Confusion matrix comparison

5.2.5 Experiment 5 – Hyperparameters tuning

In the fifth and last experiment we tried to perform hyperparameters tuning using the keras tuner library and in particular the Hyperband tuner. We tried to optimize the number of units in the dense layers, the learning rate and the dropout parameter used in the dropout layers. Using the hyperband tuner we obtained a value of 1536 units for the dense layers, 0.8 for the dropout parameter and 0.0001 for the learning rate obtaining the model described in the figure 98. In figure 99, 100, 101 and in the table 10 are shown the results obtained with this model.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 128, 128, 3  0           []
                                )]

 tf.math.truediv (TFOpLambda)   (None, 128, 128, 3)  0           ['input[0][0]']

 tf.math.subtract (TFOpLambda)  (None, 128, 128, 3)  0           ['tf.math.truediv[0][0]']

 xception (Functional)          (None, 4, 4, 2048)   20861480    ['tf.math.subtract[0][0]']

 global_average_pooling2d (Glob  (None, 2048)        0           ['xception[1][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 1536)         3147264     ['global_average_pooling2d[0][0]'
                                                                 ]

 dense_1 (Dense)                (None, 1536)         2360832     ['dense[0][0]']

 dense_2 (Dense)                (None, 1536)         2360832     ['dense_1[0][0]']

 dropout (Dropout)              (None, 1536)         0           ['dense_2[0][0]']

 dropout_1 (Dropout)            (None, 1536)         0           ['dense_2[0][0]']

 labels (Dense)                 (None, 1)            1537        ['dropout[0][0]']

 bounding_box (Dense)           (None, 4)            6148        ['dropout_1[0][0]']

==================================================================================================
Total params: 28,738,093
Trainable params: 7,876,613
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
```

Figure 98: Experiment 5 - Xception - Summary

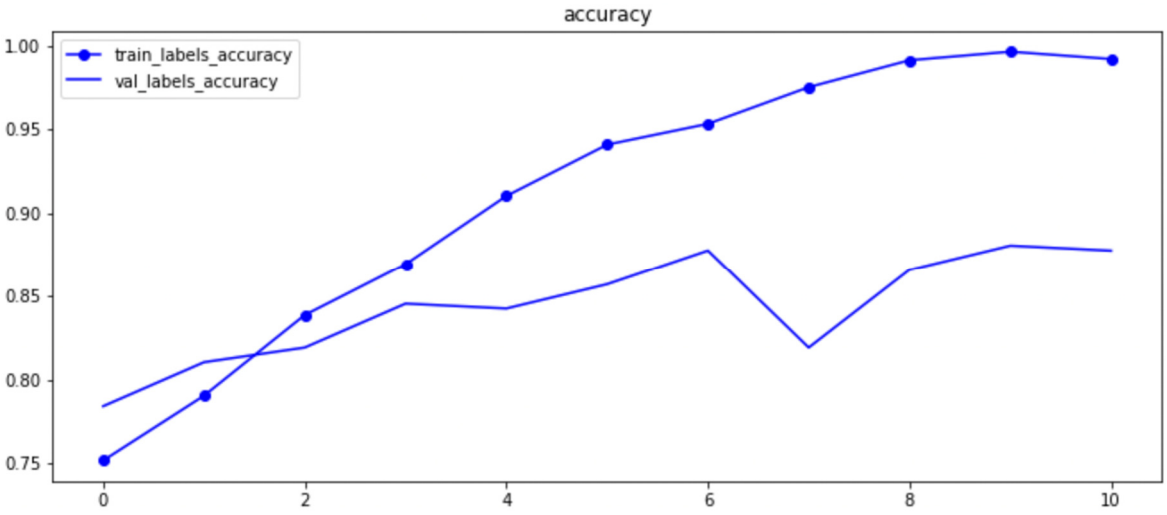


Figure 99: Experiment 5 - Xception - Accuracy

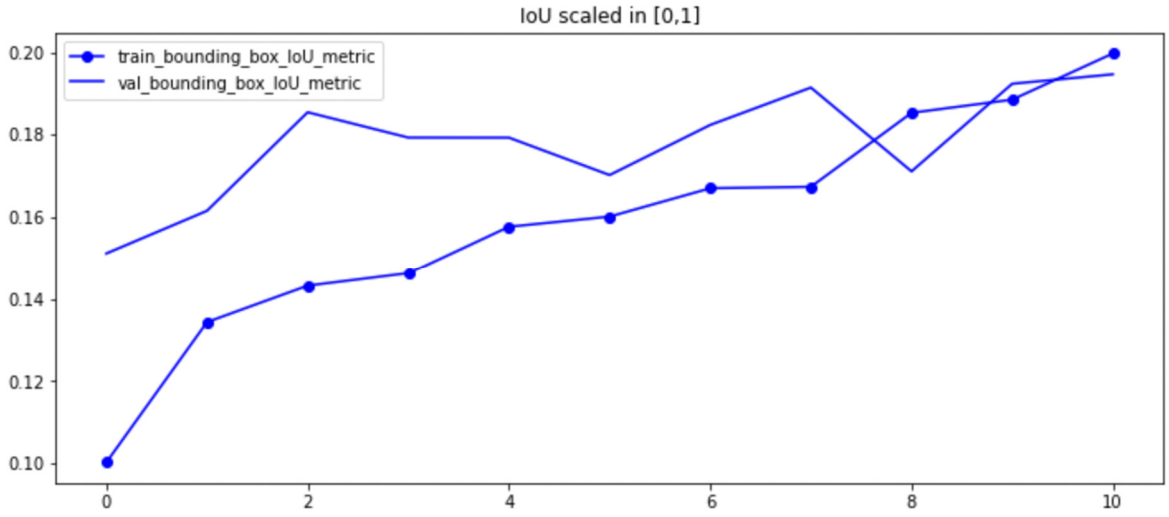


Figure 100: Experiment 5 - Xception - IoU metric

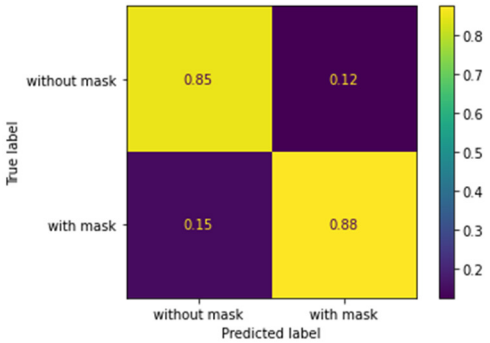


Figure 101: Experiment 5 - Xception - Confusion matrix

Experiment 5 model – Test set performance	
Accuracy	87,14%
Intersection over Union	19,05%

Table 10: Experiment 5 model test set performance

In this last experiment using hypertuning we obtain the best results for this model both in terms of accuracy and IoU metric underling the importance of hyperparameters tuning.

6. CONCLUSIONS

The documents contain the different experiment that we tried to approach the face mask detection and classification problem.
We started analyzing a from scratch implementation, trying different architectures and techniques and later we tried to perform the same task using two different pretrained models.
As expected, we obtained better results using the pretrained network with respect to the scratch network. This is because the pretrained networks use more parameters that has been pretrained on large datasets instead of the scratch network that has been trained on a smaller dataset.
The results that we obtained can be seen in the table below.

Network	Accuracy	IoU metric
Scratch network	79,44%	20,05%
Pretrained using Xception	87,14%	19,05%
Pretrained using MobileNetV2	92,75%	21,36%

As we can see, we obtained better results with the MobileNetV2 network (experiment 6).

Some further improvements that can be done to the networks are the following:

- Adjusting the class balancement of the dataset performing more sophisticated data augmentation in order to reduce overfitting and to open the possibility to perform fine tuning.
- An ensemble architecture could be used to improve the model performance
- Implement a loss function for the regression head that takes into account the correlation between the coordinates values and not the single values.
- Extend this model to perform multi-face mask detection and classification.

7. REFERENCES

[1]. Srihari Humbarwadi, *Object detection with RetinaNet* https://keras.io/examples/vision/retinanet/

[2]. Penh Zheng et al., *Object Detection with Deep Learning: A Review* *https://arxiv.org/pdf/1807.05511.pdf*

[3]. Shang Jiang et al., *Optimized Loss Functions for Object detection: A Case Study on Nighttime Vehicle Detection* https://arxiv.org/ftp/arxiv/papers/2011/2011.05523.pdf

[4]. Hanyang Peng et al., *A Systematic IoU-Related Method: Beyond Simplified Regression for Better Localization* https://arxiv.org/pdf/2112.01793.pdf

[5]. Ping Wang et al., *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression* https://arxiv.org/pdf/1911.08287.pdf

[6]. Aditya Sharma, *Mean Average Precision (mAP) Using the COCO Evaluator* https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/

[7]. Wadii Boulila et al., *A Deep Learning-based Approach for Real-time Facemask Detection* https://arxiv.org/ftp/arxiv/papers/2110/2110.08732.pdf

[8]. Li, Lisha, and Kevin Jamieson. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." Journal of Machine Learning Research 18 (2018): 1-52. https://keras.io/api/keras_tuner/tuners/hyperband/

[9]. Transfer learning and fine-tuning https://www.tensorflow.org/tutorials/images/transfer_learning

[10]. Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." https://keras.io/api/applications/mobilenet/