HW2 – Mask Detector

Submitted by:

Ron Avizemer 312146897

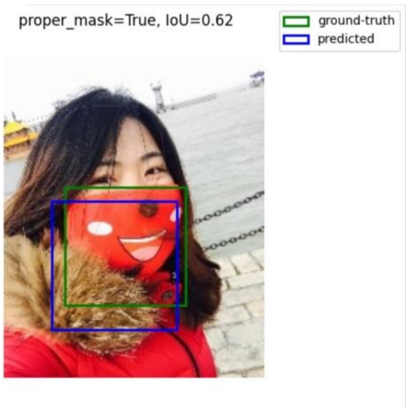
Eden Shaim 206182321

1. <u>**Exploratory Data Analysis:**</u>

   a. **Visualization:**

True Positive

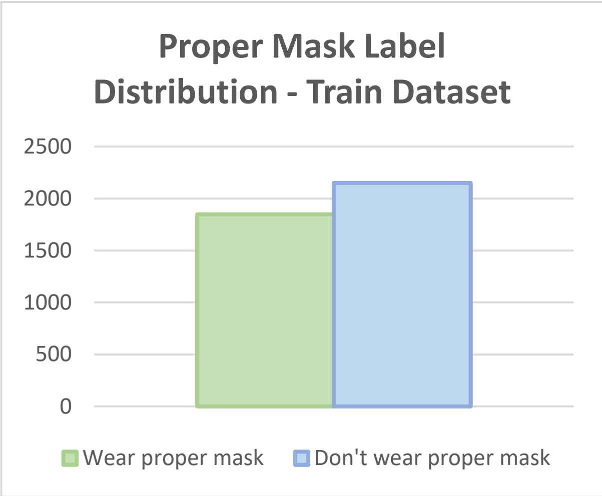
True Negative

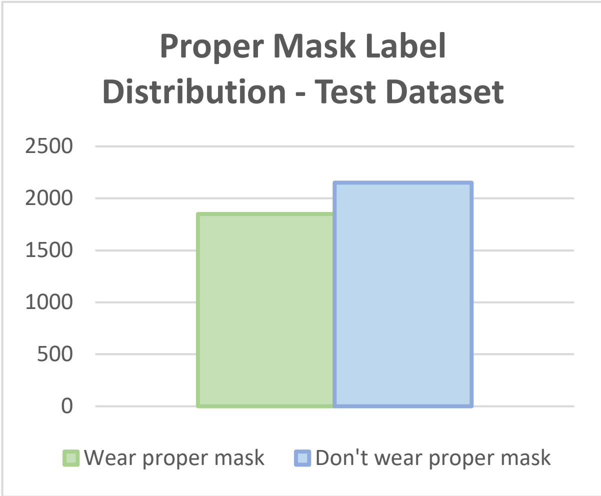


False Positive

False Negative

   b. From an initial look at the data we noticed a number of things:
      - There are incorrect labels of both types (as we showed above). That is, both images in which the person wears the mask properly but classified as False and also images in which the person does not wear the mask properly but classified as True. From that we conclude that this is not a simple task even for those who tagged the data.
      - There are images with a negative value in the 'bbox' array and images for which the height/length were equal to zero. Such images indicate an error in the data so we will delete them from our training set.
      - It can be seen that most of the people in the pictures of our dataset are Asian.



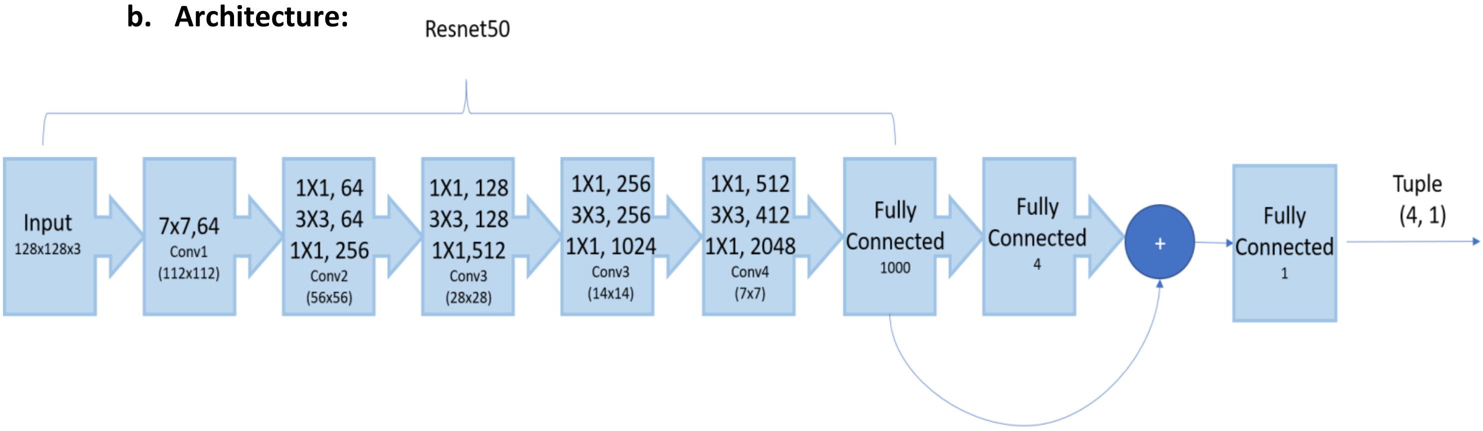


2. **<u>Experiments:</u>**

   First one:

   a. **<u>Cleaning</u>** – as we mention before we deleted all the images with the mistakes in the 'bbox' array.
   For bbox out of boundaries (x + w > img_size or y + h > img_size) we act differently with respect to the mode:
   For train mode – we skip the image
   For test mode – we fix the boundaries to be inside the image
   For bbox with x = w or y = h we act differently with respect to the mode:
   For train mode – we skip the image
   For test mode – we fix the x or y to be bigger by 1 from w, h respectively if it was possible. Else, we fix the w or h to be bigger by 1 from x, y respectively.
   Data loading - Creating a dataset of 'Pytorch' package. For each image we saved a tuple of three: the image itself, the bbox borders, label (True or False) whether the mask is properly worn.
   <u>Pre-processing</u> – We resized the images to size (256, 256), then we center crop it to (224, 224). We the images normalized according to the mean and standard deviation of them (in all three dimensions) in the dataset.
   Train mean & std:
   Mean – [0.5244, 0.4903, 4781]
   STD – [0.2614, 0.2580, 0.2534]
   Test mean & std:
   Mean - [0.485, 0.456, 0.406]
   STD - [0.229, 0.224, 0.225]
   <u>Data loading</u> – for dataloader creation we used batch_size = 32.

   b. **Architecture:**



c. **Loss functions –** We used two loss functions:
For predict the mask we used `BinaryCrossEntropyLoss`. We decided to classify to 'proper musk' (class 1) if the probability we got was higher than 0.5.
For predict the bbox we used L1Loss.
In order to perform the backward step we sum the two functions values.

d. **Optimizers –** we used Adam with learning rate of 0.02.

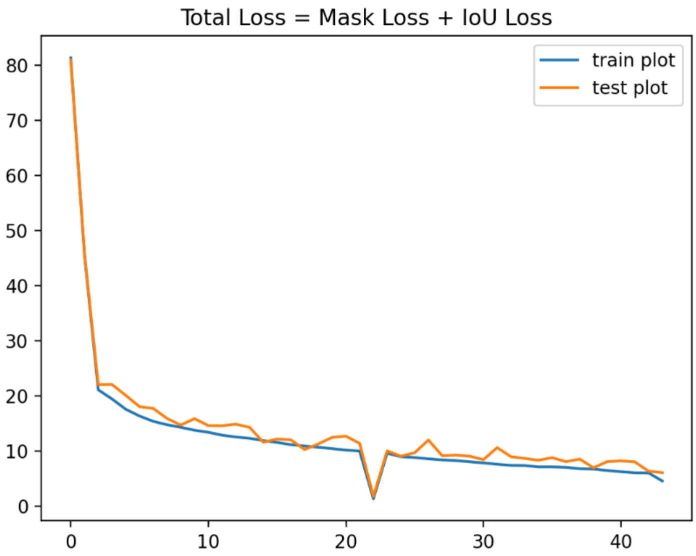
e. **Regularization –** As part of the network's architecture there are mechanisms designed to deal with over-fitting: residual connections, batch norm.

f. **Hyper parameters tuning –**
We mostly played with the learning rate where the defaulted one (0.001) was too small and after many tried, we decided to use lr= 0.02.
We tried to use both `BinaryCrossEntropyLoss` & `CrossEntropyLoss` and it didn't affect much on the results. For the `CrossEntropyLoss` we change the network to return 2 neurons where each one is the probability for each class.

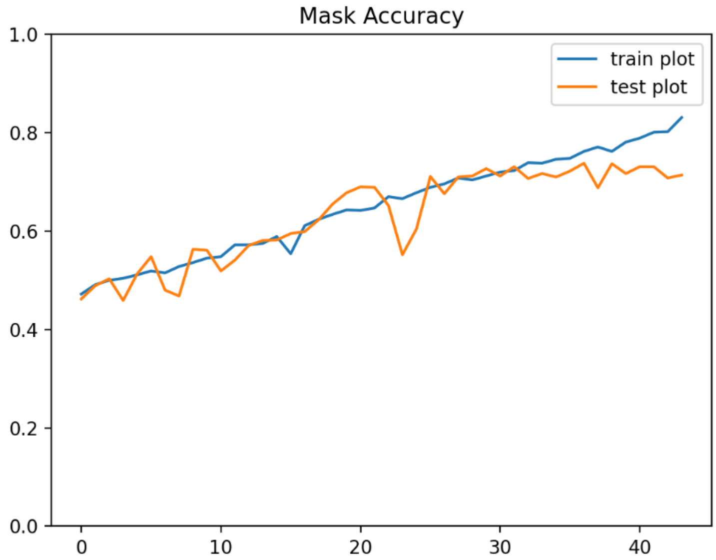
<u>Finely we used the following Hyper parameters which achieved these results:</u>

Batch size	Learning rate	optimizer	Train IOU	Train Accuracy	Test IOU	Test Accuracy
32	0.02	Adam	0.697	0.831	0.517	0.738

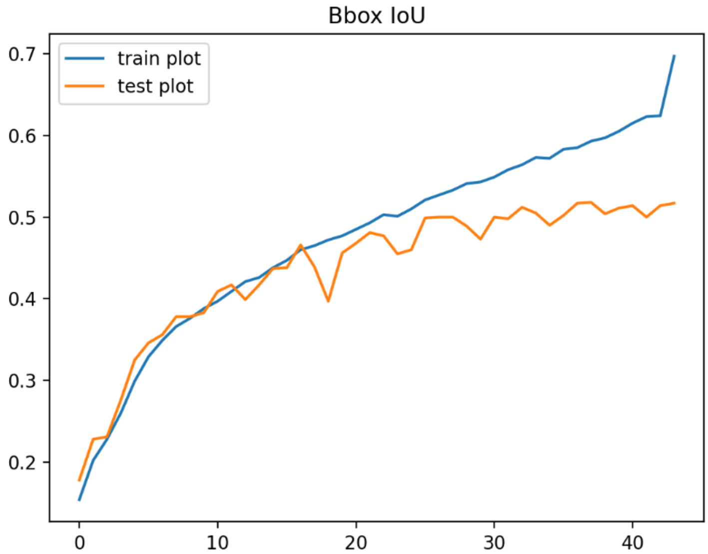
g. **Train & test Loss –** we plot the total loss



h. Mask accuracy –



i. Bbox IoU -



Second one:

a. **<u>Cleaning</u> –** as we mention before we deleted all the images with the mistakes in the 'bbox' array.

<u>Pre-processing</u> – we tried to use 'RandomHorizontalFlip' augmentation which simply randomly flip the image on the training dataset. Finally we didn't use it because it didn't help much.

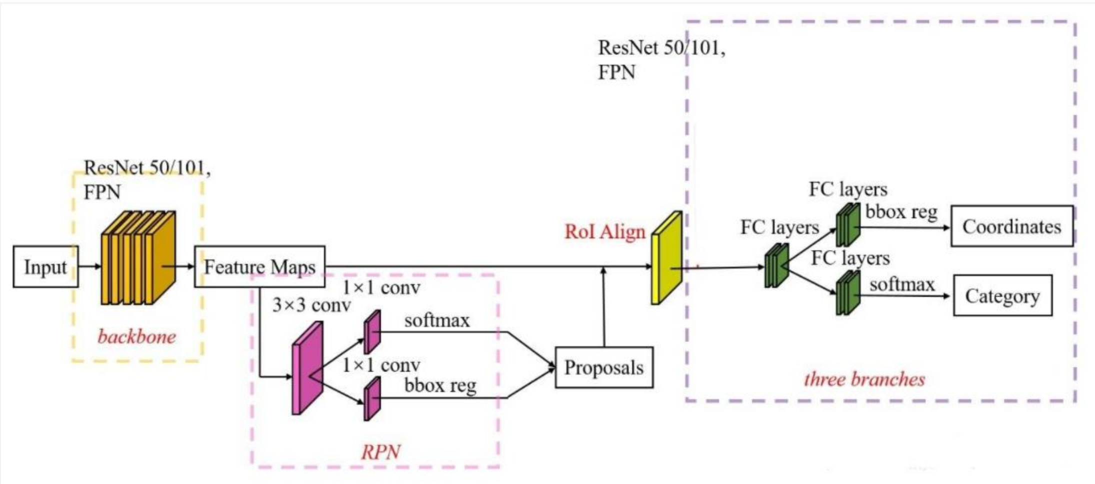
For our second experiment we used an implemented torchvision model called 'fasterrcnn_resnet50_fpn'.

This model requires a specific dataset structure:

The __getitem__ function need to return an (image, target) tuple. the resize and the normalization are part of the model itself – therefore we were only required to send min_size, max_size values to the model. We used min_size=224, max_size=224 .

<u>Data loading</u> – for dataloader creation we used batch_size = 32.

b. **Architecture:**



- <u>Backbone –</u> we used the ResNet50 (the net we described above in the first experiment) whose role in this model is to extract the features from the image as well.
- <u>Region Proposal Network (RPN) –</u> " A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score." from section 3.1 in paper [1].
  This part recommends to the Fast RCNN on areas in the image that have a high probability that the object will appear in them. It functions as attention mechanism. The RPN can predict multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. We used k = 1 because we only have one bbox.
- <u>Fast RCNN –</u> This is the object detector. The special property of our network is the shared convolutional layers between RPN and Fast RCNN. This property gives our network its name (Faster RCNN)!
  The Fast RCNN receives as input the convolution layers from the RPN and then it does not need to train from scratch. In addition, he gets the bbox.

c. **Loss function –** The loss function is part of the implemented model we used and its value is obtained as an output from the training process of the model. The loss function per image is defined as follows:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

We will define Anchors – the proposed boxes over the image.
i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes (object vs. not object). For the regression loss, we use $L_{reg}(\mathrm{t_i}, \mathrm{t_i^*}) = \mathrm{R}(\mathrm{t_i} - \mathrm{t_i^*})$ where R is the robust loss function (smooth L1). The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$).
In our case $i = 1$.

d. **Optimizer –** we used Adam optimizer with learning rate = 0.001

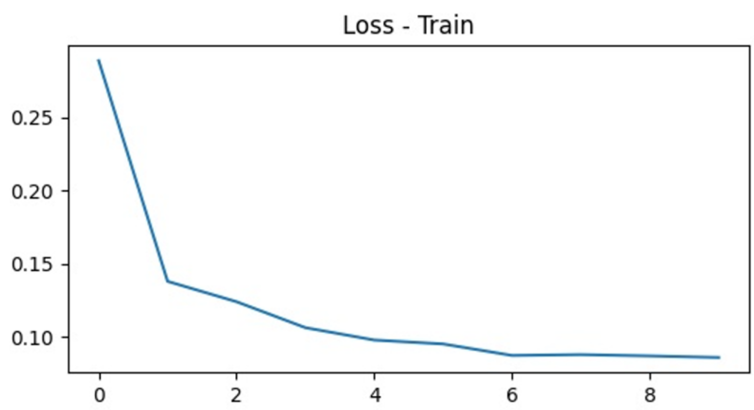
e. **Regularization –** As part of the network's architecture there are mechanisms designed to deal with over-fitting: residual connections, batch norm. We also tried additional regularization with SGD optimizer that did not improve with the performance on the test sample and therefore we did not use it in the final training. In addition, we applied gradient clipping by norm 2 for more stable gradients.

f. **Hyper parameter tuning –**
Learning rate – in the scale of [0.0001, 0.01]. all the learning higher than 0.001 resulted with worse performance (some of them even caused vanishing gradient).
Batch size – we tried batch size of 64 and resulted with cuda out of memory.
Optimizer – we tried SGD with momentum = 0.05 (regularization) but it got lower results than Adam with no regulation.

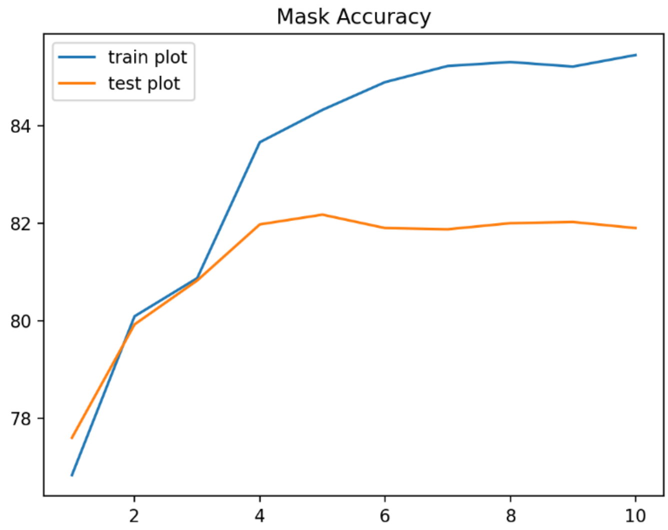
Finely we used the following Hyper parameters which achieved these results:

Batch size	Learning rate	optimizer	Train IOU	Train Accuracy	Test IOU	Test Accuracy
32	0.001	Adam	0.756	85.304	0.697	82.000

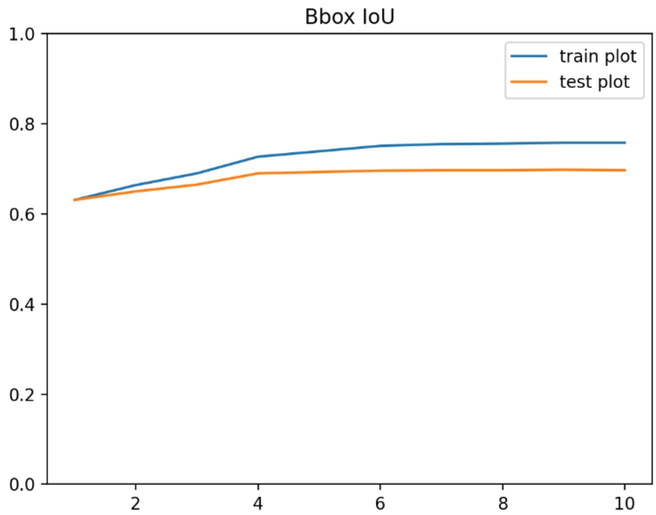
g. **Train Loss –** Note that we did not present the loss on the test set because we are using a built-in model fasterrcnn_resnet50_fpn. As mentioned earlier the model calculates the los values itself. These values are returned only when we are in model training mode. In evaluation mode the model returns only the predictions.



h. Mask Accuracy –



i. Bbox IoU –



3. **<u>Conclusions:</u>**

The use of a model that is designed to perform object detection yields more accurate results than the basic model we chose to build for this task.
In addition, it needed far fewer epochs to get his best results compared to the basic model that needed at least 15 epochs to basic results.
But even for the object detection model this was a difficult task from several reasons:

- These models specialize in recognizing several objects in an image and because the task was a bit degenerate in that we only had to predict one bbox (the left one) the difficulty was in predicting this specific object.
- The data we received was with quite a few errors in its labeling (whether it is on the label or whether it is within the limits of the bbox) which made it difficult for the model.

We want to show two images that we gave as an example for misclassification on the training set. These two images represent False Positive, False Negative. While our model classify them to their real-world label (by assignment definition).

Label - True

Our model - False

Label - False

Our model - True