Face mask detection using Convolutional neural networks

Tin Oroz
FER
Zagreb, Hrvatska
tin.oroz@fer.hr

Marin Mrčela
FER
Zagreb, Hrvatska
marin.mrcela@fer.hr

Martin Pavić
FER
Zagreb, Hrvatska
martin.pavic@fer.hr

Matija Roglić
FER
Zagreb, Hrvatska
matija.roglic@fer.hr

I. Introduction

This project will be focused on building a neural network using TensorFlow to detect if a person is wearing a face mask or not. In these COVID-19 times this kind of project sounds interesting and quite useful. Detecting if an image contains a person wearing a mask or not is a simple classification problem.

Convolutional neural network (CNN), a class of artificial neural networks that has become dominant in various computer vision tasks, is attracting interest across a variety of domains. CNN is designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers.

II. Working with data

A. Dataset

For our project, we used the dataset provided by Prajna Bhandary [3] on her GitHub site. The provided dataset consists of 1376 images, split into two discrete classes. 686 images represent the first discrete class of data, the faces of people without the face mask, while the remaining 690 images represent the second discrete class of data, which is the faces of people with the face mask. All the images contain just one face, typically shot as a portrait, with people directly facing the camera.

B. Data augmentation

The original dataset contained a few instances of data augmentation in form of blurring and adding noise to a small number of randomly picked images while keeping their originals. For the further augmentation, we created a Python script which flips and rotates each provided image and saves the result as a copy. Each image was flipped horizontally (left to right), subsequently, a random number between -45 and 45 was generated for each instance and applied as the angle of rotation, therefore, rotating each image by an angle up to 45 degrees, either clockwise or counterclockwise. This doubled the number of available images in the dataset to 2752, which is now comprised of 1372 images of faces without the face mask, and 1380 images of faces without the face mask.

C. Labeling the images for training

To prepare the images for training process, an XML file containing the information about the location of an object on a picture and a label for such object must be created. To create these XML files, a simple open source program LabelImg is used. LabelImg is a graphical image annotation tool written in Python and uses Qt for its graphical interface.[4] The usage of this tool is simple, as user can go through images in dataset in a slideshow and draw a bounding box for each image, put a label and save the data in XML file. The program uses PASCAL VOC format to save the annotations and an example of a saved XML file for a labeled image can be seen in figure 1.

```
<annotation verified="yes">
      <folder>train</folder>
      <filename>228.jpg</filename>
      <path>D:\tensorflow1\models\research\object_detection\images\train
28.jpg</path>
      <source>
            <database>Unknown</database>
      </source>
      <size>
            <width>503</width>
            <height>450</height>
            <depth>3</depth>
      </size>
      <segmented>0</segmented>
      <object>
            <name>without mask</name>
            <pose>Unspecified</pose>
            <truncated>0</truncated>
            <difficult>0</difficult>
            <bndbox>
                  <xmin>185</xmin>
                  <ymin>135</ymin>
                  <xmax>313</xmax>
                  <ymax>287</ymax>
            </bndbox>
      </object>
</annotation>
```

Figure 1 XML annotation for image 228.jpg

This example shows an image named 228.jpg that is located on a D: directory in a train folder. The size of image is 503x450 pixels. The label on the image marks a person without a mask. The bounding box is always at a right angle, so the locations of each side are represented by their x and y coordinates perspectively as we are talking about 1D lines. The bounding box that marks the face without a mask is stretched from 135th pixel to 287th pixel on a Y axis and 185th pixel to 313th pixel on X axis.

While marking the bounding boxes for each picture, it is not extremely important to pay extra attention to the borders of the bounding boxes as long as the main features of the desired detectable objects are within those bounding boxes. Since the idea is to detect a face and then classify if it has a mask or not, two labels are used: „with mask“ and „without mask“. Bounding boxes are therefore drawn around the faces as to include the eyes, nose and the mouth in case the subject in the picture is not wearing a mask. In case when the subject is wearing a mask, the bounding boxes were drawn to catch the eyes, nose and the mask of the subject. When drawing the bound boxes, an attention must be paid to ensure all possible detectable faces have bounding boxes drawn around then and correctly labeled since the dataset had several pictures with multiple subjects in the image which have been wearing or not have been wearing a mask. This process was repeated for all pictures in training and test folder.

Once an xml file has been created for every single picture in test and train folder, a simple Python script has been run that reads every single xml file and writes down a filename, width and height of a picture, label and locations of bounding boxes into a single csv file. For an already mentioned example, the summary of the xml file that was created by program LabelImg is seen in Figure 2.

228.jpg,503,450,without mask,185,135,313,287

Figure 2 Summary of XML file in CSV file

Lastly, we run another Python script to turn the csv files into TFRecord files. TFRecord file stores data as a sequence of binary strings. This format is specific for Tensorflow and it gives many advantages to use. The binary data takes up less space on a disk and can be read much efficiently from a disk. This is especially true when the data is stored on a HDD. It is also optimized for usage with Tensorflow and can therefore speed up the training time massively.[5] Once these record files are created, we can use them to train the model.

III. Neural network model

For detecting whether a person wears mask we are using object detection system called Faster Region-based Convolutional Neural Network (Faster R-CNN) **Pogreška! Izvor reference nije pronađen.**. It is consisted of two modules. The first module is CNN that proposes regions called Region Proposal Network (RPN), and the second one is the Fast R-CNN detector [1] that uses previously proposed regions. As shown on Fig. 1, the whole system works as a unified network where the RPN module tells the Fast R-CNN module where to look.

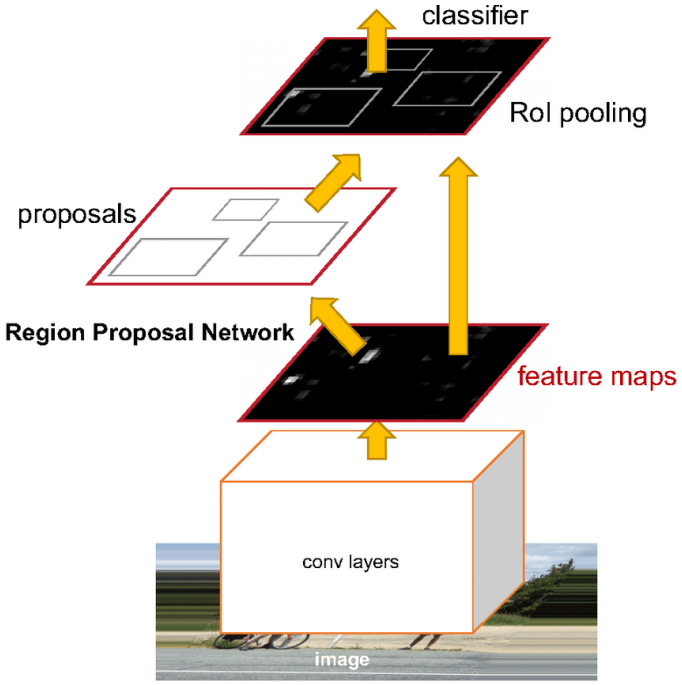

Figure 3: Faster R-CNN Architecture[1].

A. *Regional proposal networks*

Input for RPN is an image of any size and its output is a set of rectangular object proposals. Those proposals are valued with a measure for membership to a set of object classes. Region proposals are generated by sliding a small network over the convolutional feature map output by the last shared convolutional layer, shown on Fig 1. Input to this small network is an n × n spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully connected layers, a box-regression layer (reg) and a box-classification layer (cls).

The predictions of multiple region proposals are done simultaneously, at each sliding-window location, where the number of maximum possible proposals for each location is denoted as k. The reg layer has 4k outputs encoding the coordinates of k boxes, and the cls layer outputs 2k scores that estimate probability of object or not object for each proposal. The k proposals are parameterized relative to k reference boxes, which we call anchors. An anchor is centered at the current sliding window and is associated with a scale and aspect ratio. For a convolutional feature map of a size W × H (typically ∼2,400), there are WHk anchors in total.

Translation-Invariant anchors

They have an important property that is if one translates an object in an image, the proposal should translate and the same function should be able to predict the proposal in either location. It also reduces the model size.

Multi-Scale anchors as regression references

A novel scheme for addressing multiple scales and aspect ratios. Feature maps and deep convolutional features are computed for each scale of the image that is resized at multiple scales.

For training RPNs, we assign a binary class label (in our case wearing a mask or not wearing a mask) to each anchor. Positive label is assigned to two kinds of anchors: the anchor with the highest Intersection-over-Union (IoU) overlap with a ground-truth box, or an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. Note that a single ground-truth box may assign positive labels to multiple anchors. Usually the second condition is enough to determine the positive samples; but we still adopt the first condition for the reason that in some rare cases the second condition may find no positive sample. We assign a negative label to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the training objective.

We minimize an objective function following the multi-task loss in Fast R-CNN [2]. Loss function is defined as:

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

The outputs of box-classification layer (cls) and box-regression layer (reg) are denoted by $\{p_i\}$ and $\{t_i\}$ respectively. These two terms are normalized by N_{cls} and N_{reg} and weighted by a balancing parameter λ. p_i is the predicted probability of anchor *i* being an object. p_i^* is the ground-truth label and its value is 1 if the anchor is positive, therefore 0 if the anchor is negative. t_i represents a vector that represents the 4 coordinates of the predicted bounding box. t_i^* is the ground-truth box associated with positive anchor. L_{cls} is log loss over two classes (wearing mask *vs.* not wearing mask), and L_{reg} is the regression loss [1].

B. Fast R-CNN

Input to Fast R-CNN network is an entire image and a set of object proposals. At first, it processes the whole image with several convolutional and max pooling layers to produce a convolutional feature map. For each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. A sequence of fully connected layers is fed with all feature vectors, that sequence finally branch into two sibling output layers. One that produces softmax probability estimates over K object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes. The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of H × W, where H and W are layer hyper-parameters that are independent of any particular RoI. Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).

There are two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The network is trained end-to-end with a multi-task loss. RoI max pooling works by dividing the h × w RoI window into an H × W grid of sub-windows of approximate size h/H × w/W and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer.

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations. First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer. Second, the network's last fully connected layer and softmax are replaced with the two sibling layers described earlier (a fully connected layer and softmax over K + 1 categories and category-specific bounding-box regressors). Third, the network is modified to take two data inputs: a list of images and a list of RoIs in those images.

Each RoI may have a very large receptive field, often spanning the entire input image which may cause an unwanted inefficiency. Since the forward pass must process the entire receptive field, the training inputs are large, often the entire image. There is a method that takes advantage of feature sharing during training. In Fast RCNN training, stochastic gradient descent (SGD) minibatches are sampled hierarchically, first by sampling N images and then by sampling R/N RoIs from each image. Critically, RoIs from the same image share computation and memory in the forward and backward passes. Making N small decreases mini-batch computation. For example, when using N = 2 and R = 128, the proposed training scheme is roughly 64× faster than sampling one RoI from 128 different images. One concern over this strategy is it may cause slow training convergence because RoIs from the same image are correlated. This concern does not appear to be a practical issue.

Mini-batch sampling. During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random (as is common practice, we actually iterate over permutations of the dataset). These RoIs comprise the examples labeled with a foreground object class, i.e. $u \geq 1$. The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5). These are the background examples and are labeled with u = 0. During training, images are horizontally flipped with probability 0.5. No other data augmentation is used.

Back-propagation through RoI pooling layers. Backpropagation routes derivatives through the RoI pooling layer. For clarity, we assume only one image per mini-batch (N = 1), though the extension to N > 1 is straightforward because the forward pass treats all images independently. Training all network weights with back-propagation is an important capability of Fast R-CNN.

Fast R-CNN detection. For each test RoI r, the forward pass outputs a class posterior probability distribution p and a set of predicted bounding-box offsets relative to r (each of the K classes gets its own refined bounding-box prediction). We assign a detection confidence to r for each object class k using the estimated probability $P_r(class = k \mid r) \triangleq p_k$. We then perform non-maximum suppression independently for each class using the algorithm and settings from R-CNN [2].

C. Sharing Features between RPN and Fast R-CNN

RPN and Fast R-CNN that are trained independently and will modify their convolutional layer differently. Therefore, we need a technique that allows for sharing convolutional layers between two networks, rather than learning two separate networks. There are three ways for training networks with features shared.

Alternating training. In this solution, we first train RPN, and use the proposals to train Fast R-CNN. The network tuned by Fast R-CNN is then used to initialize RPN, and this process is iterated.

Approximate joint training. In this solution, the RPN and Fast R-CNN networks are merged into one network during training. In each SGD iteration, the forward pass generates region proposals which are treated just like fixed, pre-computed proposals when training a Fast R-CNN detector. The backward propagation takes place as usual, where for the shared layers the backward propagated signals from both the RPN loss and the Fast R-CNN loss are combined.

Non-approximate joint training. The bounding boxes predicted by RPN are also functions of the input. The RoI pooling layer in Fast R-CNN accepts the convolutional features and also the predicted bounding boxes as input, so a theoretically valid backpropagation solver should also involve gradients. These gradients are ignored in the above approximate joint training. In a non-approximate joint training solution, we need an RoI pooling layer that is differentiable. This is a nontrivial problem and a solution can be given by an "RoI warping" layer, which is beyond the scope of this paper [1].

D. Preparing the model for training

To prepare the model for training, several files need to be configured before the training can start. First a label map must be created. The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers as seen in figure 3. In this case, the label map only consists of two entries as there are only 2 different objects that need classification, the "without mask" face and "with mask" face.

```
item {
  id: 1
  name: 'with mask'
}

item {
  id: 2
  name: 'without mask'
}
```

Figure 4 Label map

Lastly the config file for the model needs to be configured before running the training. Inside of the config file, the number of classes which we want the classifier to detect must be set, in this case 2. Furthermore, the directories of TFRecord files for both training data and testing data need to be pointed to in specific lines in the config file as well as the directory of label map file. The config file needs to also know the correct number of images in test directory before the training can be started. In the config file a number of steps can be set, and it has been set for 200,000 times. This number of steps does not need to be reached as the model during the training creates various checkpoints which can be used to either create a frozen inference graph once the loss has become small enough, or to pause the training and continue at later time. Once this config file has been edited and saved, the training of the model can be started by running a Python script train.py which is provided by Tensorflow developers and can be found on their GitHub and therefore will not be explained in further details.

E. Training the model and creating inference graph

In the end, the training has been running for 143,781 steps before manually ending the training. The average time per step was around 0,4 seconds so the total training time was around 16 hours. The training has been stopped manually because the loss has reached below 0,05 consistently and therefore it was believed that the model has been sufficiently trained and longer time training would not achieve much of improvement. After ending the training, the last created checkpoint which was automatically created after 143,781st step has been used to create a frozen inference graph by using another Python script named export_inference_graph.py provided by Tensorflow developers. Frozen inference graph is a frozen graph that cannot be trained anymore and therefore can be used for inference purposes. This keeps all the weights same and constant for later usage in any program it needs to be used in. One of the examples where the frozen inference graph can be used is for inferencing the video feedback from a web camera to see if the person facing the web camera is wearing a mask or not.

F. Mask detection using a web camera

For the usage of the model we have made a Python script that uses computers webcam. The model will predict the possibility of each of the two classes ([without_mask, with_mask]). Based on which probability is higher, the label will be chosen and displayed around our faces. Examples of two classes recognition are presented in Figure 5 and Figure 6.

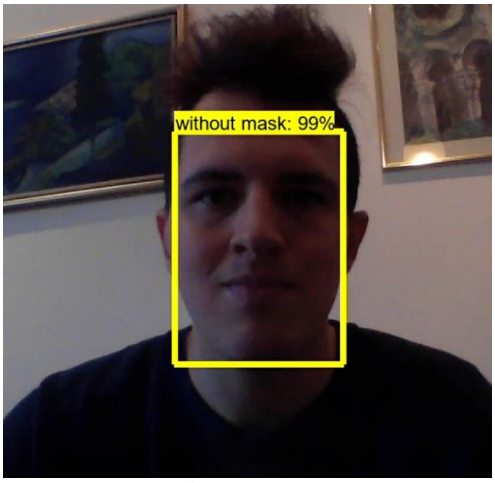

Figure 5: Example of model recognition without_mask

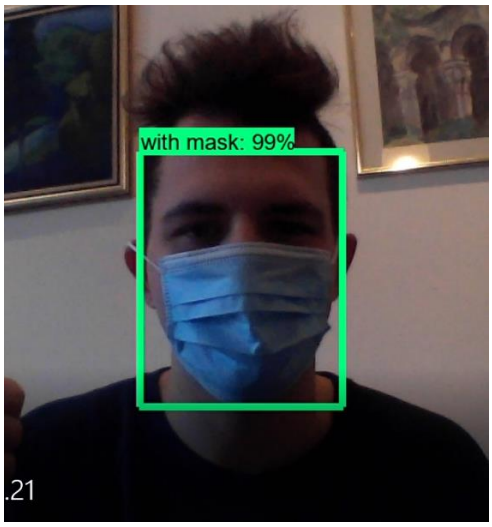

Figure 6: Example of model recognition with_mask

IV. Conclusion

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Convolutional neural networks are mostly used for tasks regarding image and video recognition. Therefore, they make a great pick for our chosen task of detecting weather a person is wearing a face mask or not. Faster R-CNN proved to be a good solution for our problem with its combination of accuracy and time efficiency. This kind of project seems quite useful in these COVID-19 times. It can be used in many different situations i.e. as a safety measure while entering certain objects where it is necessary to wear a mask.

References

[1] S. Ren, K. He, R. Girshick and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2016.

[2] R. Girshick, “Fast R-CNN”, 2015, Microsoft.

[3] https://github.com/prajnasb/observations/tree/master/experiements/data Pictures used as data

[4] tzutalin, "labelimg," 9 1 2021. [Online]. Available: https://github.com/tzutalin/labelImg.

[5] T. Gamauf, "Mostly AI," Medium, 20 March 2018. [Online]. Available: https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564. [Accessed 10 January 2021].