HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

*

Computer Vision

Capstone Project Report

Project's topic: Face detection

Team members:

Full name	Student ID	School Email
Phạm Văn Cường	20194421	cuong.pv194421@sis.hust.edu.vn
Nguyễn Tuấn Dũng	20194427	dung.nt194427@sis.hust.edu.vn
Phạm Cát Vũ	20194465	vu.pc194465@sis.hust.edu.vn

I. Introduction

Face detection is an important task in Computer Vision which involves finding faces in photos. It can be applied in various fields -- including security, biometrics, law enforcement, entertainment and personal safety -- to provide surveillance and tracking of people in real time.

Face detection has progressed from rudimentary computer vision techniques to advances in machine learning (ML) to increasingly sophisticated artificial neural networks (ANN) and related technologies; the performance of face detectors has been improving continuously. It now plays an important role as the first step in many key applications -- including face tracking, face analysis and facial recognition. Face detection has a significant effect on how sequential operations will perform in the application.

In this project, we apply three face detection methods, from classical to more modern ones: HOG + SVM, Single Shot Detector (SSD), Faster RCNN, and examine their performance.

II. Dataset

Our chosen dataset is the WIDER FACE dataset, a popular benchmark for the task of face detection. The full dataset consists of over 32,203 images with 393,703 labeled faces, with a variety of poses, scales and occlusions. This diversity makes the dataset a good benchmark to evaluate the performance of our face detectors.

The dataset is officially split into 3 smaller subsets: training, validation and test sets, making up 40%, 10%, 50% of the entire training data, respectively. However, the official test set's bounding box ground truth is not released by the authors. Since we were not able to submit our results, we will choose the official validation set with over 3000 images as the test set to evaluate our detection models. We further split the original training set into our own training set and a validation set with a ratio of 80%-20%. The final proportion of the data we used is as follows:

Subset	No. Images
Train	10304
Validation	2576
Test	3226

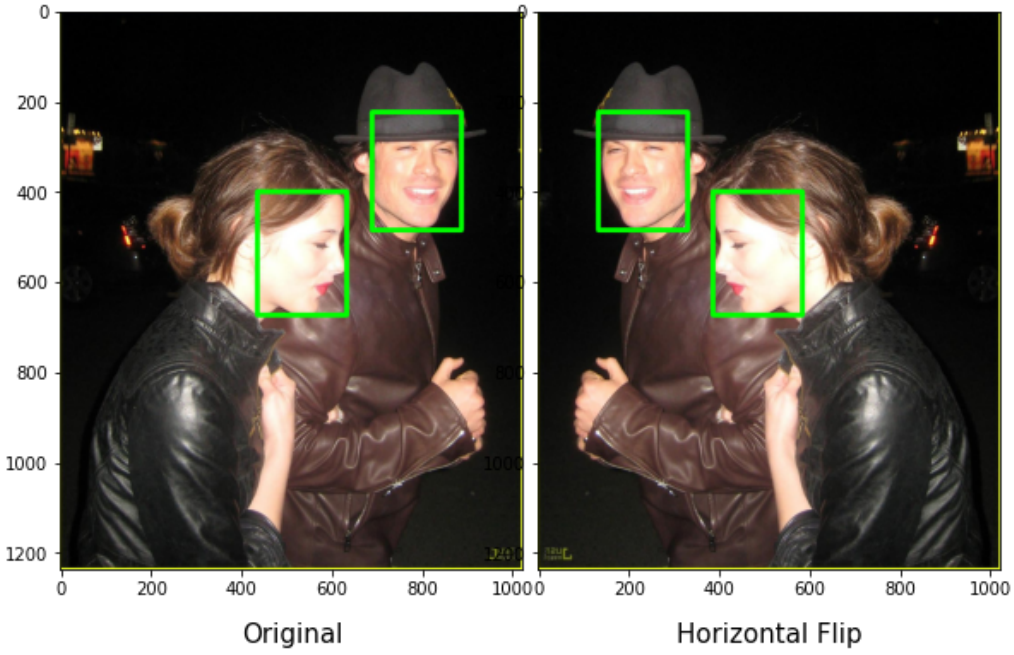
The authors of the dataset group the official test set and validation set into 3 subsets based on the detection rate of EdgeBox: Easy, Medium and Hard. In our case the official validation set is our test set.

III. Data Augmentation

A. Methods which affect bounding boxes

These methods, when applied, will make the augmented images' bounding boxes different from the originals

1. Horizontal Flip



2. Scaling

The image is scaled up/down and then cut off/filled with black to preserve the size. If 25% of a bounding box is left out of the new image, then that bounding box is removed.

3. Translation

If 25% of a bounding box is left out of the new image, then that bounding box is removed.

4. Shearing

Shear horizontally the image, expand the bounding boxes to the shearing direction. The image is resized afterward.

B. Methods which do not affect bounding boxes

1. Blurring

We use Gaussian blur and Motion blur to augment the data, these transformations do not affect the bounding boxes.

2. Distorting

We use Contrast distortion, Brightness distortion, HSV distortion to augment the data, these transformations do not affect the bounding boxes.

C. Apply on Dataset

Each image in the dataset will be passed through a pipeline of augmentation techniques which will be applied:

- 50% chance at each method of Methods which affect bounding boxes
- 50% chance of Gaussian Blur, 50% chance of Motion Blur
- 50% chance at each method of Distortion (Contrast distortion, Brightness distortion, HSV distortion)

IV. Methods

1. HOG + SVM

1.1. Theoretical Backgrounds

1.1.1. Histogram of Oriented Gradients (HOG)

Given an input image, this method aims to classify every small patch of that input image as either the positive class, which is defined as the object of interest that we want to detect (the human faces in our particular case), or the negative class, which includes everything that is not a face. In order to perform this classification, our system needs a good feature descriptor to extract the most relevant information from the image patches, which helps increase the performance of our classifier. The feature descriptor chosen is Histogram of Oriented Gradients (HOG), a popular feature extraction algorithm for classification/detection tasks.

The HOG descriptor is a descriptor that encodes information based on the distribution of the local intensity and the orientation of the edges of the image patches. First, the algorithm divides our image patches into smaller regions (or “cells”). For each of these cells, we calculate a histogram of the gradient directions over the pixels in the current cell. We then concatenate these histogram vectors to construct the feature representation for the patches. Finally, to make our representation less susceptible to varying lighting, shadowing…, contrast normalization can be done by accumulating a measure of local histogram “energy” over somewhat larger spatial regions (“blocks”) and using the results to normalize all of the cells in the block.

Gradient calculation

Suppose we have a grayscale image patch as the input. Call the gradient of the image patch I, the two components of the gradient is computed as:

$I_x(r,c) = I(r,c+1) - I(r,c-1)$ and $I_y(r,c) = I(r-1,c) - I(r+1,c)$

This is equivalent to applying the following filters to our image:

-1	0	1

-1
0
1

Next, the gradient is transformed into polar coordinates ranging from 0 to π.

$$\mu = \sqrt{I_x^{\,2} + I_y^{\,2}} \quad \text{and} \quad \theta = \frac{180}{\pi}\left(tan_2^{\,-1}\left(I_y + I_x\right) mod\ \pi\right)$$

Essentially, the gradients help us remove the less important information in the image and highlight the more noticeable ones.

Calculate Histogram of Gradients in cells

Like we have stated above, this requires us to divide our image into separate cells of size $C \times C$. We calculate the gradient magnitude and the gradient direction for every one of these cells.

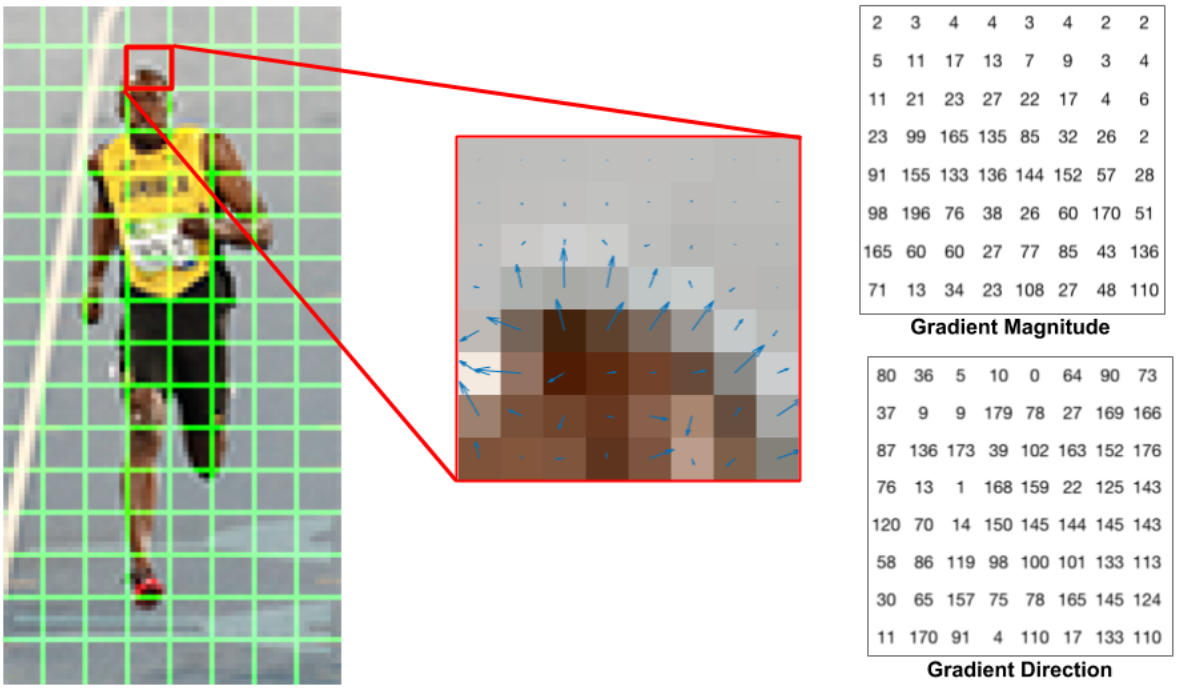


Figure: An example of Histogram of Gradients

Above is an example of this operation, with the cells' size being 8 x 8. Observing the image in the center and the gradient magnitude and direction on the right, we can see that the patches of the image overlaid with arrows denoting the gradients, with the arrow direction denoting the gradient direction showing the direction of the change in intensity in our image, and the arrow's length denoting the gradient magnitude to show how big the difference is.

Afterwards, the descriptor creates a histogram of gradients calculated for these cells binned into B bins. The bins are numbered from 0 to $B - 1$ and have a width of $w = \frac{180}{B}$. Bin i has boundaries $[wi, w(i + 1)]$. A bin is selected based on the

direction, and the vote (the value that goes into the bin) is selected based on the magnitude. A pixel with magnitude μ and orientation θ contributes a vote

$$v_j = \mu \frac{c_{j+1} - \theta}{w} \quad \text{to bin number } j = [\frac{\theta}{w} - \frac{1}{2}] \, mod \, B$$

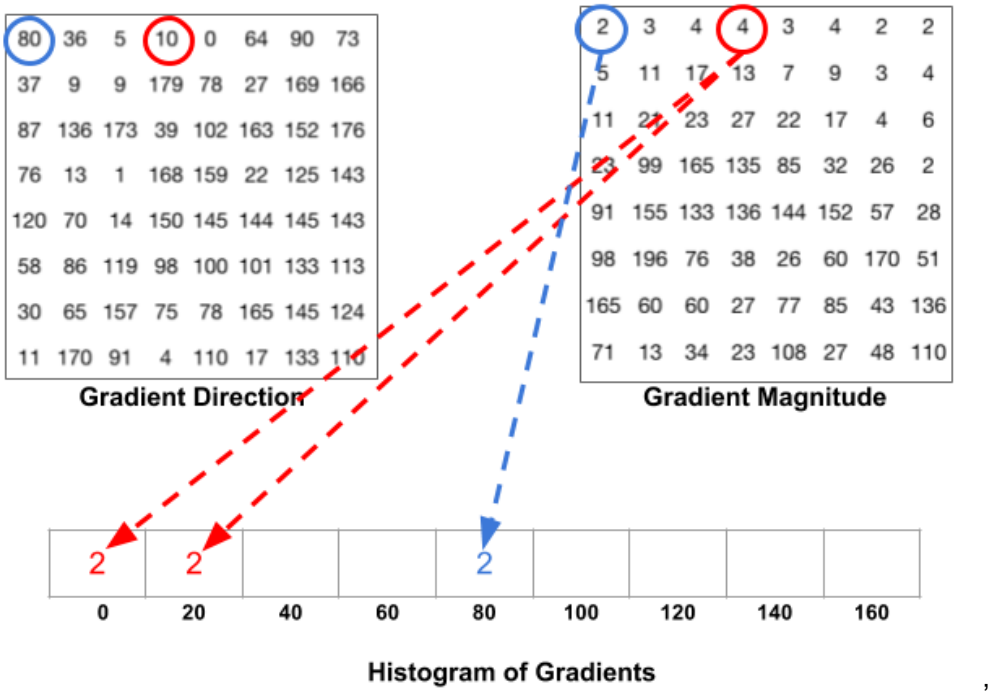
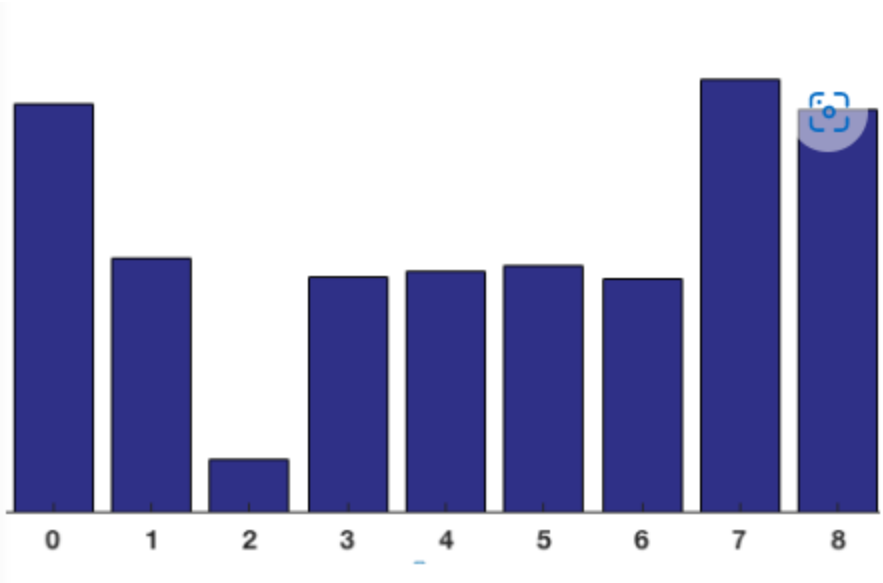


Figure: An example of Histogram of Gradients calculation

For example, for the pixel highlighted in blue, it has an angle of 80 degrees and a magnitude of 2. So it adds 2 to the 5th bin corresponding to the gradient direction of 80. The gradient at the pixel encircled using red has an angle of 10 degrees and magnitude of 4. Since 10 degrees is halfway between 0 and 20, the vote by the pixel splits evenly into the two bins.

The contributions of all the pixels in the $C \times C$ cells are added up to create the B bins histogram. For the patch above, it looks like this



Block Normalization

After calculating the histogram, we group the cells into overlapping blocks of 2 × 2 cells each, so that each block has size 2C×2C pixels. Two horizontally or vertically consecutive blocks overlap by two cells, that is, the block stride is C pixels. As a consequence, each internal cell is covered by four blocks. Concatenate the four cell histograms in each block into a single block feature b and normalize the block feature by

its Euclidean norm:

$$b \leftarrow \frac{b}{\sqrt{||b||^2 + \epsilon}}$$

The Block Normalization operation ensures the cells are less affected by changes in contrasts.

HOG Feature Vector

Finally, we concatenate all normalized block features into a single HOG feature vector h for the entire patch.

1.1.2. Classification with Support Vector Machine (SVM)

For an input image for detection, we first apply a sliding window operation from left to right on the image. Every time we iterate through a window in the image, we extract that window's HOG features, and use a linear SVM classifier to classify whether that patch is a face or not. In our particular case, the size of each window patch will be a 38 x 46 gray scale image patch.

To train such a SVM classifier, we need to prepare our data with image patches of 2 classes: A positive class with instances being human faces' image patches, and a negative class being image patches of non-human face objects or backgrounds.

Since our sliding window used for classification has a fixed size, it will not be able to detect faces at multiple scales. Thus, we need to perform the sliding window and classification procedure on multiple scales of the original image.

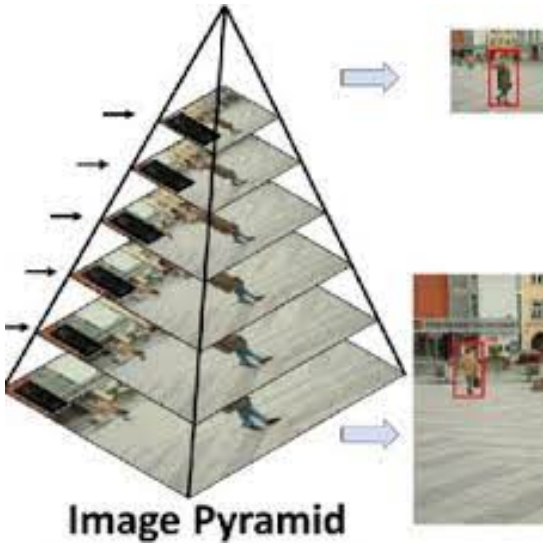


Figure: An Image Pyramid

From the original image, we incrementally reduce the image size until the image size is barely larger than the size of a sliding window patch. We ran a sliding window and classified images of all sizes in the image pyramids, and then recorded all positive predictions.

1.2. Implementation

1.2.1. Training

Training data

First, we have to prepare the data for our SVM classifier to learn.

Firstly, for positive training data, we extracted 20000 cropped faces from the training set of our WIDER FACE dataset. However, after briefly training a baseline model, we found that this data yielded very poor results.

Thus, we decided to use outside training data from the Labeled Faces in the Wild (LFW) dataset, which contains more than 13,000 images of faces collected from the web and cropped.

Here are a few examples of the positive training data that we used:

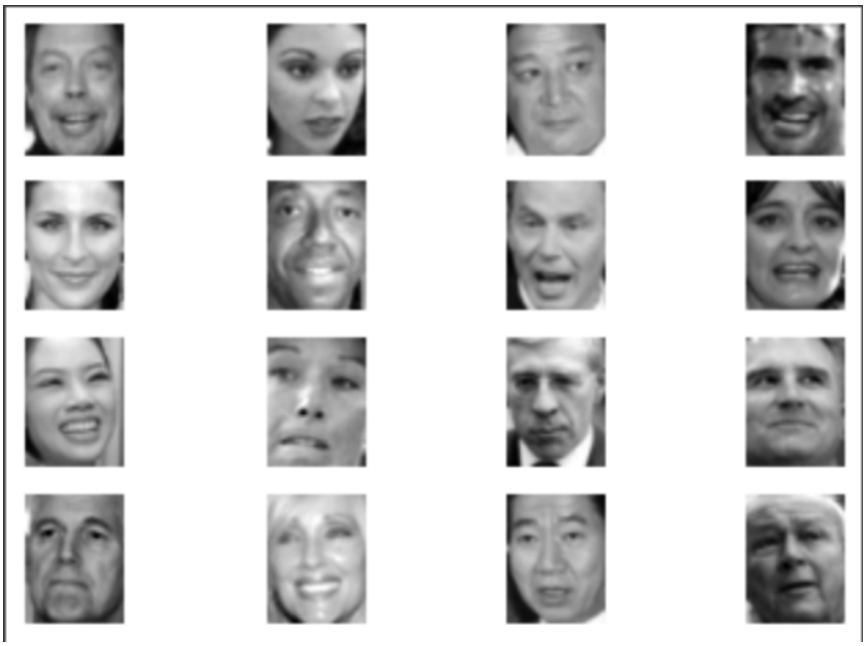


Figure: Examples of positive training data

We convert all the face patches into grayscale, and resize them to a size of 36 x 48.

We also tried combining both cropped faces from our datasets and additional training data from the LFW dataset, but the results yielded are still quite terrible.

For negative training data, we first collected images of random objects and sceneries like fields, papers… We can retrieve these images using default images provided by *skimage.* Afterwards we extract random patches from these images, resize them to the size of 36 x 48, and convert them to grayscale. We finally ended up with about 30000 negative image patches for training

Figure: Examples of negative training data

HOG Feature Extractor

To extract features from our training images, we utilized the HOG feature descriptor from OpenCV *cv2.HogDescriptor.* We choose a cell size of 6×6, a number of histogram bins equals to 9. We set a block of size 2×2 for block normalization.

After computing the histogram features, each image patch will be represented by a vector of size 1260. This vector will be fed through the SVM classifier

SVM Classifier

For classification, we implemented a simple Linear SVM with the library sklearn. We performed grid search on the C values of the SVM algorithm to find the model with the optimal performance.

1.2.2. Inference

Like we've stated before, for an input image we perform the sliding window operation on the input image, with the window size being the same size as the patches that we trained on (36×48).

Since this method is known to have a false positive problem, we can choose to predict a window as positive only if the class probability is above a certain threshold. From our experience, a threshold of about 0.8 - 0.9 seems to yield the best results.

Finally, we performed Non-Maximum Suppression with an overlap threshold of 0.5 to get rid of overlapping bounding boxes.

1.2.3. Hard Negative Mining

After implementing and testing our model, the results we got are quite low, and the prediction results are riddled with False Positives.

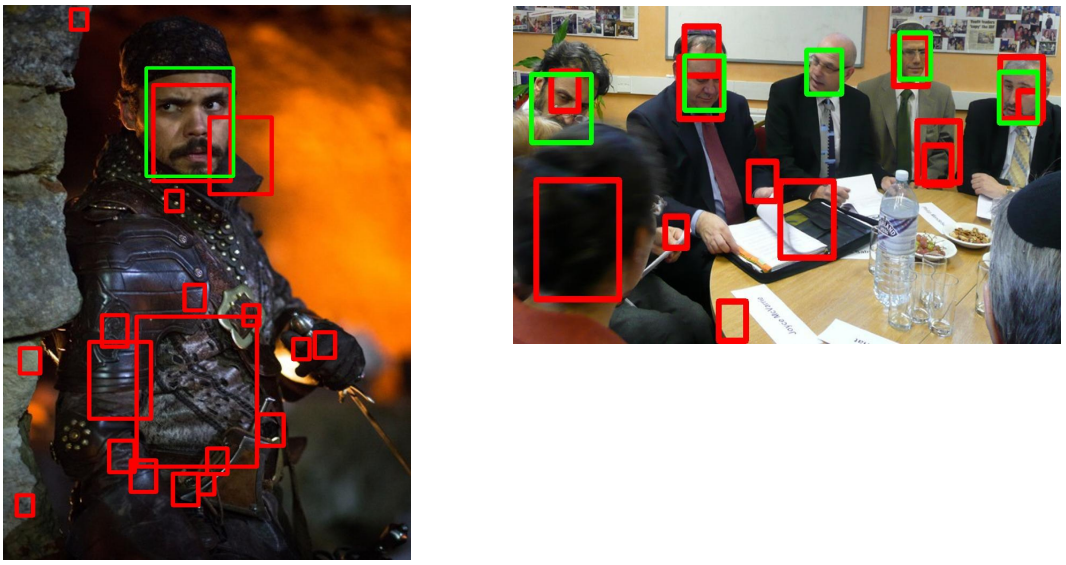


Figure: Some predictions of SVM model, with probability threshold of 0.9

To mitigate this problem, we performed Hard Negative Mining to improve on our baseline model.

To do this, we perform detection on the training set of the WIDER FACE dataset. We record and store all the false positive patches in our predictions, and use them as training examples to retrain our models. The idea here is to train the models on the instances where it is deceived as a positive.

2. Faster-RCNN

2.1. Theoretical Backgrounds

Faster R-CNN is an extension of Fast R-CNN. As its name suggests, Faster R-CNN is faster than Fast R-CNN thanks to the region proposal network (RPN).

The main improvement from Fast R-CNN are:

- Region proposal network (RPN): a fully convolutional network that generates proposals with various scales and aspect ratios. The RPN implements the terminology of neural network with attention to tell the object detection (Fast R-CNN) where to look.
- The convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time.

The architecture of Faster R-CNN is shown in the next figure. It consists of 2 modules:

- RPN: For generating region proposals.
- Fast R-CNN: For detecting objects in the proposed regions.

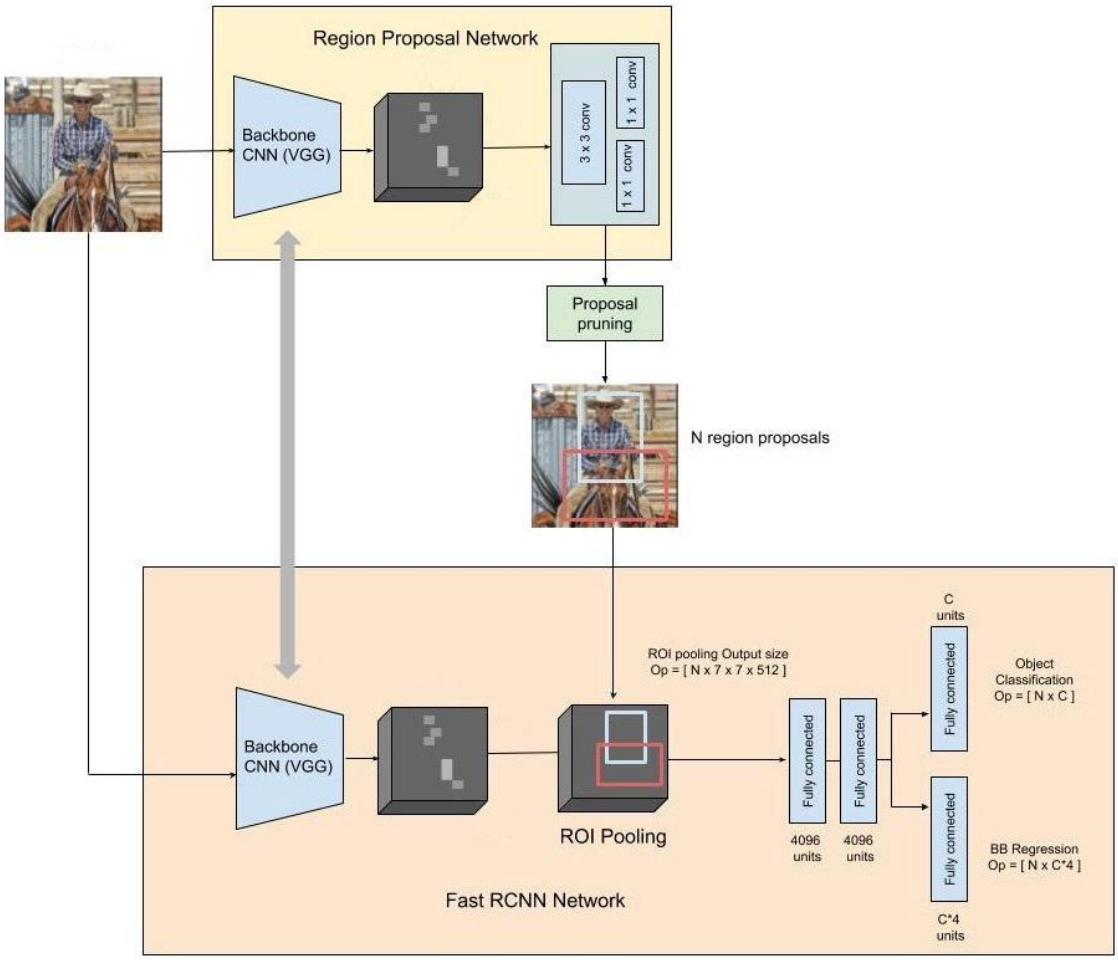


Figure: Faster R-CNN's overall structure [Faster R-CNN for object detection by Shilpa Ananth (towardatascience.com)]

The RPN module is responsible for generating region proposals. It applies the concept of attention in neural networks, so it guides the Fast R-CNN detection module to where to look for objects in the image. Note how the convolutional layers (e.g. computations) are shared across both the RPN and the Fast R-CNN modules.

The Faster R-CNN works as follows:

- The RPN generates region proposals.

- For all region proposals in the image, a fixed-length feature vector is extracted from each region using the ROI Pooling layer
- The extracted feature vectors are then classified using the Fast R-CNN.
- The class scores of the detected objects in addition to their bounding-boxes are returned.

Region Proposal Network (RPN)

The R-CNN and Fast R-CNN models depend on the Selective Search algorithm for generating region proposals. Each proposal is fed to a pre-trained CNN for classification. The Region Proposal Network (RPN) produces region proposals which has some advantages:

- The region proposals are now generated using a network that could be trained and customized according to the detection task.
- Because the proposals are generated using a network, this can be trained end-to-end to be customized on the detection task. Thus, it produces better region proposals compared to generic methods like Selective Search and EdgeBoxes.
- The RPN processes the image using the same convolutional layers used in the Fast R-CNN detection network. Thus, the RPN does not take extra time to produce the proposals compared to the algorithms like Selective Search.

Due to sharing the same convolutional layers, the RPN and the Fast R-CNN can be merged/unified into a single network. Thus, training is done only once.

The RPN works on the output feature map returned from the last convolutional layer shared with the Fast R-CNN. This is shown in the next figure. Based on a rectangular window of size $n \times n$, a sliding window passes through the feature map. For each window, several candidate region proposals are generated. These proposals are not the final proposals as they will be filtered based on their "objectness score" (explained below).

Anchor

According to the next figure, the feature map of the last shared convolution layer is passed through a rectangular sliding window of size $n \times n$, where $n = 3$ (In another word, passed through a 3×3 convolution layer). For each window, k region proposals are generated. Each proposal is parameterized according to a reference box which is called an anchor box. The 2 parameters of the anchor boxes are:

- Scale

- Aspect Ratio

Generally, there are 3 scales and 3 aspect ratios and thus there is a total of $k = 9$ anchor boxes. But k may be different than 9. In other words, k regions are produced from each region proposal, where each of the k regions varies in either the scale or the aspect ratio. Some of the anchor variations are shown in the next figure.

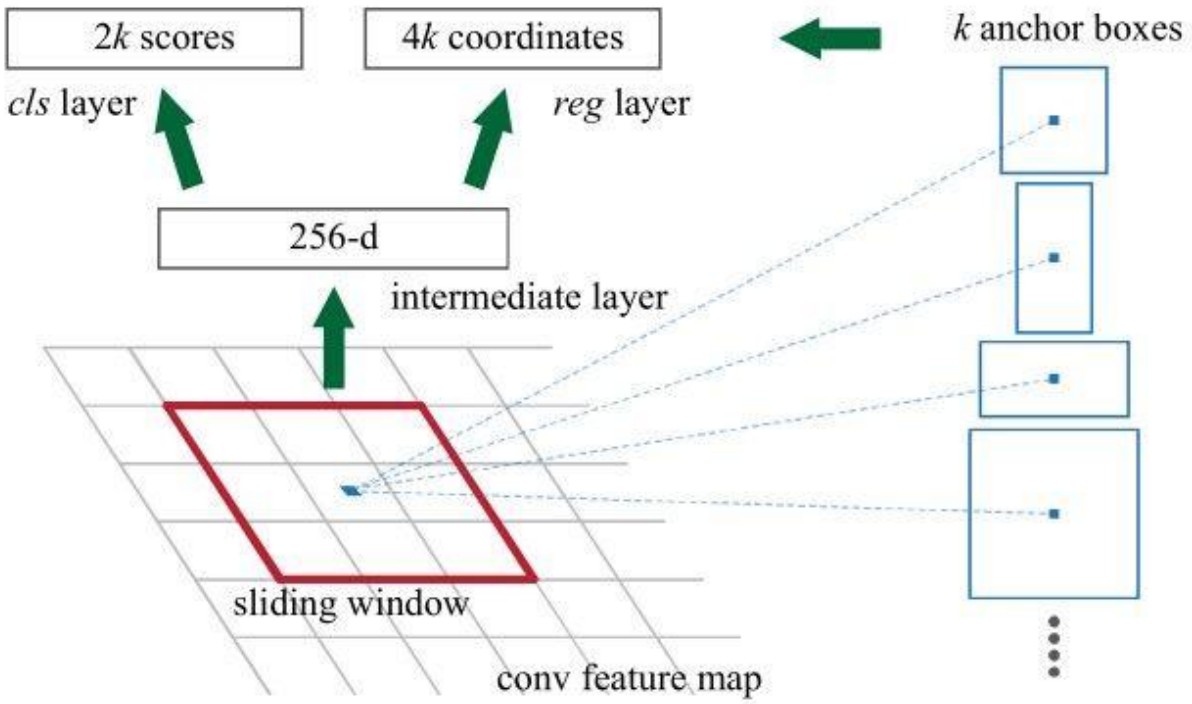


Figure: Region Proposal Network [Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks."]

Using reference anchors (i.e. anchor boxes), a single image at a single scale is used while being able to offer scale-invariant object detectors, as the anchors exist at different scales. This avoids using multiple images or filters. The multi-scale anchors are key to share features across the RPN and the Fast R-CNN detection network.

For each $n \times n$ region proposal, a feature vector (of length 256 in the above figure, 512 for our implementation) is extracted. This vector is then fed to 2 sibling fully-connected layers (or a 1×1 convolution layer on said feature vector):

- The first FC layer is named cls, and represents a binary classifier that generates the objectness score for each region proposal (i.e. whether the region contains an object, or is part of the background).

- The second FC layer is named reg which returns a 4-D vector defining the bounding box of the region.a

The first FC layer (i.e. binary classifier) has 2 outputs. The first is for classifying the region as a background, and the second is for classifying the region as an object. The next section discusses how the objectness score is assigned to each anchor and how it is used to produce the classification label.

Proposal pruning

The anchors from each image go through a series of post-processing steps to send in the object proposal bounding boxes, in our implementation:

- The regression coefficients are applied to the anchors for precise localization. This gives precise bounding boxes.
- All the boxes are arranged according to their cls scores.
- Take top N region proposal with the highest cls score (Test: $N = 1000$, Train: $N = 2000$)
- Then, a non-maximum suppression (NMS) is applied with a threshold of 0.7. From the top down, all of the bounding boxes which have an IoU of greater than 0.7 with another bounding box are discarded. Thus the highest-scoring bounding box is retained for a group of overlapping boxes.
- Take top N region proposal with the highest cls score after NMS (Test: $N = 1000$, Train: $N = 2000$)
- The cross-boundary bounding boxes are retained and clipped to the image boundary.

Objectness Score

The cls layer outputs a vector of 2 elements for each region proposal. If the first element is 1 and the second element is 0, then the region proposal is classified as background. If the second element is 1 and the first element is 0, then the region represents an object.

For training the RPN, each anchor is given a positive or negative objectness score based on the Intersection-over-Union (IoU).

The next 4 conditions use the IoU to determine whether a positive or a negative objectness score is assigned to an anchor:

- An anchor that has an IoU overlap higher than 0.7 with any ground-truth box is given a positive objectness label.

- If there is no anchor with an IoU overlap higher than 0.7, then assign a positive label to the anchor(s) with the highest IoU overlap with a ground-truth box.

- A negative objectness score is assigned to a non-positive anchor when the IoU overlap for all ground-truth boxes is less than 0.3. A negative objectness score means the anchor is classified as background.
- Anchors that are neither positive nor negative do not contribute to the training objective.

The next equation summarizes the 4 conditions.

$$\boldsymbol{Objectness_{score}(IoU)} = \begin{cases} \boldsymbol{Positive \rightarrow IoU > 0.7} \\ \boldsymbol{Positive \rightarrow 0.5 < IoU \leq 0.7} \\ \boldsymbol{Negative \rightarrow IoU < 0.3} \\ \boldsymbol{Not\ Negative/Positive \rightarrow 0.3 \leq IoU \leq 0.5} \end{cases}$$

Fast R-CNN Network and ROI Pooling

The Fast R-CNN Network accepts an image as an input and returns the class probabilities and bounding boxes of the detected objects.

The feature map from the last convolutional layer is fed to an ROI Pooling layer. The reason is to extract a fixed-length feature vector from each region proposal.

The extracted feature vector using the ROI Pooling is then passed to some FC layers. The output of the last FC layer is split into 2 branches:

- Softmax layer to predict the class scores
- FC layer to predict the bounding boxes of the detected objects

ROI Pooling

The ROI Pooling layer works by splitting each region proposal equally into a grid of cells. The max pooling operation is applied to each cell in the grid to return a single value. All values from all cells represent the feature vector.

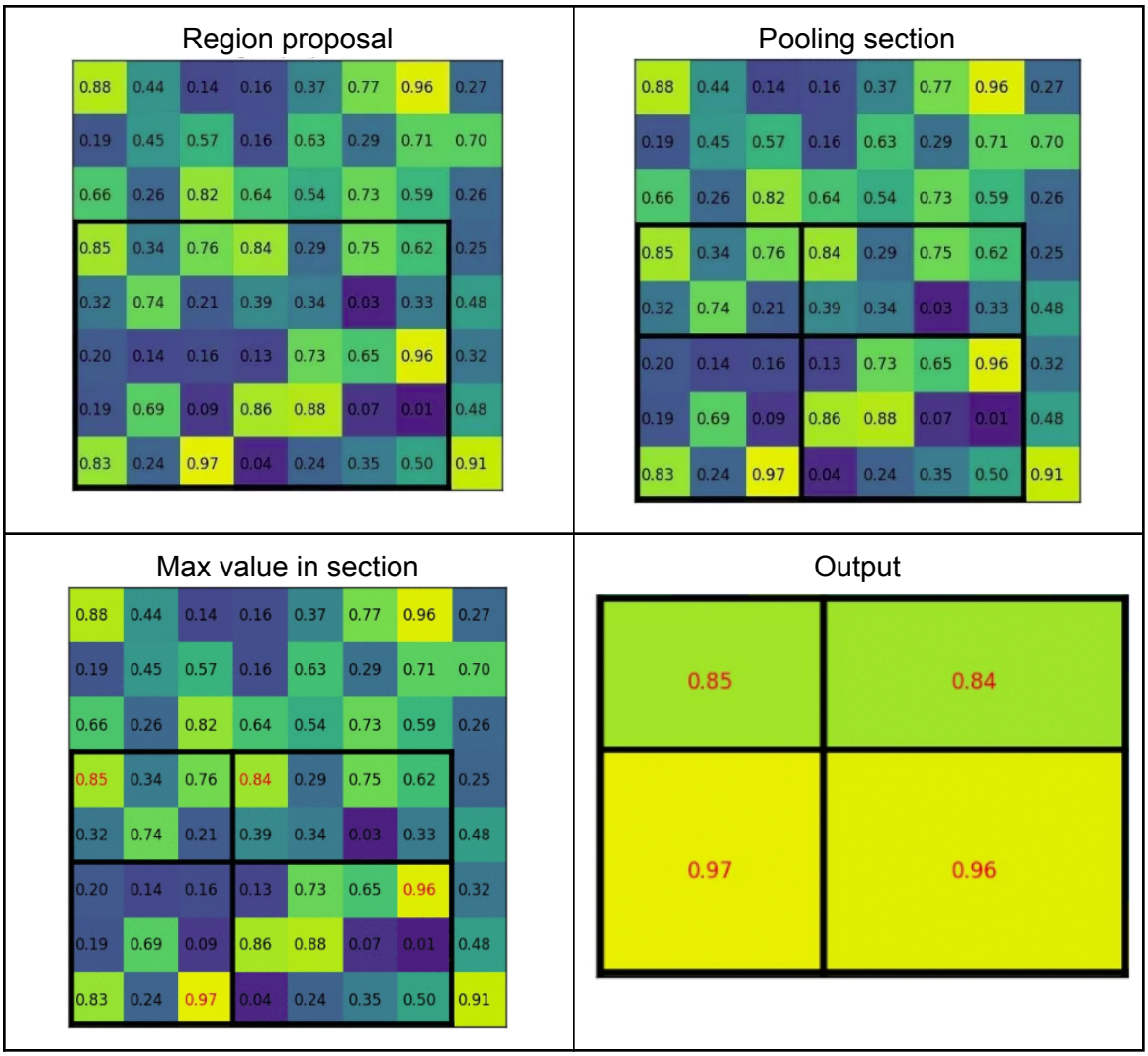


Figure: Example of ROI pooling [Region of interest pooling explained by Tomasz Grel (deepsense.ai)]

Loss Functions

RPN

Each mini-batch for training the RPN comes from a single image. Sampling all the anchors from this image would bias the learning process toward negative samples, and so 128 positive and 128 negative samples are randomly selected to form the batch, padding with additional negative samples if there are an insufficient number of positives.

The Faster R-CNN propose the following loss function for the RPN, for each image:

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

Here,

- i: The index of an anchor in a mini-batch
- p_i: The predicted probability of anchor i being an object.
- p_i^*: The ground-truth label, = 1 if the anchor is positive, = 0 if the anchor is negative.
- t_i: A vector representing the 4 parameterized coordinates of the predicted bounding box
- t_i^*: the ground-truth box associated with a positive anchor.
- The classification loss L_{cls} is log loss over two classes (object vs. not object).
- The regression loss $L_{reg}(t_i, t_i^*)$ = $R(t_i - t_i^*)$ 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).
- The outputs of the cls and reg layers consist of $\{p_i\}$ and $\{t_i\}$ respectively.
- The two terms are normalized by N_{cls} and N_{reg} and weighted by a balancing parameter λ. In the paper implementation:
    - the cls term is normalized by the mini-batch size (i.e., N_{cls} = 256)
    - the reg term is normalized by the number of anchor locations (i.e., N_{reg} ~ 2,400).
    - By default λ = 10, and thus both cls and reg terms are roughly equally weighted.

For bounding box regression, the parameterizations of the 4 coordinates follows:

$$t_x = (x - x_a)/w_a,\ t_y = (y - y_a)/h_a,$$

$$t_w = log(w/w_a)/w_a,\ t_h = log(h/h_a),$$

$$t_x^* = (x^* - x_a)/w_a,\ t_y^* = (y^* - y_a)/w_a,$$

$$t_w^* = log(w^*/w_a)/w_a,\ t_h^* = log(h^*/h_a),$$

In the formulation, the features used for regression are of the same spatial size 3×3 on the feature maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor is responsible for one scale and one aspect ratio, and the k regressors do not share weights. As such, it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale, thanks to the design of anchors.

Fast R-CNN

As for the Fast R-CNN module, it uses the same loss function as in the original paper of Fast R–CNN, for each regional proposal:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

Where:

- u: ground truth class
- v: bounding-box regression target
- p: probability distribution over the data classes plus the background class (output of softmax layer)
- $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$: a bounding-box tuple for class g (part of the output of regression layer)
- $L_{cls}(p, g) = -\log p_g$
- $L_{loc}(t^g, r) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^g, r_i)$

Note: the $[g \geq 1]$ term indicates that there is no L_{loc} for background classes (which are the regional proposals where $p_i^* = 0$ (or where the regional proposal's anchor is negative))

Training Faster R-CNN

For Faster R-CNN it is possible to build a unified network in which the RPN and Fast R-CNN are trained at once.
The core idea is that both the RPN and Fast R-CNN share the same convolutional layers. These layers exist only once but are used in the 2 networks.
The Faster R-CNN paper mentioned 3 different ways to train both the RPN and Fast R-CNN while sharing the convolutional layers:

- Alternating Training
- Approximate Joint Training
- Non-Approximate Joint Training

Alternating Training is the preferred way to train the 2 modules and the authors developed a 4-step training process.
“In the first step, we train the RPN as described above. This network is initialized with an ImageNetpre-trained model and fine-tuned end-to-end for the region proposal task. In the second step, we train a separate detection network by Fast R-CNN using the proposals generated by the step-1 RPN. This detection network is also initialized by the ImageNet-pre-trained model. At this point the two networks do not share conv layers. In the third step, we use the detector network to initialize RPN training, but we fix the shared conv layers and only fine-tune the layers unique to RPN. Now the two networks share conv layers. Finally, keeping the shared conv layers fixed, we fine-tune the fc layers of the Fast R-CNN. As such, both networks share the same conv layers and form a unified network.”

2.2. Implementation

In this project, we utilized the torchvision library to implement the Faster RCNN model.

The toolkit also provided us with a pretrained weight, trained for the task of object detection on the COCO dataset. We trained this model both with the default weights and with the pretrained weights from COCO and compared the results obtained.

Before passing an image to model, the image also needs to be resized and normalized. However, the Faster RCNN implementation already took care of this, where an image is automatically resized with the width of 800, and normalized using the mean and standard deviation of the ImageNet dataset’s RGB channels.

mean = [0.485, 0.456, 0.406]

std = [0.229, 0.224, 0.225]

As for the training procedure, we trained our model for a total of 60000 iterations. The model is trained with the Stochastic Gradient Descent with Momentum optimizer, with a learning rate equals to 0.05, a momentum of 0.9 and weight decay of 0.05.

The chosen batch size for our dataset is 16. The training data loader consists of both the original images in the training set and the augmented data applied on the whole training set, with the same pipeline as we discussed prior.

We evaluate the loss on the validation set every 300 iterations, and only save the best models' checkpoint so far, which is the model's checkpoint with the lowest validation loss. The learning rate will be decreased with a factor of 0.1 if there are no improvements after 3 checkpoints.

3. Single Shot MultiBox Detector (SSD)

While most classic detectors are based on either sliding window approach, or two-stage detection using a region proposal network before a classifier network, single shot detectors can detect objects in a single pass, using a single deep neural network. Among them, Single Shot MultiBox Detector (SSD for short) emerged as a powerful network for this task, with much faster computation time and comparable accuracy.

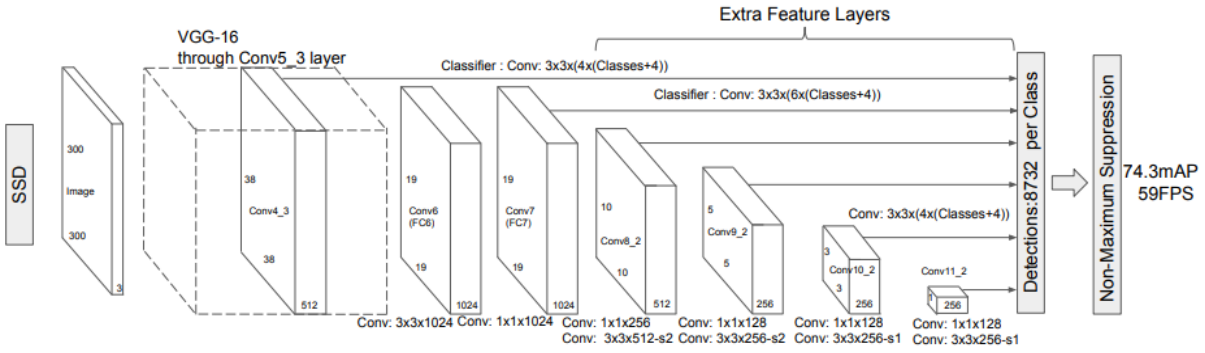
3.1. Theoretical Backgrounds

SSD uses a convolution network to produce a fixed-size collection of bounding boxes and their confidence scores. A non-maximum suppression step is then applied to this output to make the final detections. A special trait of SSD among single shot detectors is that it produces predictions at different scales from multiple feature maps.

SSD model consists of 2 main part:

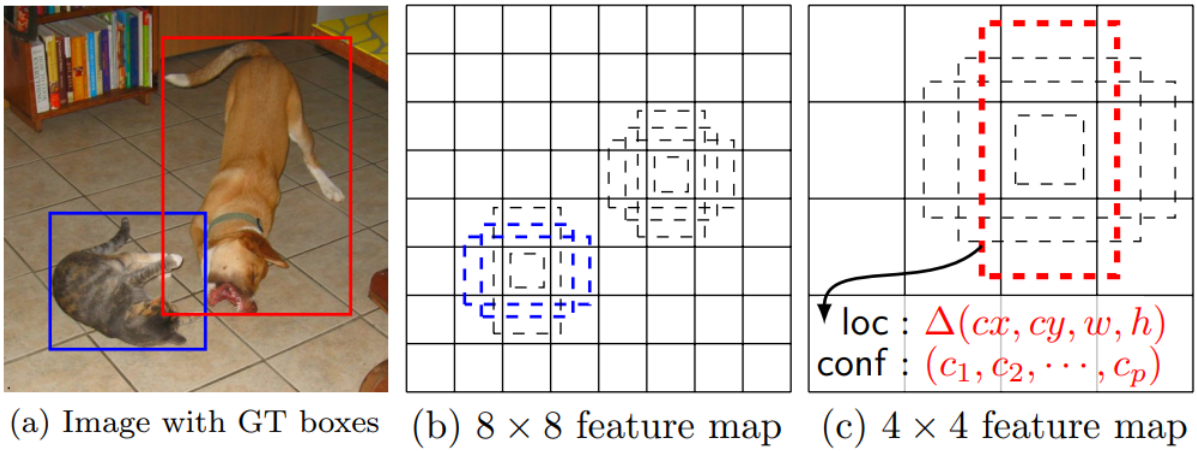
- The first part is a truncated *base network*. It is derived from a standard image classification architecture with the classification layers truncated as its purpose is to extract feature maps from the input image. A backbone model pre-trained on a reliable classification dataset (ImageNet for example) can be used to make this base network to take advantage of the knowledge from closely related tasks. The author of the original SSD paper employed VGG-16 as their base network.

- The second part is an auxiliary structure (often called *detection heads*) added at the end of the base network to produce detection. This structure includes convolution feature layers that allow detections at multiple scales, and convolutional predictors for each feature layer.



Demonstration of the SSD model architecture

As mentioned above, SSD outputs a fixed-size collection of bounding boxes, which are the offsets from the default bounding boxes, and their confidence scores, which are the probability scores for each class. The detection problem can thus be formulated as the regression of the bounding boxes combined with the classification of the objects. To solve this problem, each feature map is divided into grids, and then each grid is associated with a set of default bounding boxes (these are often called *anchors*), which serve as initial guesses for the prediction of the actual predicted bounding boxes. In other words, we predict the box offsets from these default bounding boxes, along with the per-class confidence scores. The default bounding boxes tile the feature maps in a convolutional manner so that the position of each box relative to its grid is fixed.



Demonstration of SSD default bounding boxes

To be more specific, suppose we have k bounding boxes at a given grid, for each box we have to compute 4 offset values and c confidence scores (corresponding to c

classes). That makes $(4 + c) * k$ output values for a grid, or $(4 + c) * k * m * n$ values for a feature map of size $m \times n$.

To match the predicted bounding boxes with the ground truth, the author proposed to first match each ground truth box to the default box with the best IoU, called *positive matches*, then match default boxes to any ground truth with IoU higher than a threshold. The unselected default boxes (IoU < threshold) are considered *negative matches*, and there are often much more negative matches than positive ones. A way to deal with this problem is to sort these negative matches by using the highest confidence loss and then pick only the top ones to keep the negative/positive ratio to be at most 3:1. By doing this, the model will focus on the predictions it found hardest to recognize that there are no objects.

The training objective for SSD is:

$$L = L_{conf} + \alpha * L_{loc}$$

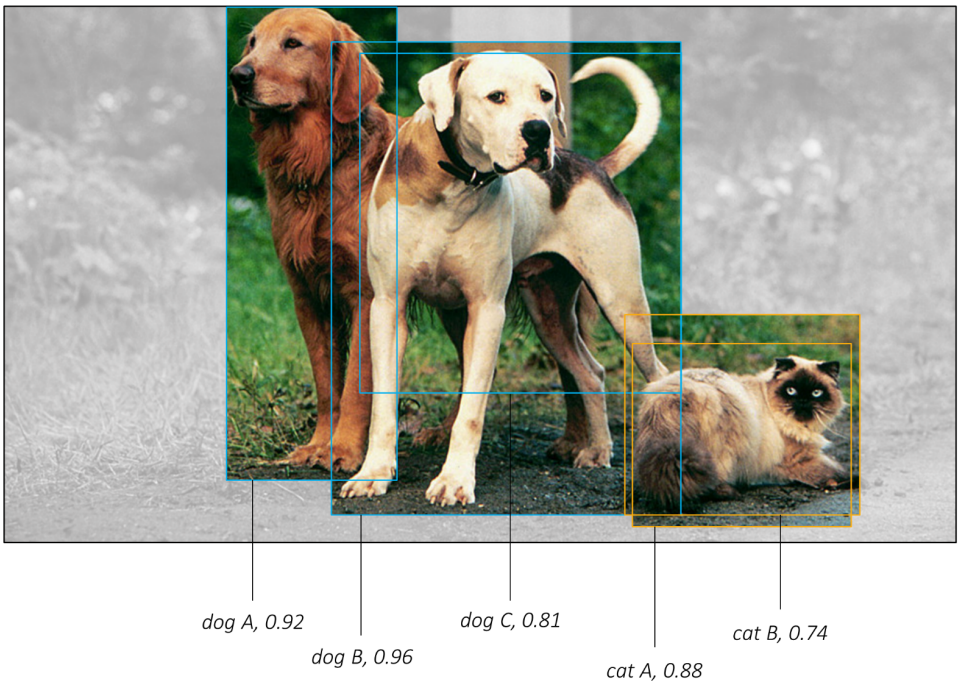
where:

- $L_{conf} = \frac{1}{n_{positives}} (\Sigma_{positives} CELoss + \Sigma_{hard\ negatives} CELoss)$
- $L_{loc} = \frac{1}{n_{positives}} (\Sigma_{positives} SmoothL_1 Loss)$
- α: a hyperparameter.

Here, the localisation loss L_{loc} calculates how accurately the positive matches are regressed to the ground truth bounding boxes. The confidence loss L_{conf} is just the cross entropy loss over multiple class confidences. These confidences are produced by a softmax layer. Since we're using some of the hard negatives in the calculation of the confidence loss, we must also include a "background" class in the set of class labels. In other words, predicted boxes that don't contain an object will most likely contain background.

Finally, there are often multiple bounding boxes predicted for the same object. To avoid this problem, a Non-maximum suppression step is applied to remove overlapping boxes, keeping only the one with the highest confidence score for each detected object.

There are usually multiple predictions for the same object



Non-maximum suppression is needed to deal with overlapping predictions

3.2. Implementation

We implemented the SSD model using the PyTorch library.

Firstly, before passing an image through the model, we resize the image to $300 \times 300 \times 3$. Afterwards, the image is normalized with the mean standard deviation of the ImageNet images' RGB channels:

mean = [0.485, 0.456, 0.406]

std = [0.229, 0.224, 0.225]

As for the training procedure, we also trained our model for a total of 60000 iterations. The model is trained with the Adam optimizer, with a learning rate equals to 1e-4; beta1 ,beta2's values of respectively 0.9 and 0.999; weight decay of 0.05.

The chosen batch size for our dataset is 16. For SSD, we trained the model with and without augmented data, and observed if data augmentation has any effects on the model.

We evaluate the loss on the validation set every 300 iterations, and only save the best models' checkpoint so far, which is the model's checkpoint with the lowest validation

loss. The learning rate will be decreased with a factor of 0.1 if there are no improvements after 3 checkpoints.

IV. Evaluation and Results

1. Evaluation metric

We use Average Precision (AP) to evaluate the results.

This AP is given by the area under the precision/recall (PR) curve for the detections. The PR curve is constructed by first mapping each detection to its most-overlapping ground-truth object instance. All of the detection with $\geq 50\%$ IoU with its most-overlapping ground-truth object instance are counted as a true-positive, and all other detections as false-positives.

Next, we compute recall and precision values.

- Recall is defined as the ratio of true-positive detections to ground-truth instances
- Precision as the ratio of true-positive detections to all detections.

The PR curve is then given by plotting these recall-and-precision pairs. Finally, dips in the curve are filled in (interpolated) by replacing each precision with the maximum of itself and all precisions occurring at higher recall levels.

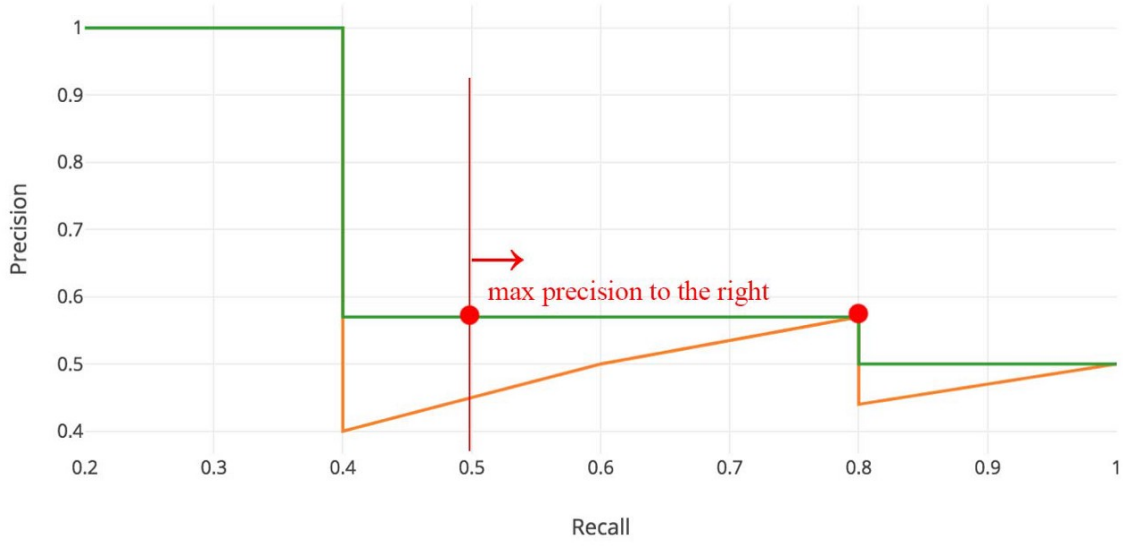


Figure: An example of interpolating PR curve [mAP (mean Average Precision) for Object Detection by Jonathan Hui (medium.com)]

The area under the interpolated PR curve is the AP value for the class.

Our code for evaluation is implemented based on the code in the following repository:

[wondervictor/WiderFace-Evaluation: Python Evaluation Code for Wider Face Dataset (github.com)]

As we have mentioned previously, the authors of the dataset group the official test set and validation set into 3 subsets based on the detection rate of EdgeBox: Easy, Medium and Hard. In our case the official validation set is our test set. We will compare our model performance on these 3 subset.

2. Results

2.1. HOG + SVM

	Easy	**Medium**	**Hard**
Baseline	0.004	0.003	0.001
Hard Negative Mining	0.195	0.12	0.05

We evaluated the results of our HOG SVM trained before and after Hard Negative Mining. We observe a big increase in performance of the model.

2.2. SSD

	Easy	**Medium**	**Hard**
No Augmentation	0.744	0.572	0.282
Augmented	0.785	0.632	0.328

As we can see in this result, data augmentation helps improve the performance of SSD on all 3 datasets. The positive effect of augmentation is more significant in harder datasets.

2.3. Faster R-CNN

	Easy	**Medium**	**Hard**
Default Weights	0.846	0.758	0.442
COCO's Weights	0.853	0.768	0.445

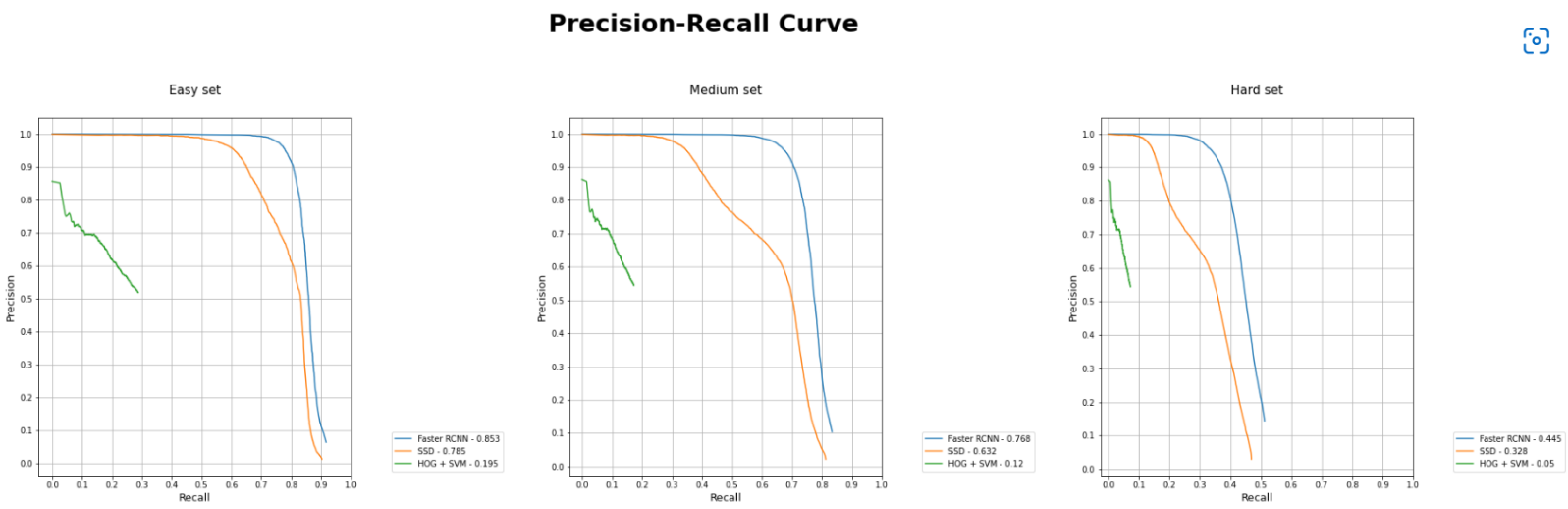
Pre-training helps improve the model performance. Though the numbers are not much of a difference, the effect of pre-training is crucial since it saves a lot of training time, given that Faster R-CNN is very computationally expensive.

2.4. Comparison

The table below shows the APs between our 3 chosen methods

Method	**Easy**	**Medium**	**Hard**
HOG + SVM	0.195	0.120	0.050
SSD	0.785	0.632	0.328
Faster R-CNN	**0.853**	**0.768**	**0.444**

And these are the Precision - Recall curves of our methods

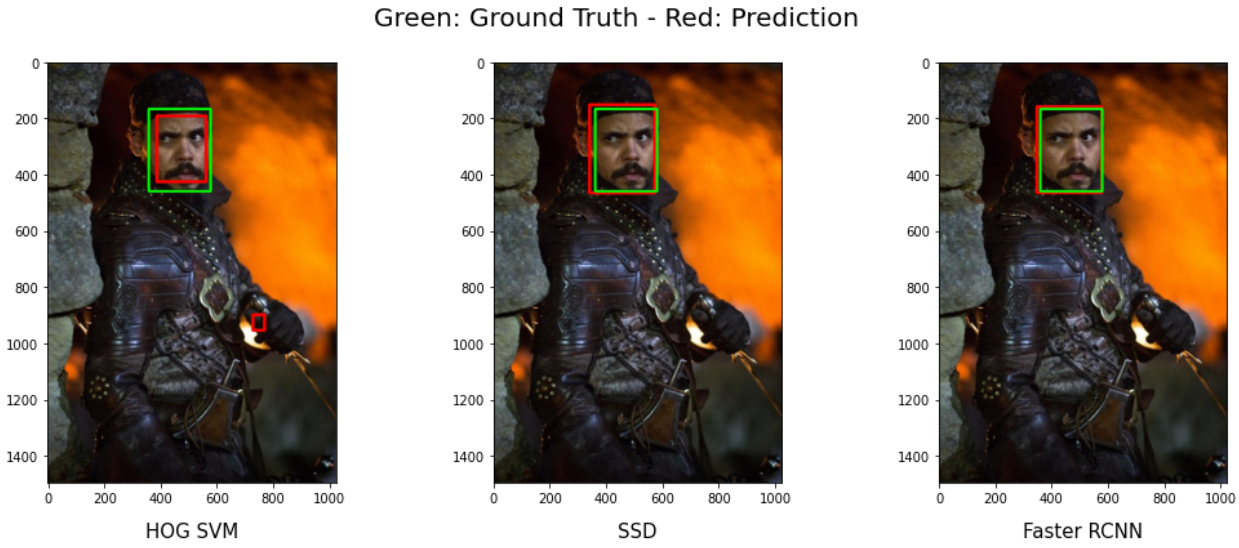


We can easily see that the classical method HOG + SVM gets outperformed significantly by the two modern Deep Learning models, with just 0.195, 0.12 and 0.05 on the 3 Easy-Validation-Test sets, respectively. It is true that handcrafted features usually cannot generalize as well as machine learning based methods.

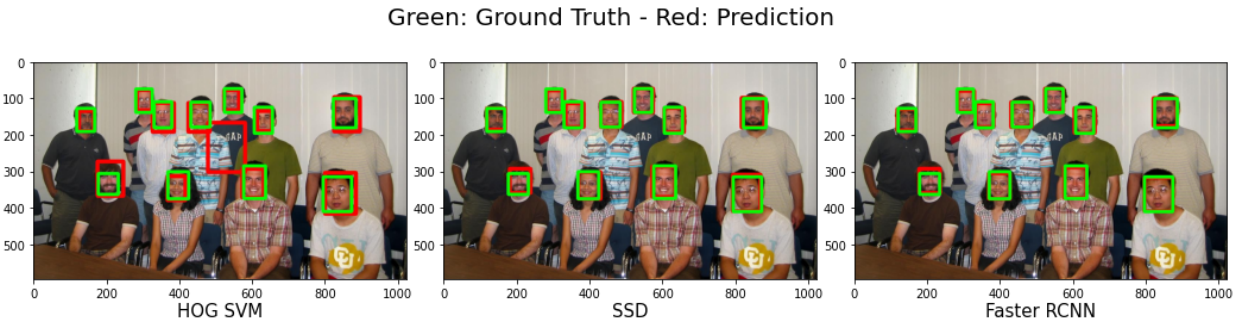
Among the two Deep Learning methods, Faster R-CNN achieved the superior results. The literature in this field also showed that two-stage methods tend to achieve better accuracy at the cost of computational time.

Next, we will take a look at some of the predictions of our models on the test set.

Easy set

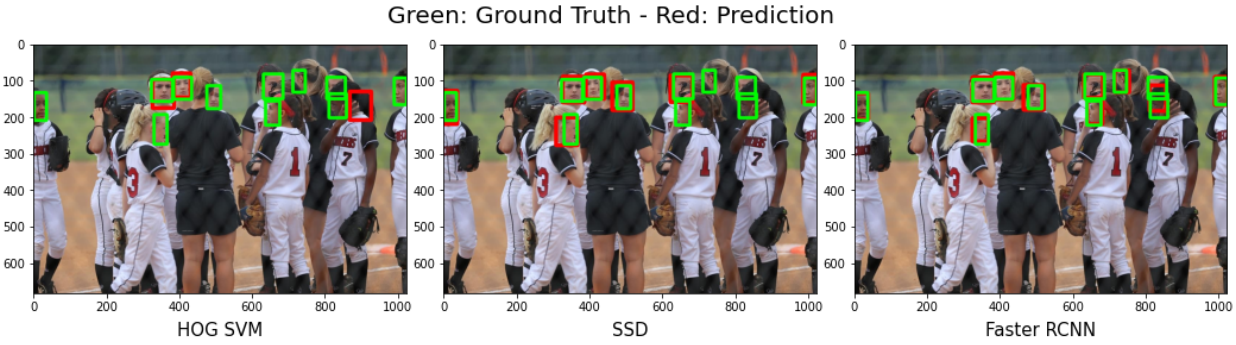


This image is taken from the easy set of our data.As we can see, this is a simple case for detection, since the only face in this image is detected by all of our models. However, HOG SVM misclassified a small patch as false positive

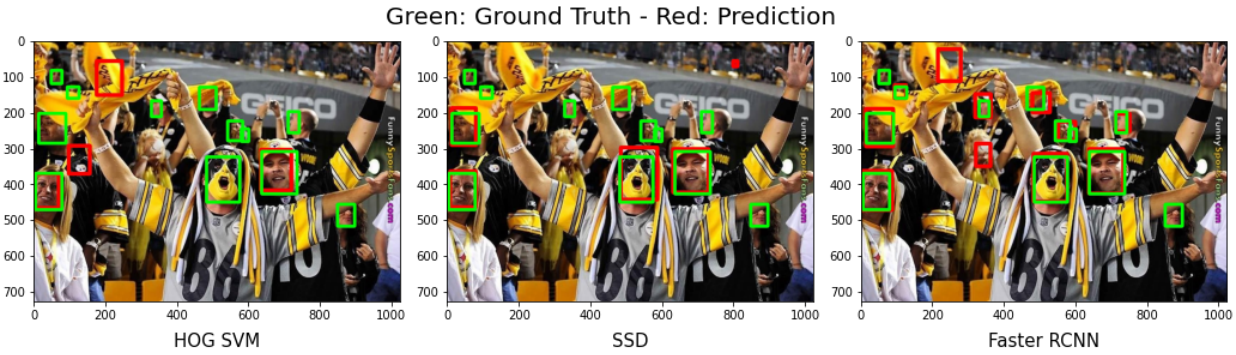


Here is another easy image in our dataset. We can see that all of our models generally do well with detection, since the faces here are front facing with no occlusion or special poses.

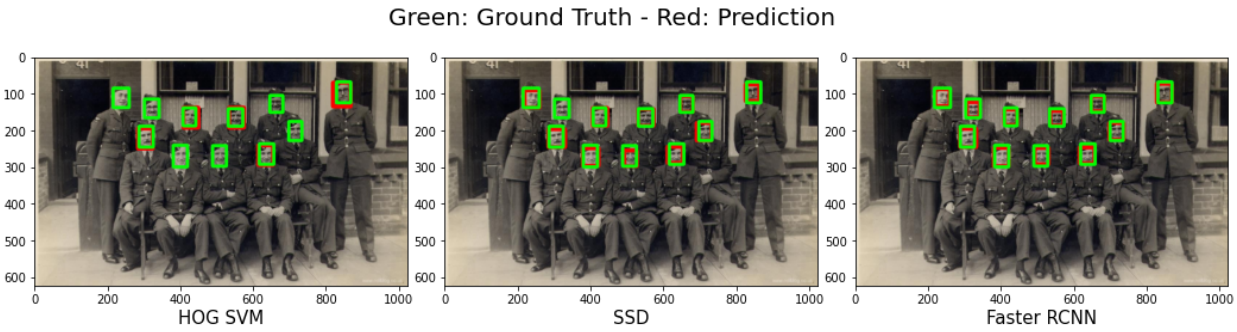
Medium



For this image, we can see that the 2 Deep Learning models did better compared to HOG SVM. HOG SVM was only able to detect 2 biggest and front-facing faces. While SSD and Faster RCNN was able to detect a lot of smaller faces with less convenient poses.

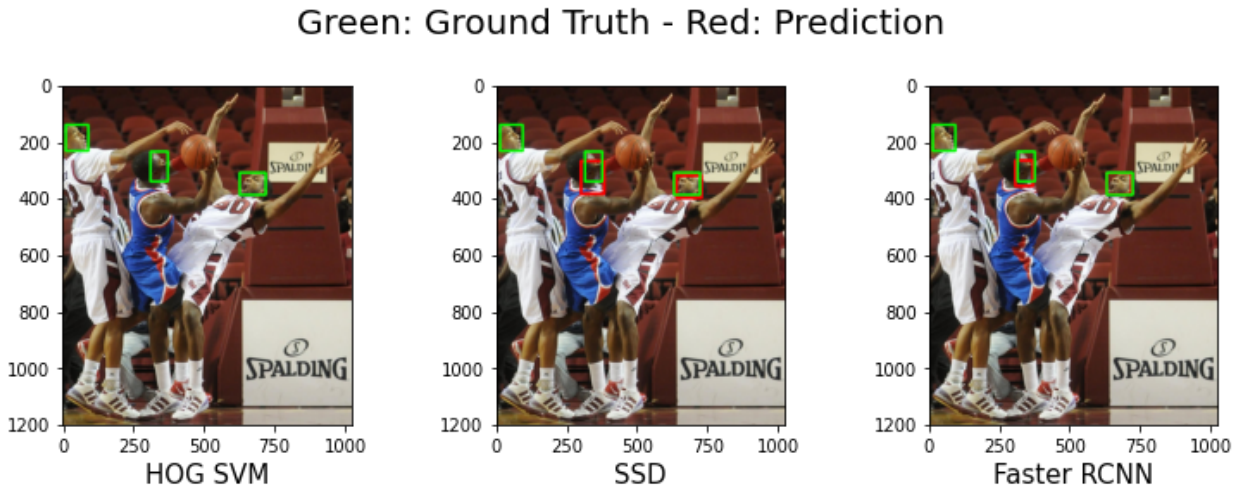


In this image, we see that HOG SVM failed to detect the person wearing face paint while the 2 deep learning models were able to capture it. Also, Faster RCNN seems to be able to capture the smaller faces.

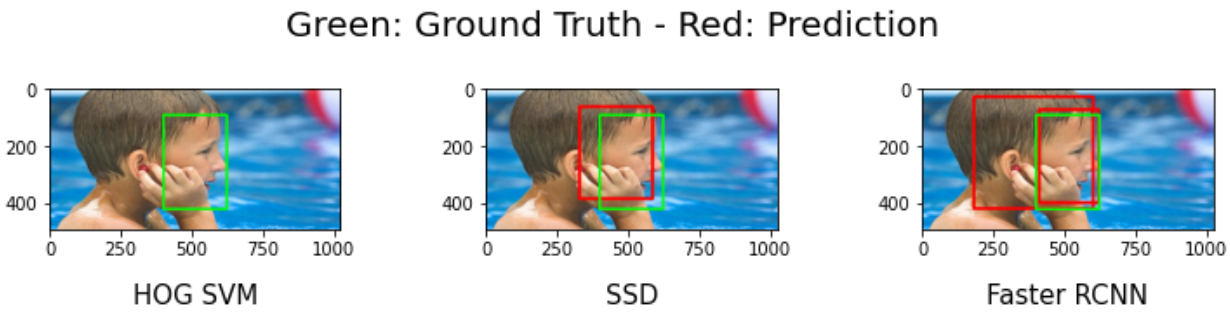


Now that the number of faces per image has increased, and their size is much smaller, it is harder for HOG SVM to detect accurately. But still, SSD and Faster R-CNN can perform quite well.

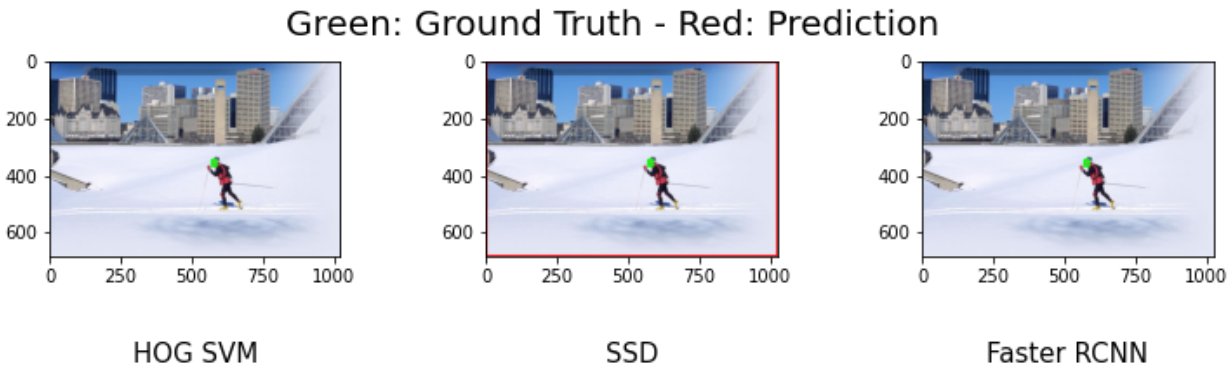
Hard



In this image, we can see that the faces are quite small, and all 3 of them are in quite unique poses and not front-facing. HOG SVM missed all of 3 of them, while SSD and Faster R-CNN managed to detect 2 faces.



In this picture, while the face's scale is quite large, it can be challenging for face detection systems since the kid's face is sideways and not front-facing, and part of his face is occulted by his hand.We see HOG SVM also missed this prediction. Both SSD and Faster RCNN managed to detect this, with Faster RCNN ending up with 1 False Positive alongside the true prediction.



This image is quite challenging, since the ground truth face is tiny. And all 3 of our models failed to predict anything in this case.

The 2 deep neural network models have their performance dropped significantly in the Hard set as compared to the Normal and the Easy set. An explanation could be found in the network structure of both models.

In the paper "Face Detection through Scale-Friendly Deep Convolutional Networks", Yang et al. discuss the shortcoming of both Faster-RCNN and SSD in face detection task:

- Faster-RCNN extracts scale-invariant features through region of interest (ROI) pooling but are not specifically designed to find faces in a wide range of scales. Specifically, the foreground and background ROIs of Faster R-CNN map to the same location on deep features, causing ambiguity to the classifier.
- SSD makes use of scale-variant templates based on the deep features and tries to detect objects of various scales at different stages/layers of the network. Nevertheless, a direct application of SSD for small face detection still does not

return satisfactory results since the scale-variant templates at early layers cannot cope well with large-scale variance. While in the later stages, SSD will suffer similar overlap mapping problems as in Faster R-CNN.

V. Conclusions

In this project, we have explored the usage of 3 face detection methods, from classical to more modern ones: HOG + SVM, Single Shot Detector (SSD), Faster RCNN, and examined their performance. Predictably, HOG + SVM has the worst result partially due to handcrafted features usually cannot generalize as well as machine learning based methods. The SSD models have better results and also proves that Data Augmentation can improve APs. Finally, Faster-RCNN is the best method out of the three with APs of 0.853, 0.768, 0.444 for the Easy, Medium and Hard subset respectively.

VII. References

WIDER FACE: A Face Detection Benchmark (shuoyang1213.me)
Histogram of Oriented Gradients explained using OpenCV (learnopencv.com)
Dalal-cvpr05.pdf (inrialpes.fr)
LFW Face Database : Main (umass.edu)
Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks."
Girshick, Ross. "Fast r-cnn."
Faster R-CNN for object detection by Shilpa Ananth (towardatascience.com)
Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation."
Region of interest pooling explained by Tomasz Grel (deepsense.ai)
Wei, Liu, et al. “SSD: Single Shot MultiBox Detector.”
mAP (mean Average Precision) for Object Detection by Jonathan Hui (towardatascience.com)
Shuo. Yang, et al. “Face Detection through Scale-Friendly Deep Convolutional Networks”