ECSE 415 Project Report

McGill University, Fall 2020

Group #: 17

Group Members:

-Ismail Faruk 260663521

-Xirui Zhang 260656808

-Sameen Mahtab 260737048

Description of Person Detector (part 1):

The code of Detectron2 is available on the Detectron2 GitHub repository.

For the initial part of the project, Detectron2, an object detection platform written on PyTorch has been utilized [1]. Detectron2 offers several models; in this project, the object detection model is used. Here, instances are classified and localized with bounding boxes [2]. Object detection is performed on images in the 'mall' dataset to detect people and obtain bounding boxes around them.

The model is pretrained on the COCO dataset. In this case, the faster_rcnn_R_50_FPN_3x model is used. Faster R-CNN is composed of two modules: a fully convolutional network and a Fast R-CNN detector. It is a "single, unified network" for object detection [3]. The features of the images are extracted and the regions where person objects exist are found. Regions are then bounded by boxes. The results can be seen in the image below (*Figure 1)*.

Figure 1: Using Faster-RCNN to detect person objects

After that, person objects from 100 images of the dataset have been cropped out and saved in a separate file. These images will serve as positive examples. Negative examples have been produced by randomly cropping sections of different dimensions. These are illustrated below in figure 2.

(a) *(b)*

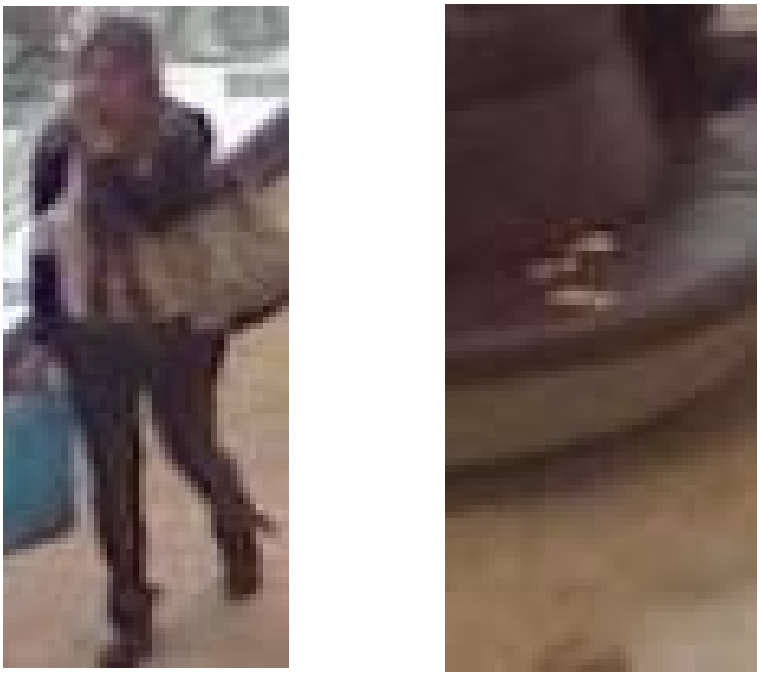

Figure 2. (a) Positive example. (b) Negative example.

The two sets of cropped images are saved in two folders, “positive” and “negative”.

Description of Person Detector (part 2):

In this part a person detection algorithm trained with a Support Vector Machine (SVM) classifier is built from scratch. SVM has the highest precision among other classifiers for human detection [5]. SVM is a binary classification algorithm that makes use of an optimal hyperplane as a decision boundary [4]. Histogram of Oriented Gradients (HOG) values are used as the feature sets and SVM is used to train the classifier.

Feature extraction is one of the fundamental goals. The dataset initially contains 2000 frames. Extracted ground truth using detectron. Found random samples of positive and negative. The cropped images from part one will be used to train the model. From the cropped images of the detectron, 1200 positive samples and 280 negative samples are taken. Performed hog of the samples.

In HOG, the features comprise the pattern in the directions of gradients. Corner and edge detection is fundamental to image detection. The appearance and shape of a local object can be identified reasonably well using the direction of edges [6]. A higher magnitude of edge correlates to a corner or edge. HOG descriptor is calculated by finding the horizontal and vertical gradients. The image is divided into cells, for each cell a histogram of the directions of the gradients is made. Accumulation of the histograms forms the HOG representation [6].

The dataset does not contain train and test sets separately. Initially the dataset is split into a training set and a test set by manually splitting the dataset by a 80:20 ratio figure#. SVC is trained using the training set and the accuracy is found using the test set. Optimal hyperparameters for svc are found by running a grid search. The accuracy after running the grid search can be seen in figure#. The optimal parameters were, gamma as the scale and 10 for C. The accuracy was 96.62%.

```
k_fold = 5
pos_count = Positive_Images
neg_count = 280
pos_train_split = int(pos_count*4/k_fold)
neg_train_split = int(pos_count+neg_count*4/k_fold)

print(f"train_size: {pos_train_split+neg_train_split-pos_count}")
print(f"test_size: {pos_count-pos_train_split+neg_count-neg_train_split+pos_count}")

# splitting all pos and neg into 4/5 for train and 1/5 split for test
train_features_split = np.concatenate((train_features[: pos_train_split],  train_features[pos_count: neg_train_split]), axis=0)
train_labels_split = np.concatenate((train_labels[: pos_train_split], train_labels[pos_count: neg_train_split]), axis=0)

val_features_split = np.concatenate((train_features[pos_train_split:pos_count],  train_features[neg_train_split:]), axis=0)
val_labels_split = np.concatenate((train_labels[pos_train_split:pos_count],  train_labels[neg_train_split:]), axis=0)

print(f"train_split: {train_features_split.shape} and {train_labels_split.shape}")
print(f"val_split: {val_features_split.shape} and {val_labels_split.shape}")
```

Figure 3. Manually splitting the dataset

```
Gamma: auto, C: 0.01,  Accuracy: 81.08%
Gamma: auto, C: 0.1,  Accuracy: 81.08%
Gamma: auto, C: 1,  Accuracy: 81.08%
Gamma: auto, C: 10,  Accuracy: 87.16%
Gamma: auto, C: 100,  Accuracy: 95.27%
Gamma: auto, C: 1000,  Accuracy: 95.95%
Gamma: scale, C: 0.01,  Accuracy: 81.08%
Gamma: scale, C: 0.1,  Accuracy: 83.45%
Gamma: scale, C: 1,  Accuracy: 95.95%
Gamma: scale, C: 10,  Accuracy: 96.62%
Gamma: scale, C: 100,  Accuracy: 96.62%
Gamma: scale, C: 1000,  Accuracy: 96.62%
Best parameters:  {'gamma': 'scale', 'C': 1000, 'accuracy': 96.62}
```

Figure 4. Accuracy after using 1 fold

```
Gamma: auto, C: 0.01,  Accuracy: 81.08%, time taken to train/test: 27.7
Gamma: auto, C: 0.1,  Accuracy: 85.14%, time taken to train/test: 23.55
Gamma: auto, C: 1,  Accuracy: 80.07%, time taken to train/test: 29.23
Gamma: auto, C: 10,  Accuracy: 89.19%, time taken to train/test: 32.69
Gamma: auto, C: 100,  Accuracy: 97.64%, time taken to train/test: 25.18
Gamma: auto, C: 1000,  Accuracy: 95.27%, time taken to train/test: 19.66
Gamma: scale, C: 0.01,  Accuracy: 81.76%, time taken to train/test: 34.18
Gamma: scale, C: 0.1,  Accuracy: 83.45%, time taken to train/test: 33.72
Gamma: scale, C: 1,  Accuracy: 97.97%, time taken to train/test: 25.51
Gamma: scale, C: 10,  Accuracy: 97.97%, time taken to train/test: 25.06
Gamma: scale, C: 100,  Accuracy: 90.88%, time taken to train/test: 23.57
Gamma: scale, C: 1000,  Accuracy: 96.96%, time taken to train/test: 24.37
Best parameters:  {'gamma': 'scale', 'C': 10, 'accuracy': 97.97}
```

Figure 5. Accuracy after running 5-fold cross validation

In order to improve the accuracy and to find more concrete hyperparameters, K-fold cross validation is used [8], figure#. Cross validation is a more expressive validation; k represents the number of data instances that are available. The dataset is split into 5 groups of training sets and validation sets (ratio of 80:20). Finally, the average accuracy from the 5 folds is taken. The svm is optimized by doing a grid search with several hyperparameters, Figure 5.

```
GammaList = ['auto', 'scale']
C_List = [0.01, 0.1, 1, 10, 100, 1000]
Best_SVM = {"gamma":None, "C":None, "accuracy":0}
for gamma in GammaList:
    for C in C_List:
        clf = NonLinear_SVM(train_features, train_labels, test_features, test_labels, gamma, C)
        accuracy = Use_SVM(clf, img)

        if accuracy > MIN_ACCURACY:
            print(f"Gamma: {gamma}, C: {C},  Accuracy: {round(accuracy, 2)}%")
        if accuracy > Best_SVM["accuracy"]:
            Best_SVM["gamma"] = gamma
            Best_SVM["C"] = C
            Best_SVM["accuracy"] = round(accuracy, 2)
print("Best parameters: ", Best_SVM)
```

Figure 6. Hyper-parameter tuning code snippet

The results after performing the 5-fold cross validation can be seen in figure#. The optimal values for C and scale remained the same. Accuracy improved slightly to 97.97%. The Accuracy for C set to 1 and C set to 10 was the same, C as 10 was finally selected as it had a slightly better run time.

```
def HoG(image):
  img_size = (128, 64) # h x w
  cell_size = (2, 2) # h x w in pixels
  block_size = (8, 8)  # h x w in cells
  nbins = 9  # number of orientation bins
  # create HoG Object
  # winSize is the size of the image cropped to multiple of the cell size
  # all arguments should be given in terms of number of pixels
  hog = cv2.HOGDescriptor(_winSize=(img_size[1] // cell_size[1] * cell_size[1],
                                    img_size[0] // cell_size[0] * cell_size[0]),
                          _blockSize=(block_size[1] * cell_size[1],
                                      block_size[0] * cell_size[0]),
                          _blockStride=(cell_size[1], cell_size[0]),
                          _cellSize=(cell_size[1], cell_size[0]),
                          _nbins=nbins)

  features = []
  image = image*255
  features.append(hog.compute(image.astype(np.uint8)).reshape(1, -1))
  return features
```

Figure 7. HOG code snippet

Pyramid of images was performed to count the number of people in an image. The images are taken, they are reshaped to different sizes, each time a lower resolution of image is taken. Sliding window is run on the images to detect the person.

Libraries used: Detectron2, numpy, os, cv2, random, sklearn, csv

Evaluation of Person Detector:

IoU is used to evaluate the accuracy of the object detector. In this case the ground truth is the cropped samples from detectron2. The ground truth is compared to the prediction provided by SVM. SVM is fed with images containing only people to check for misclassification errors. This is found by first cropping positive samples from the detectron and then feeding then to svm. It is then checked if svm returns an accuracy less than 100% . An accuracy less than 100% will indicate that svm is failing to detect.

Discussion:

One of the shortcomings of LBP is that it cannot handle changes in illumination with much success. The results could be improved if a combination of HOG and LBPHF is used. This will greatly improve detection performance as can be seen in [5] where the HOG-LBPHF combination greatly outperforms HOG_LBP.

Use a sliding window and non maximum suppression to crop the images and detect the person without detectron.

Describe any difficulties you faced in the project.

Difficulty was faced in regard to RAM allocation. After cropping, 3567 positive images were obtained. A memory error was encountered where 11.8 GigaBytes of ram was required to handle 3847 images. Due to insufficient RAM our system crashed a few times.

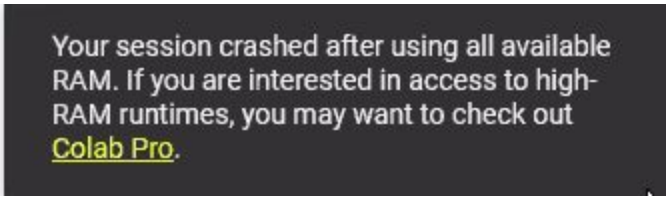

Figure 8. RAM problem encountered during the project

Changes the number of bins from 4 to 1 and the cell size from 8x8 to 4x4. The new configuration was successful. The previous configuration with bin number as 4 and cell size as 8x8 introduced crashes.

Another difficulty was with the dataset. After cropping out positive and negative samples, the number of positive examples were much greater than the number of negative samples. This could have led to over counting.

HOG has been utilized in this project, Haar-like cascade classifier is another object detection approach that could have been utilized. In this method Haar like features are extracted from images and used as the attributes. Haar like features can be seen in figure 5 [7]. Image subsections are categorized by subtracting the value of the white area from the value of the black area [7]. From [7] it can be inferred that all of the classifiers perform satisfactorily. However the results of HOG were slightly inferior to Haar LBP approaches. Combining HOG with LBP improves performance as discussed earlier. It can be concluded that a wide variety of classifiers can be utilized to detect people to reasonable success.

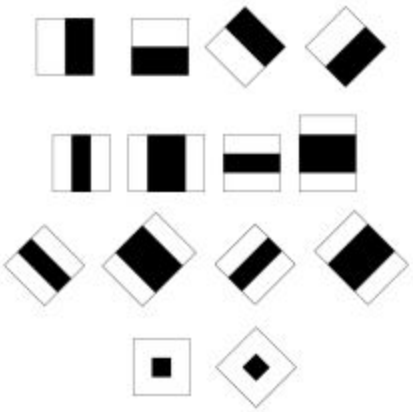

Figure 9. Haar-like features

A constraint of HOG features is that it performs poorly when the background is noisy [5]. LBP in conjunction with HoG could have been utilized to yield better results. LPB is used to detect textures in order to overcome this shortcoming. LBP has been widely used in human detection and achieves good performance when combined with HOG [5].

Local binary patterns (LBP) is used to describe the texture of the images. LBP is extracted in a symmetric circular neighbourhood by comparing each pixel to its neighbour [5].

$$LBP(P,R) = \sum_{l=1}^{p-1} u(\mathrm{g}_l - \mathrm{g}_c)2^l$$

Here P is the number of neighbours and R is the radius to build the circular binary pattern, gl is the intensity of the neighbouring pixel and gc is the intensity of the center pixel [5]. u(x) is a step function where u(x) =1 when the difference is positive and u(x) = 0 otherwise. A histogram is created where there are bins for each uniform pattern. All non-uniform patterns are assigned to a single bin [5].

```
def LBP(images, radius):
    P = 8 * radius
    R = radius
    features = []
    eps = 1e-7
    for img in images:
        lbp = feature.local_binary_pattern(img, P, R)
        (hist, _) = np.histogram(lbp.ravel(),bins=np.arange(0, P + 3),range=(0, P + 2)
        # normalize the histogram
        hist = hist.astype("float")
        hist /= (hist.sum() + eps)
        if np.isnan(hist.any()):
            print("nan")

        features.append(hist)
    features=np.array(features)
    return features
```

Figure 10. LBP code snippet

Another improvement could be to use sliding windows instead of taking the images from detectron.

References

[1]Detectron2: A PyTorch-based modular object detection library
[2]https://towardsdatascience.com/object-detection-in-6-steps-using-detectron2-705b92575578
[3]S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.
[4]M. Bertozzi, A. Broggi, M. Del Rose, M. Felisa, A. Rakotomamonjy and F. Suard, "A Pedestrian Detector Using Histograms of Oriented Gradients and a Support Vector Machine Classifier," 2007 IEEE Intelligent Transportation Systems Conference, Seattle, WA, 2007, pp. 143-148, doi: 10.1109/ITSC.2007.4357692.
[5]Aili Wang, Shiyu Dai, Mingji Yang and Y. Iwahori, "A novel human detection algorithm combining HOG with LBP histogram Fourier," 2015 10th International Conference on Communications and Networking in China (ChinaCom), Shanghai, 2015, pp. 793-797, doi: 10.1109/CHINACOM.2015.7498045.
[6]N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.
[7] Cruz, J., et al. “A Comparison of Haar-like, LBP and HOG Approaches to Concrete and Asphalt Runway Detection in High Resolution Imagery.” *Journal of Computational Interdisciplinary Sciences*, vol. 6, no. 3, 2016, doi:10.6062/jcis.2015.06.03.0101.
[8 [S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, 2016, pp. 78-83, doi: 10.1109/IACC.2016.25.