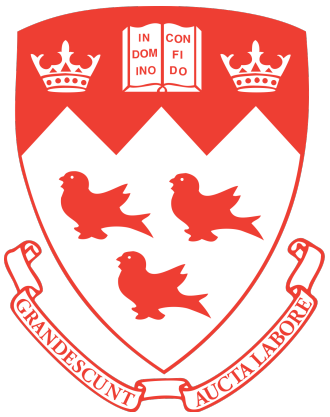

ECSE 415 - Introduction to Computer Vision
Final Project

Matthew Lesko-Krleza
260692352

Tristan Saumure Toupin
260688073

Alexander Harris
260688155

Filip Bernevec
260689062

Abbas Yadollahi
260680343

December 2, 2018

Contents

1 Introduction

Identification of objects is a long-standing challenge in the field of machine learning and computer vision. Object classification and localization are two well known and mastered tasks by computers. In this report, the students accomplish both tasks using techniques learned in their ECSE-415 class. The data set used is the MIO-TCD, consisting of a total 786,702 images acquired at different times of the day and different periods of the year by thousands of traffic cameras deployed all over Canada and the United States. Algorithms such as Support Vector Machine for classification and YOLO for localization/classification are exploited to achieve impressive results. In the bonus section of this project, the students achieved a near state-of-art classifier with a simple, yet thorough technique on a small percentage of the data.

2 Classification

2.1 Description of Contents of Data Set

2.1.1 Number of Samples

The data set used is a subset of the entire MIO-TCD-Classification data set. Given the resources at the students' disposal, they wouldn't have been able to use the entirety of the data set for analysis and machine learning training. The students used 46017 images from the data set. Out of the subset of images taken from the whole data set, the amount of images for each of the 11 categories are:

1. 5000 background images;
2. 2284 bicycle images;
3. 1751 non-motorized vehicle images;
4. 5000 bus images;
5. 5000 single unit truck images;
6. 5000 pedestrian images;
7. 1982 motorcycle images;
8. 5000 work van images;
9. 5000 car images;
10. 5000 pickup truck images;
11. 5000 articulated truck images;

The students wanted to import a large number of images and have the most equal distribution of images from each category. The rational behind this approach can be explained by the exhaustive preprocessing required and the efficiency affected by such imbalanced data sets. However, there are categories with significantly less images, that was just the way the images were provided by the data set.

The students also decided to train SVM models in four different ways. Two SVM models, one trained on plain images, and the other on HoG extracted images, were trained with the 10-Fold Cross Validation approach. For the sake of the report, the report calls these two models the "GridSearch" models. The name comes from the fact that they were generated from the use of scikit-learn's GridSearch function. The best parameters from the two 10-Fold Cross Validation approaches were then used to train two other SVM models, one with plain images, and the other on HoG extracted images. These two last SVM models were trained on a larger subset of the data set. The rationale behind this decision is that the students wanted to experiment on training with a small amount of images, and a large amount of images. However, scikit-learn's GridSearch function takes far too long to compute when trained on large subsets of images. Hence the reason why two other SVM models were computed last.

2.1.2 Image Sizes

Furthermore, the images' sizes weren't of a common aspect ratio, and they were too large to use for machine learning training. For the SVM model to be trained, each sample fed to the model must be of the same size. Hence, the students implemented a solution to resize and pad the images so that they would all be of the same aspect ratio and wouldn't be warped from resizing. Their solution, imported directly from their jupyter notebook file, can be seen in the code snippet below:

```
    def pad_resize_images(img_list, output_size):
    """
    @brief Resize each image within a list to a given shape while also keeping its aspect ratio by the use of padding
    @param img_list The list of images to resize
    @param output_size The shape of the output images are output_size x output_size
    """
    BLACK = 0
    result = np.empty_like(img_list)

    for i, img in enumerate(img_list):

        height, width = img.shape
        ratio = float(output_size) / max([height, width])
        height_new, width_new = tuple([int(val * ratio) for val in (height, width)])
        img_resized = cv2.resize(img, (height_new, width_new))

        height_adjust = output_size - height_new
        width_adjust = output_size - width_new

        top = math.ceil(height_adjust / 2)
        bot = height_adjust - top
        left = math.ceil(width_adjust / 2)
        right = width_adjust - left

        result[i] = cv2.resize(
            cv2.copyMakeBorder(img_resized, top, bot, left, right, cv2.BORDER_CONSTANT, value=BLACK),
            (output_size, output_size))
    return result
```

Further down in their notebook, the students set the images to a common size of 128 x 128 pixels and flattened each image matrix to a vector, as seen below. In scikit-learn's[1] implementation of the SVM, the samples fed to the model fitting must be of a vector shape, hence the reason why each resized picture is also flattened.

```
    size = 128
    padded_images = np.array([x.flatten() for x in pad_resize_images(classification_images, size)])
    padded_images_hog = np.array([x.flatten() for x in pad_resize_images(hog_images, size)])
```

2.2 Feature Extraction Method

The students developed two different SVM models: one trained on a large amount of images without any extracted features and one trained on a smaller amount of images with their HoG features extracted. The students used a feature extraction method they were familiar with from a previous assignment. They also believed that HoG would be helpful for classifying vehicles. The rationale behind such a design choice, is that HoG feature extraction has been used previously to identify pedestrians. The students desired to experiment with HoG feature extraction and vehicle classification to determine if this method could actually be used in the real-world for such an application.
The reasoning behind the training of a model on data without any features extracted is that the students wanted to experiment with and analyze the performance of cross-validation and training models without extracted features.

2.3 Feature Extraction Parameters

The parameters used can be seen in the code snippet below taken directly from the notebook:

```
1     fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
2                         cells_per_block=(1, 1), visualise=True)
```

The students used scikit-image's[2] implementation of the HoG feature description extractor. The students chose parameters that they were familiar with. The pixels_per_cell parameter is set to a small number, 16 by 16. This was chosen because then there would be a larger amount of detail if less pixels were chosen for each cell. There would therefore be a larger number of cells and greater detail in HoG bin orientations for the whole image. They decided to use 1 cell per block for convenience as it would mean one block is one cell.

2.4 Cross-Validation Method

The Support Vector Machine algorithms were fined tuned to obtained the best results over a total of 12 candidates. Each candidate had a total of 3 different parameters: C, *gamma* and the *kernel type*. Each of these 12 instances of the algorithms were cross validated over 10 folds. That means that a grand-total of 120 fits were accomplished. The parameters for best average accuracy over these folds was kept and used for training the two last SVM models.
The folds were determined at random over 1200 training images for the plain SVM and 1200 images for the HoG feature extracted classifiers. Each instances of the algorithm was compose of a sequence of unique hyper-parameters. The students tried two types of kernels:

1. Linear;
2. RBF.

For both types, the students varied the *gamma* values between 0.0001 and 0.001 and the C values between 1, 10 and 100. In order to speed up the 120 fits, the students ran the algorithms in parallel.

2.5 Evaluation of Performance and Interpretation of Results

2.5.1 Performance Over 10-Fold Cross Validation Approach

One can see in Figure 1 that the average accuracy for plain images is of 41.67 percent. Furthermore, a quick analysis of Figure 2 reveals that the average accuracy of the HoG feature SVM is of 30.33 percent. Their explained variance scores are of -0.163 and of -0.495 respectively as seen in the Figures 1 and 2. One can see in Figures 3 and 4 that the average precision and recall values are consistent with the average accuracy. For the plain image SVM, the average accuracy is of 0.400 and its average precision and recall values are of 0.41 and 0.42 respectively. Similarly, for the SVM with HoG feature extracted images, the average accuracy is of 0.303 and the average precision and recall values are both of 0.30. These values are not representative of the overall data set of 600,000 images. The overall data set's distribution of images vary greatly from one category to another. However, the subset of images chosen for the experiment has a smaller variance of images between categories as opposed to the overall data set. The students attempted to import 5000 images from each category, and almost achieved an equal distribution of images across all categories. In a situation where there's a large variance in number of images for each category, the precision and recall would vary greatly from the model's accuracy. Conversely, in a situation where there's a small variance in number of images for each category, the precision and recall would be similar to the model's accuracy. In the case of this experiment, the precision and recall are similar to the accuracy because the students are close to having an equal distribution of images across categories.

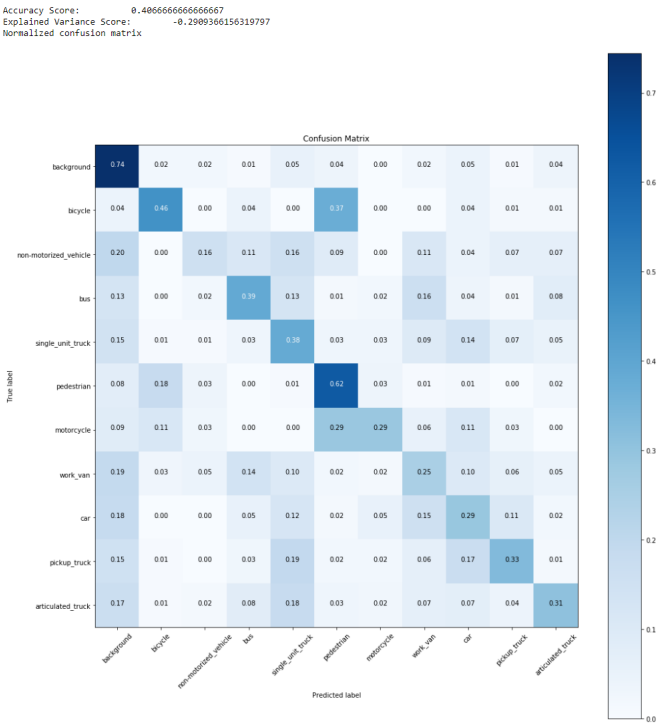

Figure 1: Normalized Confusion matrix for Plain Images

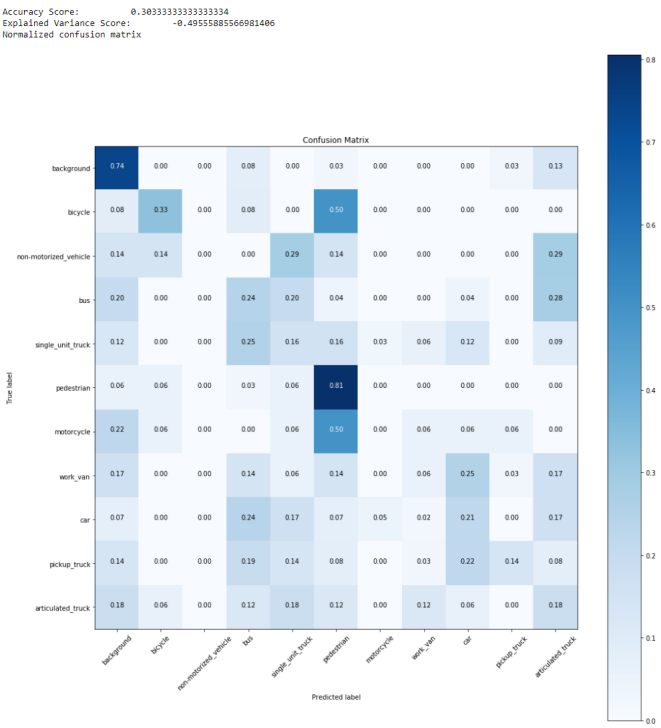

Figure 2: Normalized Confusion matrix for Images with HoG Features Extracted

The confusion matrix in Figure 1 is averaged over the 10 fold validation sets. Our classifier for plain images shows good accuracy on the pedestrian and background, with 62% and 74% accuracy

for both respectively. A common mistake our classifier made was to confuse the pedestrian with bicycle since most images containing bicycles also had humans on them. On the other hand, the non-motorized vehicle performances are terrible considering its poor representation in the data set.

The second classifier with HoG features extracted over a 10 fold is shown in Figure 2. There are similar patterns as the previous method. For instance, this classifier is best at classifying the pedestrians and the background images. The non-motorized vehicle is not classified correctly, just like the previous model. The pedestrian is also often confused with either the motorcycle or bicycle classes. A noticeable difference with the HoG based SVM is that it's almost incapable in correctly classifying pickup trucks, articulated trucks, work vans and motorcycles.

	precision	recall	f1-score	support
0	0.40	0.74	0.52	133
1	0.44	0.46	0.45	67
2	0.25	0.16	0.19	45
3	0.50	0.39	0.44	137
4	0.29	0.38	0.33	117
5	0.59	0.62	0.61	143
6	0.27	0.29	0.28	35
7	0.29	0.25	0.27	131
8	0.31	0.29	0.30	131
9	0.52	0.33	0.40	140
10	0.48	0.31	0.37	121
avg / total	0.42	0.41	0.40	1200

Figure 3: Average Precision and Recall over 10 Fold-Cross Validations

	precision	recall	f1-score	support
0	0.45	0.74	0.56	38
1	0.44	0.33	0.38	12
2	0.00	0.00	0.00	7
3	0.14	0.24	0.18	25
4	0.16	0.16	0.16	32
5	0.45	0.81	0.57	36
6	0.00	0.00	0.00	18
7	0.22	0.06	0.09	36
8	0.27	0.21	0.24	42
9	0.62	0.14	0.22	37
10	0.08	0.18	0.11	17
avg / total	0.30	0.30	0.26	300

Figure 4: Average Precision and Recall over 10 Fold-Cross Validations with HoG Features Extracted

2.5.2 Performance with a Non GridSearch Approach

The best accuracy was achieved on the plain Support Vector Machine algorithms, as seen in Figure 5. The final accuracy score was 88.2116% with a linear kernel, a C value of 1 and a *gamma* value of 0.001. The model performs surprisingly better as the number of samples (36813) is much higher than the dimension (16384).

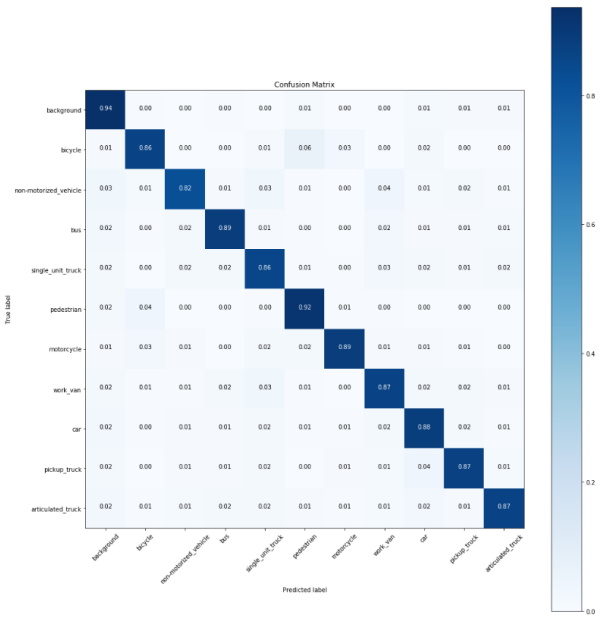

Figure 5: Normalized Confusion matrix for Plain Images

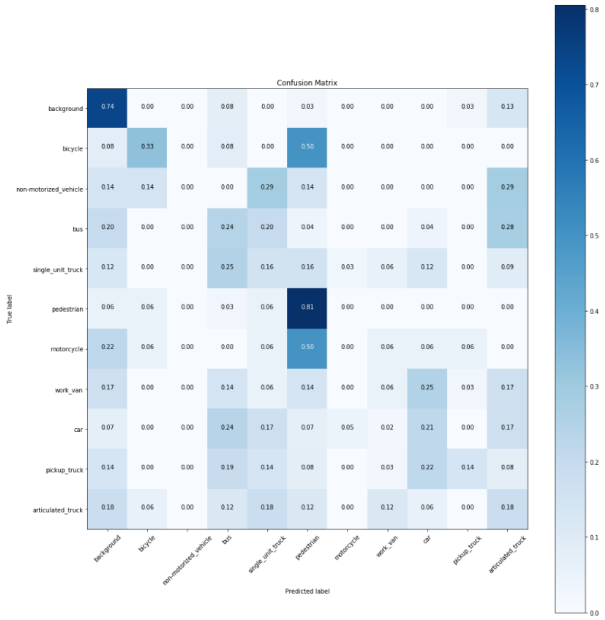

Figure 6: Normalized Confusion matrix for HoG Feature Extracted Images

As for the HoG extracted feature support vector machine classifier, its maximal accuracy reached exactly 32.0%. Just like the non-HoG extracted feature SVM, this classifier performed better with a linear kernel. This is not surprising since the number of training images (1200) is much lower than the dimension of the input vector (16384). The hyper-parameters C and *gamma* with respective values of 1 and 0.001.

The python environment required the following python packages: os, cv2, matplotlib, sklearn, skimage, numpy, math, random, and joblib. The code can be found online[3].

3 Localization and Classification

3.1 Description of Contents of Dataset

The data set used for the localization and classification task for the project is the MIO-TCD-Localization dataset. Originally, it has 137,743 high resolution images of urban traffic scenarios, each containing one or more objects of interest from the 11 possible labels:

- Articulated truck
- Bicycle
- Bus
- Car
- Motorcycle
- Motorized vehicle
- Non-motorized vehicle
- Pedestrian
- Pickup truck
- Single unit truck
- Work van

For the train portion of the set, a csv file contains ground truth data describing the x,y coordinates of two opposite corners of a bounding box and the corresponding label of each object in the images. The test set however contains no ground truth data. This means that the validation set will be a subset of the training set in order to compare the outputs of the localizer to the ground truth.

3.2 Localization Method

3.2.1 Architecture

For localization, we used a custom-trained version of the popular object detection algorithm YOLOv3. YOLO's advantage lies in its relative speed compared to other state-of-the-art detection algorithms, as well as the fact that it combines both classification and detection into a single evaluation, meaning it uses a single neural network to predict both bounding boxes and class probabilities. This results in a simpler architecture and faster computation speed. Therefore we chose to use YOLO for our localization implementation due to its high accuracy and good performance. [4] YOLOv3 uses the darknet-53 network [5], which consists of only 53 convolutional layers.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 7: Darknet-53 Architecture [4]

3.2.2 Training

When it came to training the MIO-TCD localization dataset on this network, there were a few things we had to do first. In order to train on a custom dataset, darknet needs some configuring. We needed to first modify the default YOLOv3 config file with our number of classes (11), filters and some other parameters. We then had to create a names file containing the names of our classes, as well as a data file containing the paths to our training and validation sets, names file, as well as number of classes and location to save the weights.

Then we had to modify the ground truth data given in the MIO-TCD dataset to a format that YOLO recognizes. The bounding box coordinates in the original dataset are given as two x,y coordinates representing two opposing corners of the bounding box, along with the image number, all in a single CSV file. Meanwhile YOLO's bounding box coordinates are the relative x,y coordinate of the center, as well as the relative height and width of the bounding box. There is a separate text file for each image containing the coordinates of the bounding boxes along with the class index. We also need a text file containing the paths to all the images.

In order to do this for the entire dataset of 110,000 images, we wrote a Python script to process each image, compute the relative coordinates and write them to a text file. Since we had no ground truth data for the test portion of the original dataset, we decided to 80/20 split the original training set into a training and validation set. We wrote the paths of the corresponding images to train and test text files using the same script. The network was trained using darknet on an NVIDIA GTX

1080 for over 20,000 iterations, we eventually stopped when the average loss starting levelling out around 20,000 iterations due to time constraints.

3.3 Cross-Validation Approach

In order to evaluate how well our custom YOLO (You Only Learn Once) model generalizes results to an independent data set, the training images were split in a 80/20 proportion. Hence, out of the 110,000 images that had ground truths bounding boxes associated to them, around 88,000 were used to train the model and approximately 22,000 were used as the validation set. After the model was successfully trained, every image in the validation set was passed in the trained model to generate the predicted bounding boxes. However, because those images come from the original training images, they have the coordinates for the ground truth bounding boxes. The predicted and ground truth bounding boxes are used to output meaningful results like DICE coefficients, discussed in the following section.

3.4 Evaluation of Localization Performance and Interpretation of Results

To properly evaluate the performance of our YOLO model and algorithm, we had to compute the DICE coefficients of our predicted bounding boxes against the real boxes from the ground truth data. We also needed to evaluate the precision and recall rate of our model.

3.4.1 DICE Coefficient Computation

To calculate the DICE coefficients, the precision and the recall, we needed to compare the true negatives and positives and the false negatives and positives. This gives us the following equations to be computed.

$$DICE = \frac{2 * TP}{2 * TP + FN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

3.4.2 DICE Coefficients Distribution

In order to calculate these coefficients, we resorted to the already existing algorithm that darknet provides us, which allowed us to compute the average DICE coefficient, the precision and recall across our custom testing data set (20% of the training data). Across the 48 hours of training, we saved the weights after every several iterations and used these to validate our model. Three important key points during the training were after 7,000, 18,000 and after 20,000 iterations.

After 7,000 iterations, we see that the DICE coefficient, also called the intersection of union (IoU), reaches an average of 63.45% across all test images. The precision and recall rates reach relatively high numbers, coming in at 82% and 81% respectively, although as previously mentioned, to properly train our YOLO model, we should iterate approximately 22,000 times.

```
detections_count = 364199, unique_truth_count = 70406
class_id = 0, name = articulated_truck,          ap = 71.86 %
class_id = 1, name = bicycle,    ap = 45.74 %
class_id = 2, name = bus,        ap = 83.35 %
class_id = 3, name = car,        ap = 87.19 %
class_id = 4, name = motorcycle,         ap = 65.10 %
class_id = 5, name = motorized_vehicle   ,         ap = 34.89 %
class_id = 6, name = non-motorized_vehicle,       ap = 16.28 %
class_id = 7, name = pedestrian,         ap = 29.91 %
class_id = 8, name = pickup_truck,       ap = 83.61 %
class_id = 9, name = single_unit_truck,          ap = 42.02 %
class_id = 10, name = work_van,          ap = 57.67 %
 for thresh = 0.25, precision = 0.78, recall = 0.80, F1-score = 0.79
 for thresh = 0.25, TP = 56193, FP = 15658, FN = 14213, average IoU = 63.45 %

 mean average precision (mAP) = 0.561471, or 56.15 %
Total Detection Time: 438.000000 Seconds
```

Figure 8: DICE, Precision and Recall Coefficients - 7 000 Iterations

After 18,000 iterations, we saw a net increase of almost 8% for the average IoU ratio, peaking at 70.04% across all test images. The increase can clearly be seen from the average DICE coefficients in each of the 11 classes in Figure 9. The precision and recall ratios also moved, although not as substantially as the DICE coefficient, reaching 84% and 80% respectively. As we can see, the recall rate decreased a few ticks, which can be attributed to the large decrease in false positives that we see over using the weights with 7,000 iterations. We thought the ratios would continue increasing since we had not yet hit our 22,000 iterations goal.

```
detections_count = 226782, unique_truth_count = 70406
class_id = 0, name = articulated_truck,          ap = 81.13 %
class_id = 1, name = bicycle,    ap = 67.51 %
class_id = 2, name = bus,        ap = 89.88 %
class_id = 3, name = car,        ap = 87.61 %
class_id = 4, name = motorcycle,         ap = 84.59 %
class_id = 5, name = motorized_vehicle   ,         ap = 37.76 %
class_id = 6, name = non-motorized_vehicle,       ap = 37.19 %
class_id = 7, name = pedestrian,         ap = 44.74 %
class_id = 8, name = pickup_truck,       ap = 87.07 %
class_id = 9, name = single_unit_truck,          ap = 56.36 %
class_id = 10, name = work_van,          ap = 67.03 %
 for thresh = 0.25, precision = 0.84, recall = 0.80, F1-score = 0.82
 for thresh = 0.25, TP = 56656, FP = 10679, FN = 13750, average IoU = 70.04 %

 mean average precision (mAP) = 0.673520, or 67.35 %
Total Detection Time: 442.000000 Seconds
```

Figure 9: DICE, Precision and Recall Coefficients - 18 000 Iterations

Finally, after 20,000 iterations, we hit a stop. The DICE coefficient actually decreased by a full 6%, hitting an average of 64.60%. When looking at the individual class DICE coefficients, we see that they fluctuate very little, except for the pedestrian class which sees quite a large increase of 9%. At this point we started to run out of time, and we also believed that we had begun over-training our model. The fact that the precision actually reduced by 8%, which is large amount by any standard, confirmed our beliefs.

```
detections_count = 283916, unique_truth_count = 70406
class_id = 0, name = articulated_truck,          ap = 77.04 %
class_id = 1, name = bicycle,    ap = 72.00 %
class_id = 2, name = bus,        ap = 75.69 %
class_id = 3, name = car,        ap = 88.06 %
class_id = 4, name = motorcycle,         ap = 83.02 %
class_id = 5, name = motorized_vehicle   ,         ap = 39.06 %
class_id = 6, name = non-motorized_vehicle,      ap = 40.16 %
class_id = 7, name = pedestrian,         ap = 44.39 %
class_id = 8, name = pickup_truck,       ap = 87.19 %
class_id = 9, name = single_unit_truck,          ap = 61.26 %
class_id = 10, name = work_van,          ap = 68.45 %
 for thresh = 0.25, precision = 0.78, recall = 0.84, F1-score = 0.81
 for thresh = 0.25, TP = 59222, FP = 16776, FN = 11184, average IoU = 64.60 %

 mean average precision (mAP) = 0.669369, or 66.94 %
Total Detection Time: 775.000000 Seconds
```

Figure 10: DICE, Precision and Recall Coefficients - 20 000 Iterations

As a result, we used the weights with 18,000 iterations when it came time to visualize the localization and classification results. As can be seen from the boxes surrounding the vehicles in the images of Figure 11, our localizer does a good job at detected the cars both in the foreground and the backround of the image. It is also able to detect vehicles when they are partially cutout of the image, something that could have proved troublesome using a sliding window algorithm.

Figure 11: Images 00100539.jpg (left) and 00107294.jpg (right)

3.4.3 Classifier Evaluation

When looking at the classifier from the classification section, the SVM accuracy was 41.67 percent. The SVM was trained using 46,017 images from the data set. In contrast, the YOLO model had an accuracy 67.35 percent, and was trained using 88,000 images. There is a difference of 25.68 percent in the accuracy when comparing the two techniques. The reason being is that first of all, the YOLO algorithm was trained using around twice as more images as the SVM algorithm. Second of all, the YOLO technique is a state of the art neural network approach that is proven to have more accurate results than traditional machine learning techniques like SVM. The YOLO approach has 2 possible points of inducing errors as it is localizing the objects and also classifying them. However, it yields better accuracy than the SVM implemented using plain images as the feature space.

When looking at the labels of both the classification and localization data set, the "background" label is not included in the latter. This label should not be included when evaluating the performance of the localizer as it will might influence the objects of interest and hence become a source of error to the accuracy for the model.

Figure 12: Images 00102224.jpg (left) and 00102790.jpg (right)

4 Deep Learning Bonus - Classifier

4.1 Architecture

The architecture of the model is conventional. It is inspired from the MobileNet neural network[6] which had impressive results on the ImageNet[7] data set. This model is composed of a sequence of three convolutions blocks. One convolution block has two layers of 2 dimensional convolution with identical kernel size and number of filters. The activation layer on each convolution layer is relu for its good performances and faster computations than other activation layers such as tanh[8]. All convolution layers are followed by a batch normalization. This technique improve the general performance of the model by denoising the image and also speed up the process [9]. We add at the end of each convolutions block a max pooling with a squared pool size of two. To generalize the performance of the model and prevent over fitting[10].

The convolution blocks are implemented in keras[11]. They have the following number of filters: 32, 64 and 128 pixels. The kernels all have the same squared dimension of 3. The output of this sequence is then flattened and fed into a densely-fully-connected layer with 512 perceptions. Dropout is applied to finally make a prediction over the eleven classes.

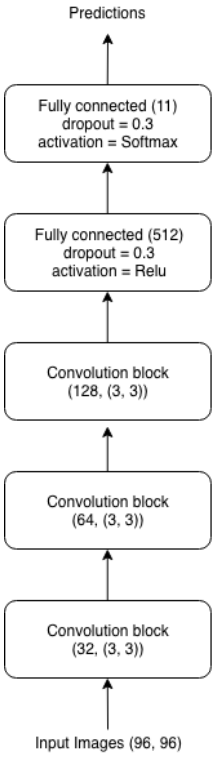

Figure 13: Architecture of the Deep-Learning Model

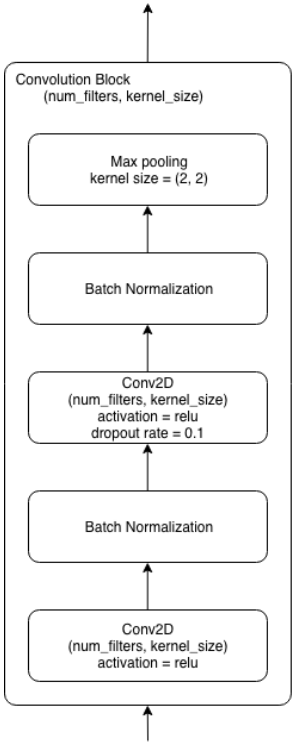

Figure 14: Modular Architecture of a Convolution Block

Nesterov Adam optimizer was chosen for this neural network for its fast convergence properties when combined with dropout[12]. This learning rate was originally set to 0.001 but we used a learning rate decay when a plateau was reached on the validation categorical accuracy to speed up the process and boost the performance[13].

4.2 Training

The model was trained with only 25% of the total classification data set which is equal to 129787 images. All the images were pre-process in the same matter. They were first re-sized to a maximum shape of 96 pixels on both axis while making sure to keep the dimensions of the images. Then a black padding was applied to obtain a square image of 96 pixels squared.

These 129787 images were divided into two sets a training set and a validation set. Considering the generous size, 90% of the original images were passed as training data and 10% to the validation data. This equals to 129787 and 12979 respectively in both sets. Our team then generated more training images by applying rotations and shifting the images width and height by 20%. We also applied horizontal flip to the images to generalize the predictions even more. We finally generated more data by zooming in and out by a factor of 0.1. The model was fed 256 images at a time.

4.3 Evaluation of Performance

As the official website of the MIO-TCD data set does, we prioritized overall accuracy. When training the categorical accuracy was monitored to prevent under and over fitting. Other performance metrics will be used to evaluate the model on the outputs. For instance, the accuracy of the model on each classes will be observed.

4.4 Validation

The model was trained over 40 epoch and was able to reach an overall accuracy of 93.8516%. This accuracy ranks the classifier in place 11 on the official website. It performs better than the AlexNet[8] which is well known to achieve impressive results on image classification tasks. These results were made on the original unbalanced dataset and ca be visualized in figure 15 and figure 16.

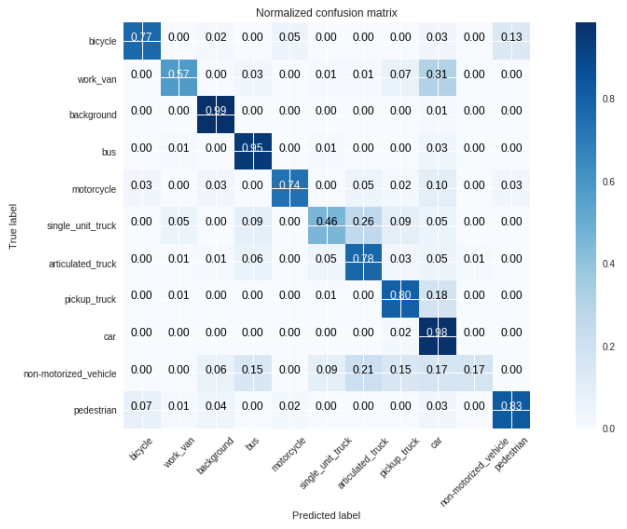

Figure 15: Normalized Confusion matrix with the Deep Learning Model

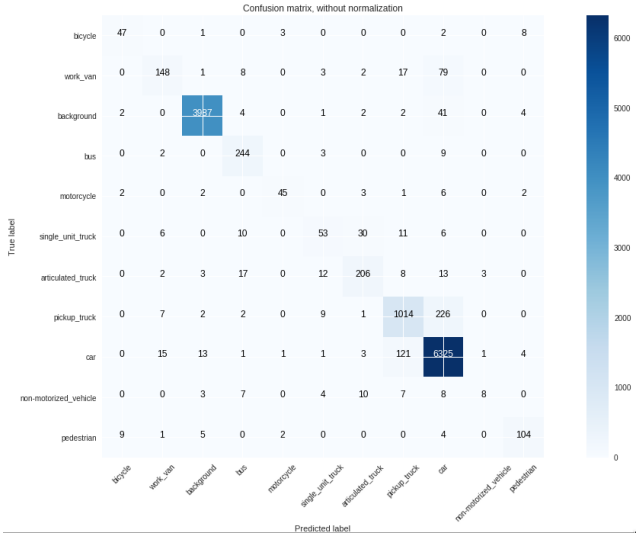

Figure 16: Confusion matrix with the Deep Learning Model

4.5 Comparison with Other Methods

This classifier performed much better than the SVM classifier. As mentioned previously, the SVM classifier achieved a overall 88.2116% on the data set. That being said the SVM classifier did not use the same unbalanced class the original data set had, which help its results. Therefore, it is same to assume that the deep learning solution performs better than any other method.

4.6 Code and Description of Environment

The code was executed in a Google collaborator notebook server. This server had a total of 13GB of RAM and 2vCPU at 2.2GHz each. In addition, the server provide a 33GB of disk space. The python environment required the following python packages: os, cv2, matplotlib, pyplot, numpy, math, itertools, keras and random. The code can be found online[3].

5 Conclusion

In conclusion, we were able to implement both classification and localization for this project. For classification, we implemented two different types of classifier, one using SVM which attained an accuracy of 41.67%, while our CNN-based implementation attained an overall accuracy of 93.85%. This shows how effective deep neural networks can be when it comes to classification, especially CNN which are particularly suited towards processing images.

For our localization implementation, we went straight to a neural network based solution. We used a version of the YOLOv3[4] object detection algorithm which we trained using darknet[5] on our custom dataset. After training for over 20,000 iterations over the course of around 48 hours, we were able to obtain an average DICE coefficient of 70.04% and a mean average precision (mAP) of 67.35%. We noted our network was particularly adept at localizing objects such as cars, buses, motorcycles, pickup trucks and articulated trucks, with average precision over 85%. However we did have issues with smaller objects such as pedestrians, with average precision below 50%. If we were to retrain the network, we could use a larger network resolution to combat this issue, as this can make it easier to detect smaller objects, however this would increase computation time by a fair amount.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and douard Duchesnay, "scikit-learn: Machine learning in python," 2011.

[2] S. van der Walt, J. L. Schnberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: Image processing in python," 2011.

[3] T. Saumure Toupin *et al.*, "Github repository - ecse 415 final project." https://github.com/tristantoupin/ECSE415-FinalProject, 2018.

[4] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[5] J. Redmon, "Darknet: Open source neural networks in c," 2018.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[9] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising," *CoRR*, vol. abs/1608.03981, 2016.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[11] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[13] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," tech. rep., Amherst, MA, USA, 1987.