On-Road Vehicle & Pedestrian Image Classifier

A Deep Learning model for classifying on-road entities using CNN with Progressive Resizing

Team – validation7407

Ang Boon Yew A0096966E
Kartik Chopra A0198483L
Karamjot Singh A0198470U

Institute of Systems Science,
National University of Singapore

Contents

1. Introduction

Image classification has become an integral part of computer vision applications. Classification of images can help to solve several problems and the algorithms such as Convolutional Neural Network (CNNs) help us to extract features from images that the human eye cannot distinguish. This project aims to classify various entities present on-road into 4 major categories:

1. Bike
2. Bus
3. Car
4. Pedestrian

The inspiration of this project came from a previous project in which traffic light times were calculated in real-time based on the number of vehicles present on the road. As an extension to that, the model developed in this project can help count individual number of each type of vehicle to provide an accurate measure for calculating traffic times. Apart from this, digging deeper into the bus category, several other problems can be solved such as the NUS buses waiting at checkpoints inside NUS. All this can be achieved if there are sensors such as cameras present that can capture images in real-time. However, the scope of this project only deals with images captured beforehand and classifying them into the abovementioned categories. In addition, the project also aims to find ways to create datasets for this problem.

This classifier takes input of images of size 300*300 pixels and classifies them into one of the 4 categories. The code for resizing is implemented in the test script.

2. Methodology

1. Dataset Creation

For the creation of image dataset, there are a few procedures that can be considered:

- Scraping images from a search engine
- Using an API service

Although using an API service like Flickr is suitable for such tasks, scraping method was chosen as it allows visualisation of data before downloading it. There are a few ways for scraping images from a search engine like Google:

- Python libraries like google_images_download
- Scraping from the webpage by search queries
- Automated tools like Chrome Extension - Image Downloader[1] & Fatkun Batch Download Image[2].

Creation of a dataset with all the above-mentioned methods was then attempted in this project. Using python library as mentioned above is preferable but is slower than the second approach. Hence in this assessment, we chose option 2 in order to gather our image data. This approach is inspired by PyImageSearch's[3]. Search queries for the relevant classes were executed on images.google.com giving the appropriate results. Results in which a majority of the images are not optimal, can be fine-tuned by executing a different query.

On the same page a small JavaScript snippet is executed in the browser's JavaScript console, which returns a TXT file containing the URL of every image present in the search results. URL's are then passed to a python script to sequentially download images from the source webpages. Renaming, size manipulations & applying filters like grey-scaling etc., can all be done at this step. Hence, this method is preferable over the other methods as it provides a simpler API usage. Images are downloaded to folders named with the class labels and allows them to be used with Keras' Data Generators methods.

The procedure to create dataset is described below.

- **Querying the search engine for the required class labels**

  The required class label can be queried on images.google.com (**Figure 1**). If majority of the results are satisfactory, proceed. Else a different query can be fired.

- **Fetching the URL's of resulting images through JavaScript**

  The TXT file containing the URL's of the above search results can be downloaded by executing the provided "js_console.js" file, step by step (**Figure 2**) in the JavaScript console of the web browser (under the "developer" menu-bar item of the browser). A file named "urls.txt" gets downloaded at this step.

- **Downloading the images through python script**

  This "urls.txt" file should be placed in the directory where the "download_images.py" script file is present. In terminal, execute the head command to see the format for providing the "urls.txt" file & destination directory path to the script (**Figure 3**). Download script should be executed in the following way:

  python3 **download_images.py** –urls **urls.txt** --output **path/to/dir**

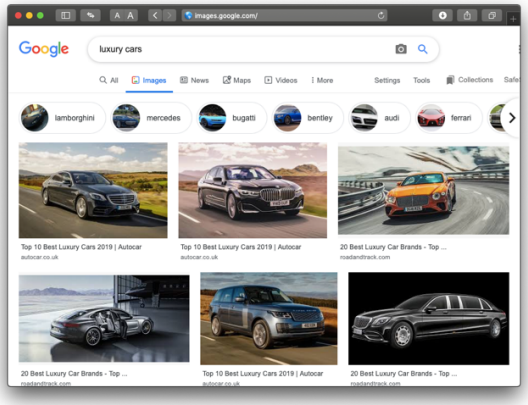


***Figure 1**. Google Image Search*

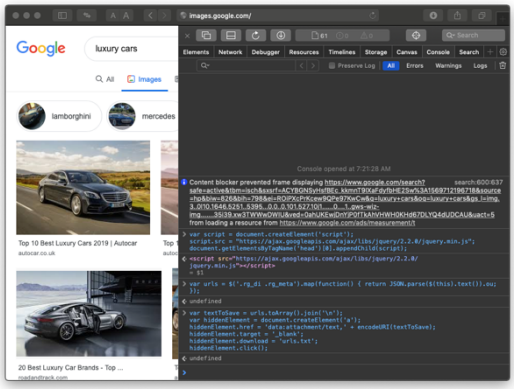


***Figure 2**. **js_console.js** snippet execution*

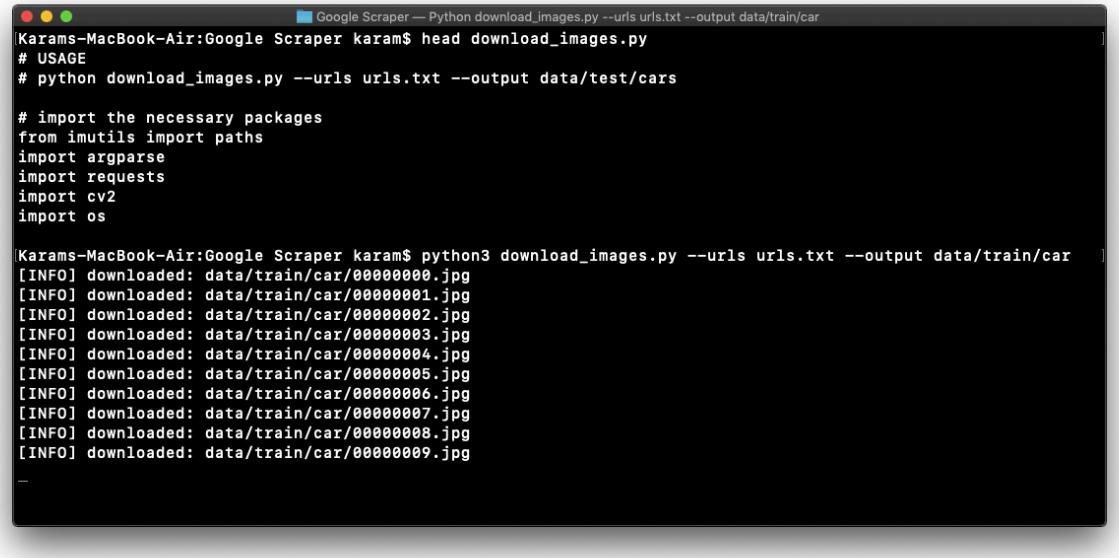


***Figure 3. download_images.py** script execution*

2. Data Cleaning

Once the downloaded images are available in respective class-labelled folders, the images were inspected and duplicate or irrelevant images were removed. Examples of irrelevant images to this dataset include animated images, which are not aligned with the real-world images of a particular class & hence are not applicable to the scenario.

Another example is the images with multiple entities of same/different classes in one image. Images with an alpha channel were also downloaded. These can either be ignored from the image-downloading script OR handled within the data-generator itself. The latter method is appropriate in this model building approach, where images with an alpha channel, colour-mode = 'rgba' get converted to the generic 3-channel colour-mode - 'rgb'. The files were then divided into train-val-test folders. Each class contains 1000 images,

of which 90% were kept in the training data and the rest were equally divided in validation & test datasets. **Figure 4** shows a subset of the data after cleaning.

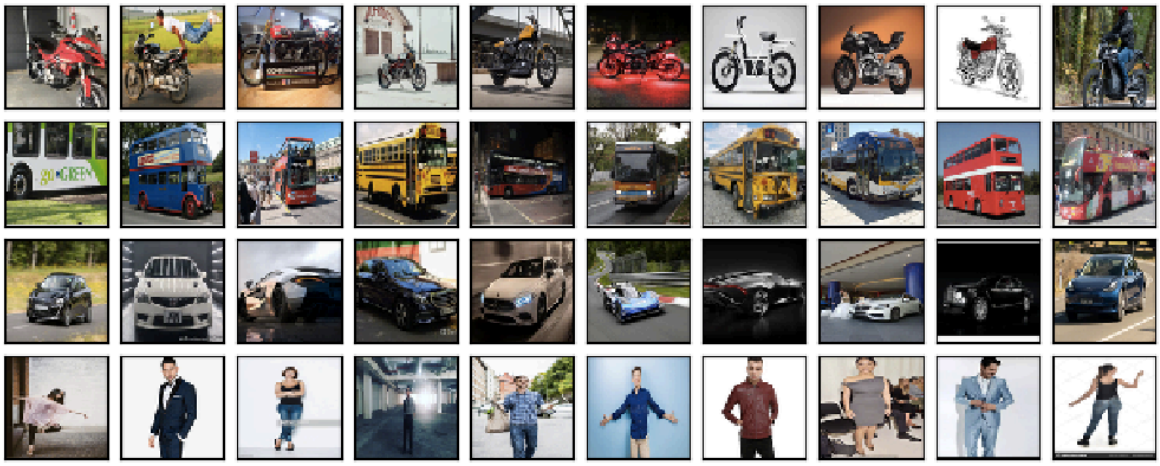


Figure 4.1 Training samples

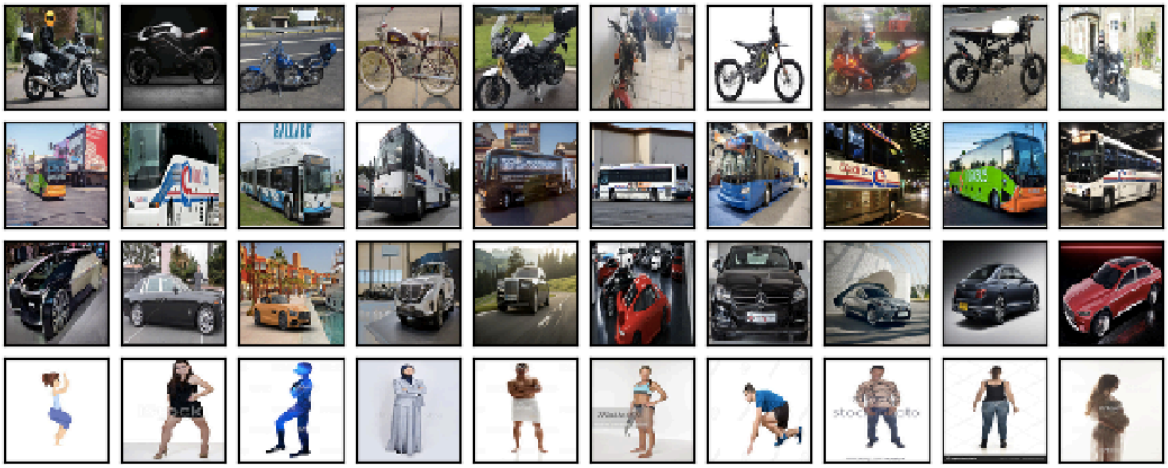
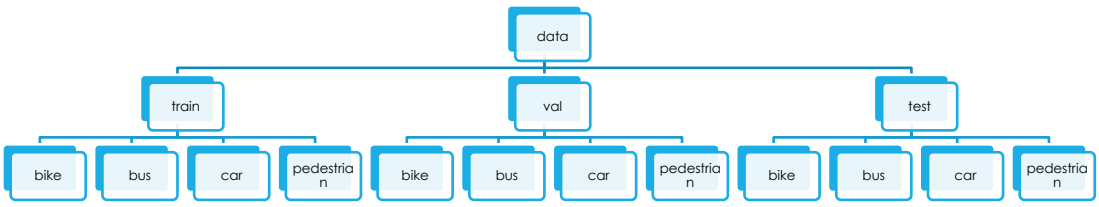


Figure 4.2 Validation samples

The cleaned data was then placed in hierarchical folder structure to use the Keras' Image-Data-Generators for model training, as explained in the next section. Hierarchy followed is represented below:



3. Training Approach

Since the number of images is relatively small, one of the ways to increase accuracy is to perform image augmentation. Images in our dataset were augmented using the powerful Keras' ImageDataGenerator. Each image is augmented with several version with different shear, flip & zoom value so that the model can learn more from each training image. **Fig. 5** shows one such example from the dataset. The augmented images are then fed into a simple deep learning model directly from the class directories using Keras' flow_from_directory module.

Data normalisation was carried out through rescaling all images in data-generator as the dataset contains images at their full resolution. Training a model on the full resolution images can be computationally expensive and will take a large time to complete. The target image size was chosen as 150 by 150 for training and testing the model. For transfer learning approach (as explained in section 3.5) the dimensions were set to 300 by 300 pixels. The same is done by providing these dimensions in the data-generator.

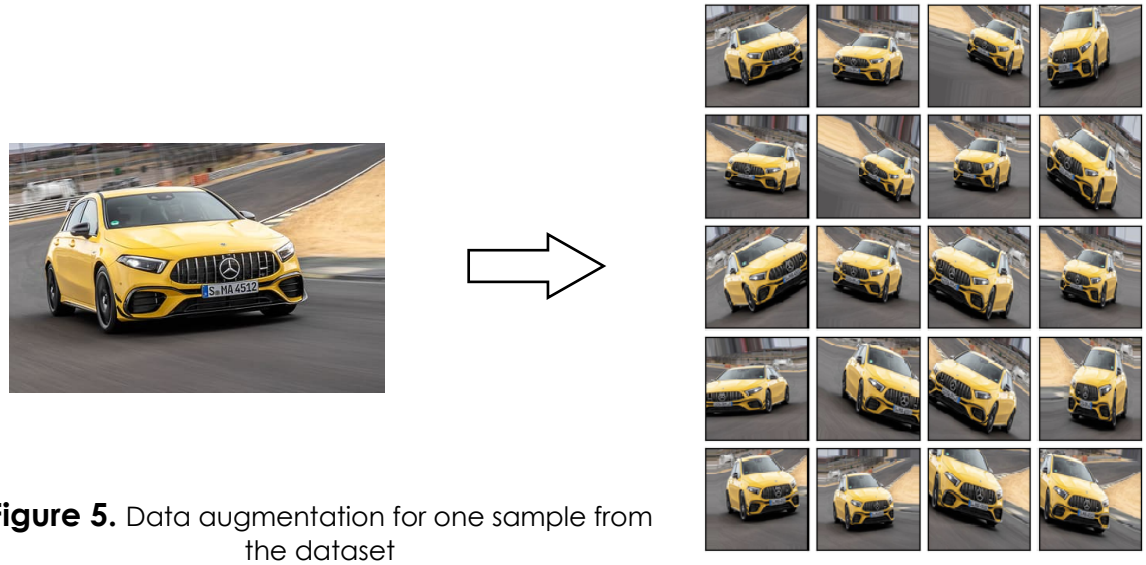


Figure 5. Data augmentation for one sample from the dataset

4. The Model

A simple CNN with various Conv2D and MaxPooling layers was designed to predict the class labels for this dataset. This model takes input from the data-generator which provides the input in a dimension of 150 by 150 pixels. Various approaches to achieve highest model accuracy were carried out, which will be explained further in section 3.5. The model summary is shown in **Fig. 6.**

Model: "Model-150"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
conv2d (Conv2D)	(None, 150, 150, 32)	896
batch_normalization (BatchNo	(None, 150, 150, 32)	128
activation (Activation)	(None, 150, 150, 32)	0
conv2d_1 (Conv2D)	(None, 150, 150, 32)	9248
activation_1 (Activation)	(None, 150, 150, 32)	0
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 32)	9248
batch_normalization_1 (Batch	(None, 75, 75, 32)	128
activation_2 (Activation)	(None, 75, 75, 32)	0
dropout (Dropout)	(None, 75, 75, 32)	0
conv2d_3 (Conv2D)	(None, 75, 75, 32)	9248
activation_3 (Activation)	(None, 75, 75, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 37, 37, 32)	0
conv2d_4 (Conv2D)	(None, 37, 37, 64)	18496
batch_normalization_2 (Batch	(None, 37, 37, 64)	256
activation_4 (Activation)	(None, 37, 37, 64)	0
dropout_1 (Dropout)	(None, 37, 37, 64)	0
conv2d_5 (Conv2D)	(None, 37, 37, 64)	36928
activation_5 (Activation)	(None, 37, 37, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 18, 18, 64)	0
conv2d_6 (Conv2D)	(None, 18, 18, 128)	73856
batch_normalization_3 (Batch	(None, 18, 18, 128)	512
activation_6 (Activation)	(None, 18, 18, 128)	0
dropout_2 (Dropout)	(None, 18, 18, 128)	0
conv2d_7 (Conv2D)	(None, 18, 18, 128)	147584
activation_7 (Activation)	(None, 18, 18, 128)	0
max_pooling2d_3 (MaxPooling2	(None, 9, 9, 128)	0
conv2d_8 (Conv2D)	(None, 9, 9, 256)	295168

```
_________________________________________________________________
activation_8 (Activation)    (None, 9, 9, 256)         0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
activation_9 (Activation)    (None, 4, 4, 512)         0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 64)                131136
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 1,913,252
Trainable params: 1,912,740
Non-trainable params: 512
```

Figure 6. Model Summary for "Model-150" – input dimensions (150, 150)

The above model, **Model-150** provides an accuracy of **94.25**% on the **validation** dataset. When tested on **test** dataset model-150 returns an accuracy **88.97**%. A subset of test dataset is shown in **Fig. 7**.

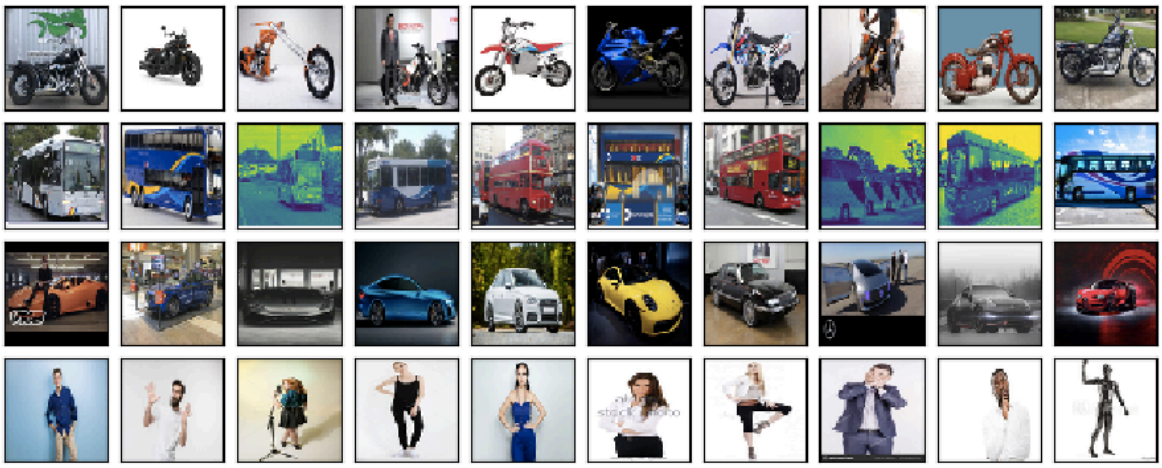


Figure 7. Subset of test samples

5. Model Tuning

To achieve the best possible training and test accuracies, a few methods were applied, including:

- changing the layers, addition and removal of different types
- changing the dropouts (to avoid overfitting)
- changing the input shape to a lower/higher size
- progressive resizing
- implementing shared layers

The first three approaches marginally improved the validation accuracy but however, did not improve the test accuracy. The misclassification rate was

similar to or more than model-150. An implementation with shared layers, **Fig. 8**, was also attempted where the model takes 2 input sizes, learns from both and predicts the class. The training accuracy in this implementation fluctuated below 40% due to which this approach was not chosen.

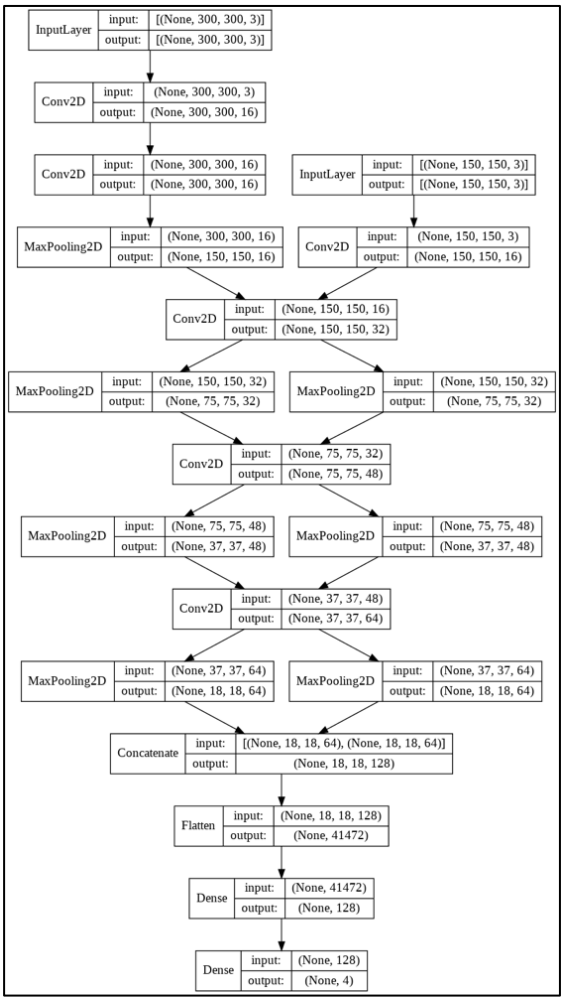


Figure 8. Model implemented with shared layers having inputs (150, 150) & (300, 300)

Progressive resizing was also used and was selected as the best approach for the dataset. The weights of model-150, trained on image size (150, 150) pixels with maximum validation accuracy were loaded to another model, **Model-300** which takes as inputs; images with an input size of (300, 300) pixels. Through this transfer learning approach, there is potential for improvements in accuracy as previously extracted features were available. This model took 1.5 times the training time of model-150.

The model summary is shown in **Fig. 9**. Model-300 is similar in terms of layers to model-150, but is tuned for taking a different input.

Model: "Model-300"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 300, 300, 3)]	0
conv2d_20 (Conv2D)	(None, 300, 300, 32)	896
batch_normalization_8 (Batch	(None, 300, 300, 32)	128
activation_20 (Activation)	(None, 300, 300, 32)	0
conv2d_21 (Conv2D)	(None, 300, 300, 32)	9248
activation_21 (Activation)	(None, 300, 300, 32)	0
max_pooling2d_12 (MaxPooling	(None, 150, 150, 32)	0
conv2d_22 (Conv2D)	(None, 150, 150, 32)	9248
batch_normalization_9 (Batch	(None, 150, 150, 32)	128
activation_22 (Activation)	(None, 150, 150, 32)	0
dropout_8 (Dropout)	(None, 150, 150, 32)	0
conv2d_23 (Conv2D)	(None, 150, 150, 32)	9248
activation_23 (Activation)	(None, 150, 150, 32)	0
max_pooling2d_13 (MaxPooling	(None, 75, 75, 32)	0
conv2d_24 (Conv2D)	(None, 75, 75, 64)	18496
batch_normalization_10 (Batc	(None, 75, 75, 64)	256
activation_24 (Activation)	(None, 75, 75, 64)	0
dropout_9 (Dropout)	(None, 75, 75, 64)	0
conv2d_25 (Conv2D)	(None, 75, 75, 64)	36928
activation_25 (Activation)	(None, 75, 75, 64)	0
max_pooling2d_14 (MaxPooling	(None, 37, 37, 64)	0
conv2d_26 (Conv2D)	(None, 37, 37, 128)	73856
batch_normalization_11 (Batc	(None, 37, 37, 128)	512
activation_26 (Activation)	(None, 37, 37, 128)	0
dropout_10 (Dropout)	(None, 37, 37, 128)	0
conv2d_27 (Conv2D)	(None, 37, 37, 128)	147584
activation_27 (Activation)	(None, 37, 37, 128)	0
max_pooling2d_15 (MaxPooling	(None, 18, 18, 128)	0
conv2d_28 (Conv2D)	(None, 18, 18, 256)	295168
activation_28 (Activation)	(None, 18, 18, 256)	0
max_pooling2d_16 (MaxPooling	(None, 9, 9, 256)	0
conv2d_29 (Conv2D)	(None, 9, 9, 512)	1180160
activation_29 (Activation)	(None, 9, 9, 512)	0
max_pooling2d_17 (MaxPooling	(None, 2, 2, 512)	0
flatten_2 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 64)	131136
dropout_11 (Dropout)	(None, 64)	0

```
dense_5 (Dense)              (None, 4)                 260
=================================================================
Total params: 1,913,252
Trainable params: 1,912,740
Non-trainable params: 512
```

Figure 9. Model Summary for "Model-300" – input dimensions (300, 300)

Model-300 provides a validation accuracy of 95.7% on the same data. Test accuracy increased from model-150's 88% to 97.79% with misclassifications happening in only 1 class.

***Table 1**. Summary of results*

Model-150

Best accuracy (on validation dataset): 94.25%

acc	0.956500
val_acc	0.942513

Best accuracy (on testing dataset): 88.97%

	precision	recall	f1-score	support
Bike	0.8837	0.9500	0.9157	40
Bus	0.7632	1.0000	0.8657	29
Car	1.0000	0.6750	0.8060	40
Pedestrians	0.9643	1.0000	0.9818	27
accuracy			0.8897	136
macro avg	0.9028	0.9062	0.8923	136
weighted avg	0.9082	0.8897	0.8859	136

	Bike	Bus	Car	Ped
Bike	38	2	-	-
Bus	-	29	-	-
Car	5	7	27	1
Pedes.	-	-	-	27

Model-300

Best accuracy (on validation dataset): 95.72%

acc	0.964500
val_acc	0.957219

Best accuracy (on testing dataset): 97.79%

	precision	recall	f1-score	support
Bike	1.0000	1.0000	1.0000	40
Bus	0.9667	1.0000	0.9831	29
Car	1.0000	0.9250	0.9610	40
Pedestrians	0.9310	1.0000	0.9643	27
accuracy			0.9779	136
macro avg	0.9744	0.9812	0.9771	136
weighted avg	0.9792	0.9779	0.9778	136

	Bike	Bus	Car	Pedes.
Bike	40	-	-	-
Bus	-	29	-	-
Car	-	1	37	2
Pedes.	-	-	-	27

Table 1 summarizes the classification report and confusion matrix for both the models. For a certain test data example, an accuracy increase of 8.8% was achieved.

3. Challenges

1. Handling corrupt images

    - One of the main challenges in the initial stage was to handle corrupt images present in dataset. Since data-generators are unable to handle this, this task was handled in the download images script, where we remove any such images if found.

2. Computational Power

    - A good performance GPU may be required if the images are of sizes greater than 150x150 pixels. The training time heavily increases moving from 150x150 to 300x300 image size.

4. Learning Outcomes

1. Various methods to create image datasets

    - Through this project, we did research on different ways to obtain images to build our dataset and learnt how to use readily-available APIs and packages to help us in this task. These tools will be useful in future projects that involve generating or gathering additional data.

2. Image data augmentation & manipulation

    - In this project, we also recognized the importance of data augmentation, especially for image data in order to generate more relevant examples for the model to learn correctly. In addition, scaling of images in order to optimize the training of the CNN model was another learning outcome from this project.

3. Methods to improve accuracy of Deep Learning models

    - One of the key learning outcomes of this project for us was to learn the concepts used for training a model having high accuracy on a relatively small dataset. To achieve this, the approach of data-generators & progressive resizing was applied to the model inputs.

5. Future Work

Since this classifier was selected as a learning assignment from a previous project of improving traffic lights wait-times, the knowledge gained from the project can be applied to build an object classifier that provides the number of vehicles at a traffic junction to an expert system for optimal wait-time calculation.

TagUI can be also be used to automate the image-downloading through JavaScript process for image scrapping from images.google.com.

6. User Guide

1. **Creating dataset**
   - Steps mentioned in **section 2.1** should be followed for the same.
   - Dataset used for training the model can be downloaded from – https://drive.google.com/file/d/1yAvTpaNdeqVqi7TPo1FHkMltU2wM82ns/view?usp=sharing

2. **Choosing a pre-defined dataset**
   - Make sure the class labels are same as the ones defined in **section 1.**
   - The dataset should be arranged in the hierarchical structure as explained in **section 2.2**

3. **Training**
   - To train the model, place the "train.py" file in the same directory as dataset's "data" folder.
   - **Execute** – python3 train.py

4. **Testing**
   - To test the model, place the "test.py" file in the same directory as dataset's "data" folder.
   - Make sure the weight files, created by the training script are present in the same directory.
   - **Execute** – python3 test.py

7. References

1. https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf?hl=en
2. https://chrome.google.com/webstore/detail/image-downloader/cnpniohnfphhjihaiiggeabnkjhpaldj
3. https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/