Classification Using CNNs For Autonomous Vehicles

Syd Proom (N0788203), Supervised by Dr. Archontis Giannakidis

Department of Physics & Mathematics
Nottingham Trent University

INTRODUCTION

With the sheer number of companies aiming to create the first fully autonomous vehicle, it often feels like these companies are competing in the 21st century's version of the "space race". Players in this space such as Tesla, Waymo (Google) and Uber are all using different techniques for collecting data and testing their model. It is clear that self-driving cars will revolutionise the transport industry with possibilities for driverless taxis and trucks that will aim to be drastically safer than any human.

In the majority of these examples, machine learning and neural networks will make the foundation of the solutions, as these techniques allow for constant improvement with more data. This is critical given the limitless possibilities of scenarios drivers can find themselves within. Detecting objects and classifying objects is fundamental for navigating roads, the later of which this poster will discuss.

The aim of this poster will be to create a model that predicts whether or not there is/are car(s) in front of the dash cam car. Classification will be used to predict a discrete value, where 0 means there are no cars in the image and 1 means there is at least 1 car in the image. In the full report, predicting continuous values (regression) such as the steering angle will be considered, along with predicting more than two classes such as traffic signs.

NEURAL NETWORK THEORY

A subset of machine learning is neural networks (NN) which are made of layers of neurons each with weights and thresholds [3]. The general form of a neural network can be seen in Figure 1. For image datasets, each pixel of the image will have a neuron to represent it in the input layer x_n which are all attached to neurons within i hidden layers h^i_m and finally reach the output layer y_k representing k classes.

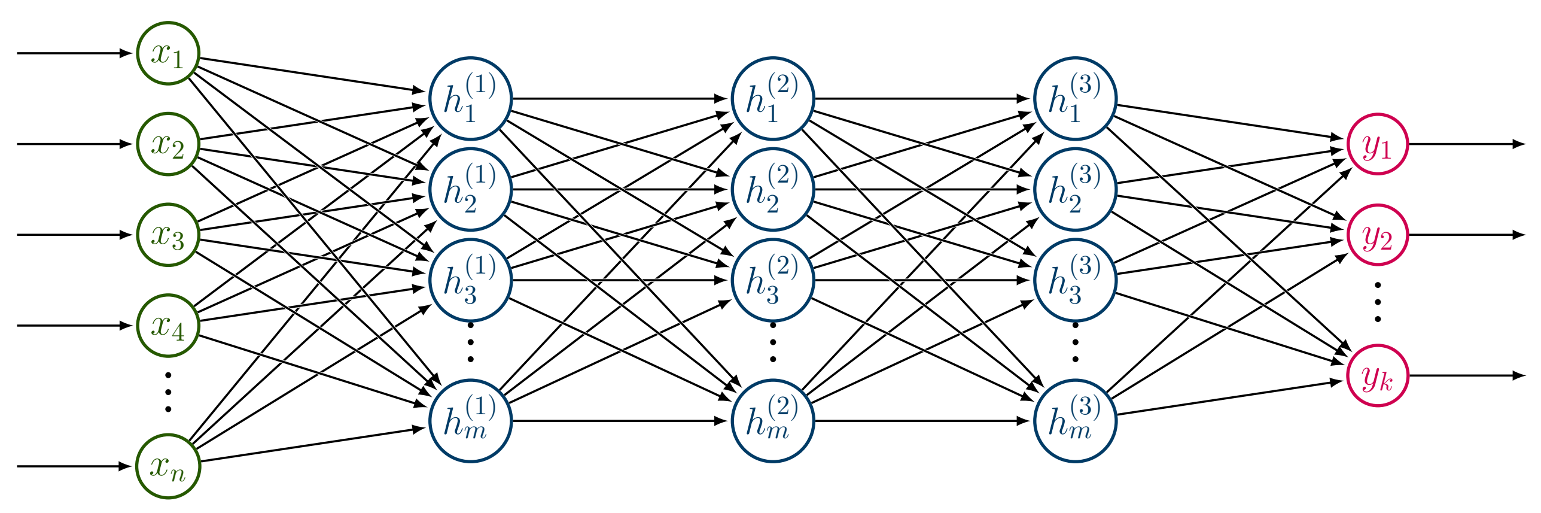

Figure 1: General neural network layer structure.

The value for each neuron is known as its activation, which is calculated using Equation (1) where x_i is the activation of the i^{th} neuron in the previous layer and w_i is the weight of the connection between the i^{th} neuron in the previous layer and the current neuron.

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n + b = \sum_{1}^{n} w_ix_i + b = \boldsymbol{x}^T\boldsymbol{w} + b \tag{1}$$

PRE-PROCESSING DATA

The dataset that will be used in this task will be the Cityscapes dataset [4] featuring approximately 20,000 labelled dashcam images taken within major cities in Germany. It is important to pre-process the dataset for training and testing to ensure any unnecessary information is reduced to maximise performance and model accuracy.

Each image label contains semantic segmentation data, which has labels for individual objects within the image and forms clusters [8] as can be seen in Figure 2. This needs to be converted to a binary $\{0, 1\}$ label using a simple piece of code. It is also important to ensure there is an equal number of images for each label. In the Cityscapes dataset, it reduces to the following data in Table 1

Table 1: Number of images for each class before and after pre-processing has completed for the binary classification task.

Binary Classification	Number of Images					
	Before Pre-Processing				After Pre-Processing	
Dataset	Training	Extra Training	Validation	Test	Training	Test
"0" Class	128	2608	18	1525	5008	500
"1" Class	2847	17390	482	0	5008	500
Total Images	2975	19998	500	1525	10016	1000

Each image has a resolution of 2048 × 1024 in full 8-bit colour. This resolution is quite high and would require lots of likely unnecessary neurons, so all images will be resized drastically to 132 × 132 (note that the aspect ratio does not matter).

Finally, the values of all pixels must be divided by 255 to normalise the dataset. Figure 2 and Figure 3 show the data before and after pre-processing.

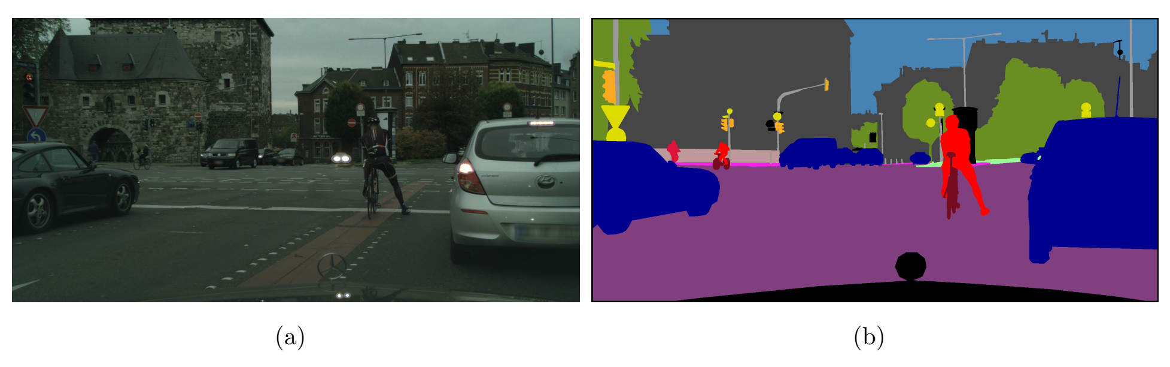

Figure 2: (a) shows unprocessed image from Cityscapes dataset (blurred) and (b) shows the corresponding semantic segmentation labels.

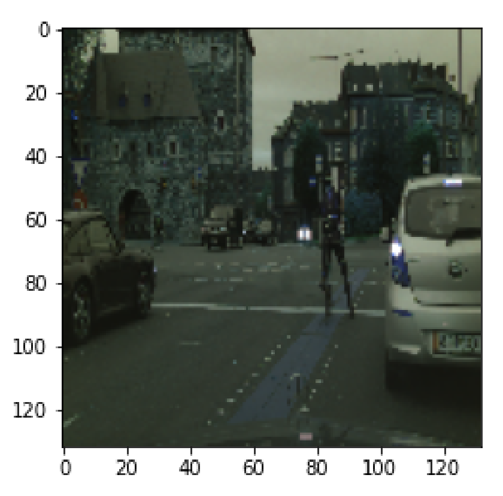

Figure 3: shows image after pre-processing is complete, and contains the label "1" (includes car)

CNN ARCHITECTURE

Now that the dataset is ready, it is time to create the model that will be trained on the dataset. A Convolutional Neural Network (CNN) will be used, which is a specialised version of a neural network for multi-dimensional data arrays including images [1]. CNNs differ from regular NNs in the layer structure, where CNN arrange neurons in 3D, and eventually reduce down to a 1D value representing the probability of the image being within the $\{1\}$ class. This means values below 0.5 represent images with no cars, and values above 0.5 represent images with cars.

The sequential model that was created and tweaked is contains the following layers, where further reasoning for tweaks can be found in the dissertation:

1. `Conv2D(256, (3, 3))`
2. `Activation('relu')`
3. `MaxPooling2D(pool_size = (2, 2))`
4. `Conv2D(256, (3, 3), name = 'visualised_layer')`
5. `Activation('relu')`
6. `MaxPooling2D(pool_size = (2, 2))`
7. `Flatten()`
8. `Dense(64)`
9. `Activation('relu')`
10. `Dropout(0.5)`
11. `Dense(1, activation = 'sigmoid')`

Where each layer means the following:

- `Conv2D`: 2-dimensional convolution is a way of collapsing a group of values into a singular value. In lines 1 and 4 it takes a 3 × 3 grid of values and condenses them each into a single value.
- `Activation`: Fully connected layer that applies an activation function to all values. In lines 2, 5 and 9 it uses the ReLU function, and line 11 uses the Sigmoid function.
- `MaxPooling2D`: Takes the maximum value of a group of values. In lines 3 and 6, it takes the maximum value of a 2 × 2 grid of values.
- `Flatten`: Converts the 3-dimensional feature maps into 1-dimensional vectors.
- `Dense`: Feeds the outputs from all previous layer to all neurons in the current layer.
- `Dropout`: A regularizer that reduces overfitting by randomly removing neurons every epoch [9] (number of times the algorithm works through the dataset [2]).

EVALUATION OF MODEL

Using the following code, the model can be trained on the dataset:

```
model.compile(loss = 'binary_crossentropy',
optimizer = 'SGD', metrics = ['accuracy'])
model.fit(X_train_norm, y_train, batch_size = 8,
epochs = 10, validation_split = 0.2)
```

The first command creates the model using the layers in the previously discussed, and identifies the optimiser and loss function. The optimiser, which in this case is Stochastic Gradient Descent (SGD), is the part of the model which tweaks the weights of the neurons. SGD does this by randomly selecting points on the loss function and making small, smooth iterations toward the minima [6]. Equation (2) shows SGD with ℓ loss function, η_j learning rate hyperparameter, H_j history of previous iterations including the gradient of the loss function and the loss function.

$$\text{SGD}(H_j, \eta_j): \quad \theta_{j+1} = \theta_j + \eta_j \nabla \ell(\theta_j) \tag{2}$$

The function that needs to be minimised is the loss function, which in this case is the binary cross entropy function. There are two separate cases of this function depending on the predicted label y_i being 0 or 1 (p_i being the probability distribution). Equation (3) shows the equation for binary cross entropy and Figure 4 shows both cases for the predicted labels.

$$\ell_i(y_i, p_i) = -\left(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right) \tag{3}$$

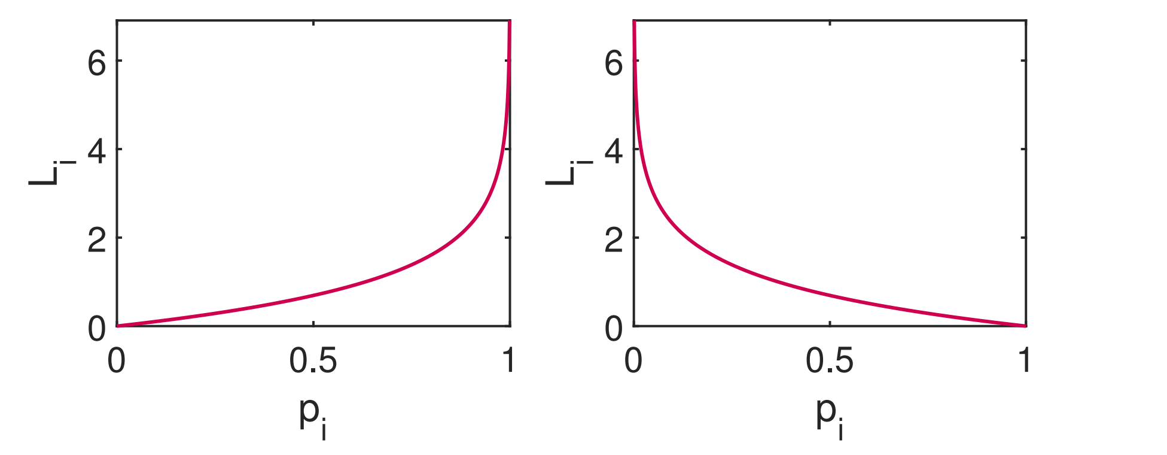

Figure 4: Two graphs showing loss function with the left predicting label "0" and the right predicting label "1".

Now fitting the model on the training dataset with a 20% validation split yields the results seen in Table 2 below and feature heatmap seen in Figure 5 on the right.

Table 2: Accuracy results on each dataset.

Dataset	Accuracy
Training	**90.38%**
Validation	**88.32%**
Test	**86.70%**

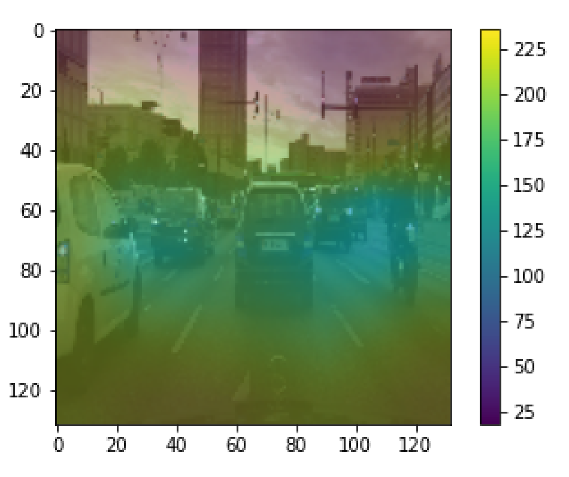

Figure 5: Heatmap of most useful features on test image.

CONCLUSION & DISCUSSION

The numbers from Table 2 show a decent accuracy that can be further improved with more tweaking to the hyperparameters in the model. Given that the accuracy for each dataset is similar, it is unlikely that the model is overfitting to the training set. From the heatmap in Figure 5, which aims to show behind the scenes of what the model is picking to decide the class of the image by using the final convolutional layer [7], it is correctly prioritising the road section in front of the dashcam car. However, since the data used was only from cities, the model is currently overfitting on city environments and will likely not perform as great when the images do not have buildings in every direction. Future work includes this dataset for multi-class classification for additional objects, along with object detection using techniques like YOLO.

This is where big companies such as Tesla and Google have a much better chance of creating a system that in theory could be used on public roads. For example, Tesla has started rolling out beta software updates to a select number of "safe drivers" [5] of the cars they have sold to test out their "Full Self-Driving package" using only vision (cameras). This is possible because they can request the fleet of cars on the road currently to provide them with more data for training which other companies simply cannot do. It would seem that for this reason, Tesla is currently winning this generations space race.

REFERENCES

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. *Understanding of a convolutional neural network*. 2017. doi: 10.1109/icengtechnol.2017.8308186.

[2] Jason Brownlee. Difference between a batch and an epoch in a neural network, Oct 2019. URL `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/`.

[3] IBM Cloud Education. What are neural networks?, Aug 2020. URL `https://www.ibm.com/cloud/learn/neural-networks`.

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Lora Kolodny. Tesla drivers can now request full self-driving beta with the press of a button, despite safety concerns, Sep 2021. URL `https://www.cnbc.com/2021/09/25/tesla-drivers-can-request-fsd-beta-with-a-button-press-despite-safety-concerns.html`.

[6] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[7] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7. URL `http://dx.doi.org/10.1007/s11263-019-01228-7`.

[8] Martin Thoma. A survey of semantic segmentation, 2016.

[9] Christian Versloot. Machine-learning-articles/how-to-use-dropout-with-keras.md at main · christianversloot/machine-learning-articles, Dec 2019. URL `https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-dropout-with-keras.md`.