# Problem Analysis and Development of a Model with MLP Networks

Phase I

Eliza Czaplicka, 2022110771

## 1. Problem introduction

There are more vehicles on the streets every year causing a rising trend in traffic in urban areas. This also brings about more accidents. One feature of a smart city should be prevention as well as management of these situations. With a program being able to correctly asses whether there is traffic on the road or an accident, the appropriate public services can be dispatched, the other vehicles can be directed to other roads etc. It is also useful for applications like Google Maps to track traffic and accidents to choose the best route for the driver.

For this task I used a github dataset:
https://github.com/OlafenwaMoses/Traffic-Net

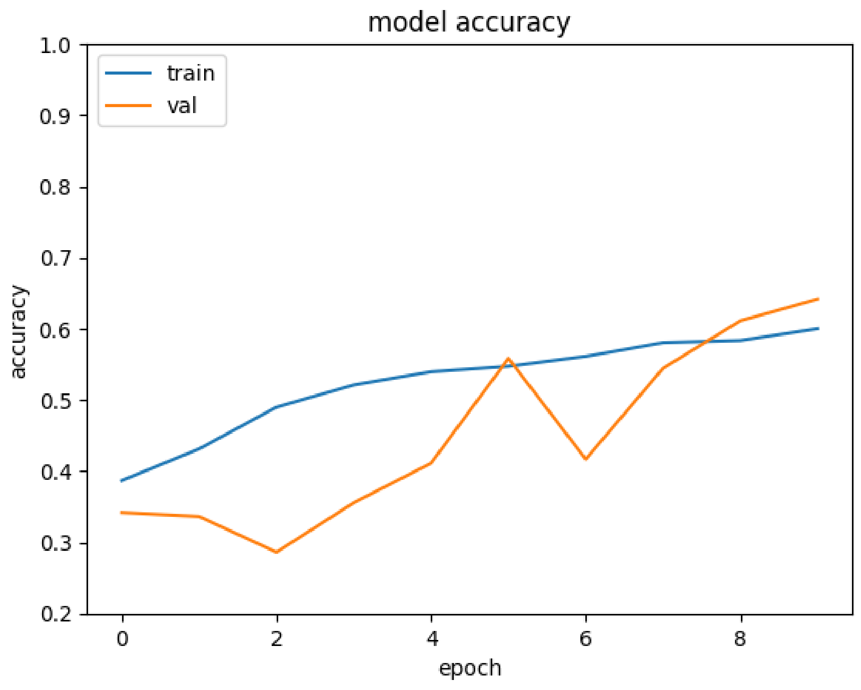
It consists of 4 categories: accident, fire, dense traffic and sparse traffic, with 900 images in each category for training and 200 in each for testing.

## 2. Implementations

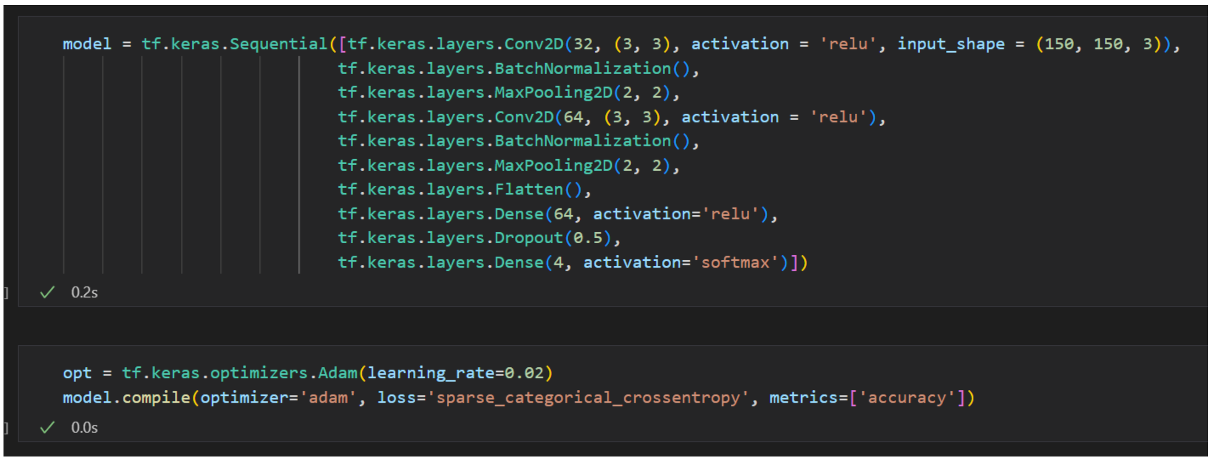
Throughout this project I used several various implementations with varying results. The good thing was the results got better with every new idea. I will start with the initial model and go through my progress in chronological order.

### 1. First experiments

My first idea was simple - a CNN. It turned out not so simple to do in practice.
I used a trial and error method while mixing convolutional, pooling, flattening and dense layers. I encountered a lot of cases where the accuracy did not change between epochs or got worse. At that time it was around 30%, almost like guessing. After some more experiments, I achieved the best I could with this approach.



At least the accuracy was rising. The model that achieved these results was:



```
model = tf.keras.Sequential([tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.MaxPooling2D(2, 2),
                             tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.MaxPooling2D(2, 2),
                             tf.keras.layers.Flatten(),
                             tf.keras.layers.Dense(64, activation='relu'),
                             tf.keras.layers.Dropout(0.5),
                             tf.keras.layers.Dense(4, activation='softmax')])
```
✓ 0.2s

```
opt = tf.keras.optimizers.Adam(learning_rate=0.02)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```
✓ 0.0s

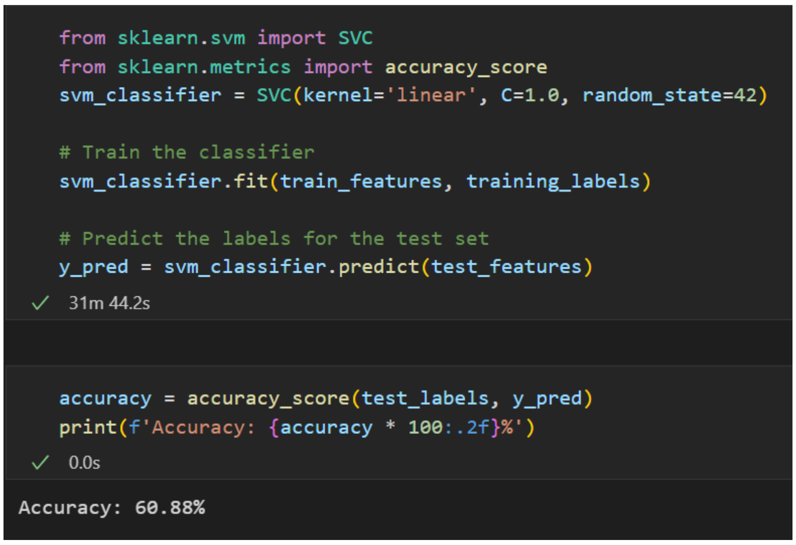
Prior to the training I shuffled the data and normalized it for better performance and variety.

However, the results were not satisfactory. 60% accuracy is not what I was looking for. So I turned to other measures.

The HOG and SVM approach

I decided to do some feature engineering beforehand. HOG - Histogram of Oriented Gradients seemed like a good idea for this task. This algorithm not only finds the magnitude of an edge in an image but also its direction (gradient and orientation). Additionally, it divides an image into regions and calculates the occurrences of gradient orientations.

With that, I had two-dimensional data to put into a linear SVM. I chose linearity because of its speed, but still, it took over an hour to compute. And after this time I achieved an accuracy of 60,88%. Very similar to the result from the previous implementation, so I went on to try my third and final idea.



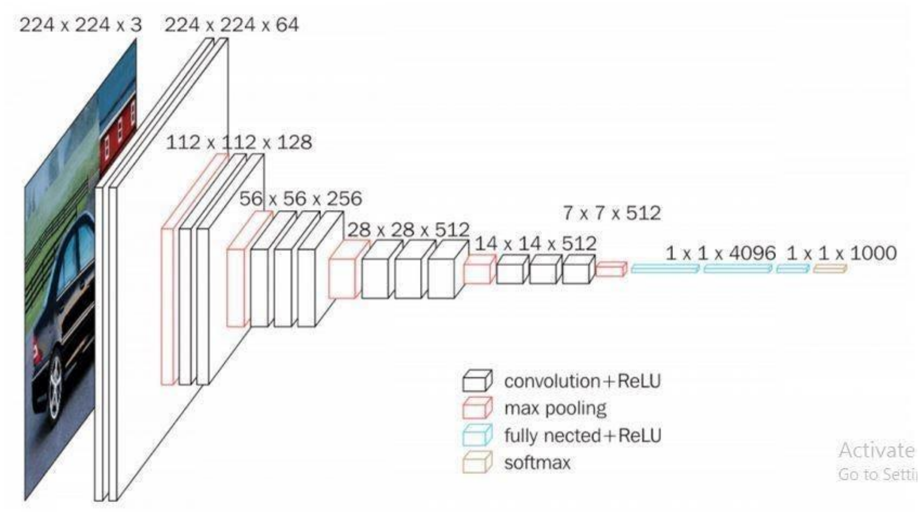
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)

# Train the classifier
svm_classifier.fit(train_features, training_labels)

# Predict the labels for the test set
y_pred = svm_classifier.predict(test_features)
```
✓ 31m 44.2s

```
accuracy = accuracy_score(test_labels, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```
✓ 0.0s

```
Accuracy: 60.88%
```

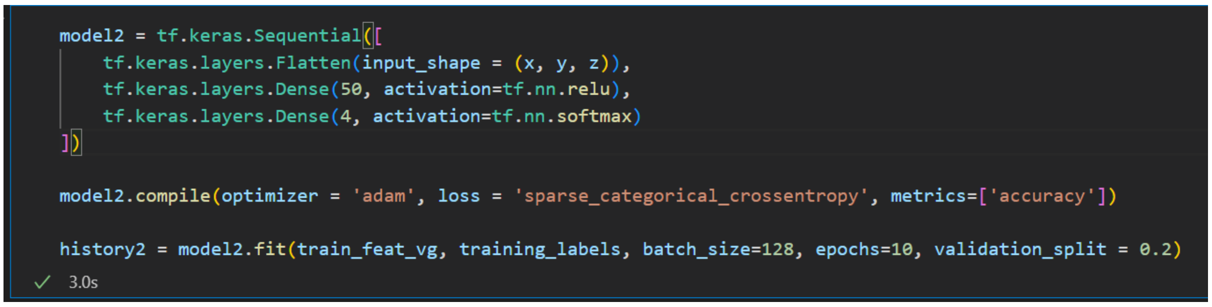
## 2. Final model

After previous failures I turned to Keras applications. I decided on VGG16.



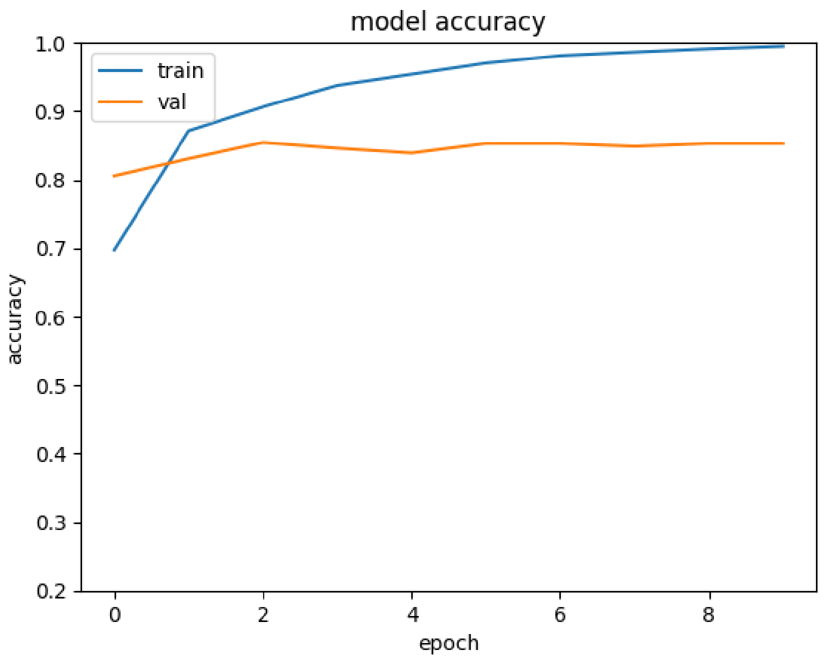
VGG16 was introduced in a paper “Very deep convolutional networks for large-scale image recognition”, which was made for the ImageNet Large-Scale Visual Recognition Challenge in 2014 and won 1st place. It is a type of convolutional neural network used for object detection and classification tasks. The architecture is shown above. The 16 in the name comes from 16 layers that have weights, convolutional and dense.

After VGG16 predictions I added a simple output network for the problem at hand.



```
model2 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape = (x, y, z)),
    tf.keras.layers.Dense(50, activation=tf.nn.relu),
    tf.keras.layers.Dense(4, activation=tf.nn.softmax)
])

model2.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])

history2 = model2.fit(train_feat_vg, training_labels, batch_size=128, epochs=10, validation_split = 0.2)
```
✓ 3.0s

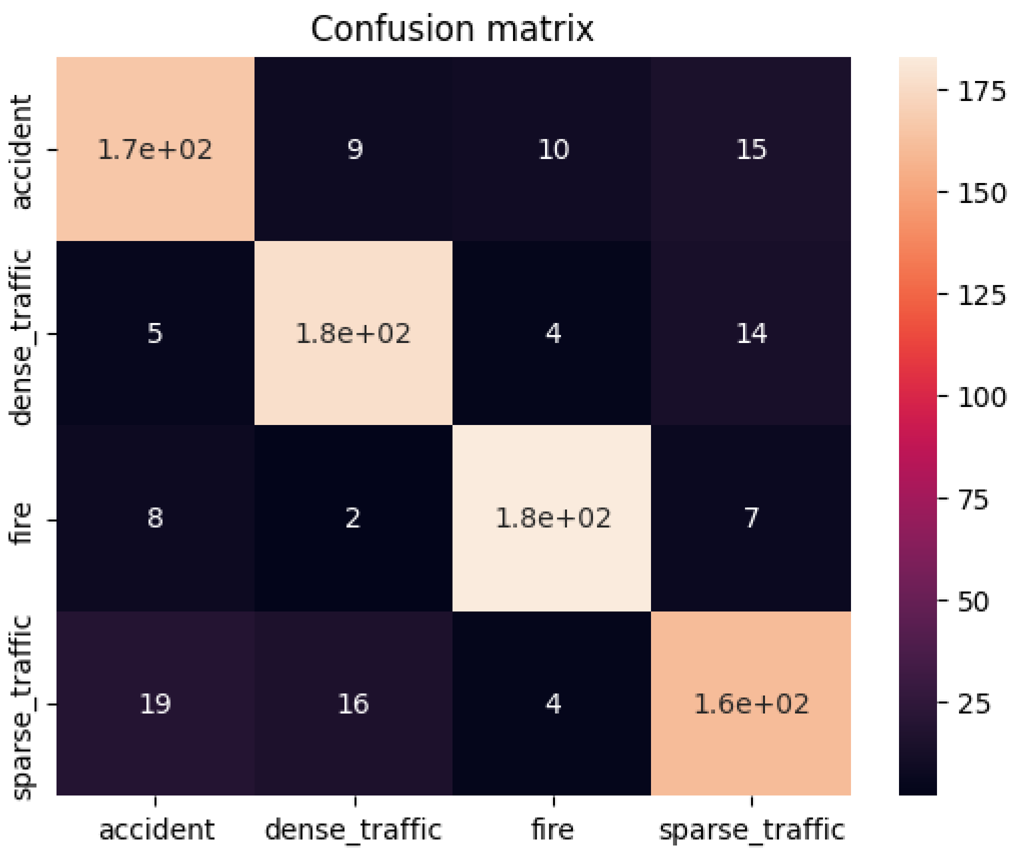
And the results were not disappointing:

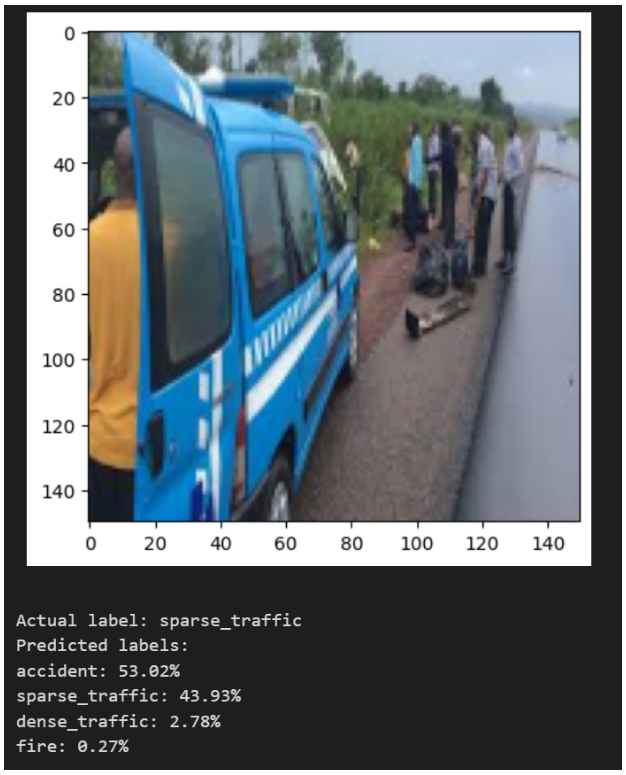


After many failed attempts the achieved accuracy for the test set was 86%

## 3. Results achieved

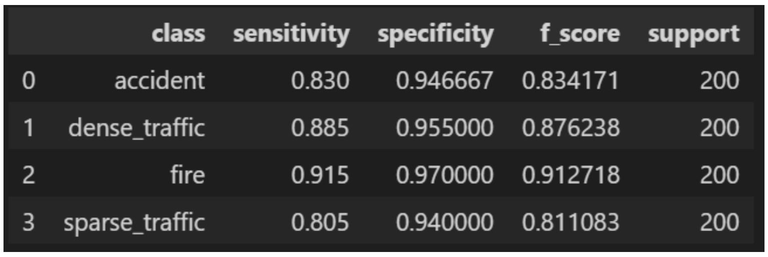
The results of the final model.



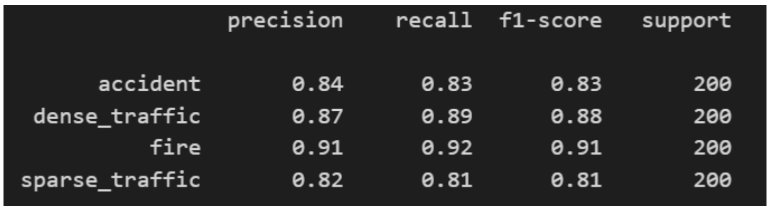


```
Actual label: sparse_traffic
Predicted labels:
accident: 53.02%
sparse_traffic: 43.93%
dense_traffic: 2.78%
fire: 0.27%
```

We notice that the algorithm has the most trouble accurately predicting the sparse traffic class. It is most often confused with accident class. This may be because many accident or fire photos were of vehicles on empty roads, with no other cars around. On the left, we can see one such case, a car on the side of the road with no other cars around was usually labelled as an accident, but not in this case.



| | class | sensitivity | specificity | f_score | support |
|---|---|---|---|---|---|
| 0 | accident | 0.830 | 0.946667 | 0.834171 | 200 |
| 1 | dense_traffic | 0.885 | 0.955000 | 0.876238 | 200 |
| 2 | fire | 0.915 | 0.970000 | 0.912718 | 200 |
| 3 | sparse_traffic | 0.805 | 0.940000 | 0.811083 | 200 |



```
               precision    recall  f1-score   support

     accident       0.84      0.83      0.83       200
dense_traffic       0.87      0.89      0.88       200
         fire       0.91      0.92      0.91       200
sparse_traffic      0.82      0.81      0.81       200
```

This algorithm performs better in terms of sensitivity (true positive rate) than specificity (true negative rate) but the results are satisfactory in both areas.

## 4. Further enhancements and ideas

The final model performed well but could be better. To further enhance it I would try out other Keras applications, looking into their structure and editing them to fit my problem. I would try teaching the algorithm on my training data to achieve the best results.
What could also be done, is a random forest classifier with various models inside in order to reach the best accuracy.

## 5. References

arXiv:1512.03385 (Deep Residual Learning for Image Recognition)
https://machinelearningmastery.com/model-averaging-ensemble-for-deep-learning-neural-networks/
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529
https://www.kaggle.com/code/aslanahmedov/automatic-number-plate-recognition
https://keras.io/api/applications/
https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918