



University  
of Windsor

**Connected Autonomous Vehicles**

**ELEC-8900-110 Special Topics**

**“Assignment 2”**

**Submitted by**

Vijayarangan Pandurengadurai Raju    110128279

**Submitted to:**

Dr. Ning Zhang

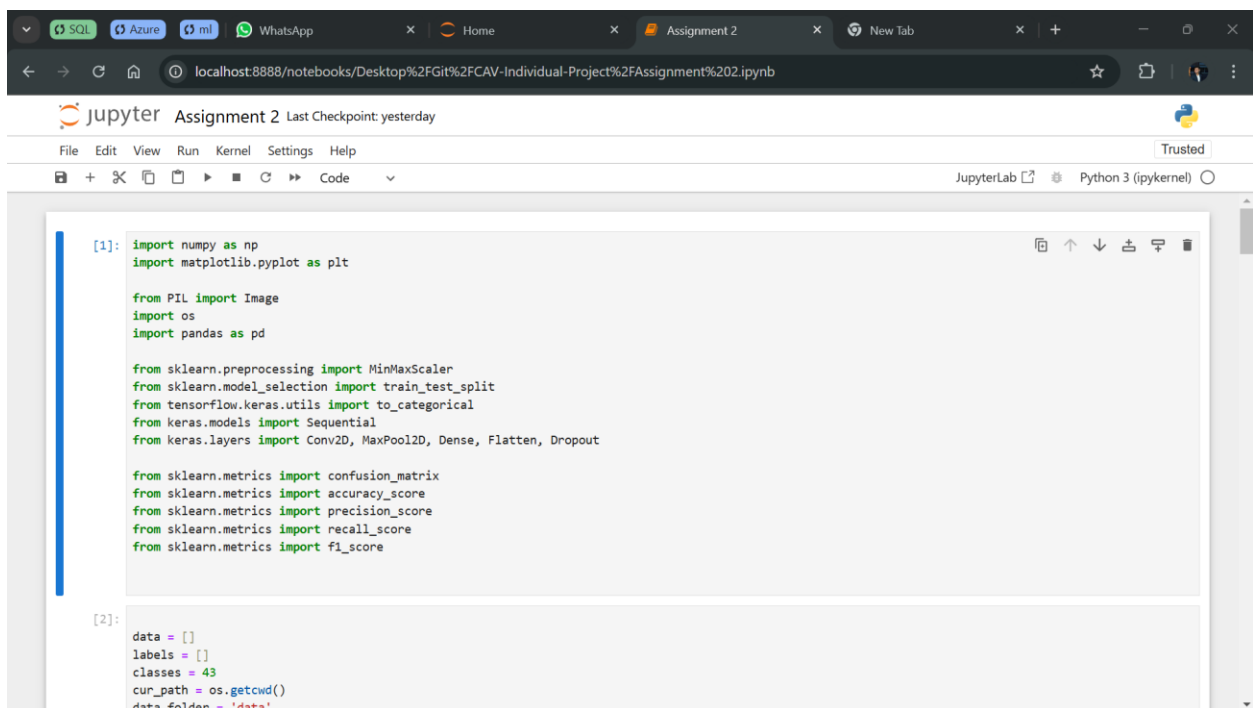
**Submitted on:**

14<sup>th</sup> November 2024

## Step 1a: Downloading Text Editor

I am using **Jupyter Notebook** which is widely used for:

1. **Interactive Coding:** It allows you to run code in chunks (cells) and see the output immediately, which is ideal for testing and iterating quickly.
2. **Data Analysis and Visualization:** With support for libraries like pandas and matplotlib, it's perfect for exploring datasets and creating visualizations.
3. **Documentation and Collaboration:** You can add markdown cells to explain code and share notebooks with others, making it great for tutorials, documentation, and collaborative projects.
4. **Reproducible Research:** Jupyter Notebooks are commonly used in data science and research to keep code, data, and results together for reproducibility.
5. **Language Flexibility:** While popular for Python, Jupyter also supports many other languages, making it a versatile tool across fields and disciplines.



```
[1]: import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
import os
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

[2]: data = []
labels = []
classes = 43
cur_path = os.getcwd()
data_folder = 'data'
```

## Step 1b: Downloading the necessary Libraries.

Downloading libraries saves time by providing pre-built functions, ensuring reliability with tested code, and speeding up development through optimized solutions. They make code maintenance easier with regular updates and offer better performance, especially for complex tasks. Libraries also standardize solutions, making your code more understandable to others and offering community support for troubleshooting.

```
PS C:\Users\prvij\Desktop\Git\CAV-Individual-Project> pip install tensorflow keras sklearn numpy matplotlib
Requirement already satisfied: tensorflow in c:\users\prvij\anaconda3\lib\site-packages (2.10.0)
Requirement already satisfied: keras in c:\users\prvij\anaconda3\lib\site-packages (3.6.0)
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Preparing metadata (setup.py) ... error
error: subprocess-exited-with-error

× python setup.py egg_info did not run successfully.
  exit code: 1
  [15 lines of output]
  The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
  rather than 'sklearn' for pip commands.

  Here is how to fix this error in the main use cases:
  - use 'pip install scikit-learn' rather than 'pip install sklearn'
  - replace 'sklearn' by 'scikit-learn' in your pip requirements files
    (requirements.txt, setup.py, setup.cfg, Pipfile, etc ...)
  - if the 'sklearn' package is used by one of your dependencies,
    it would be great if you take some time to track which package uses
    'sklearn' instead of 'scikit-learn' and report it to their issue tracker
  - as a last resort, set the environment variable
    SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this error

  More information is available at
  https://github.com/scikit-learn/sklearn-pypi-package
  [end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
  See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
PS C:\Users\prvij\Desktop\Git\CAV-Individual-Project>
```

## Importing the necessary Libraries to the file:

```
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
import os
import pandas as pd

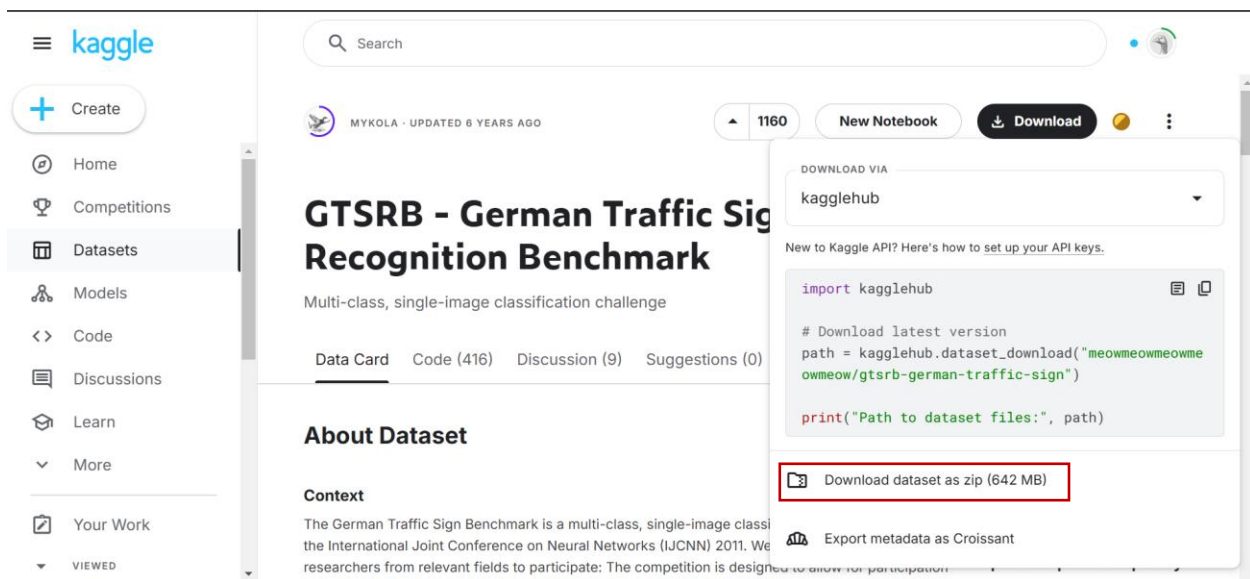
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Step 2: Download the GTSRB - German Traffic Sign Recognition Benchmark dataset from Kaggle data from **Kaggle**.

Kaggle is valuable for:

1. **Datasets and Competitions:** It offers a vast collection of public datasets and hosts competitions, allowing you to practice and showcase your skills on real-world problems.
2. **Learning Resources and Notebooks:** Kaggle provides free courses and user-generated notebooks, making it an excellent platform for learning and experimenting with code in data science and machine learning.
3. **Community and Networking:** Kaggle's community forums and ranking system enable collaboration, discussion, and visibility, connecting you with data science professionals worldwide.



Step 3: Reading the training data from “Train” and testing data from “Test” folder.

### 3a: Loading the Training data

```
data = []
labels = []
classes = 43
cur_path = os.getcwd()
data_folder = 'data'

# Loading training dataset
for i in range(classes):
    path = os.path.join(cur_path, data_folder, 'train', str(i))
    images = os.listdir(path)

    for a in images:
        try:
            image = Image.open(path + '\\' + a)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

### 3b: Loading the Test data

```
# Testing the model
path = os.path.join(cur_path, data_folder)
y_test = pd.read_csv(data_folder + '/' + 'Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values

data=[]

for img in imgs:
    image = Image.open(path + '\\' + img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test = np.array(data)
```

Step 4 a and 4 b: Building the Convolutional Neural Network which contains Convolutional, Dense, DropOut and Pooling layers.

### Explanation of Layers

1. **Conv2D Layer:** Detects features in the input image by applying convolution filters. Each Conv2D layer increases the complexity of feature detection.
2. **MaxPooling2D Layer:** Reduces the dimensionality of the feature maps, keeping the essential information.
3. **Flatten Layer:** Converts 2D feature maps into a 1D vector to pass to the dense layers.
4. **Dense Layer:** Standard fully connected layer for decision-making in classification.
5. **Dropout Layer:** Reduces overfitting by randomly setting input units to zero during training.

```
# Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
model.summary()
```

Summary of the model:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	2,432
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 256)	409,856
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 43)	11,051

Total params: 441,835 (1.69 MB)

Trainable params: 441,835 (1.69 MB)

Non-trainable params: 0 (0.00 B)

## Compilation of the model

An **epoch** represents a single complete pass through the entire training dataset. During an epoch, the model goes through each data sample in the training set once, adjusting its weights based on the error calculated after each forward and backward pass.

```
# Compilation of the model
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
epochs = 15
```

```
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
```

```
model.save('traffic_classifier.h5')
```

```
Epoch 1/15
981/981 ————— 10s 8ms/step - accuracy: 0.3743 - loss: 2.3236 - val_accuracy: 0.9365 - val_loss: 0.2704
Epoch 2/15
981/981 ————— 7s 7ms/step - accuracy: 0.8649 - loss: 0.4363 - val_accuracy: 0.9777 - val_loss: 0.1402
Epoch 3/15
981/981 ————— 7s 7ms/step - accuracy: 0.9221 - loss: 0.2561 - val_accuracy: 0.9852 - val_loss: 0.0645
Epoch 4/15
981/981 ————— 7s 7ms/step - accuracy: 0.9416 - loss: 0.1893 - val_accuracy: 0.9894 - val_loss: 0.0464
Epoch 5/15
981/981 ————— 7s 7ms/step - accuracy: 0.9532 - loss: 0.1522 - val_accuracy: 0.9894 - val_loss: 0.0429
Epoch 6/15
981/981 ————— 7s 7ms/step - accuracy: 0.9615 - loss: 0.1273 - val_accuracy: 0.9907 - val_loss: 0.0350
Epoch 7/15
981/981 ————— 7s 7ms/step - accuracy: 0.9662 - loss: 0.1084 - val_accuracy: 0.9929 - val_loss: 0.0306
Epoch 8/15
981/981 ————— 7s 7ms/step - accuracy: 0.9681 - loss: 0.1049 - val_accuracy: 0.9918 - val_loss: 0.0289
Epoch 9/15
981/981 ————— 7s 7ms/step - accuracy: 0.9725 - loss: 0.0876 - val_accuracy: 0.9926 - val_loss: 0.0283
Epoch 10/15
981/981 ————— 7s 7ms/step - accuracy: 0.9717 - loss: 0.0884 - val_accuracy: 0.9945 - val_loss: 0.0223
Epoch 11/15
981/981 ————— 7s 7ms/step - accuracy: 0.9768 - loss: 0.0766 - val_accuracy: 0.9948 - val_loss: 0.0226
Epoch 12/15
981/981 ————— 7s 7ms/step - accuracy: 0.9761 - loss: 0.0785 - val_accuracy: 0.9945 - val_loss: 0.0218
Epoch 13/15
...
Epoch 14/15
981/981 ————— 7s 7ms/step - accuracy: 0.9783 - loss: 0.0694 - val_accuracy: 0.9936 - val_loss: 0.0253
Epoch 15/15
981/981 ————— 7s 7ms/step - accuracy: 0.9776 - loss: 0.0686 - val_accuracy: 0.9959 - val_loss: 0.0168
```

Step 4 c: Showing few examples of prediction.

Getting a random choice from X\_test and Predicting labels for that.

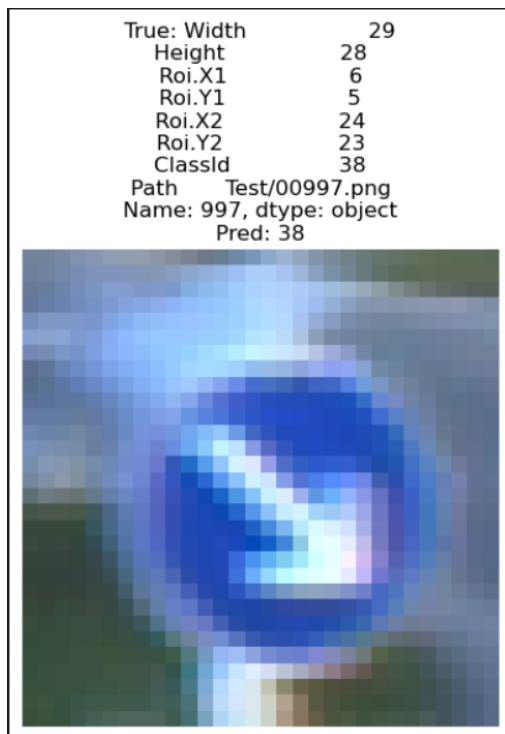
```
# Select a random sample from the test set
index = np.random.choice(len(X_test), 1)[0]
sample_image = X_test[index]

# If y_test is a DataFrame, use iloc to get the label at the same index
sample_label = y_test.iloc[index] if isinstance(y_test, pd.DataFrame) else y_test[index]

# Predict the label
predicted_label = np.argmax(model.predict(np.array([sample_image])))

# Plot the image with the true and predicted labels
plt.imshow(sample_image.reshape(30, 30, 3)) # Adjust reshape if your images are a different size
plt.axis('off')
plt.title(f"True: {sample_label}\nPred: {predicted_label}")
plt.show()
```

Image with the true and predicted labels





Step 4 d: Compare the performance with the above model.

### 1) Changing the Epochs

**Epochs** refer to how many times the model goes through the entire training dataset during training.

- **One Epoch:** The model sees each training example once.
- **Multiple Epochs:** The model keeps seeing the same data multiple times, allowing it to learn and adjust gradually.

For example, if you set epochs=10, the model will go over the entire dataset 10 times to improve its predictions.

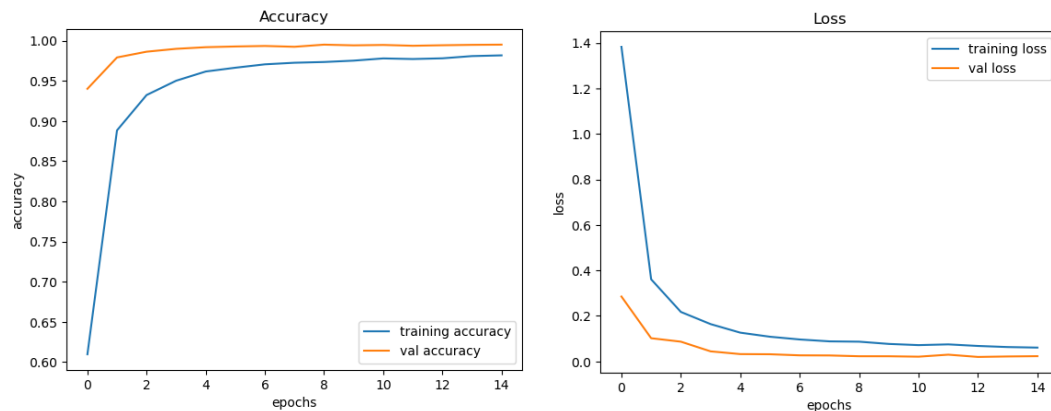
#### a. The Performance evaluation when Epochs at 15

Code:

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Performance evaluation:



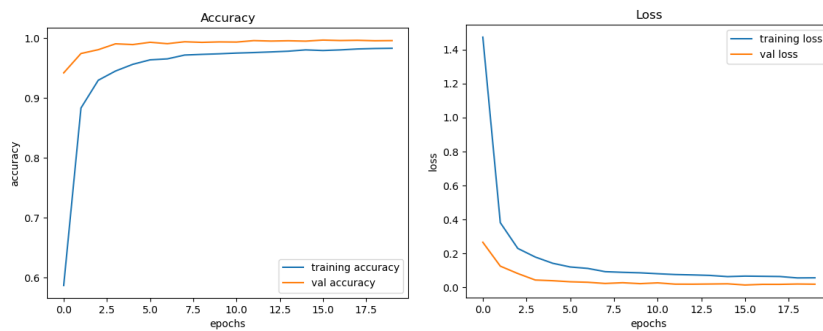
### b. The Performance evaluation when Epochs at 20

Code:

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 20
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Performance evaluation:



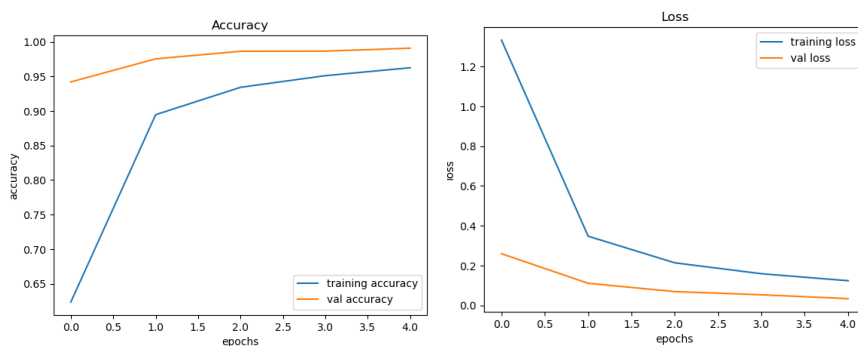
### c. The Performance measure when Epochs at 5

Code

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 5
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Performance evaluation



## 2) Changing the Batch size

The **batch size** is the number of training samples the model processes before updating its internal parameters (weights).

- **Small Batch Sizes:** The model updates weights more frequently (after every few samples), which can sometimes help it learn faster but may introduce noise.
- **Large Batch Sizes:** The model updates weights less frequently, which can be more stable but may require more memory and can slow down learning.

For example, if you have 1,000 training samples and set `batch_size=32`, the model will go through 32 samples at a time, update its weights, then move to the next 32, repeating this until it has processed all 1,000 samples (completing one epoch).

This balance between batch size and learning rate often influences training speed and model performance.

You can see the training speed and performance changes in the output

### a) The Performance evaluation when Batch size = 32

Code:

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Output:

```
Epoch 1/15
981/981 — 11s 9ms/step - accuracy: 0.3788 - loss: 2.2775 - val_accuracy: 0.9385 - val_loss: 0.3222
Epoch 2/15
981/981 — 8s 9ms/step - accuracy: 0.8612 - loss: 0.4524 - val_accuracy: 0.9763 - val_loss: 0.1117
Epoch 3/15
981/981 — 8s 8ms/step - accuracy: 0.9219 - loss: 0.2593 - val_accuracy: 0.9866 - val_loss: 0.0681
Epoch 4/15
981/981 — 9s 9ms/step - accuracy: 0.9419 - loss: 0.1884 - val_accuracy: 0.9874 - val_loss: 0.0586
Epoch 5/15
981/981 — 9s 9ms/step - accuracy: 0.9503 - loss: 0.1570 - val_accuracy: 0.9890 - val_loss: 0.0487
Epoch 6/15
981/981 — 8s 8ms/step - accuracy: 0.9584 - loss: 0.1327 - val_accuracy: 0.9916 - val_loss: 0.0325
Epoch 7/15
981/981 — 8s 8ms/step - accuracy: 0.9638 - loss: 0.1171 - val_accuracy: 0.9941 - val_loss: 0.0270
Epoch 8/15
981/981 — 9s 9ms/step - accuracy: 0.9673 - loss: 0.1057 - val_accuracy: 0.9927 - val_loss: 0.0288
Epoch 9/15
981/981 — 9s 9ms/step - accuracy: 0.9700 - loss: 0.0961 - val_accuracy: 0.9948 - val_loss: 0.0263
Epoch 10/15
981/981 — 8s 9ms/step - accuracy: 0.9726 - loss: 0.0845 - val_accuracy: 0.9948 - val_loss: 0.0239
Epoch 11/15
981/981 — 9s 9ms/step - accuracy: 0.9737 - loss: 0.0834 - val_accuracy: 0.9955 - val_loss: 0.0204
Epoch 12/15
981/981 — 9s 9ms/step - accuracy: 0.9766 - loss: 0.0777 - val_accuracy: 0.9950 - val_loss: 0.0214
Epoch 13/15
981/981 — 9s 9ms/step - accuracy: 0.9777 - loss: 0.0726 - val_accuracy: 0.9955 - val_loss: 0.0170
Epoch 14/15
981/981 — 9s 9ms/step - accuracy: 0.9792 - loss: 0.0707 - val_accuracy: 0.9943 - val_loss: 0.0205
Epoch 15/15
981/981 — 9s 9ms/step - accuracy: 0.9766 - loss: 0.0746 - val_accuracy: 0.9964 - val_loss: 0.0180

Confusion Matrix:
[[ 58   1   0 ...   0   0   0]
 [   0 715   0 ...   0   0   0]
 [   0   7 741 ...   0   0   0]
 ...
 [   0   0   0 ... 84   0   0]
 [   0   0   0 ...   0 39   0]
 [   0   0   0 ...   0   1 89]]
Accuracy: 0.971971
Precision: 0.966679
Recall: 0.951783
F1 score: 0.955791
```

## b) The Performance evaluation when Batch size = 16

Code:

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15
history = model.fit(X_train, y_train, batch_size=16, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Output:

```
Epoch 1/15
1961/1961 — 15s 6ms/step - accuracy: 0.4348 - loss: 2.0522 - val_accuracy: 0.9515 - val_loss: 0.2209
Epoch 2/15
1961/1961 — 12s 6ms/step - accuracy: 0.8849 - loss: 0.3838 - val_accuracy: 0.9841 - val_loss: 0.0825
Epoch 3/15
1961/1961 — 12s 6ms/step - accuracy: 0.9338 - loss: 0.2245 - val_accuracy: 0.9883 - val_loss: 0.0538
Epoch 4/15
1961/1961 — 12s 6ms/step - accuracy: 0.9479 - loss: 0.1738 - val_accuracy: 0.9874 - val_loss: 0.0510
Epoch 5/15
1961/1961 — 13s 7ms/step - accuracy: 0.9574 - loss: 0.1348 - val_accuracy: 0.9899 - val_loss: 0.0501
Epoch 6/15
1961/1961 — 13s 7ms/step - accuracy: 0.9592 - loss: 0.1289 - val_accuracy: 0.9908 - val_loss: 0.0429
Epoch 7/15
1961/1961 — 13s 7ms/step - accuracy: 0.9658 - loss: 0.1089 - val_accuracy: 0.9945 - val_loss: 0.0249
Epoch 8/15
1961/1961 — 12s 6ms/step - accuracy: 0.9676 - loss: 0.1104 - val_accuracy: 0.9939 - val_loss: 0.0232
Epoch 9/15
1961/1961 — 14s 7ms/step - accuracy: 0.9685 - loss: 0.1012 - val_accuracy: 0.9946 - val_loss: 0.0234
Epoch 10/15
1961/1961 — 13s 6ms/step - accuracy: 0.9713 - loss: 0.0963 - val_accuracy: 0.9946 - val_loss: 0.0212
Epoch 11/15
1961/1961 — 13s 7ms/step - accuracy: 0.9722 - loss: 0.0944 - val_accuracy: 0.9946 - val_loss: 0.0209
Epoch 12/15
1961/1961 — 12s 6ms/step - accuracy: 0.9711 - loss: 0.0999 - val_accuracy: 0.9925 - val_loss: 0.0288
Epoch 13/15
1961/1961 — 12s 6ms/step - accuracy: 0.9707 - loss: 0.1022 - val_accuracy: 0.9958 - val_loss: 0.0186
Epoch 14/15
1961/1961 — 12s 6ms/step - accuracy: 0.9744 - loss: 0.0903 - val_accuracy: 0.9948 - val_loss: 0.0205
Epoch 15/15
1961/1961 — 13s 6ms/step - accuracy: 0.9763 - loss: 0.0897 - val_accuracy: 0.9936 - val_loss: 0.0260
```

Confusion Matrix:

```
[[ 59   1   0 ...   0   0   0]
 [   0 713   4 ...   0   0   0]
 [   0   6 737 ...   0   0   0]
 ...
 [   0   0   1 ...  87   0   0]
 [   0   0   0 ...   0  51   1]
 [   0   0   0 ...   0   4  86]]
```

Accuracy: 0.960570

Precision: 0.948972

Recall: 0.947292

F1 score: 0.946227

c) The Performance evaluation when Batch size = 64

Code:

```
# Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_val, y_val))
model.save('traffic_classifier.h5')
```

Output:

```
Epoch 1/15
491/491 ————— 9s 13ms/step - accuracy: 0.3115 - loss: 2.5764 - val_accuracy: 0.9162 - val_loss: 0.3884
Epoch 2/15
491/491 ————— 6s 12ms/step - accuracy: 0.8266 - loss: 0.5599 - val_accuracy: 0.9615 - val_loss: 0.1734
Epoch 3/15
491/491 ————— 6s 12ms/step - accuracy: 0.9022 - loss: 0.3177 - val_accuracy: 0.9796 - val_loss: 0.1041
Epoch 4/15
491/491 ————— 6s 13ms/step - accuracy: 0.9289 - loss: 0.2265 - val_accuracy: 0.9846 - val_loss: 0.0753
Epoch 5/15
491/491 ————— 6s 12ms/step - accuracy: 0.9438 - loss: 0.1820 - val_accuracy: 0.9880 - val_loss: 0.0558
Epoch 6/15
491/491 ————— 6s 12ms/step - accuracy: 0.9544 - loss: 0.1479 - val_accuracy: 0.9887 - val_loss: 0.0533
Epoch 7/15
491/491 ————— 6s 12ms/step - accuracy: 0.9605 - loss: 0.1297 - val_accuracy: 0.9918 - val_loss: 0.0387
Epoch 8/15
491/491 ————— 6s 12ms/step - accuracy: 0.9674 - loss: 0.1075 - val_accuracy: 0.9904 - val_loss: 0.0407
Epoch 9/15
491/491 ————— 6s 12ms/step - accuracy: 0.9637 - loss: 0.1240 - val_accuracy: 0.9934 - val_loss: 0.0304
Epoch 10/15
491/491 ————— 6s 12ms/step - accuracy: 0.9723 - loss: 0.0865 - val_accuracy: 0.9923 - val_loss: 0.0357
Epoch 11/15
491/491 ————— 6s 13ms/step - accuracy: 0.9749 - loss: 0.0814 - val_accuracy: 0.9946 - val_loss: 0.0253
Epoch 12/15
491/491 ————— 7s 13ms/step - accuracy: 0.9751 - loss: 0.0800 - val_accuracy: 0.9940 - val_loss: 0.0223
Epoch 13/15
491/491 ————— 6s 12ms/step - accuracy: 0.9783 - loss: 0.0713 - val_accuracy: 0.9948 - val_loss: 0.0213
Epoch 14/15
491/491 ————— 6s 12ms/step - accuracy: 0.9782 - loss: 0.0691 - val_accuracy: 0.9971 - val_loss: 0.0165
Epoch 15/15
491/491 ————— 6s 12ms/step - accuracy: 0.9791 - loss: 0.0674 - val_accuracy: 0.9968 - val_loss: 0.0154
```

Confusion Matrix:

```
[[ 59   1   0 ...   0   0   0]
 [   0 714   3 ...   0   0   0]
 [   0   8 738 ...   0   0   0]
 ...
 [   0   1   0 ...  88   0   0]
 [   0   0   0 ...   0  45   0]
 [   0   0   0 ...   0   0  90]]
```

Accuracy: 0.963341

Precision: 0.948767

Recall: 0.940687

F1 score: 0.942891

### 3) Decreasing the number of filters

By reducing the filters from 32 to 16 in the first convolutional layer and from 64 to 32 in the second convolutional layer, the model's ability to capture detailed patterns will be slightly reduced, this might decrease the model's capacity slightly but will also make it faster to train.

You can see the training speed and performance changes in the output.

#### a) Filter when first layer is 32 and Second layer is 64

Code:

```
# Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
model.summary()
```

Output:

```
Epoch 1/15
981/981 — 11s 9ms/step - accuracy: 0.3788 - loss: 2.2775 - val_accuracy: 0.9385 - val_loss: 0.3222
Epoch 2/15
981/981 — 8s 9ms/step - accuracy: 0.8612 - loss: 0.4524 - val_accuracy: 0.9763 - val_loss: 0.1117
Epoch 3/15
981/981 — 8s 8ms/step - accuracy: 0.9219 - loss: 0.2593 - val_accuracy: 0.9866 - val_loss: 0.0681
Epoch 4/15
981/981 — 9s 9ms/step - accuracy: 0.9419 - loss: 0.1884 - val_accuracy: 0.9874 - val_loss: 0.0586
Epoch 5/15
981/981 — 9s 9ms/step - accuracy: 0.9503 - loss: 0.1570 - val_accuracy: 0.9890 - val_loss: 0.0487
Epoch 6/15
981/981 — 8s 8ms/step - accuracy: 0.9584 - loss: 0.1327 - val_accuracy: 0.9916 - val_loss: 0.0325
Epoch 7/15
981/981 — 8s 8ms/step - accuracy: 0.9638 - loss: 0.1171 - val_accuracy: 0.9941 - val_loss: 0.0270
Epoch 8/15
981/981 — 9s 9ms/step - accuracy: 0.9673 - loss: 0.1057 - val_accuracy: 0.9927 - val_loss: 0.0288
Epoch 9/15
981/981 — 9s 9ms/step - accuracy: 0.9700 - loss: 0.0961 - val_accuracy: 0.9948 - val_loss: 0.0263
Epoch 10/15
981/981 — 8s 9ms/step - accuracy: 0.9726 - loss: 0.0845 - val_accuracy: 0.9948 - val_loss: 0.0239
Epoch 11/15
981/981 — 9s 9ms/step - accuracy: 0.9737 - loss: 0.0834 - val_accuracy: 0.9955 - val_loss: 0.0204
Epoch 12/15
981/981 — 9s 9ms/step - accuracy: 0.9766 - loss: 0.0777 - val_accuracy: 0.9950 - val_loss: 0.0214
Epoch 13/15
981/981 — 9s 9ms/step - accuracy: 0.9777 - loss: 0.0726 - val_accuracy: 0.9955 - val_loss: 0.0170
Epoch 14/15
981/981 — 9s 9ms/step - accuracy: 0.9792 - loss: 0.0707 - val_accuracy: 0.9943 - val_loss: 0.0205
Epoch 15/15
981/981 — 9s 9ms/step - accuracy: 0.9766 - loss: 0.0746 - val_accuracy: 0.9964 - val_loss: 0.0180

Confusion Matrix:
[[ 58   1   0 ...   0   0   0]
 [   0 715   0 ...   0   0   0]
 [   0   7 741 ...   0   0   0]
 ...
 [   0   0   0 ...  84   0   0]
 [   0   0   0 ...   0  39   0]
 [   0   0   0 ...   0   1  89]]
Accuracy: 0.971971
Precision: 0.966679
Recall: 0.951783
F1 score: 0.955791
```

b) Filter when first layer is 16 and Second layer is 32

Code:

```
# Building the model
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(5,5), activation='relu',
input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
model.summary()
```

Output:

```
Epoch 1/15
981/981 ————— 7s 5ms/step - accuracy: 0.3062 - loss: 2.5643 - val_accuracy: 0.8769 - val_loss: 0.4349
Epoch 2/15
981/981 ————— 4s 4ms/step - accuracy: 0.8059 - loss: 0.6210 - val_accuracy: 0.9625 - val_loss: 0.1699
Epoch 3/15
981/981 ————— 4s 4ms/step - accuracy: 0.8867 - loss: 0.3653 - val_accuracy: 0.9776 - val_loss: 0.1044
Epoch 4/15
981/981 ————— 5s 5ms/step - accuracy: 0.9084 - loss: 0.2862 - val_accuracy: 0.9820 - val_loss: 0.0766
Epoch 5/15
981/981 ————— 5s 5ms/step - accuracy: 0.9244 - loss: 0.2359 - val_accuracy: 0.9875 - val_loss: 0.0654
Epoch 6/15
981/981 ————— 5s 5ms/step - accuracy: 0.9351 - loss: 0.2082 - val_accuracy: 0.9898 - val_loss: 0.0554
Epoch 7/15
981/981 ————— 5s 5ms/step - accuracy: 0.9428 - loss: 0.1743 - val_accuracy: 0.9885 - val_loss: 0.0466
Epoch 8/15
981/981 ————— 5s 5ms/step - accuracy: 0.9476 - loss: 0.1698 - val_accuracy: 0.9922 - val_loss: 0.0400
Epoch 9/15
981/981 ————— 5s 5ms/step - accuracy: 0.9499 - loss: 0.1616 - val_accuracy: 0.9925 - val_loss: 0.0364
Epoch 10/15
981/981 ————— 5s 5ms/step - accuracy: 0.9538 - loss: 0.1445 - val_accuracy: 0.9927 - val_loss: 0.0321
Epoch 11/15
981/981 ————— 5s 5ms/step - accuracy: 0.9579 - loss: 0.1334 - val_accuracy: 0.9908 - val_loss: 0.0369
Epoch 12/15
981/981 ————— 5s 5ms/step - accuracy: 0.9582 - loss: 0.1347 - val_accuracy: 0.9941 - val_loss: 0.0281
Epoch 13/15
981/981 ————— 4s 5ms/step - accuracy: 0.9597 - loss: 0.1234 - val_accuracy: 0.9935 - val_loss: 0.0262
Epoch 14/15
981/981 ————— 5s 5ms/step - accuracy: 0.9625 - loss: 0.1153 - val_accuracy: 0.9948 - val_loss: 0.0237
Epoch 15/15
981/981 ————— 5s 5ms/step - accuracy: 0.9630 - loss: 0.1149 - val_accuracy: 0.9934 - val_loss: 0.0275
```

Confusion Matrix:

```
[[ 60  0  0 ...  0  0  0]
 [  0 711  0 ...  0  0  0]
 [  0  10 738 ...  0  0  0]
 ...
 [  0  0  0 ... 88  0  0]
 [  0  0  0 ...  0 48  0]
 [  0  0  0 ...  0  0 90]]
```

Accuracy: 0.962154  
Precision: 0.944546  
Recall: 0.943314  
F1 score: 0.942022

#### 4) Changing the dropout rate

Dropout is a regularization technique that helps prevent overfitting by randomly deactivating some neurons during training. By adjusting the dropout rate, you can control how much regularization is applied.

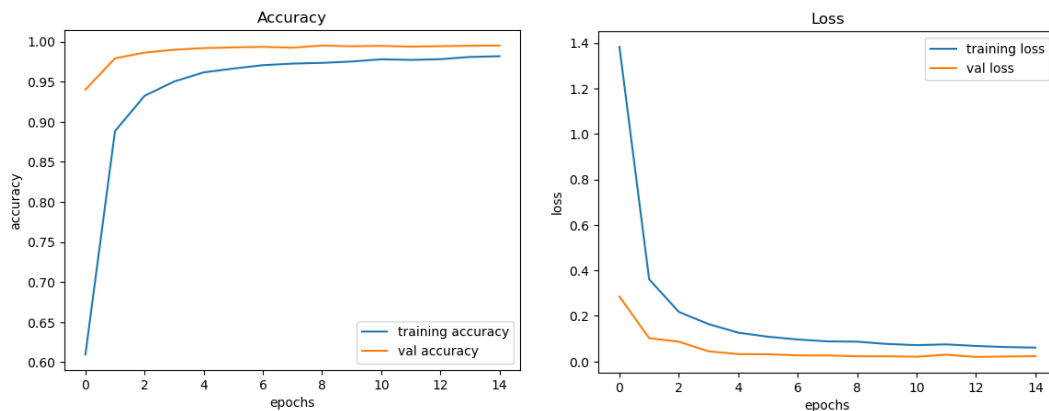
The reduced performance in the second code suggests that the increased dropout rate may have led to some underfitting, where the model was unable to capture the complexity of the data as effectively as it did with a lower dropout rate.

##### a) Default dropout rate which was used

Code:

```
# Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
model.summary()
```

Output:



```
Confusion Matrix:
[[ 58   1   0 ...   0   0   0]
 [   0 715   0 ...   0   0   0]
 [   0   7 741 ...   0   0   0]
 ...
 [   0   0   0 ...  84   0   0]
 [   0   0   0 ...   0  39   0]
 [   0   0   0 ...   0   1  89]]
Accuracy: 0.971971
Precision: 0.966679
Recall: 0.951783
F1 score: 0.955791
```

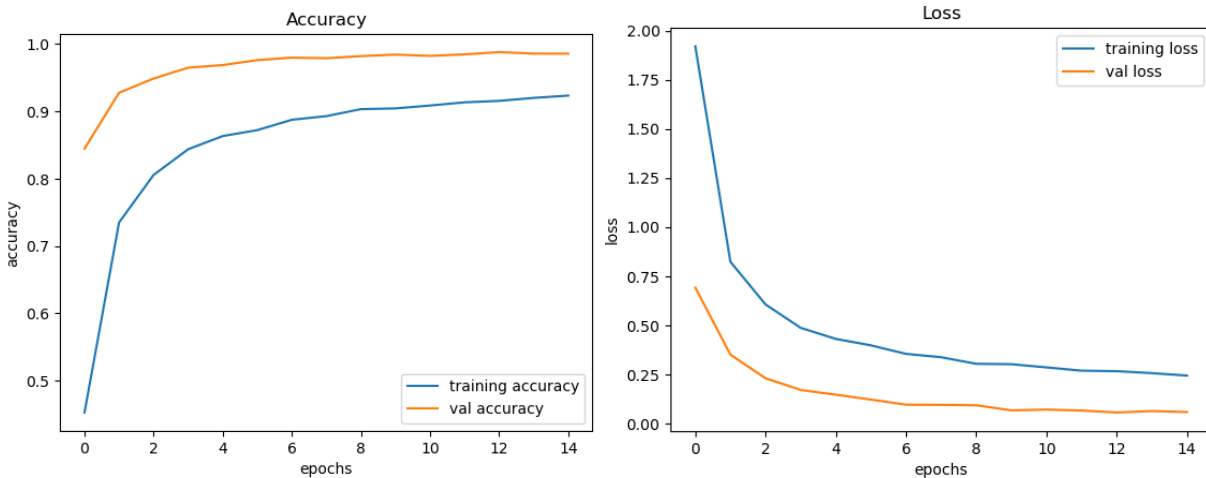


b) Increasing the dropout rate

Code:

```
# Building the model with increased dropout rate
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(5,5), activation='relu',
input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.35)) # Increased from 0.25 to 0.35
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.35)) # Increased from 0.25 to 0.35
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.6)) # Increased from 0.5 to 0.6
model.add(Dense(43, activation='softmax'))
model.summary()
```

Output:



Confusion Matrix:

```
[[ 60  0  0 ...  0  0  0]
 [  1 688 21 ...  0  0  0]
 [  0  5 742 ...  0  0  0]
 ...
 [  0  0  2 ... 84  0  0]
 [  0  0  0 ...  0 34  0]
 [  0  0  0 ...  0  1 87]]
```

Accuracy: 0.951148

Precision: 0.934684

Recall: 0.920860

F1 score: 0.924028