

**Connected Autonomous Vehicles**

**ELEC-8900-110 Special Topics**

**“Assignment 2”**

**Submitted by**

Vijayarangan Pandurengadurai Raju 110128279

**Submitted to:**

Dr. Ning Zhang

**Submitted on:**

14th 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.

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer program

Description automatically generated

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](https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign) 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.

A screenshot of a computer

Description automatically generated

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())

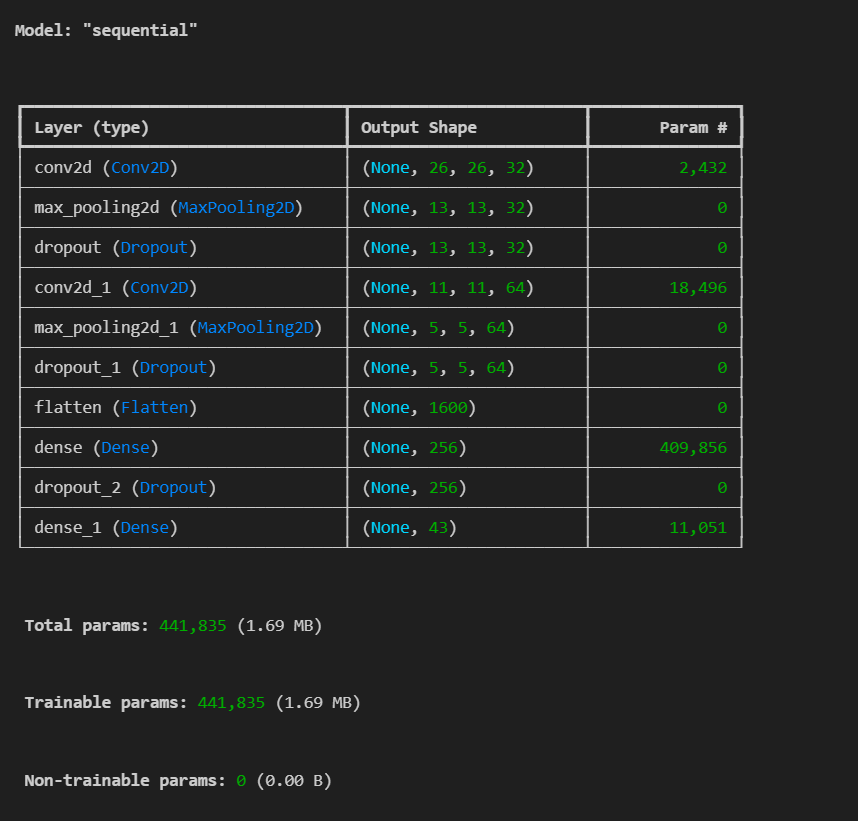
model.add(Dense(256, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(43, activation='softmax'))

model.summary()

Summary of the model:



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')

A screenshot of a computer program

Description automatically generated

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

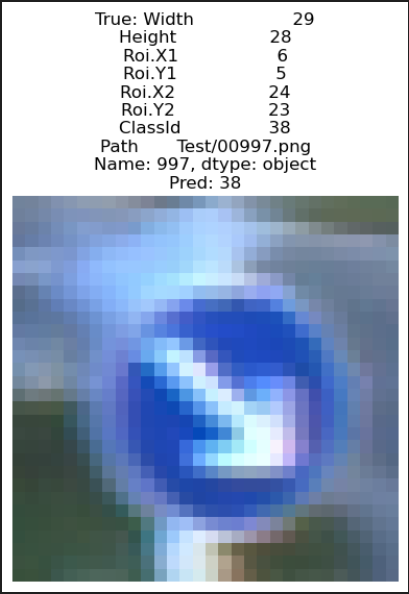
plt.imshow(sample\_image.reshape(30, 30, 3))  # Adjust reshape if your images are a different size

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.

* 1. 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:

A graph of accuracy and training

Description automatically generatedA graph of loss and loss

Description automatically generated

* 1. 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:

A graph of accuracy and training

Description automatically generated A graph of loss and loss

Description automatically generated

* 1. 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

A graph of accuracy and training accuracy

Description automatically generated with medium confidenceA graph of loss and loss

Description automatically generated

1. 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

* 1. 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:

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated

* 1. 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:

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

1. 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:

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

1. 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.

1. 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:

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated

1. 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())

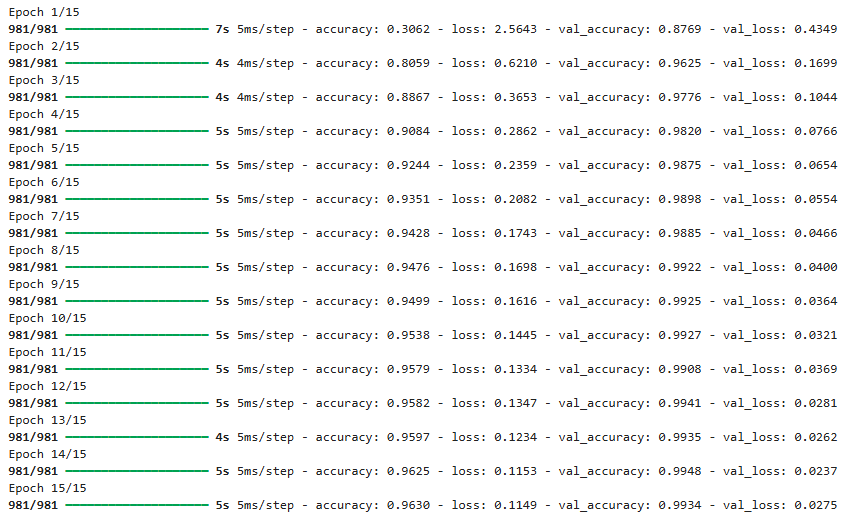
model.add(Dense(256, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(43, activation='softmax'))

model.summary()

Output:



A screenshot of a computer code

Description automatically generated

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.

1. 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:

A graph of accuracy and training

Description automatically generatedA graph of loss and loss

Description automatically generated

A screenshot of a computer code

Description automatically generated

1. 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:

A graph of a performance

Description automatically generated with medium confidenceA graph of loss and loss

Description automatically generated

A screenshot of a computer code

Description automatically generated