

Winter in Data Science (WiDS): AI for Self-Driving Car

Build a Traffic Sign Classifier

Submitted by: Pawni Parashar

Roll No.: 25b2456

Department: Metallurgical Engineering and Materials Science
IIT Bombay

Contents

1 Abstract	3
2 Introduction: The Role of AI in Mobility	3
3 Week 1 & 2: Python Programming & EDA Foundations	3
3.1 Objective	3
3.2 Core Competencies I Learnt	3
3.3 Numerical Operations with NumPy	4
4 Mathematics of Learning: Gradient Descent and Loss	4
4.1 Cost Function	4
4.2 Gradient Descent	4
5 Assignment 1: Exploratory Data Analysis & Cleaning	5
6 Assignment 2: Neural Networks for Regression	5
6.1 Data Preprocessing Logic	5
7 Final Assignment: Traffic Sign Recognition Capstone	7
7.1 Data Pipeline & Preprocessing	7
7.2 Understanding the Architecture	8
8 Technical Challenges & Solutions	8
9 Future Scope	8
10 Conclusion	9
11 Tech Stack Summary	9

1 Abstract

This report documents the comprehensive learning journey undertaken during the "Winter in Data Science" (WiDS) program. The primary objective of this project was to develop a deep understanding of Data Science pipelines, ranging from fundamental Python programming to advanced Deep Learning architectures tailored for computer vision in autonomous driving.

The project culminated in the development of a Traffic Sign Recognition System, a Convolutional Neural Network (CNN) capable of identifying 43 distinct classes of traffic signs with a test accuracy of approximately 98.7%. This system mimics the visual perception capabilities required for self-driving cars. Additionally, intermediate projects involved predicting vehicle fuel efficiency using regression Artificial Neural Networks (ANNs) and handwriting recognition using CNNs. This report details the methodologies, code implementations, and performance metrics for each stage of the curriculum.

2 Introduction: The Role of AI in Mobility

Artificial Intelligence (AI) serves as the "brain" of a self-driving car. While sensors like LiDAR and Radar provide distance data, visual recognition relies entirely on Deep Learning models.

- **Perception:** Identifying lane lines, traffic lights, and signs.
- **Prediction:** Forecasting the movement of pedestrians and other vehicles.
- **Planning:** Making real-time decisions based on perception and prediction.

This project focuses on the Perception layer, specifically utilizing Convolutional Neural Networks (CNNs). Unlike traditional algorithms that require manual feature extraction, CNNs learn to identify relevant features (edges, shapes, textures) directly from raw image data, making them robust to variations in lighting and orientation.

3 Week 1 & 2: Python Programming & EDA Foundations

3.1 Objective

The foundation of any data science pipeline is efficient coding. Week 1 was dedicated to mastering Python 3.x, focusing on writing modular, readable, and optimized code suitable for handling large datasets.

3.2 Core Competencies I Learnt

- **Lists & Tuples:** Used for sequencing data. I learned that Tuples are immutable (faster for fixed data), while Lists are mutable (ideal for dynamic data collection).
- **Dictionaries:** Essential for mapping unique keys to values, simulating JSON-like data often used in API responses.
- **Sets:** Utilized for operations requiring unique elements, such as finding distinct classes in a dataset.
- **Control Flow:** Mastered if-elif-else constructs and implemented for and while loops to iterate over training epochs.

3.3 Numerical Operations with NumPy

For heavy mathematical computations, I used NumPy. It enables vectorization, allowing operations on entire arrays at once rather than using slow loops. This is the underlying engine for all Deep Learning tensors. I learned that NumPy's ability to handle multi-dimensional arrays efficiently is what makes modern AI possible.

4 Mathematics of Learning: Gradient Descent and Loss

I studied the core mathematical optimization that powers neural networks.

4.1 Cost Function

I learned that for Regression, we use the Mean Squared Error (MSE):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

This function calculates the discrepancy between the predicted value and the actual target.

4.2 Gradient Descent

To minimize this cost, I studied Gradient Descent. The logic is to take steps proportional to the negative of the gradient of the function at the current point.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Where α is the Learning Rate. This iterative process allows the weights of the neural network to adjust until the error is minimized.

5 Assignment 1: Exploratory Data Analysis & Cleaning

In this assignment, I utilized the Pandas library and visualization tools to understand the dataset structure.

```
1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 # Load data
7 df = sns.load_dataset('mpg')
8
9 # I learned to identify missing values using isnull().sum()
10 print("Missing values before cleaning:\n", df.isnull().sum())
11
12 # Cleaning: Handle gaps by dropping rows or imputing values
13 df = df.dropna()
14
15 # Numerical Operations: Vectorization with NumPy
16 # I learned that operations on entire arrays are faster than loops
17 weights_in_kg = df['weight'].values * 0.453592
18
19 # Data Visualization
20 plt.figure(figsize=(10, 6))
21 sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
22 plt.title('Correlation Matrix of Vehicle Features')
23 plt.show()
24
25 # Pair Plots: To see the distribution of data classes
26 sns.pairplot(df[['mpg', 'horsepower', 'weight', 'acceleration']])
27 plt.show()
```

Listing 1: Full Implementation of EDA and Data Cleaning

6 Assignment 2: Neural Networks for Regression

The second major project involved Regression—predicting a continuous numerical value. I aimed to predict a car’s Miles Per Gallon (MPG) based on its physical attributes.

6.1 Data Preprocessing Logic

I learned that Neural Networks struggle when input features have vastly different ranges. I used Scikit-Learn’s StandardScaler to normalize all features. This scales data so that the mean is 0 and standard deviation is 1, ensuring the Gradient Descent algorithm converges faster.

```
1 import pandas as pd
2 import seaborn as sns
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import Dense
7
8 # Load and clean data
9 df = sns.load_dataset('mpg')
10 df = df.dropna()
11
12 features = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']
13 X = df[features]
14 y = df['mpg']
15
16 # Splitting 80% for Training, 20% for Testing
```

```

17 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
18         random_state=42)
19 # Normalization: I learned that scaling is critical for convergence
20 scaler = StandardScaler()
21 X_train_scaled = scaler.fit_transform(X_train)
22 X_test_scaled = scaler.transform(X_test)
23
24 # Model Architecture (ANN)
25 # I built a Feed-Forward Neural Network using Keras
26 model = Sequential([
27     Dense(64, activation='relu', input_shape=(len(features),)),
28     Dense(32, activation='relu'),
29     Dense(1) # Single output for regression
30 ])
31
32 model.compile(optimizer='adam', loss='mse', metrics=['mae'])
33 history = model.fit(X_train_scaled, y_train, epochs=100, validation_split=0.2,
34                     verbose=0)
35 # Results Analysis
36 # I used Mean Absolute Error (MAE) to evaluate performance.
37 # The final MAE was approximately 2.95 MPG.

```

Listing 2: Full Implementation of MPG Prediction Model

7 Final Assignment: Traffic Sign Recognition Capstone

The final capstone brought all previous learnings together to solve a real-world autonomous driving problem: classifying traffic signs using the German Traffic Sign Recognition Benchmark (GTSRB).

7.1 Data Pipeline & Preprocessing

Real-world images vary in size and lighting. We built a rigorous preprocessing pipeline using PIL and NumPy.

```
1 import numpy as np
2 import pandas as pd
3 import os
4 import cv2
5 from PIL import Image
6 import matplotlib.pyplot as plt
7 from sklearn.model_selection import train_test_split
8 from tensorflow.keras.utils import to_categorical
9 from tensorflow.keras.models import Sequential, load_model
10 from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
11
12 data_dir = '/kaggle/input/gtsrb-german-traffic-sign'
13 train_path = os.path.join(data_dir, 'Train')
14 IMG_HEIGHT = 30
15 IMG_WIDTH = 30
16 NUM_CATEGORIES = 43
17
18 data = []
19 labels = []
20
21 # Logic: Iterated through 43 subdirectories
22 for i in range(NUM_CATEGORIES):
23     path = os.path.join(train_path, str(i))
24     images = os.listdir(path)
25     for img in images:
26         try:
27             image = Image.open(os.path.join(path, img))
28             image = image.resize((IMG_HEIGHT, IMG_WIDTH))
29             image = np.array(image)
30             data.append(image)
31             labels.append(i)
32         except:
33             print("Error loading image")
34
35 data = np.array(data)
36 labels = np.array(labels)
37
38 # Split and Normalize
39 X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.2,
40 random_state=42)
41 X_train = X_train / 255.0
42 X_val = X_val / 255.0
43
44 # One-hot encode the labels
45 y_train = to_categorical(y_train, NUM_CATEGORIES)
46 y_val = to_categorical(y_val, NUM_CATEGORIES)
47
48 # Model Building
49 model = Sequential([
50     Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=(30, 30,
51     3)),
52     Conv2D(filters=32, kernel_size=(5,5), activation='relu'),
53     MaxPool2D(pool_size=(2, 2)),
54     Dropout(rate=0.25),
```

```

53     Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
54     Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
55     MaxPool2D(pool_size=(2, 2)),
56     Dropout(rate=0.25),
57
58     Flatten(),
59     Dense(256, activation='relu'),
60     Dropout(rate=0.5),
61     Dense(NUM_CATEGORIES, activation='softmax')
62 )
63 )
64
65 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
66 history = model.fit(X_train, y_train, batch_size=32, epochs=15, validation_data=(X_val, y_val))
67
68 # Save the model
69 model.save("traffic_classifier.h5")

```

Listing 3: Full Implementation of Traffic Sign Classifier

7.2 Understanding the Architecture

During this project, I understood the specific roles of each layer in a CNN:

- **Conv2D:** Learns spatial hierarchies. The first layers detect edges, while deeper layers detect shapes like circles and triangles.
- **MaxPool2D:** Reduces spatial dimensions, which helps in reducing computation and making the model invariant to small translations.
- **Dropout:** I learned that this is a regularization technique where neurons are randomly ignored during training to prevent overfitting.

8 Technical Challenges & Solutions

- **Overfitting:** Early experiments showed high training accuracy but lower validation accuracy. I introduced Dropout layers (0.25 and 0.5) to regularize the model.
- **Data Imbalance:** Some traffic signs (like speed limits) are more common. I validated the model using a separate test set to ensure robustness across all classes.
- **Input Consistency:** Images had different resolutions. The preprocessing pipeline resized all inputs to 30x30, ensuring compatibility.

9 Future Scope

- **Real-Time Detection:** Integrating this classifier with YOLO (You Only Look Once) to detect signs in a live video stream.
- **Edge Deployment:** Optimizing the model using TensorFlow Lite for hardware like Raspberry Pi.
- **Data Augmentation:** Artificially expanding the dataset with rotations and blurs to simulate bad weather.

10 Conclusion

The WiDS program has been a comprehensive journey into Data Science. Starting from basic Python scripts, I advanced to building neural networks that can essentially "see." The successful development of the Traffic Sign Recognition System solidified my understanding of the Deep Learning pipeline—from data cleaning to model evaluation.

11 Tech Stack Summary

- **Language:** Python 3.12
- **Manipulation:** Pandas, NumPy
- **Deep Learning:** TensorFlow, Keras
- **Computer Vision:** OpenCV, PIL (Pillow)
- **Utilities:** Scikit-Learn, Matplotlib, Seaborn