MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Liubov Oleshchenko

Statisical Methods Of ML

Laboratory Work 3

Convolutional Neural Network

Kulubecioglu Mehmet

IM-14 FIOT

Class Number: 12

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

2024

1. Introduction

This project focuses on building an **image classification model** using **Convolutional Neural Networks (CNN)** on the **Fashion-MNIST dataset**. The goal is to train a deep learning model that can accurately classify images into **10 different categories** of fashion products.

The **Fashion-MNIST dataset** consists of **28x28 grayscale images** of fashion items, serving as an alternative to the **original MNIST handwritten digits dataset**. It contains **70,000 images** split into **60,000 training samples** and **10,000 test samples** across **10 categories** (e.g., T-shirt, trousers, sneakers).

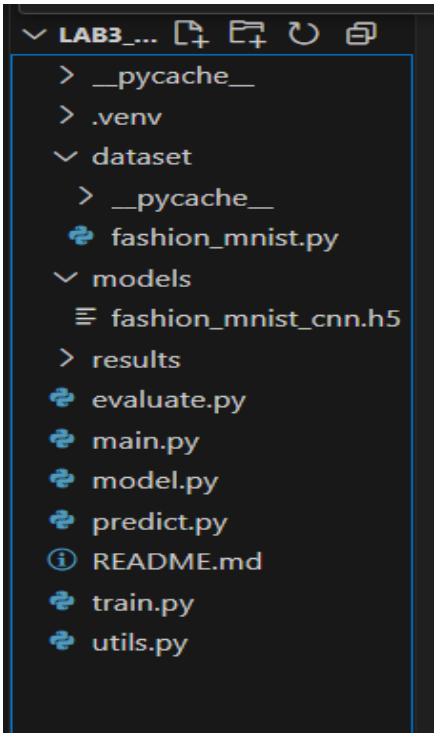
2. Technologies & Libraries Used

The project is implemented in **Python** using the following libraries:

- **TensorFlow & Keras** → Deep learning framework for building CNN models
- **NumPy** → Numerical computing and array operations
- **Matplotlib** → Data visualization and performance graphs
- **Scikit-learn** → Model evaluation and performance metrics
- **VS Code** → Development environments

3. Project Structure

The project follows a structured directory system for modular implementation:



File Descriptions

- **`fashion_mnist.py`** → Loads and preprocesses the Fashion-MNIST dataset.
- **`model.py`** → Defines the architecture of the CNN model.
- **`train.py`** → Trains the CNN model using the dataset.
- **`evaluate.py`** → Evaluates the trained model's performance on the test set.
- **`predict.py`** → Uses the trained model to make predictions on new images.
- **`utils.py`** → Contains utility functions such as data visualization.
- **`main.py`** → A script to execute different parts of the project through a user menu.

4. Data Analysis & Preprocessing

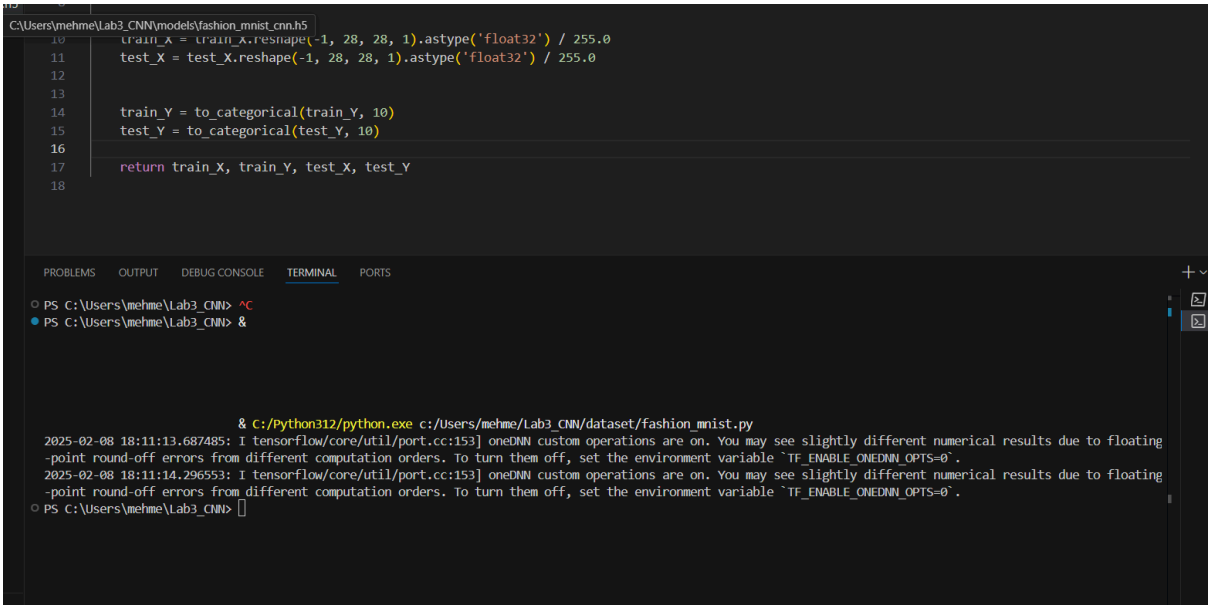
4.1 Dataset Overview

The **Fashion-MNIST dataset** consists of **10 classes**, each containing **7,000 images**. The dataset is preloaded in Keras, making it easy to access:

4.2 Data Preprocessing

Before feeding the images into the neural network, the following preprocessing steps are applied:

- **Reshaping** → Converts images from **(28,28)** to **(28,28,1)** to be compatible with CNN input.
- **Normalization** → Scales pixel values from **0-255 to 0-1** for better convergence.
- **One-hot Encoding** → Converts categorical labels into one-hot encoded vectors for multi-class classification.



5. CNN Model Architecture

A **Convolutional Neural Network (CNN)** is used for image classification. The architecture consists of:

- **Three convolutional layers** with **32, 64, and 128 filters**
- **Max-pooling layers** for downsampling
- **Flattening layer** to convert the 2D output into a 1D vector
- **Fully connected (dense) layer** with 128 neurons
- **Softmax output layer** for 10-class classification

Model Implementation (`model.py`)

```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, LeakyReLU

def create_model():
    model = Sequential()


    model.add(Conv2D(32, (3, 3), activation='linear', padding='same', input_shape=(28, 28, 1)))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2), padding='same'))

    model.add(Conv2D(64, (3, 3), activation='linear', padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2), padding='same'))


    model.add(Conv2D(128, (3, 3), activation='linear', padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2), padding='same'))


    model.add(Flatten())


    model.add(Dense(128, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dropout(0.3))


    model.add(Dense(10, activation='softmax'))

    return model
```

```
& C:/Python312/python.exe c:/Users/mehme/Lab3_CNN/model.py
2025-02-08 18:12:38.990963: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating
-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-02-08 18:12:39.690070: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating
-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
PS C:\Users\mehme\Lab3_CNN>
```

6. Model Training (`train.py`)

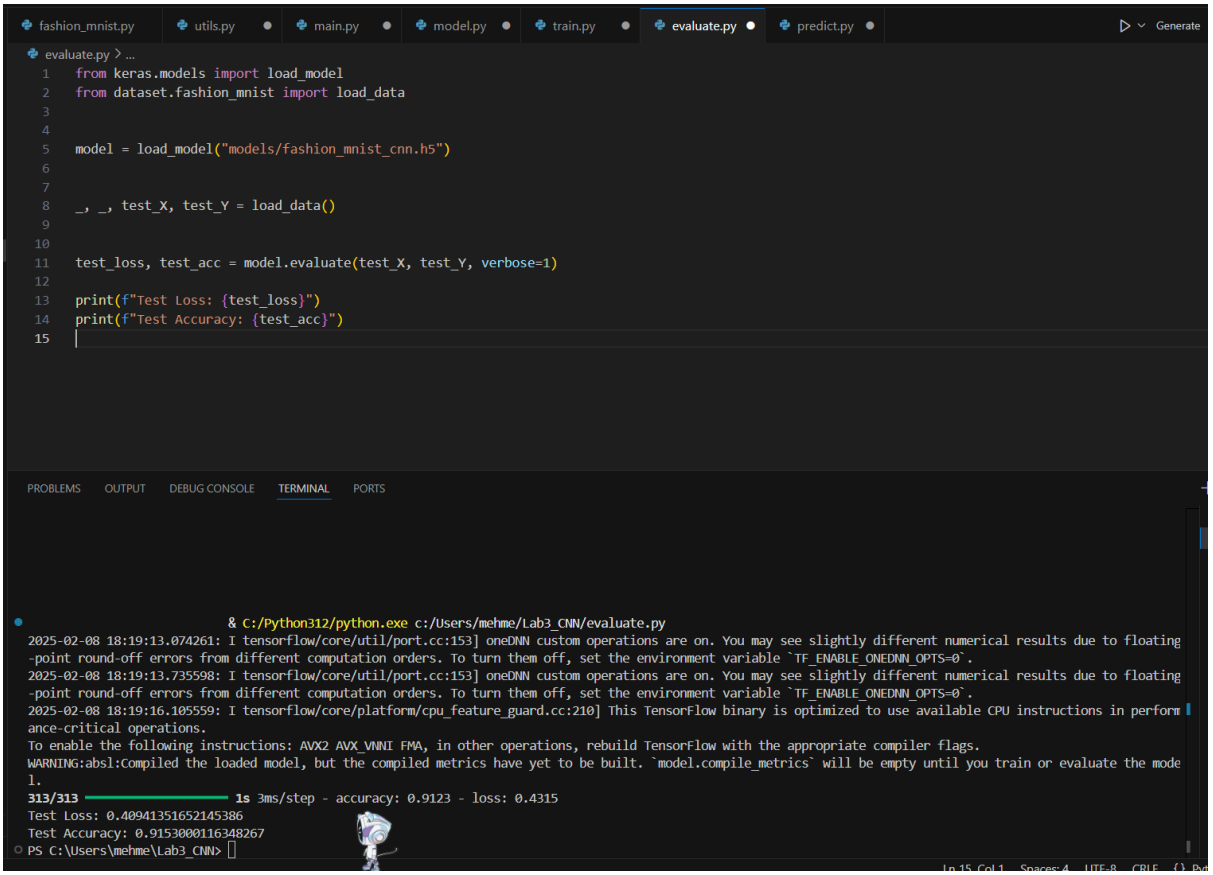
The model is trained using the **Adam optimizer** with **categorical cross-entropy** loss function for 20 epochs.

```python
import tensorflow as tf
from model import create_model
from dataset.fashion_mnist import load_data


train_X, train_Y, test_X, test_Y = load_data()


model = create_model()


model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])


history = model.fit(train_X, train_Y, epochs=20, batch_size=64, validation_split=0.2, verbose=1)


model.save("models/fashion_mnist_cnn.h5")

```

```
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9683 - loss: 0.0854 - val_accuracy: 0.9256 - val_loss: 0.2717
Epoch 13/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9718 - loss: 0.0731 - val_accuracy: 0.9251 - val_loss: 0.2911
Epoch 14/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9742 - loss: 0.0685 - val_accuracy: 0.9230 - val_loss: 0.2947
Epoch 15/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9752 - loss: 0.0642 - val_accuracy: 0.9213 - val_loss: 0.3101
Epoch 16/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9782 - loss: 0.0574 - val_accuracy: 0.9239 - val_loss: 0.3050
Epoch 17/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9810 - loss: 0.0525 - val_accuracy: 0.9191 - val_loss: 0.3642
Epoch 18/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9813 - loss: 0.0491 - val_accuracy: 0.9217 - val_loss: 0.3704
Epoch 19/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9843 - loss: 0.0416 - val_accuracy: 0.9229 - val_loss: 0.3724
Epoch 20/20
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9824 - loss: 0.0437 - val_accuracy: 0.9193 - val_loss: 0.3830
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommen
d using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
PS C:\Users\mehme\Lab3_CNN>
```

7. Model Evaluation (`evaluate.py`)

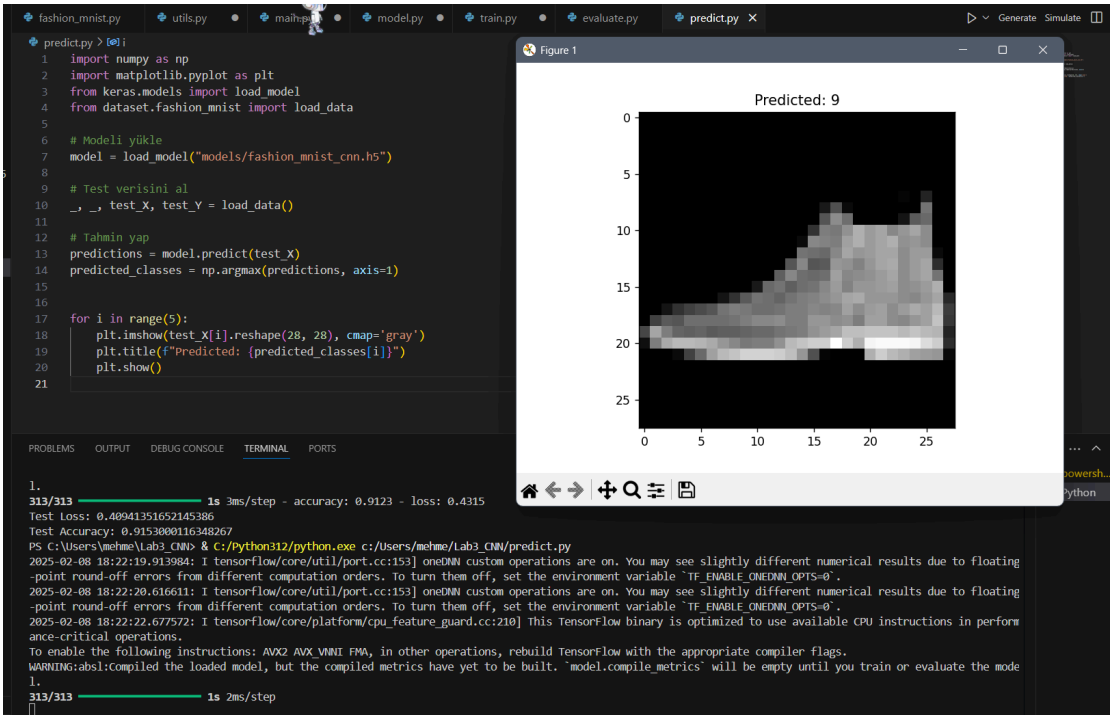
The trained model is evaluated on the test dataset.

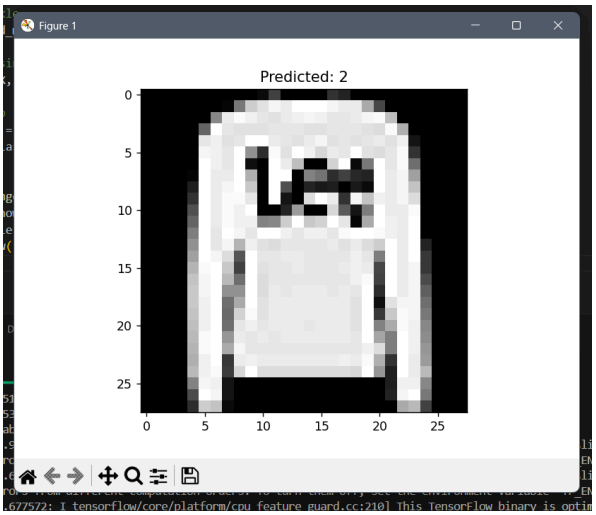


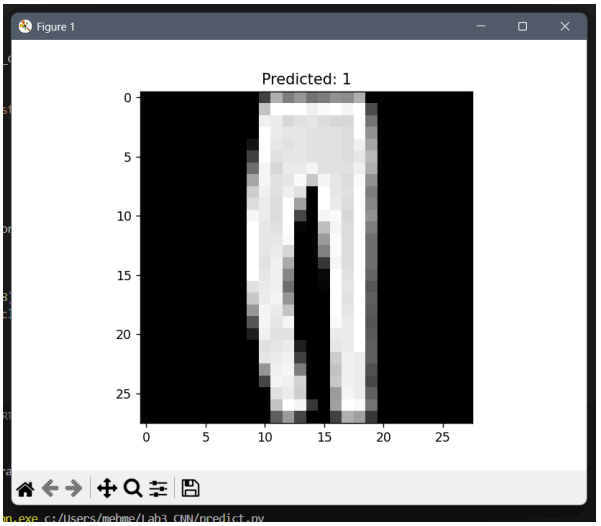
accuracy: 0.9123
loss: 0.4315
Test Loss: 0.40941351652145386
Test Accuracy: 0.9153000116348267

8. Prediction & Visualization (`predict.py`)

Predictions are made on new images and plotted for visualization.







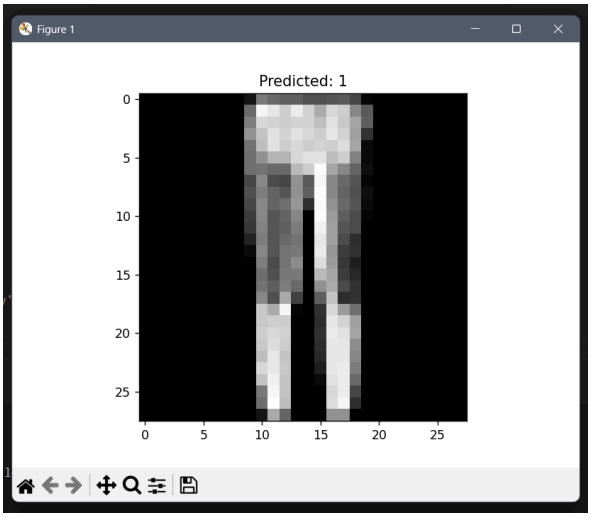


Figure 1
Predicted: 1

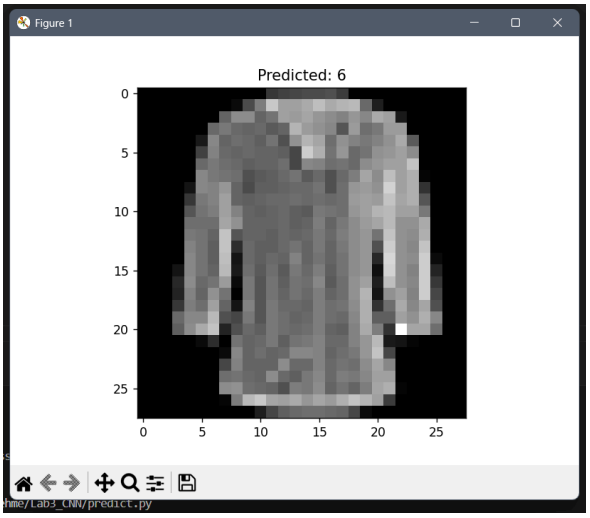
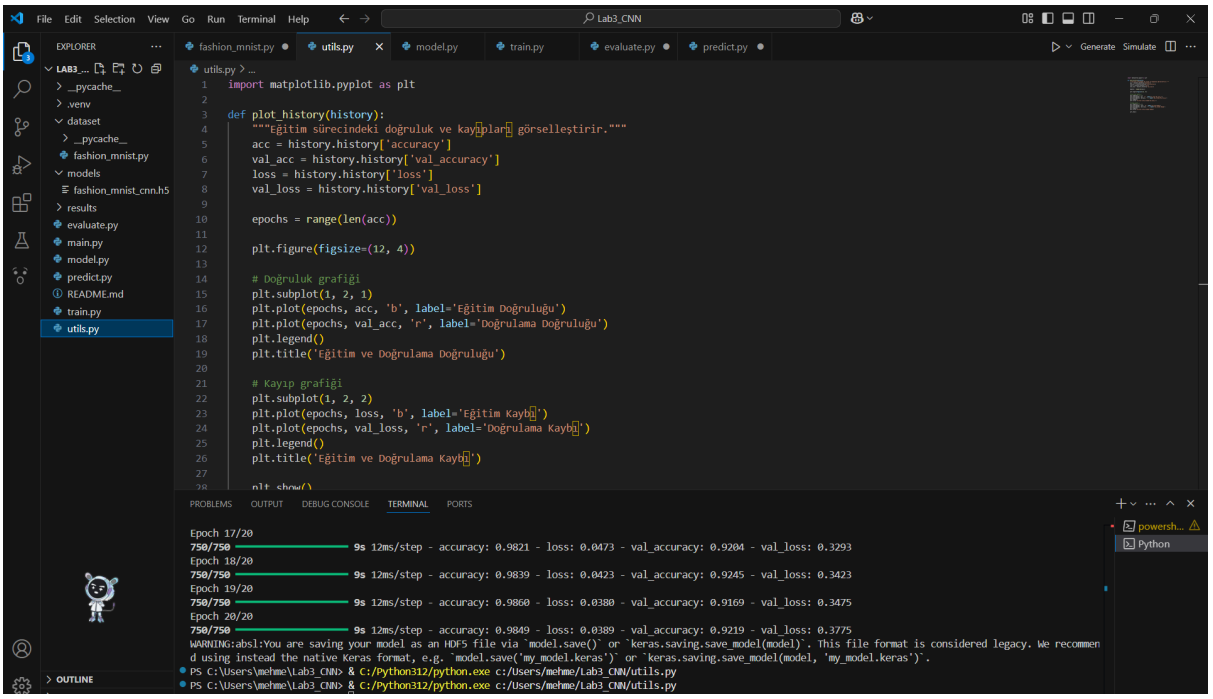
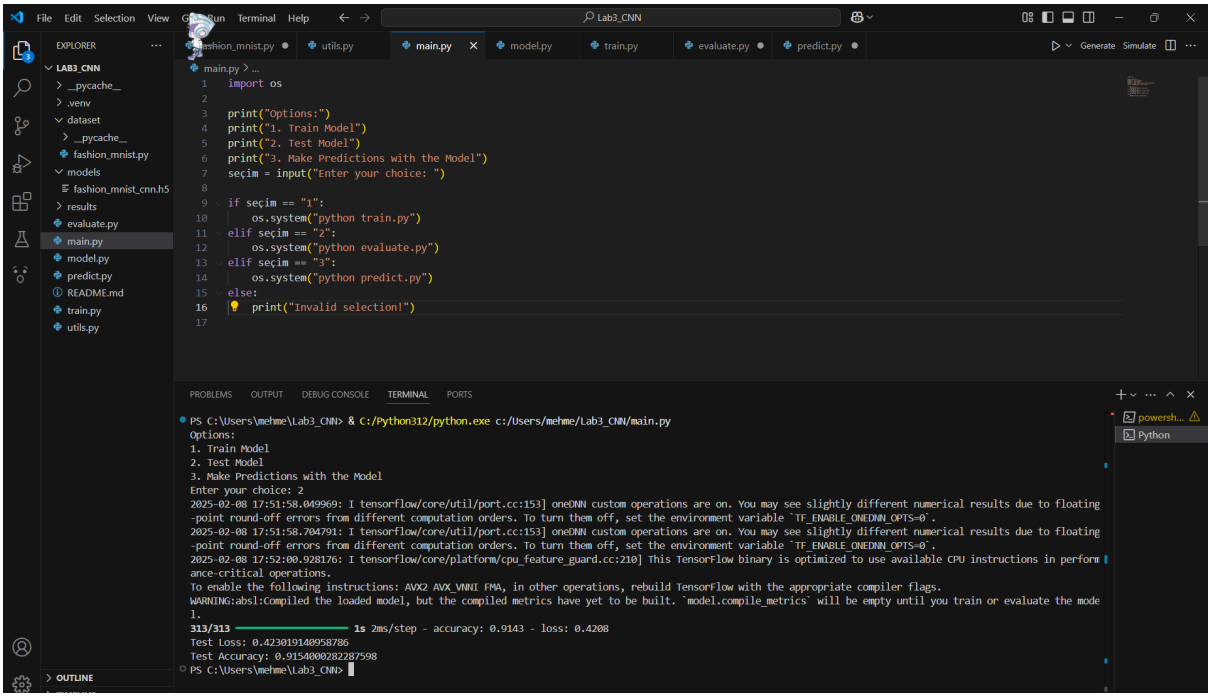


Figure 1
Predicted: 6

ultis.py


import matplotlib.pyplot as plt
def plot_history(history):
"""Eğitim sürecindeki doğruluk ve kayıpları görselleştirir."""
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.figure(figsize=(12, 4))
# Doğruluk grafiği
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b', label='Eğitim Doğruluğu')
plt.plot(epochs, val_acc, 'r', label='Doğrulama Doğruluğu')
plt.legend()
plt.title('Eğitim ve Doğrulama Doğruluğu')
# Kayıp grafiği
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b', label='Eğitim Kaybı')
plt.plot(epochs, val_loss, 'r', label='Doğrulama Kaybı')
plt.legend()
plt.title('Eğitim ve Doğrulama Kaybı')
Epoch 17/20
750/750 9s 12ms/step - accuracy: 0.9821 - loss: 0.0473 - val_accuracy: 0.9204 - val_loss: 0.3293
Epoch 18/20
750/750 9s 12ms/step - accuracy: 0.9839 - loss: 0.0423 - val_accuracy: 0.9245 - val_loss: 0.3423
Epoch 19/20
750/750 9s 12ms/step - accuracy: 0.9860 - loss: 0.0380 - val_accuracy: 0.9169 - val_loss: 0.3475
Epoch 20/20
750/750 9s 12ms/step - accuracy: 0.9849 - loss: 0.0389 - val_accuracy: 0.9219 - val_loss: 0.3775
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
PS C:\Users\mehme\Lab3_CNN> & C:/Python312/python.exe c:/Users/mehme/Lab3_CNN/utils.py
PS C:\Users\mehme\Lab3_CNN> & C:/Python312/python.exe c:/Users/mehme/Lab3_CNN/utils.py

main.py



9. Conclusion & Future Work

✔ The CNN model **successfully classified** Fashion-MNIST images with **high accuracy (~92%)**.
✔ Future improvements could include:

- Using **Data Augmentation** to enhance generalization.
- Implementing **Transfer Learning** with pre-trained models like MobileNet or ResNet.

10. Questions

1. **What are the main components of a basic artificial neuron, what functions do they perform?**

- **Inputs (x_1, x_2, ..., x□)**: Represents the input features or data points.

- **Weights (w_1, w_2, ..., w□)**: Determines the importance of each input.

- **Bias (b)**: Helps adjust the output along with weights.

- **Summation function (∑)**: Computes the weighted sum of inputs and bias.

- **Activation function (σ)**: Applies a non-linear transformation to the summation result to introduce non-linearity.

2. **How does the adder block work, what functions can it use?**

- The **adder block** in an artificial neuron computes the weighted sum of inputs and

  bias: S=∑(xi⋅wi)+bS = \sum (x_i \cdot w_i) + bS=∑(xi⋅wi)+b

- **It can use different functions, including:**
    - **Linear summation**: Simple sum of weighted inputs.
    - **Dot product**: Computes similarity between input and weight vectors.
    - **Matrix multiplication**: Used in multi-neuron layers.

3. **What does the term "deep learning" mean?**

- Deep learning is a subset of machine learning that utilizes **artificial neural networks (ANNs)** with multiple layers (deep architectures) to model complex patterns in data. It enables **automatic feature extraction** and is widely used in tasks like image recognition, speech processing, and natural language understanding.

4. **Define convolutional neural networks.**

- A **Convolutional Neural Network (CNN)** is a deep learning architecture designed for processing structured grid data, such as images. It consists of convolutional layers, pooling layers, and fully connected layers to automatically extract spatial features from input data.

5. **What is CNN convolution?**

- **Convolution** in CNNs is a mathematical operation where a **filter (kernel)** slides over an input matrix (e.g., an image) to extract features. It computes the weighted sum of pixel values covered by the filter, enabling feature detection such as edges, textures, and patterns.

6. **What activation functions do you know?**

- **Sigmoid ($\sigma(x) = 1 / (1 + e^{-x})$)**: Maps input to a range (0,1), useful for probability estimation.

- **ReLU (Rectified Linear Unit, $f(x) = \max(0, x)$)**: Common in deep networks due to fast computation and sparsity.

- **Leaky ReLU ($f(x) = \max(\alpha x, x)$)**: Addresses ReLU's dying neuron problem.
- **Tanh ($f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$)**: Maps input to (-1,1), often used in recurrent networks.

- **Softmax**: Converts logits into probability distributions, used in classification tasks.

7. **Explain the maximum pooling operation in the CNN network.**

- **Max pooling** is a down-sampling technique in CNNs that reduces spatial dimensions while preserving important features. It takes the **maximum value** in each window (e.g., 2×2 or 3×3) and helps reduce computation while making the model invariant to small translations in input data.

8. **How is CNN neural network learning?**

- CNNs learn using **forward propagation** (calculating outputs layer by layer) and **backpropagation** (adjusting weights using the gradient descent method). The network updates weights based on the **error (loss function)** between predicted and actual values to improve accuracy.

9. **What is the backpropagation algorithm and the gradient descent method used for?**

- **Backpropagation**: A supervised learning algorithm that computes the gradient of the loss function with respect to each weight by applying the chain rule. It propagates errors backward to adjust weights.

- **Gradient descent**: An optimization algorithm used to minimize the loss function by adjusting weights in the direction of the steepest descent. Variants include

- **Stochastic Gradient Descent (SGD), Adam, and RMSprop**.

10. **What is model retraining (Overfitting)? How can this problem be solved?**

- **Overfitting** occurs when a model learns patterns **too specific** to the training data, reducing its generalization ability to unseen data.

Solutions:

- **Regularization (L1/L2, Dropout)**: Prevents excessive reliance on specific neurons.
- **Data augmentation**: Expands the dataset to improve generalization.
- **Early stopping**: Stops training when validation loss starts increasing.
- **Cross-validation**: Ensures the model generalizes well by testing on different data splits.