**1. Environment Setup**

* **Create a virtual environment:**  
  python -m venv tfenv
* **Activate the environment:**  
  tfenv\Scripts\activate
* **Install required packages:**
  + TensorFlow: pip install tensorflow
  + Matplotlib: pip install matplotlib
* *Note:* If using Conda, replace pip with conda.

**2. Importing Libraries**

**import** tensorflow **as** tf

**from** tensorflow.keras.datasets **import** mnist

**from** tensorflow.keras.models **import** Sequential

**from** keras.layers **import** Input, Flatten, Dense

**from** tensorflow.keras.utils **import** to\_categorical

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

* **TensorFlow/Keras:** For building and training the neural network.
* **Matplotlib:** For visualizing images and predictions.
* **NumPy:** For numerical operations.

**3. Loading and Preprocessing the Data**

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

* **MNIST Dataset:** Handwritten digit images (28x28 pixels).
* **Normalization:** Pixel values scaled to 1 for better training performance.

**Label Encoding**

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

* **One-hot encoding:** Converts digit labels (0-9) into binary class matrices.

**4. Model Architecture**

model = Sequential([

Input(shape=(28, 28)),

Flatten(),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

* **Input Layer:** Accepts 28x28 images.
* **Flatten Layer:** Converts 2D images to 1D vectors.
* **Dense Layer (128 units):** Fully connected layer with ReLU activation.
* **Output Layer (10 units):** Softmax activation for 10 digit classes.

**5. Model Compilation**

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

* **Optimizer:** Adam (adaptive learning rate).
* **Loss Function:** Categorical cross-entropy (for multi-class classification).
* **Metrics:** Accuracy.

**6. Model Training**

model.fit(x\_train, y\_train, epochs=5, validation\_split=0.1)

* **Epochs:** 5 passes through the training data.
* **Validation Split:** 10% of training data used for validation.

**Training Output Example**

* Shows accuracy and loss for each epoch on both training and validation sets.

**7. Model Evaluation**

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

**print**(f"\nTest Accuracy:{test\_acc:.4f}")

* **Evaluates** the model on the test set.
* **Prints** the test accuracy (e.g., 0.9775 or 97.75%).

**8. Making Predictions and Visualization**

predictions = model.predict(x\_test)

index = 0

plt.imshow(x\_test[index], cmap='gray')

plt.title(f"predicted:{np.argmax(predictions[index])}")

plt.axis('off')

plt.show()

* **Predicts** the digit for the first test image.
* **Displays** the image with the predicted label as the title.