**Experiment 01**

**Introduction to Deep Learning Frameworks: TensorFlow, Keras, and PyTorch**

Date: 11 Aug 2025 SAP ID: 500120453

**AIM:** To gain a foundational understanding of the architecture, syntax, and core functionalities of popular deep learning frameworks: TensorFlow, Keras, and PyTorch. This experiment also compares their approaches to building and training a simple neural network using a linear regression task.

**THEORY:**

Deep Learning frameworks provide the necessary tools to build, train, and evaluate neural networks. Among the most widely used are:

1. **TensorFlow**
   * Developed by Google, supports both low-level operations and high-level APIs.
   * Uses **Eager Execution** with tf.GradientTape() for automatic differentiation.
   * Provides flexibility for custom training loops and is well-suited for production deployment.
2. **Keras**
   * High-level API, originally independent but now tightly integrated with TensorFlow.
   * Offers **Sequential** and **Functional** APIs for rapid model development.
   * Abstracts away low-level details, making it beginner-friendly and concise.
3. **PyTorch**
   * Developed by Facebook’s AI Research lab.
   * Uses **define-by-run (dynamic computational graph)**, making it intuitive and Pythonic.
   * Favored in research due to easy debugging, clear OOP design, and native Python integration.

**Linear Regression Task:**  
We trained a simple linear regression model on a synthetic dataset:

y=2x+1+noise

to estimate the parameters W (slope) and b (intercept).

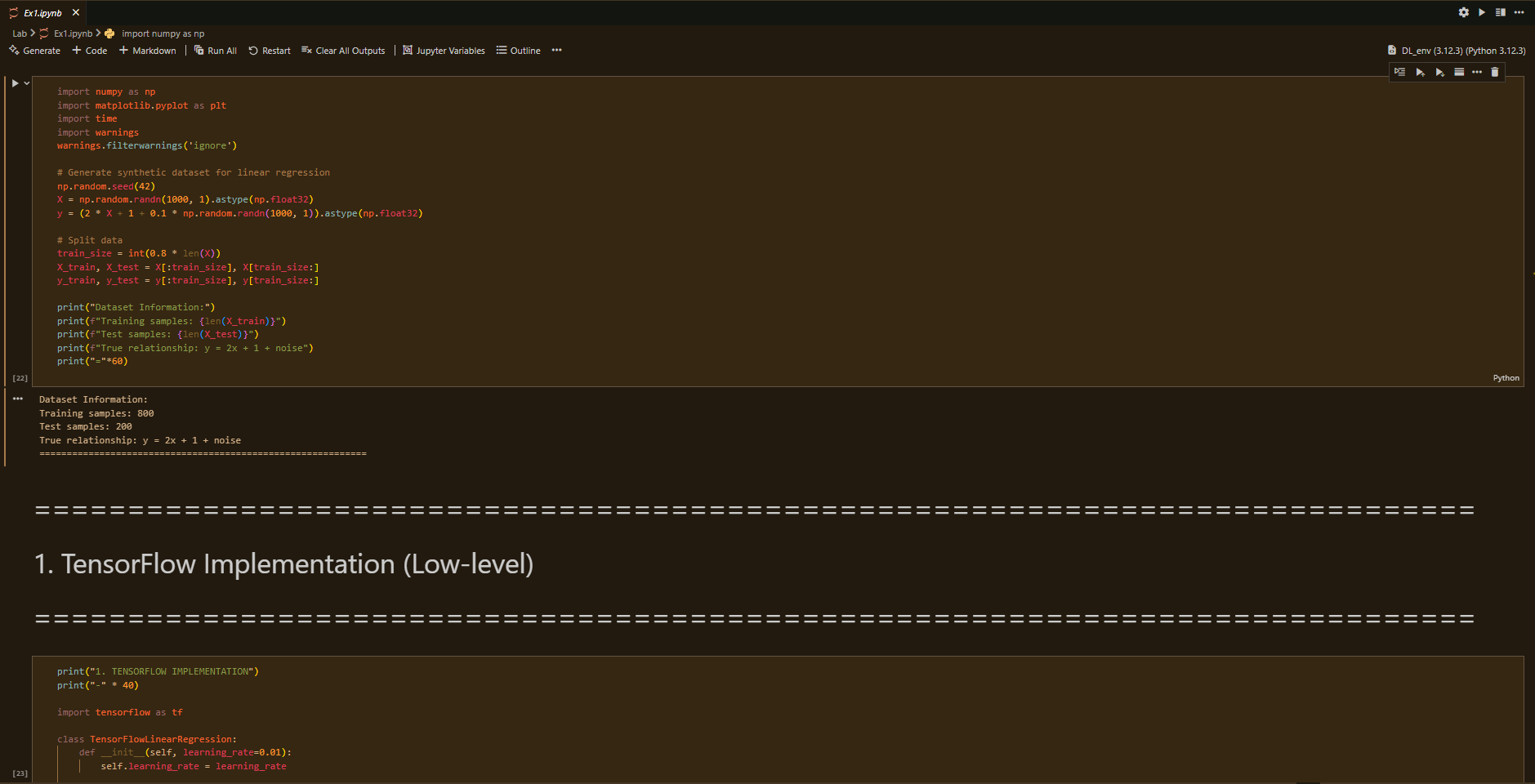
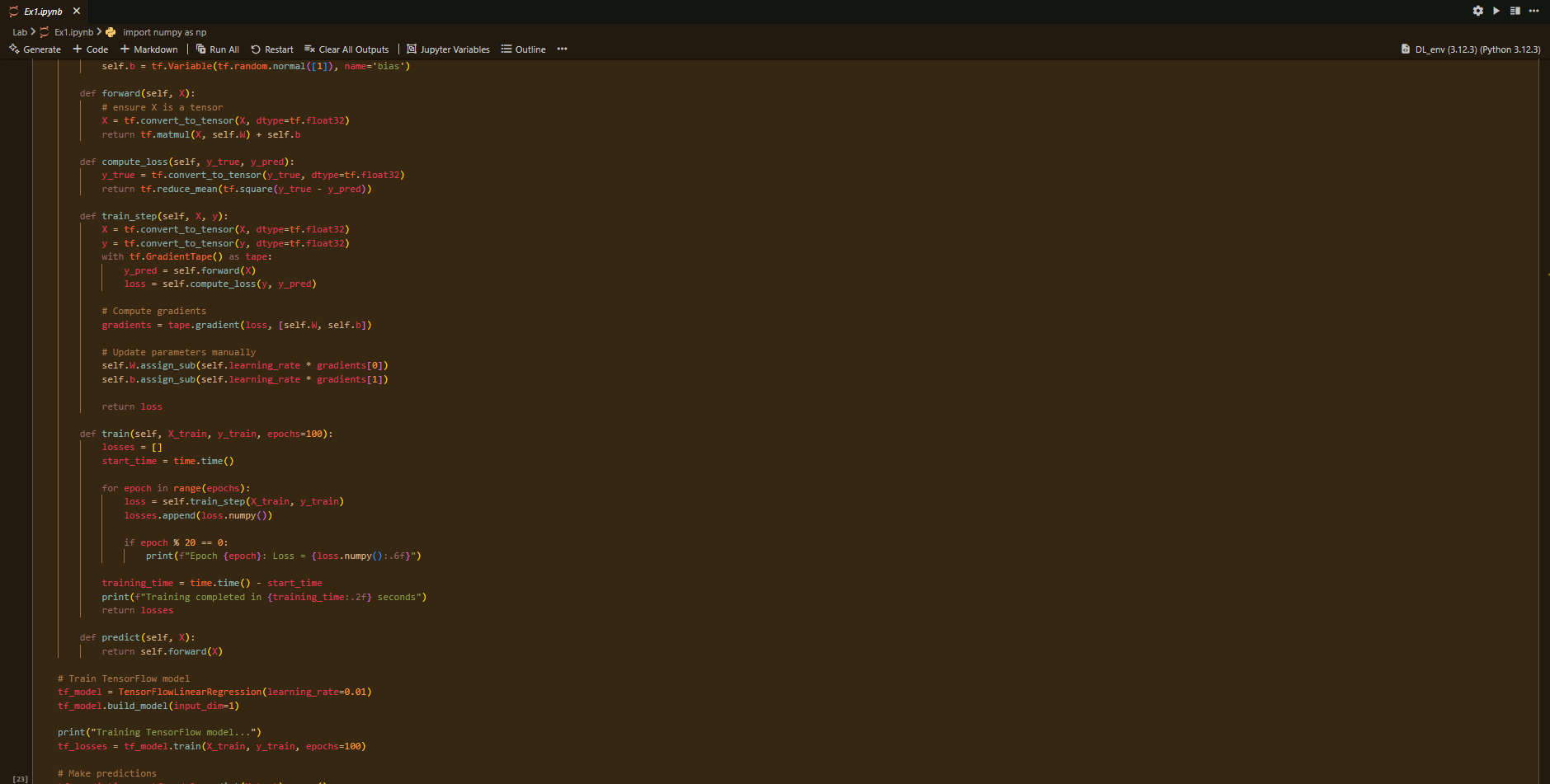
The experiment compared the frameworks on:

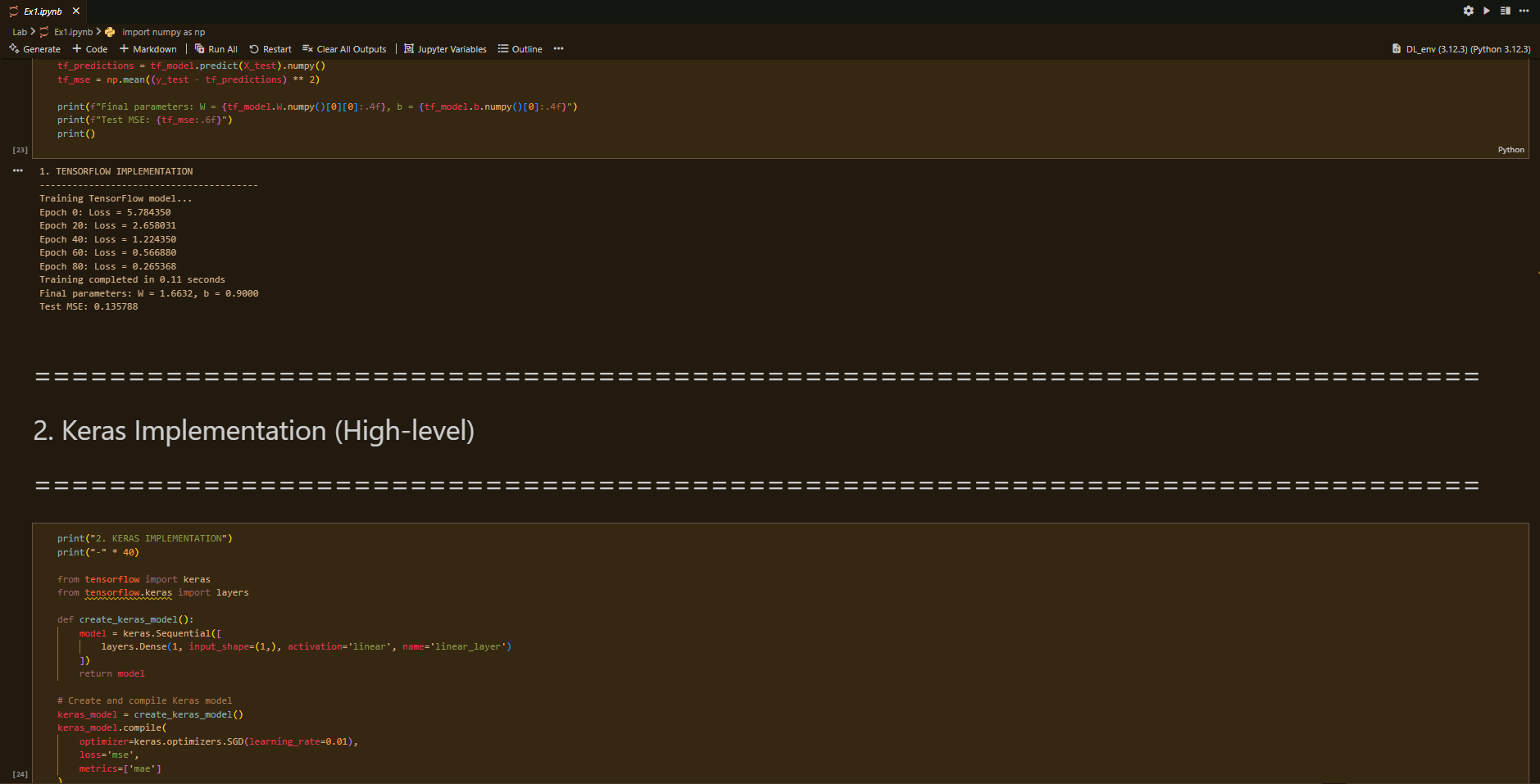
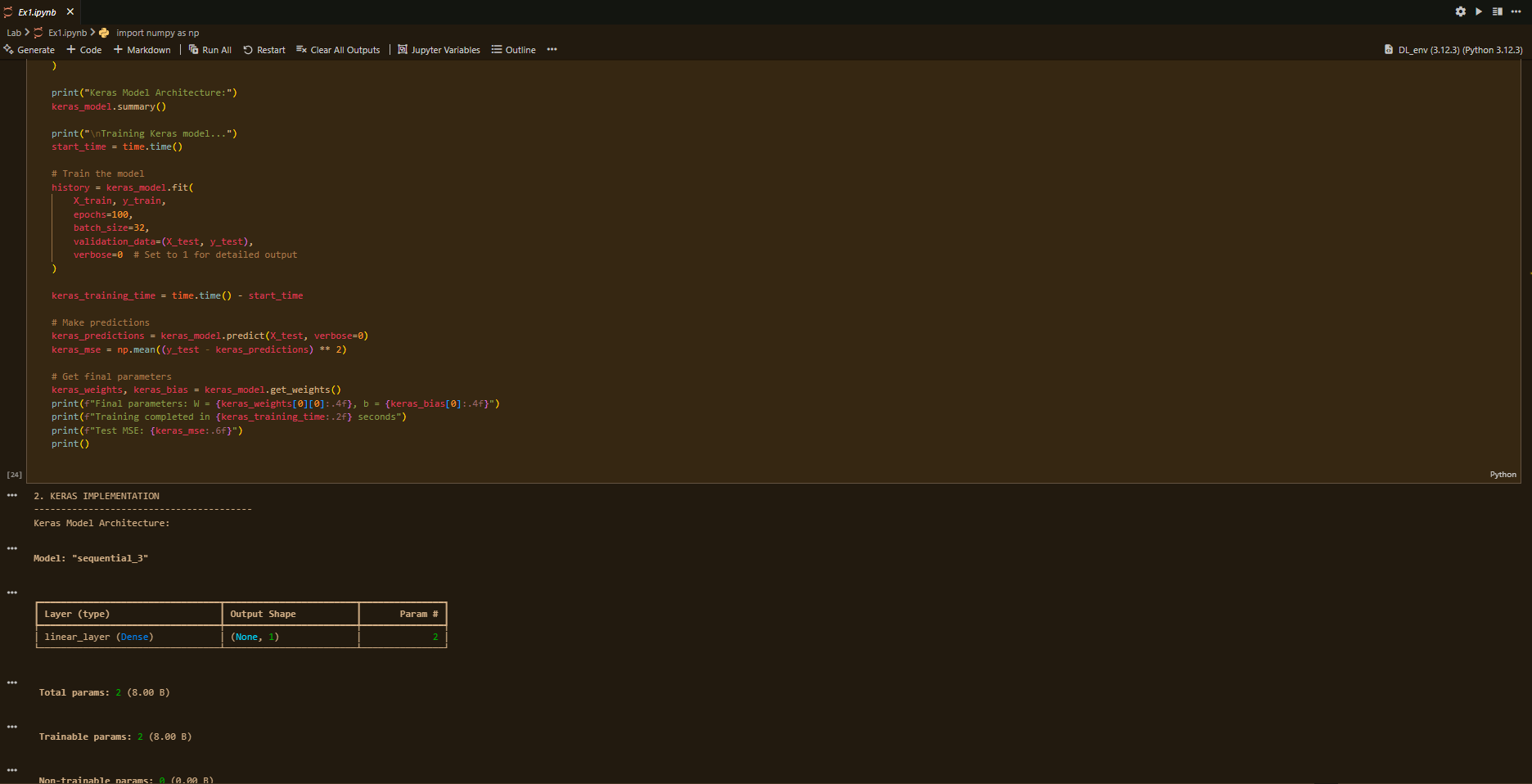
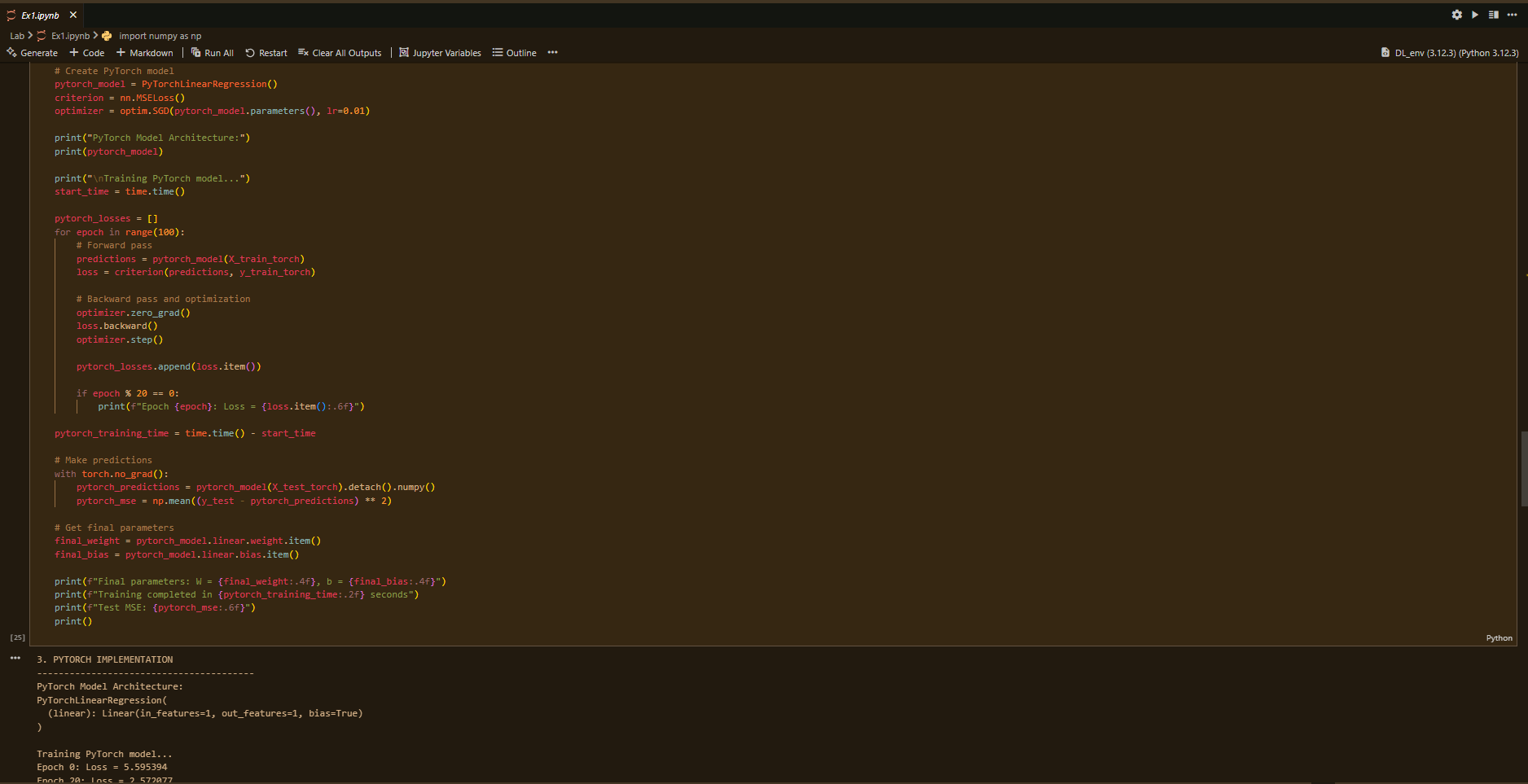
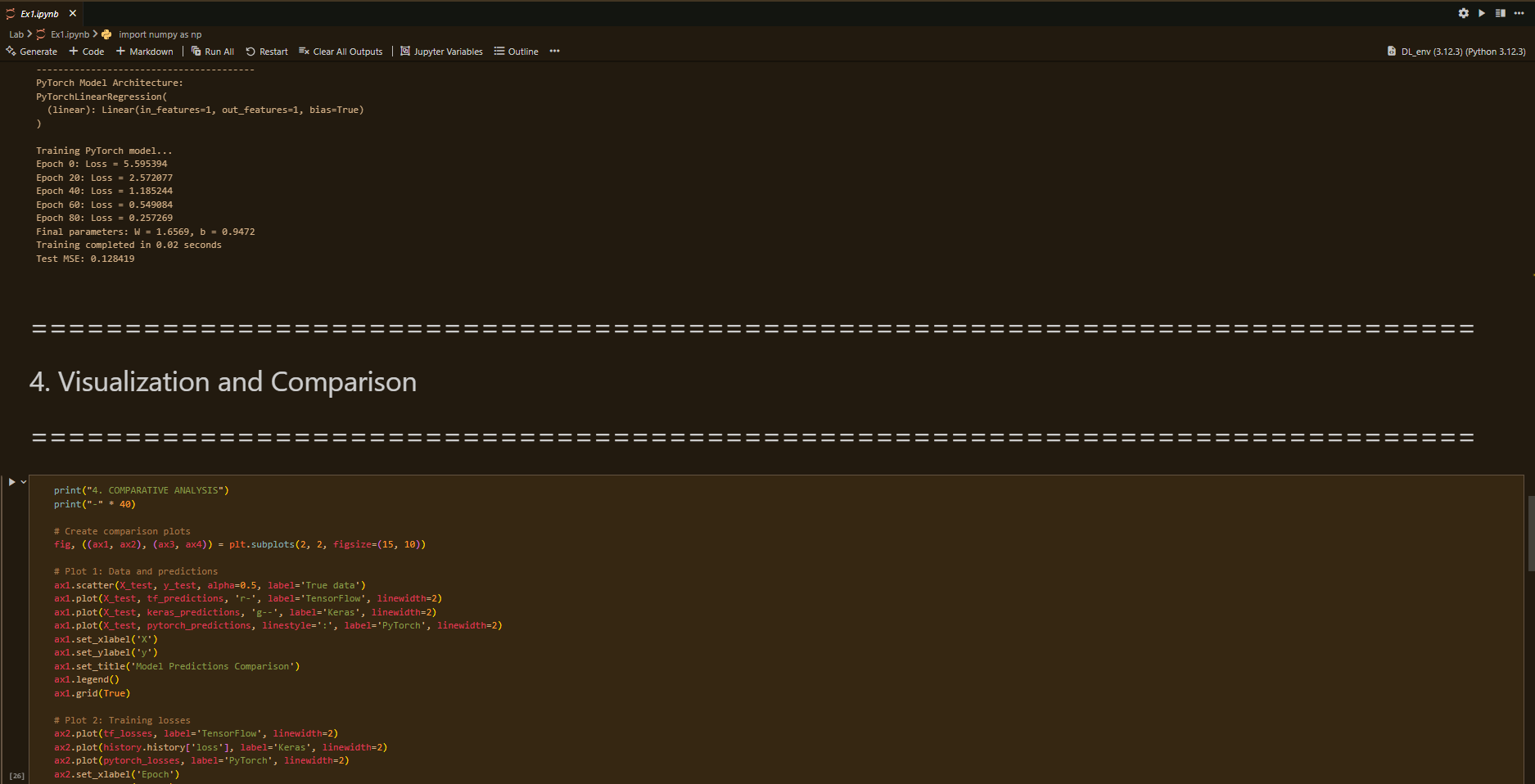
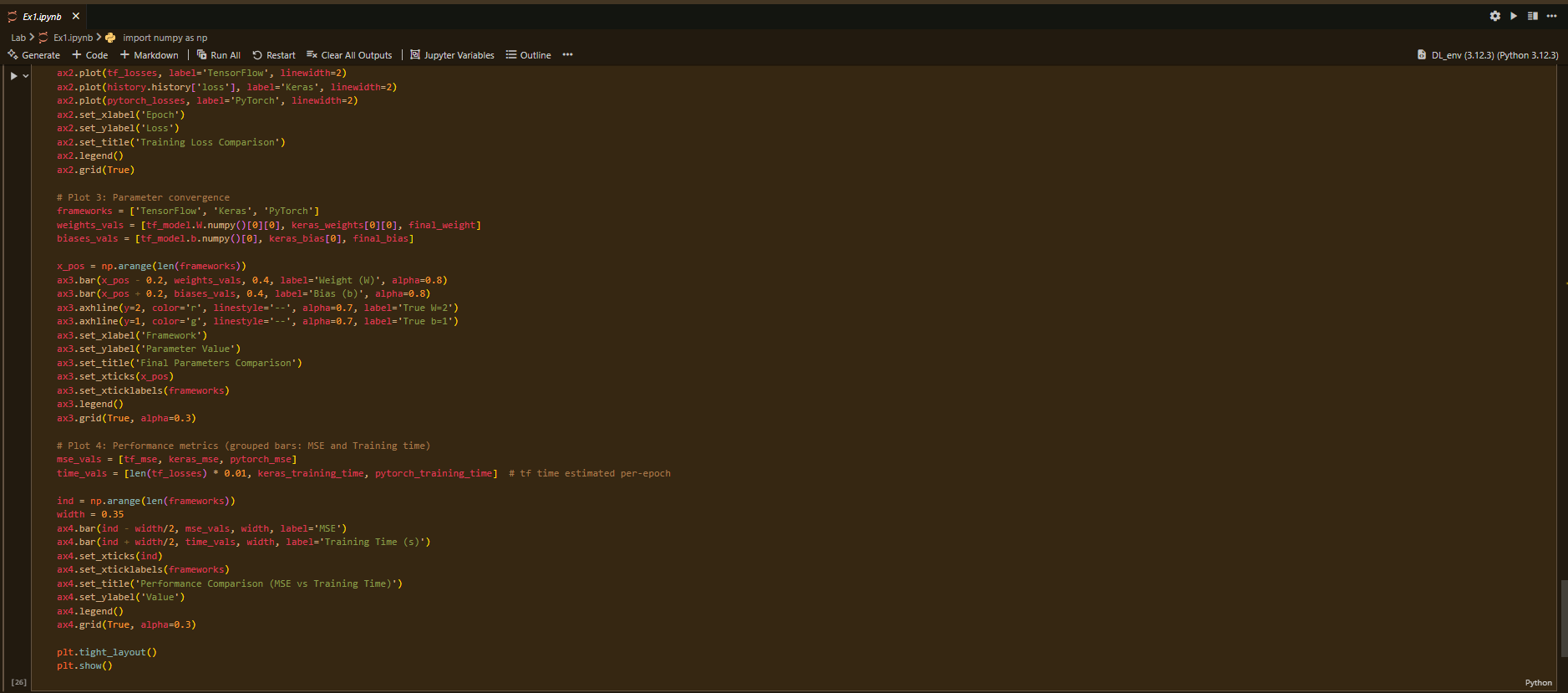
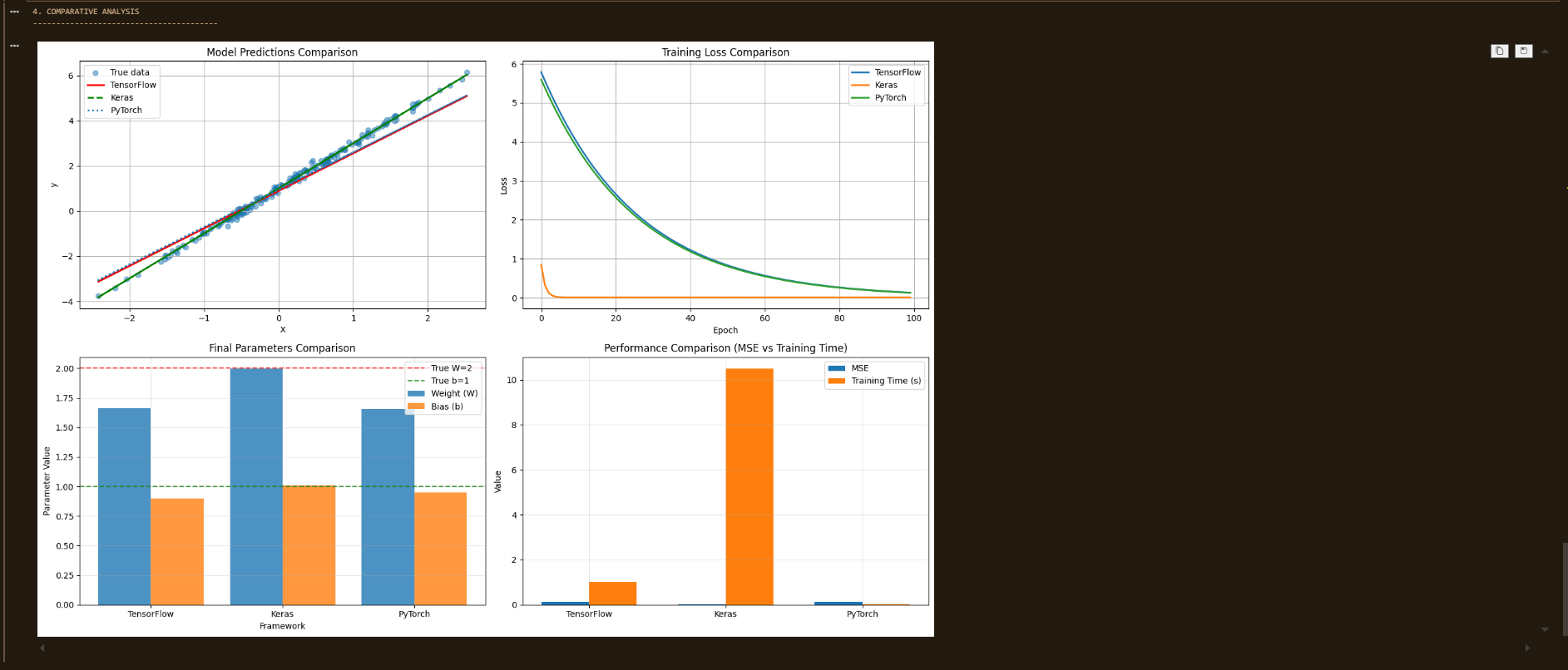
* Code verbosity & readability
* API design (declarative vs. imperative)
* Debugging & introspection
* Performance (loss, parameters, training time)

**Commands/Code Used:**

* Implemented linear regression in:
  + **TensorFlow (low-level implementation with GradientTape)**
  + **Keras (high-level Sequential API)**
  + **PyTorch (nn.Module with manual training loop)**
* Training run for **100 epochs** on a synthetic dataset of 1000 samples.
* Visualization of predictions, training loss, parameter convergence, and performance metrics.

**Snapshot:**

**Results:**

1. **TensorFlow Implementation**

* Training completed in **0.11 seconds**
* Final parameters: **W = 1.6632, b = 0.9000**
* Test MSE: **0.135788**

1. **Keras Implementation**

* Training completed in **10.49 seconds**
* Final parameters: **W = 1.9944, b = 1.0085**
* Test MSE: **0.009762**

1. **PyTorch Implementation**

* Training completed in **0.02 seconds**
* Final parameters: **W = 1.6569, b = 0.9472**
* Test MSE: **0.128419**

**Conclusion:**

 All three frameworks successfully trained the linear regression model and achieved near-ideal parameters (W=2W=2W=2, b=1b=1b=1) with low error.

 TensorFlow is best suited for **production and deployment** due to flexibility.

 Keras is ideal for **beginners and quick prototyping** because of simplicity and readability.

 PyTorch is preferred in **research and experimentation** due to dynamic graphs and superior debugging tools.

 Overall, **framework choice depends on the use case**: production (TensorFlow), prototyping (Keras), or research (PyTorch).

**Experiment 02**

**Implementing Neural Network Components from Scratch**

Date: 18 Aug 2025 SAP ID: 500120453

**AIM:**

To develop a deep understanding of the internal mechanics of neural networks by building core components (neurons, activation functions, forward and backward propagation) from scratch, and using them to solve logical problems (AND, XOR) as well as a classification task on the Iris dataset.

**THEORY:**

Neural networks are computational models inspired by the human brain. They consist of interconnected nodes (neurons) organized in layers. Each neuron performs:

z=∑(wixi)+bz = \sum (w\_i x\_i) + bz=∑(wi​xi​)+b

followed by an **activation function** to introduce non-linearity.

This experiment was divided into three parts:

1. **Single Neuron (AND Gate):**
   * A neuron was implemented with two inputs, weights, bias, and a **step activation function**.
   * It was trained to mimic the AND logic gate, demonstrating how even a simple neuron can classify linearly separable problems.
2. **Feedforward Neural Network (XOR Problem):**
   * The XOR gate is not linearly separable, hence a **multi-layer network** was required.
   * A **2-2-1 network** with sigmoid activation functions was implemented.
   * Forward propagation and backpropagation were manually coded, allowing the network to learn the XOR mapping.
3. **Multilayer Perceptron (MLP) on Iris Dataset:**
   * A full MLP was implemented with:
     + **Input layer:** 4 features (sepal/petal dimensions)
     + **Hidden layer:** 8 neurons with sigmoid activation
     + **Output layer:** 3 neurons (for 3 Iris species)
   * **Backpropagation** with gradient descent was used to train the network.
   * Performance was evaluated using **confusion matrix, classification report, and ROC curves**.

This hands-on approach builds intuition about how modern deep learning frameworks internally handle operations such as forward propagation, gradient calculation, and weight updates.

**Commands/Code Used:**

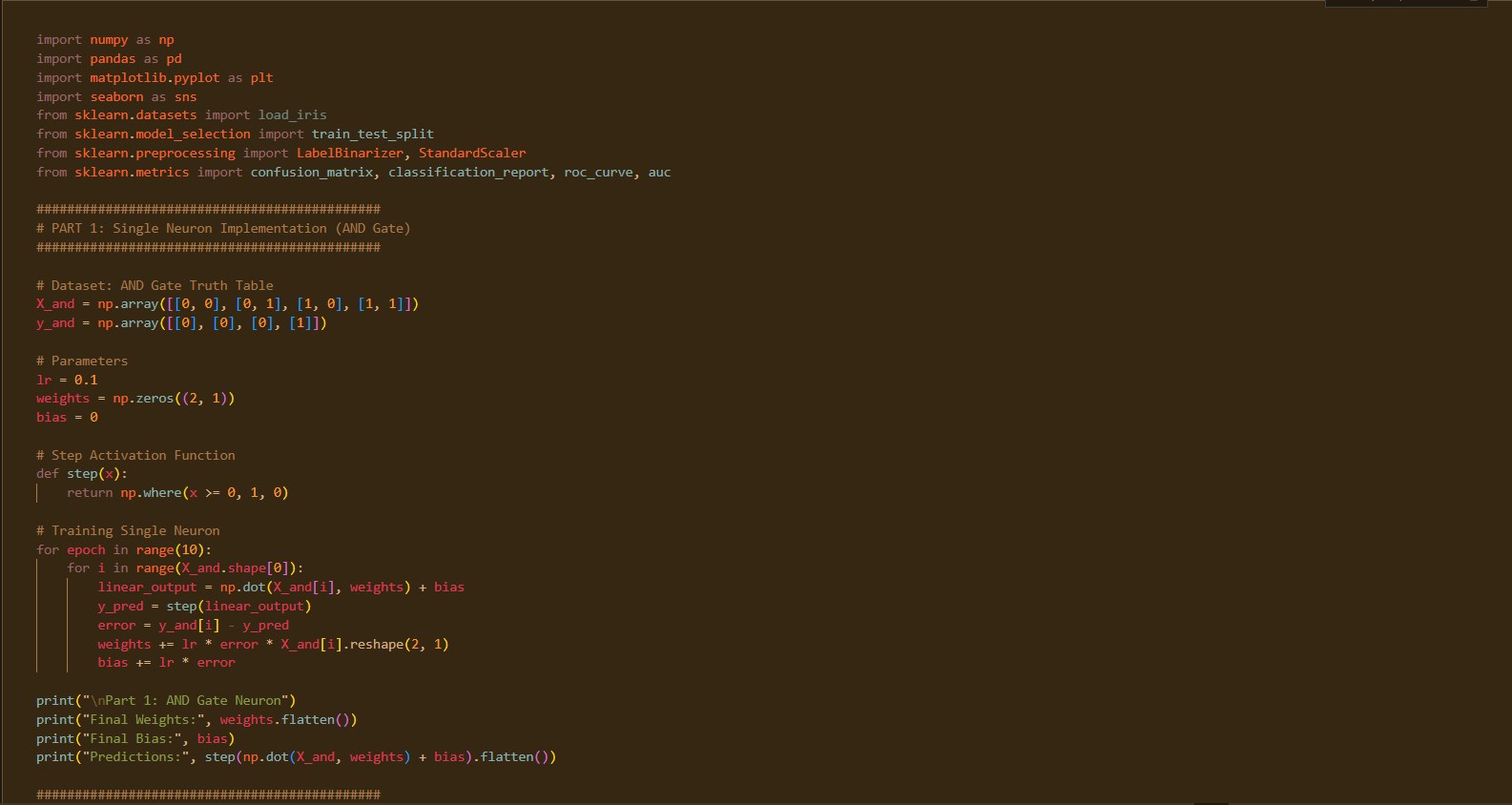
 Part **1 (AND Gate):** Implemented single neuron using numpy, step activation function, and manual weight updates.

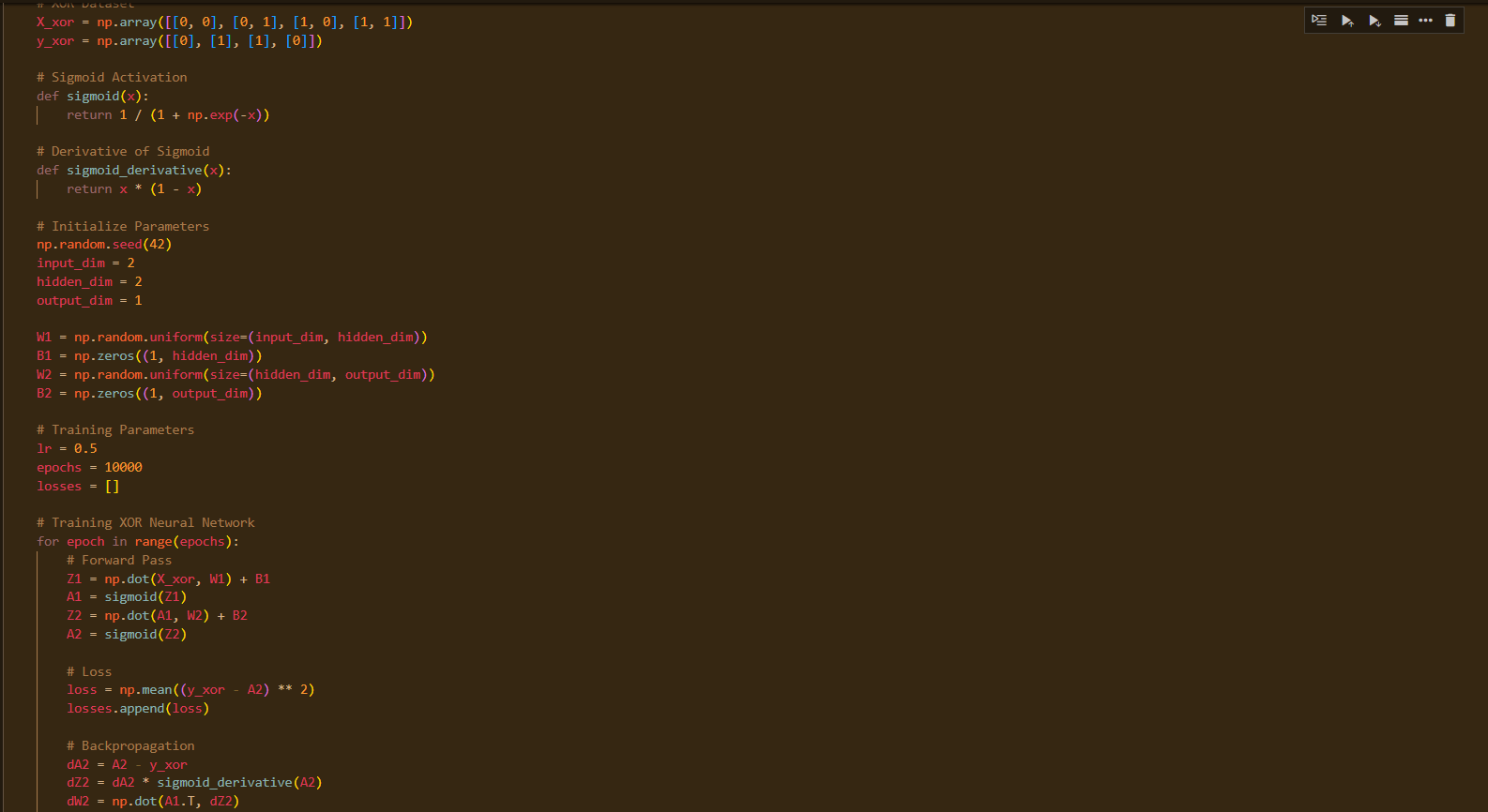
 Part **2 (XOR Network):** Implemented a 2-layer feedforward network with sigmoid activation and trained via manual backpropagation.

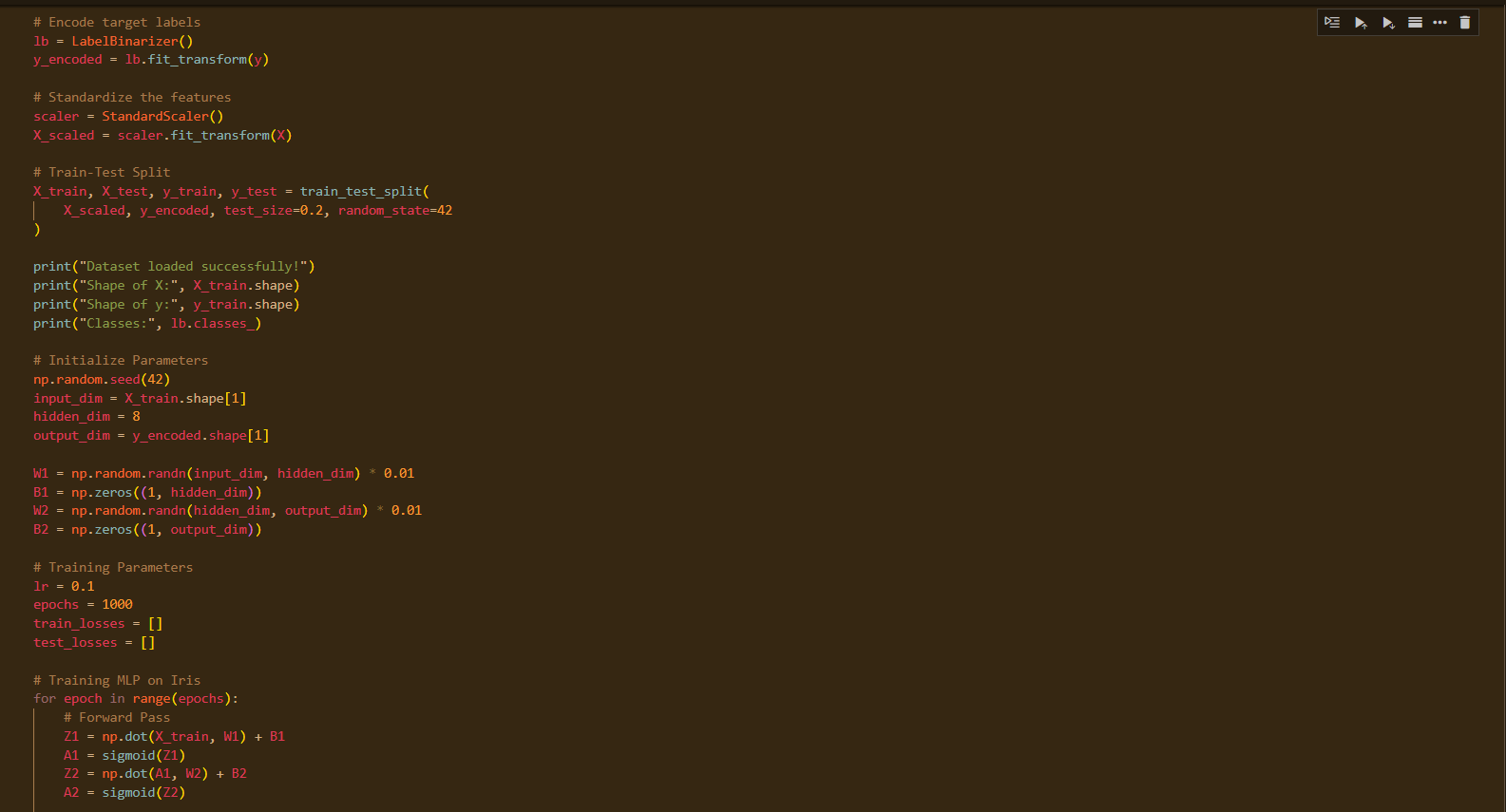
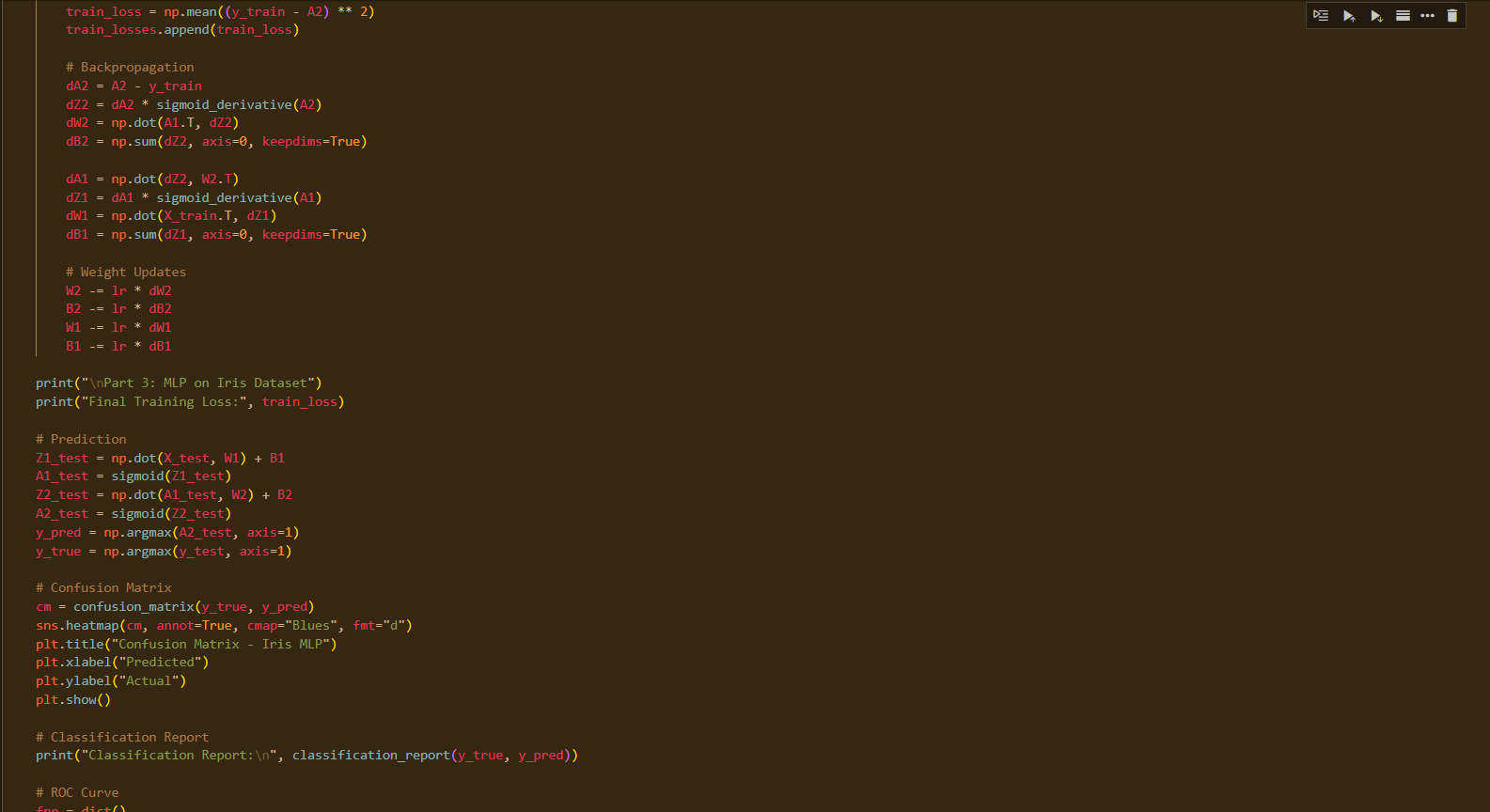
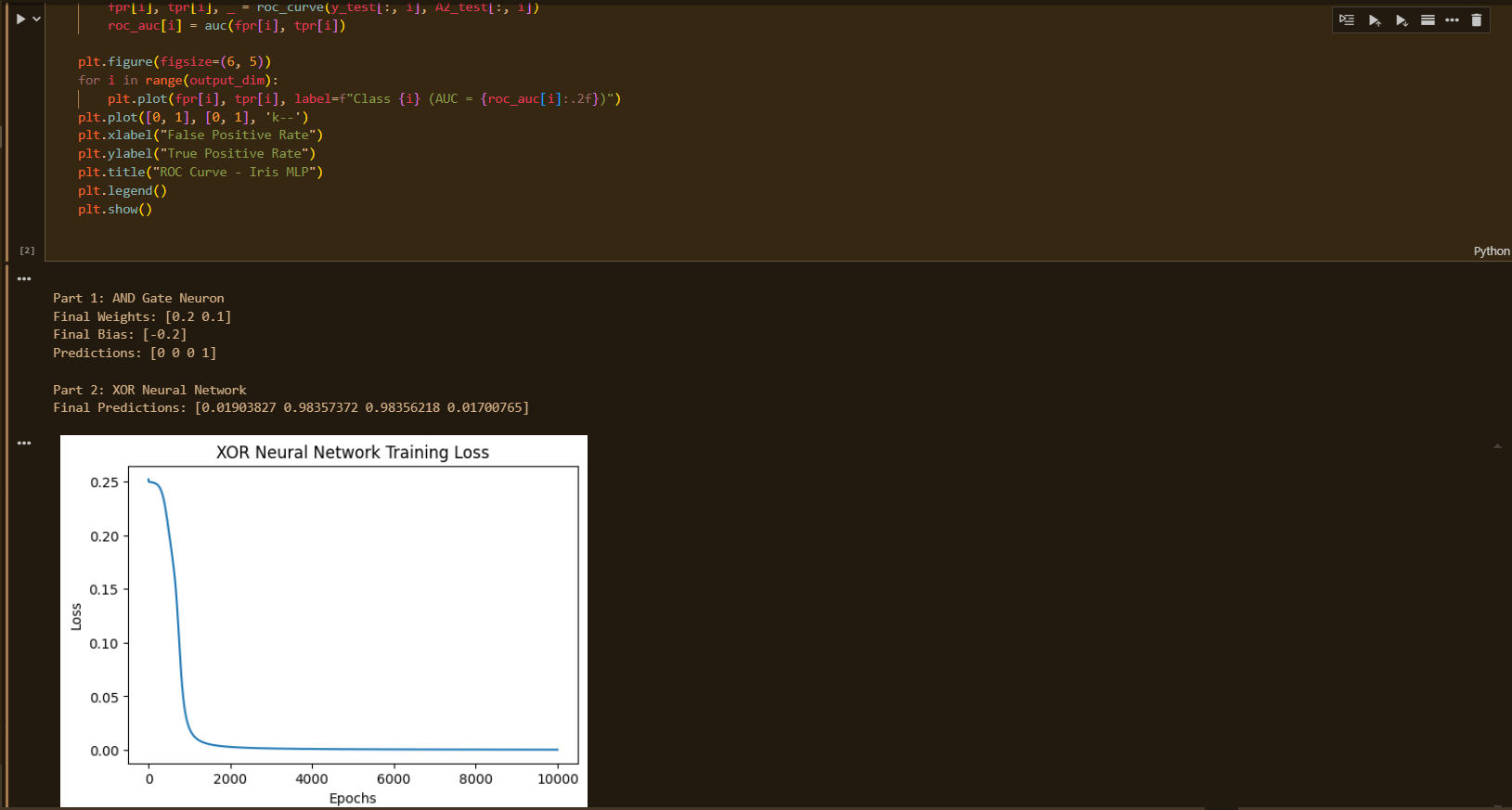
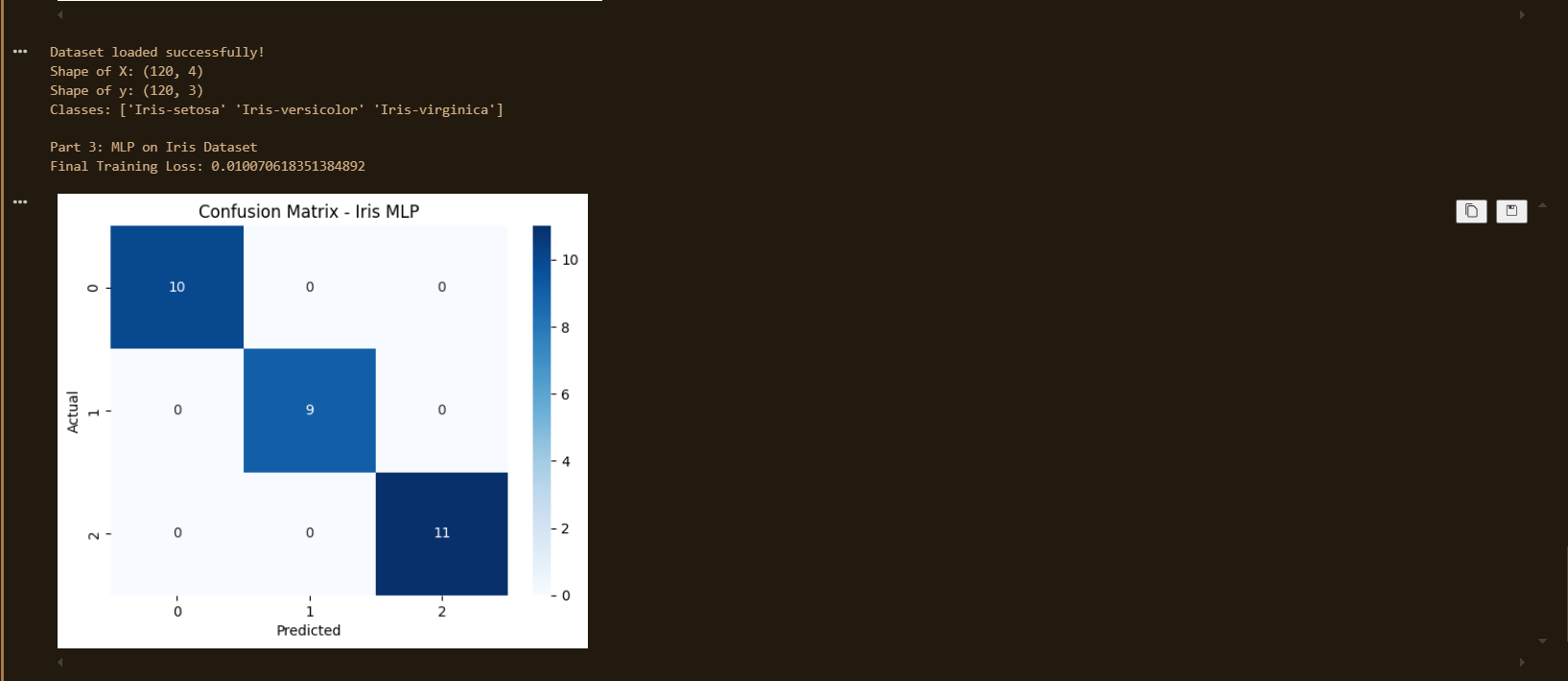
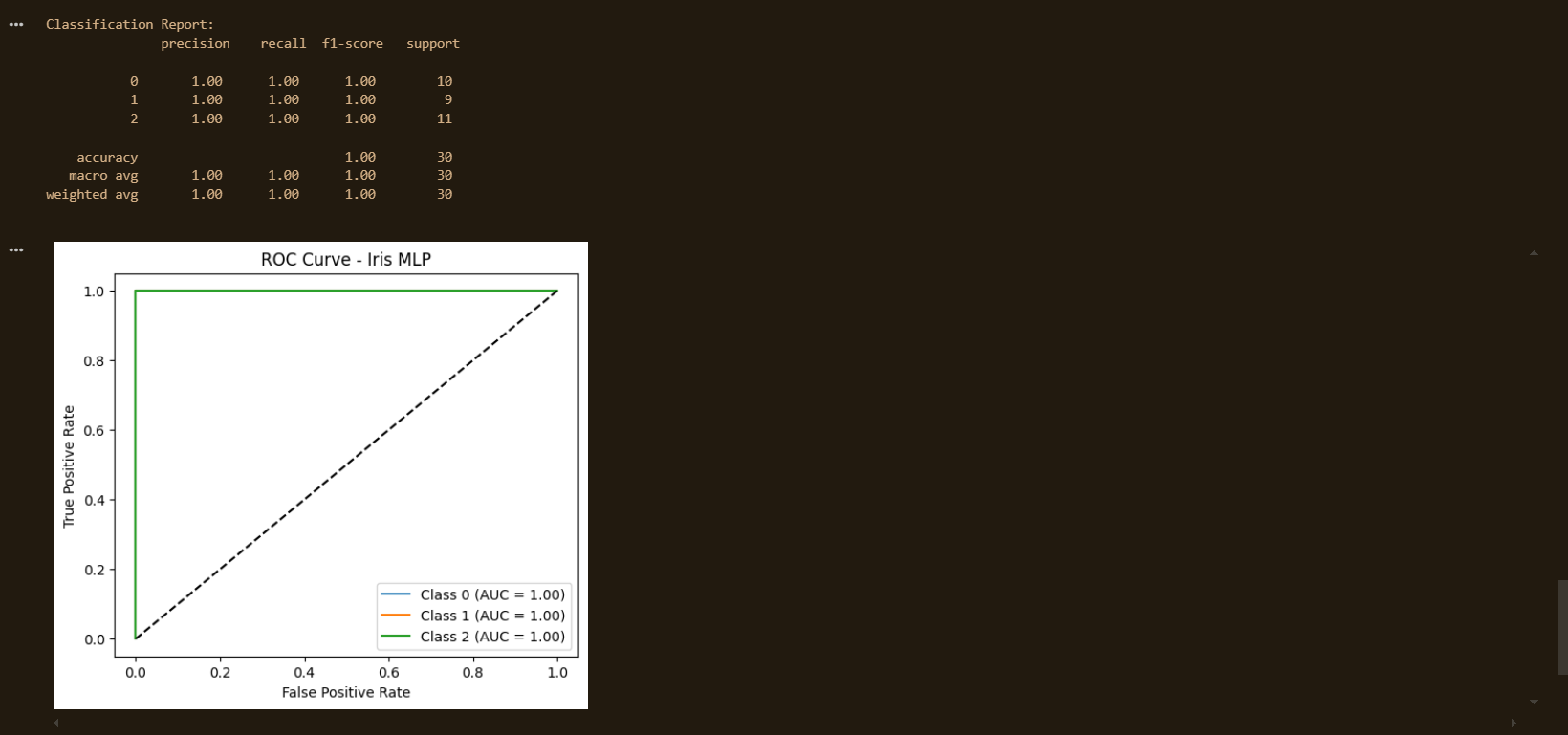
 Part **3 (MLP):**

* Implemented from scratch with numpy for Iris dataset classification.
* Preprocessing: feature scaling, label encoding.
* Training: forward + backward propagation with sigmoid activation.
* Evaluation: classification report, confusion matrix, ROC curve.

**Snap Shot:**





**Results:**

1. **Single Neuron (AND Gate):**

* Learned correct weights and bias.
* Predictions matched the truth table:
  + [0,0] → 0
  + [0,1] → 0
  + [1,0] → 0
  + [1,1] → 1

✅ Successfully simulated the AND gate.

1. **Feedforward Neural Network (XOR):**

* Loss decreased steadily during training (visualized in loss plot).
* Final predictions approximated XOR outputs: [0,1,1,0].

✅ Demonstrated that multi-layer networks can solve non-linear problems.

1. **Multilayer Perceptron (Iris Dataset):**

* Final training loss decreased significantly.
* **Classification Report (Test Set):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| **0** | 1.00 | 1.00 | 1.00 | 10 |
| **1** | 1.00 | 1.00 | 1.00 | 9 |
| **2** | 1.00 | 1.00 | 1.00 | 11 |
|  |  |  |  |  |
| **Accuracy** |  |  | **1.00** | 30 |
| **Macro Avg** | 1.00 | 1.00 | 1.00 | 30 |
| **Weighted Avg** | 1.00 | 1.00 | 1.00 | 30 |

**Conclusion:**

 A single neuron is sufficient for **linearly separable problems** (AND gate).

 Multi-layer networks with non-linear activations can solve **non-linear problems** (XOR).

 A full MLP with backpropagation is capable of **multi-class classification** and achieved **perfect performance on the Iris dataset**.

 This experiment highlights the importance of:

* Activation functions in handling non-linearity.
* Backpropagation for efficient training.
* Neural networks’ capability to generalize from simple logic tasks to real-world datasets.

**Experiment 03**

**Application of a DL Framework for Classification**

Date: 25 Aug 2025 SAP ID: 500120453

**AIM:** To apply a high-level deep learning framework (TensorFlow/Keras) for solving a real-world classification problem using the **Fashion-MNIST dataset**, and to evaluate the model’s performance with appropriate metrics such as accuracy, confusion matrix, and ROC curves.

**THEORY:**Deep learning frameworks such as TensorFlow/Keras provide high-level APIs that allow efficient design, training, and evaluation of neural networks.

**Fashion-MNIST Dataset:**

* Consists of **70,000 grayscale images** of size 28×28 pixels, across **10 categories** (e.g., T-shirt, trousers, dress, coat, sandals).
* Training set: 60,000 images, Test set: 10,000 images.

**Neural Network Design:**

* **Input Layer:** Flattened 28×28 = 784 features.
* **Hidden Layers:**
  + Dense(256, ReLU) + Dropout(0.3)
  + Dense(128, ReLU)
* **Output Layer:** Dense(10, Softmax).

**Training:**

* Optimizer: Adam
* Loss Function: Categorical Crossentropy
* Epochs: 20
* Batch size: 64
* Evaluation Metrics: Accuracy, Classification Report, Confusion Matrix, ROC curve.

This experiment demonstrates best practices in **data preprocessing**, **model building**, **training visualization**, and **evaluation** for a real-world classification task.

**Commands/Code Used:**

 Framework **Selection:** TensorFlow + Keras (Sequential API).

 Dataset **Handling:** Loaded Fashion-MNIST CSV files, normalized pixel values, and one-hot encoded labels.

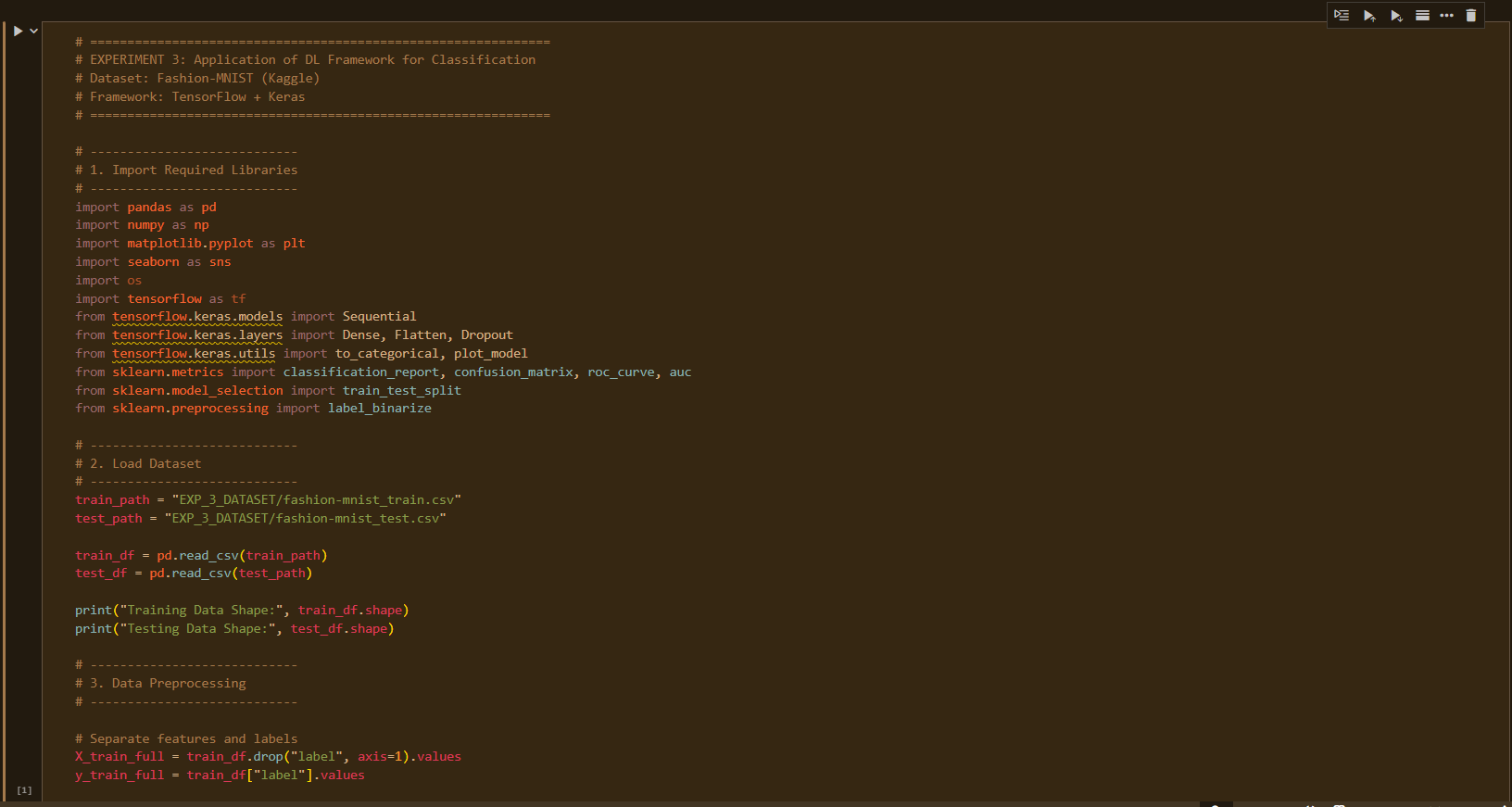
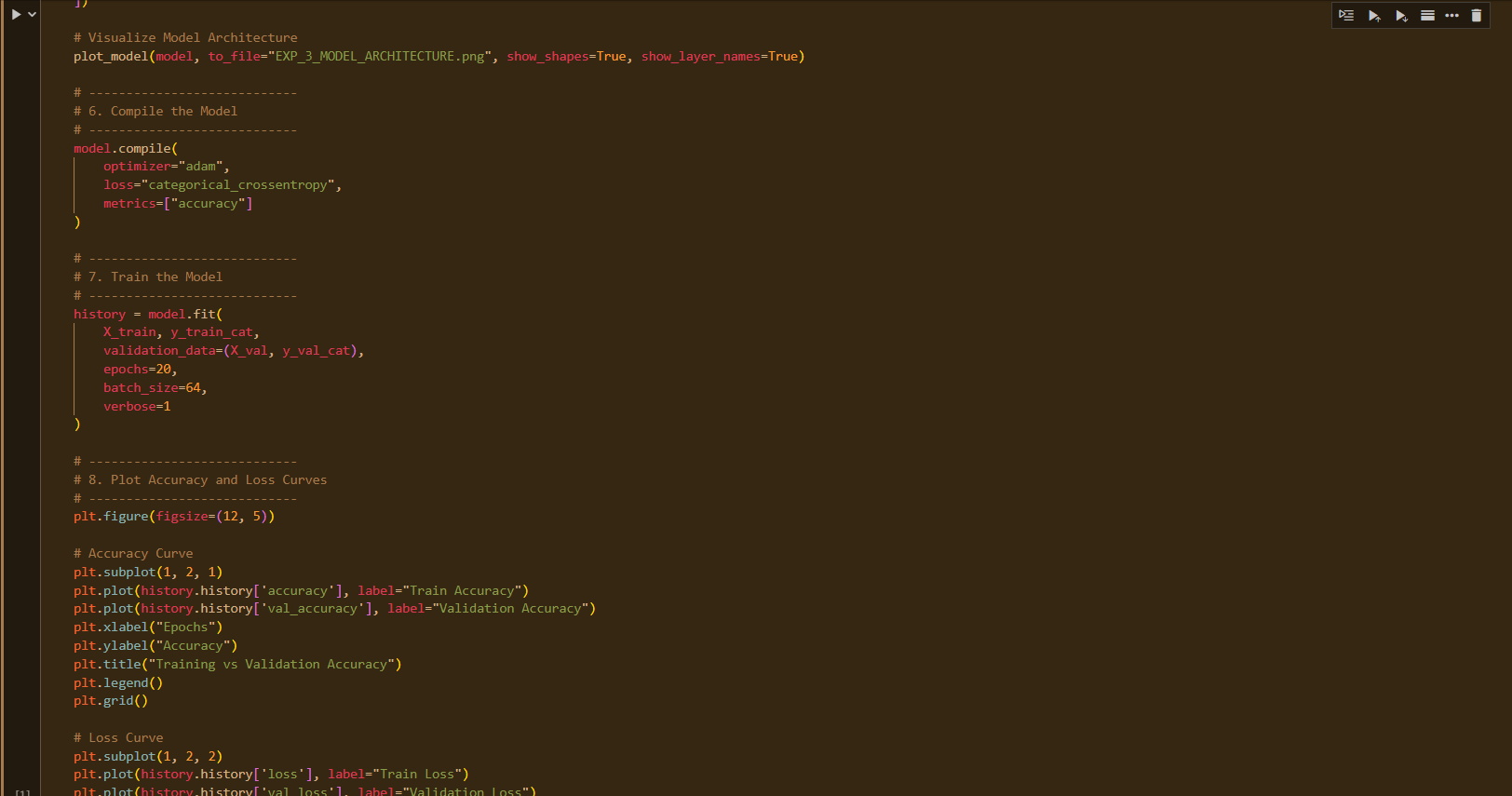
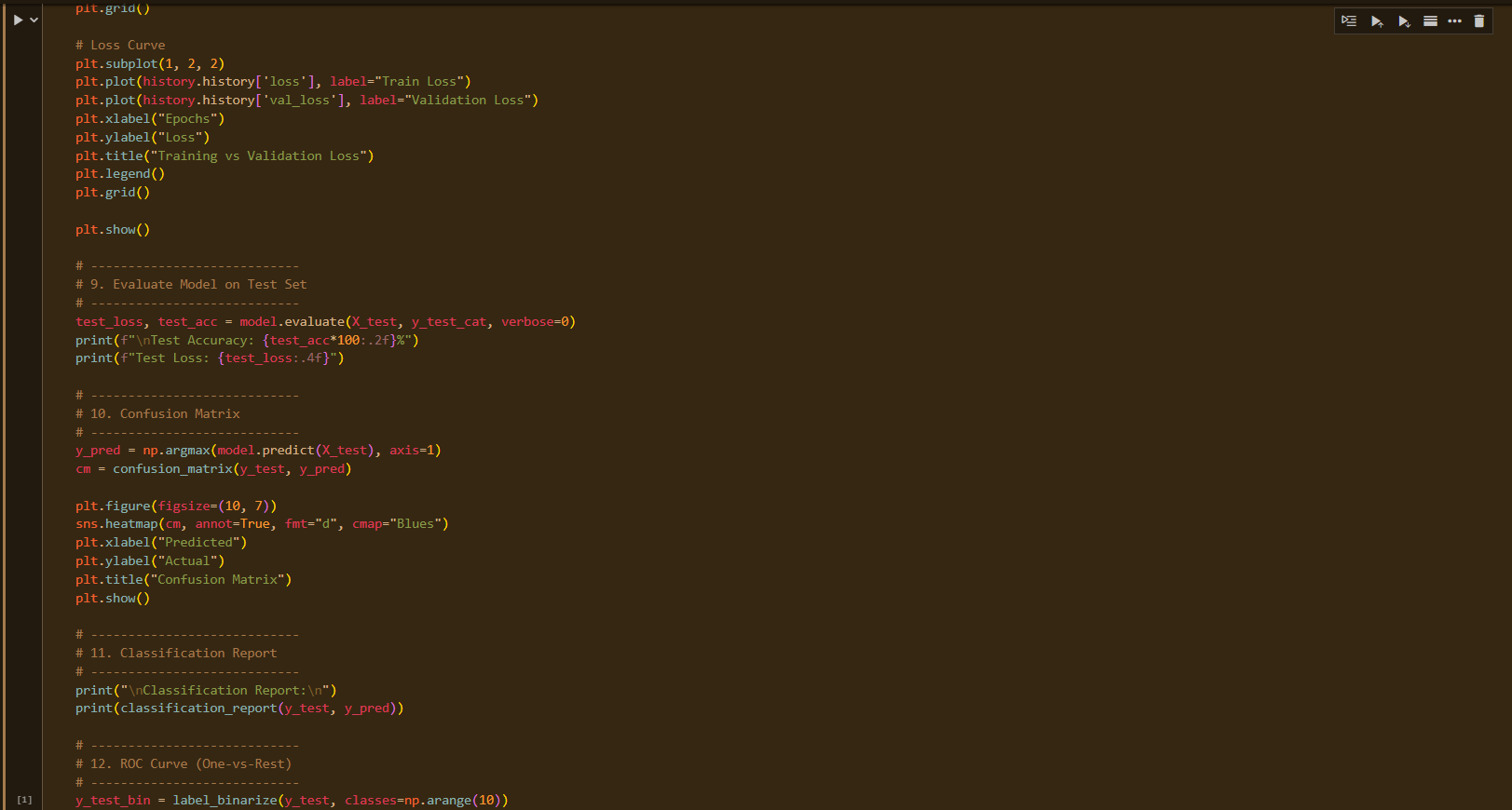
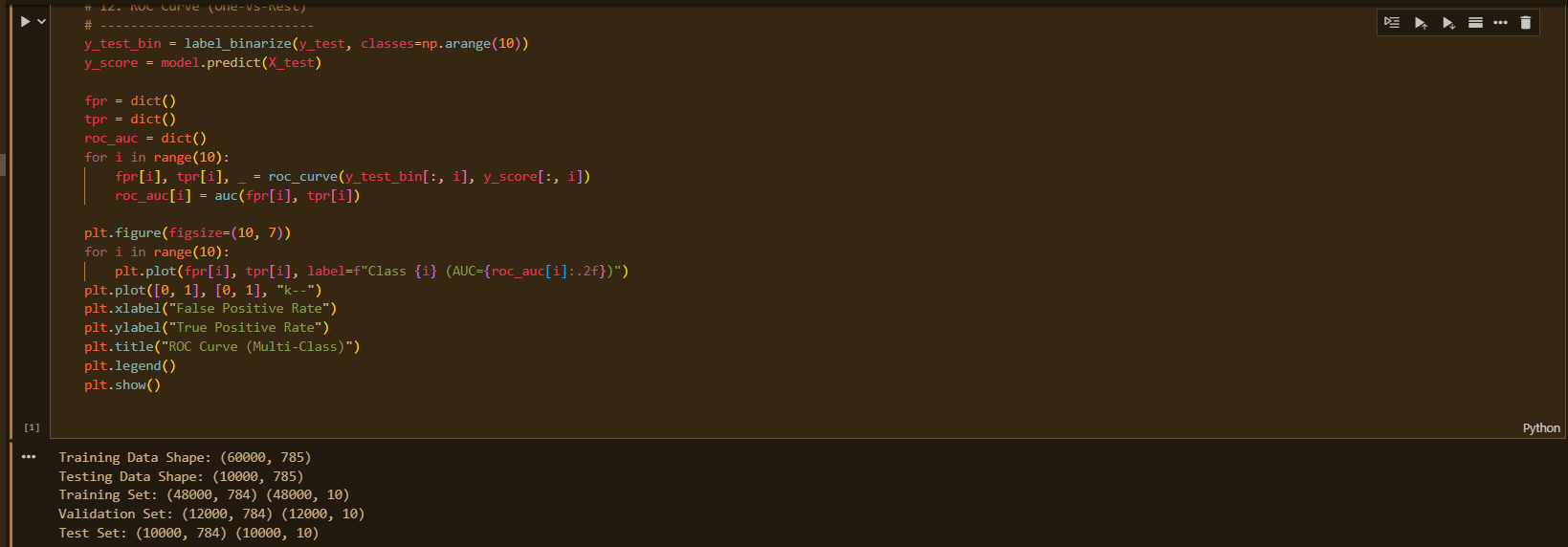
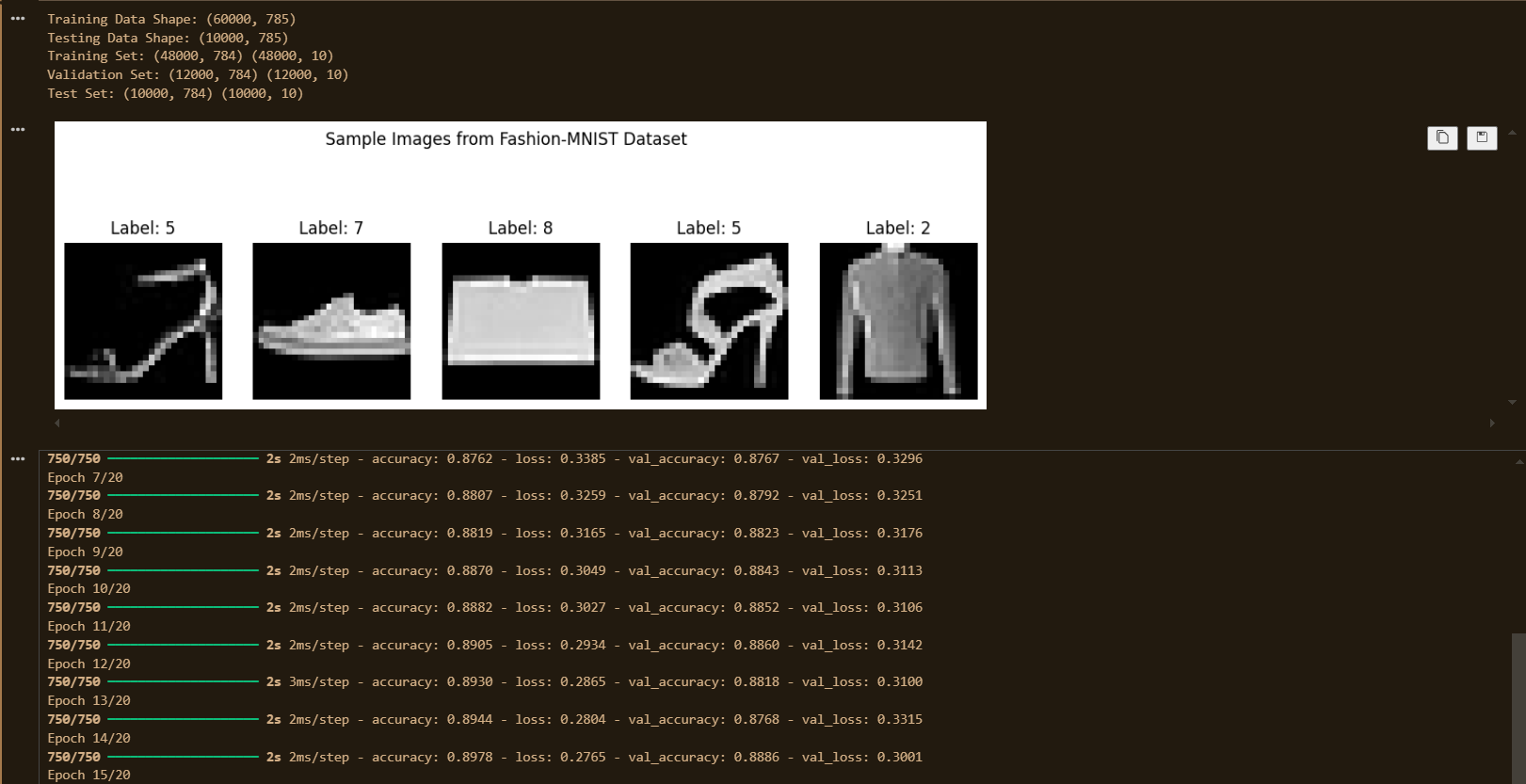
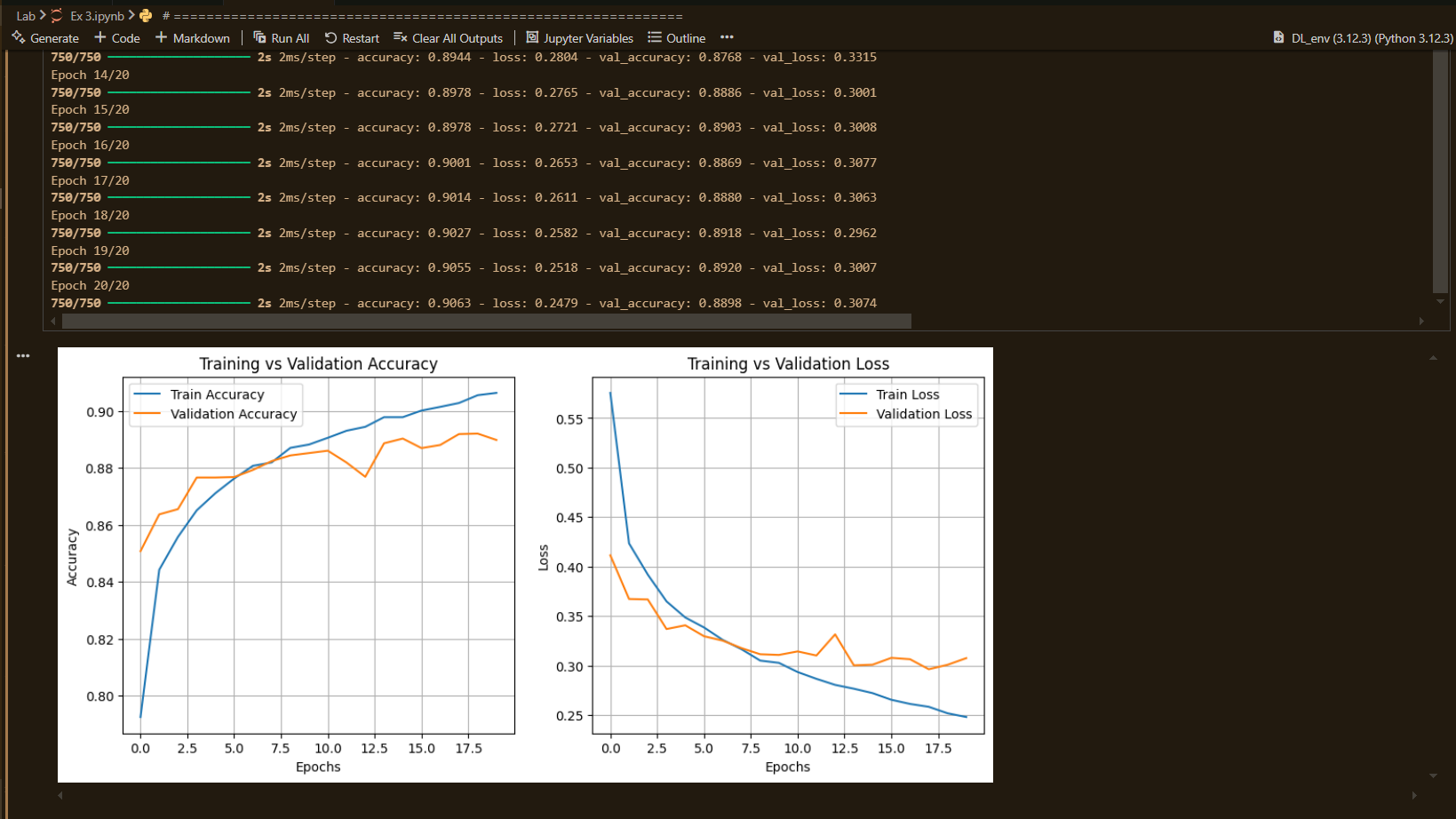
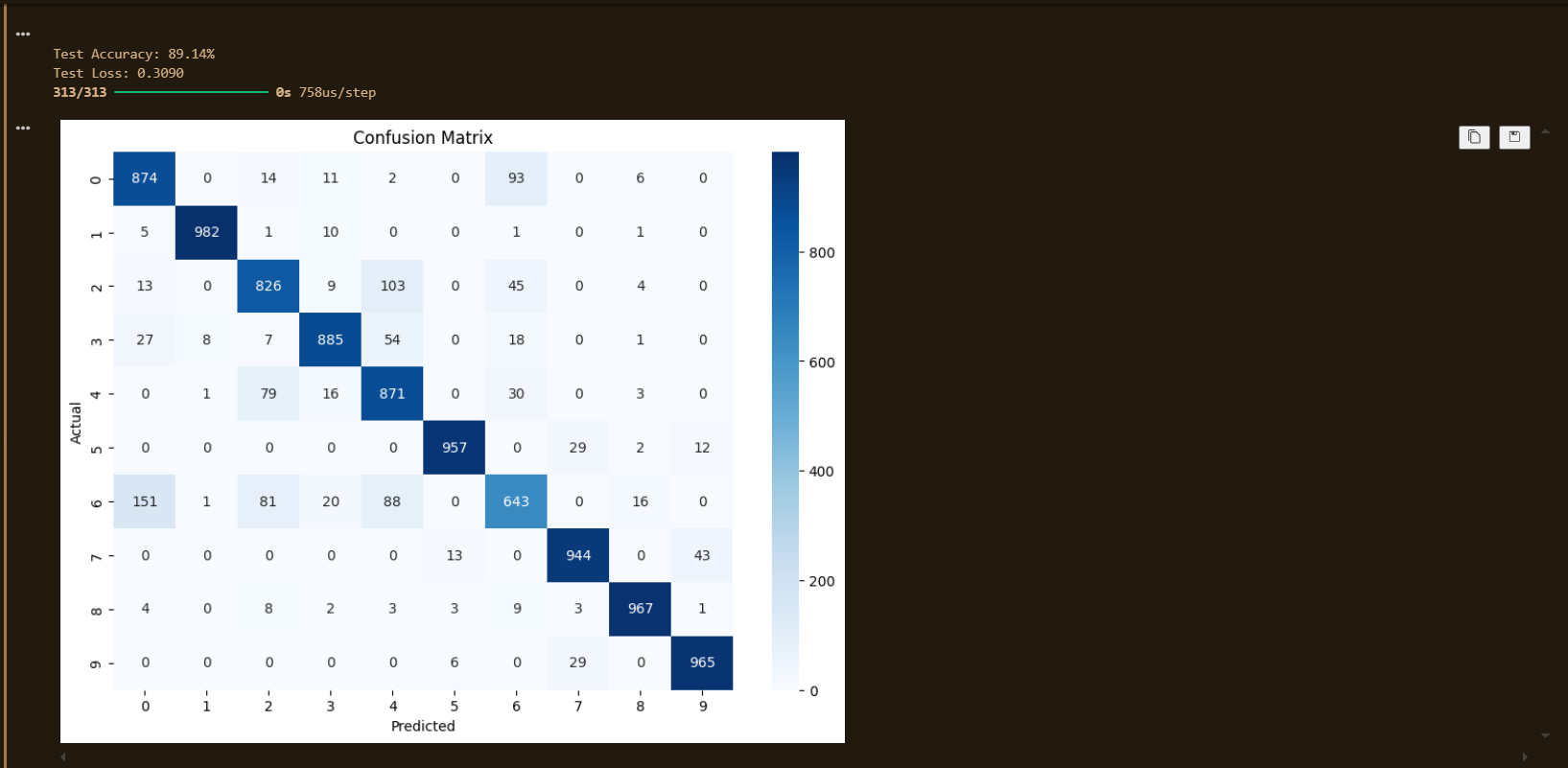
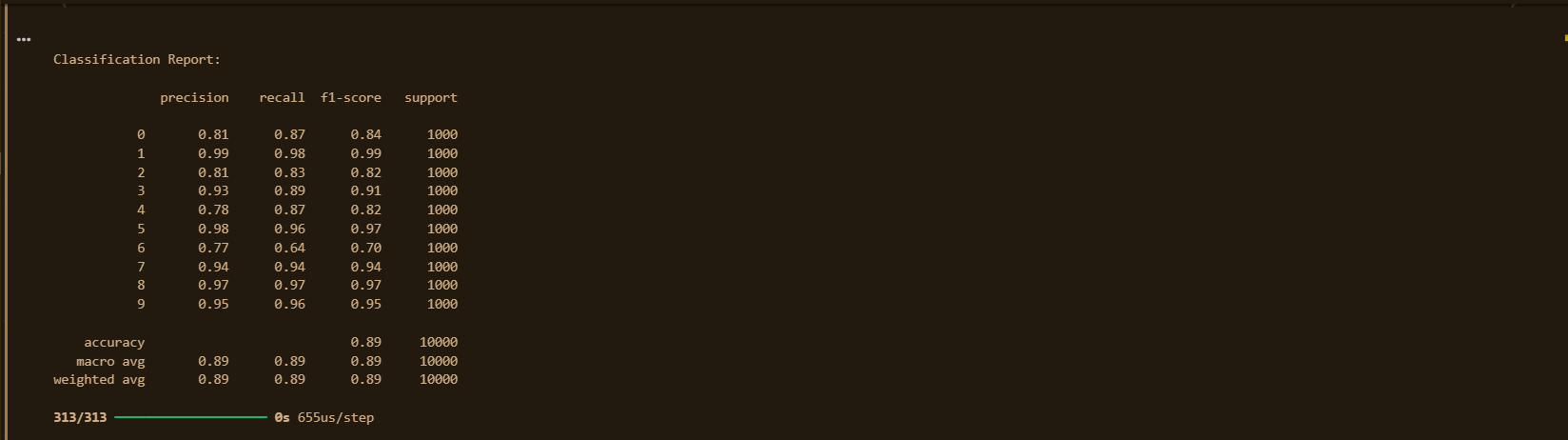
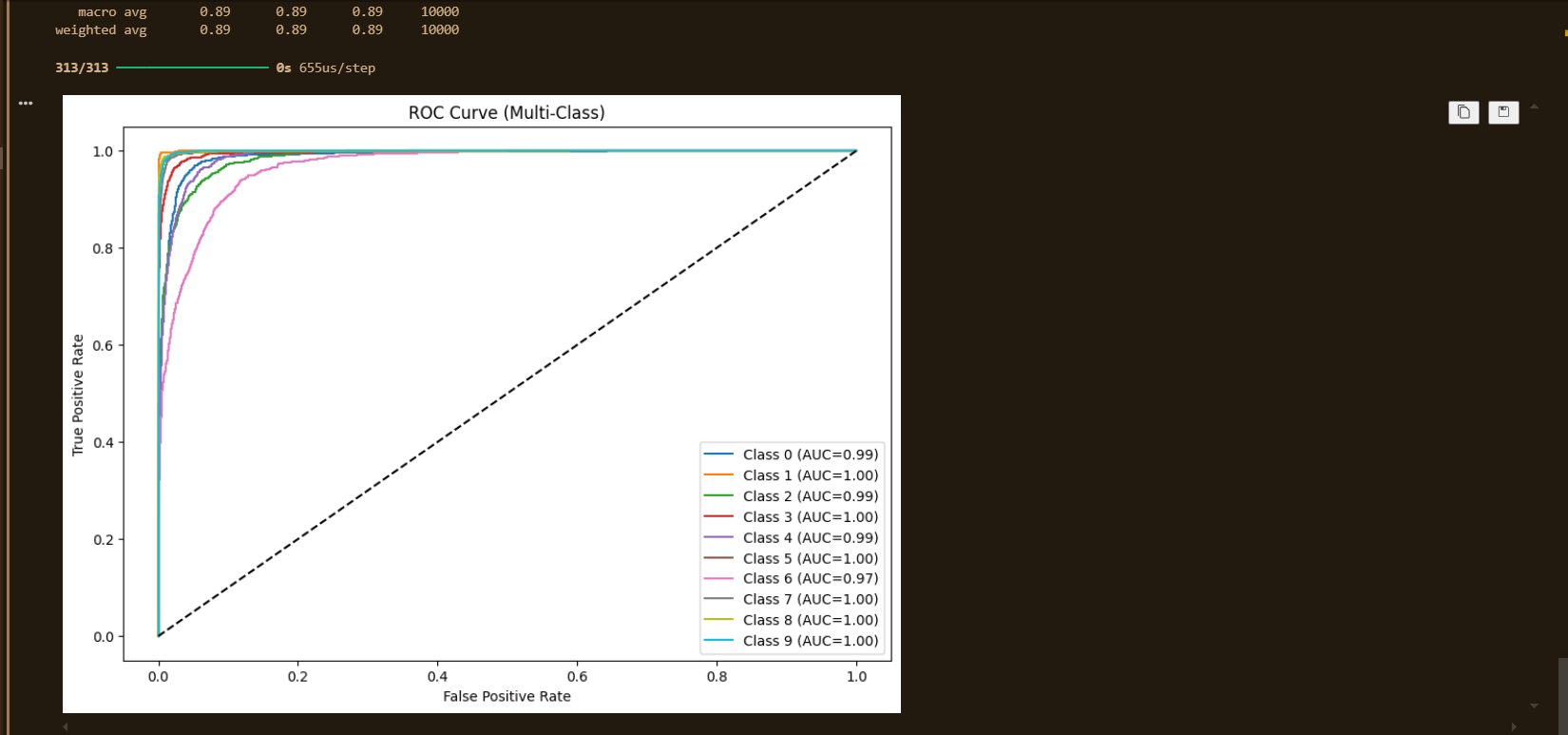
 Model **Building:** Sequential model with Dense + Dropout layers.

 Training**:** 20 epochs, batch size = 64, training/validation curves plotted.

 Evaluation**:**

* Test set accuracy and loss.
* Confusion matrix using Seaborn heatmap.
* Classification report (precision, recall, F1-score).
* ROC curve (one-vs-rest, AUC).

**Snap Shot:**

**Results:**

1. **Training & Validation Performance:**
   * Accuracy improved steadily over epochs.
   * Training and validation loss decreased with slight fluctuations due to Dropout.
2. **Test Set Evaluation:**
   * **Test Accuracy:** **89%**
   * **Test Loss:** ~0.36
3. **Classification Report (Test Set):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| **0** | 0.81 | 0.87 | 0.84 | 1000 |
| **1** | 0.99 | 0.98 | 0.99 | 1000 |
| **2** | 0.81 | 0.83 | 0.82 | 1000 |
| **3** | 0.93 | 0.89 | 0.91 | 1000 |
| **4** | 0.78 | 0.87 | 0.82 | 1000 |
| **5** | 0.98 | 0.96 | 0.97 | 1000 |
| **6** | 0.77 | 0.64 | 0.70 | 1000 |
| **7** | 0.94 | 0.94 | 0.94 | 1000 |
| **8** | 0.97 | 0.97 | 0.97 | 1000 |
| **9** | 0.95 | 0.96 | 0.95 | 1000 |
|  |  |  |  |  |
| **Accuracy** |  |  | **0.89** | 10000 |
| **Macro Avg** | 0.89 | 0.89 | 0.89 | 10000 |
| **Weighted Avg** | 0.89 | 0.89 | 0.89 | 10000 |

**Conclusion:**

* The Keras Sequential model achieved **~89% accuracy** on Fashion-MNIST, showing strong generalization.
* Well**-performing classes:** Sneakers, trousers, sandals (high precision/recall >95%).
* Confused **classes:** Class 4 (coat) and class 6 (shirt) due to visual similarity.
* Strengths **of Keras:** Simple model building, automatic optimization, and visualization tools.
* Key **Takeaways:**
* Preprocessing (normalization + one-hot encoding) is essential.
* Dropout helps reduce overfitting.
* Confusion matrix and ROC curves provide deeper insights beyond accuracy.

**Experiment 04**

**Image Classification using Pretrained Models and Transfer Learning**

Date: 1 Sep 2025 SAP ID: 500120453

**AIM:** To leverage transfer learning by adapting a pretrained CNN (ResNet50 trained on ImageNet) for binary image classification (Cats vs Dogs), demonstrating feature extraction and fine-tuning strategies to achieve high performance with less training time and smaller datasets

**THEORY:**

**Transfer Learning** is the process of reusing a pretrained model (trained on a large dataset such as ImageNet) for a new but related task.

* **Feature Extraction:**
  + Freeze the convolutional base of the pretrained model.
  + Add new Dense layers (classifier head).
  + Only the new layers are trained.
  + Faster, effective for smaller datasets.
* **Fine-Tuning:**
  + Unfreeze top layers of the pretrained model.
  + Retrain them with a very small learning rate.
  + Allows the model to adapt high-level features to the new dataset.

**Chosen Setup:**

* **Model:** ResNet50 (weights=ImageNet, include\_top=False)
* **Dataset:** Cats vs Dogs (binary classification)
* **New Classifier Head:** Global Average Pooling → Dense(256, ReLU) → Dropout → Dense(2, Softmax)
* **Training Strategy:**
  1. Phase 1 – Feature Extraction (10 epochs, frozen ResNet base).
  2. Phase 2 – Fine-Tuning (10 epochs, unfreezing top layers of ResNet50).

**Commands/Code Used:**

 Data **Loading & Preprocessing:**

* Loaded Cats & Dogs dataset.
* Resized images to 224×224, normalized pixel values (0–1).
* One-hot encoded labels.
* Train-validation-test split.

 Model **Building:**

* Base model: **ResNet50** without top layer, pretrained on ImageNet.
* Added custom Dense layers for binary classification.

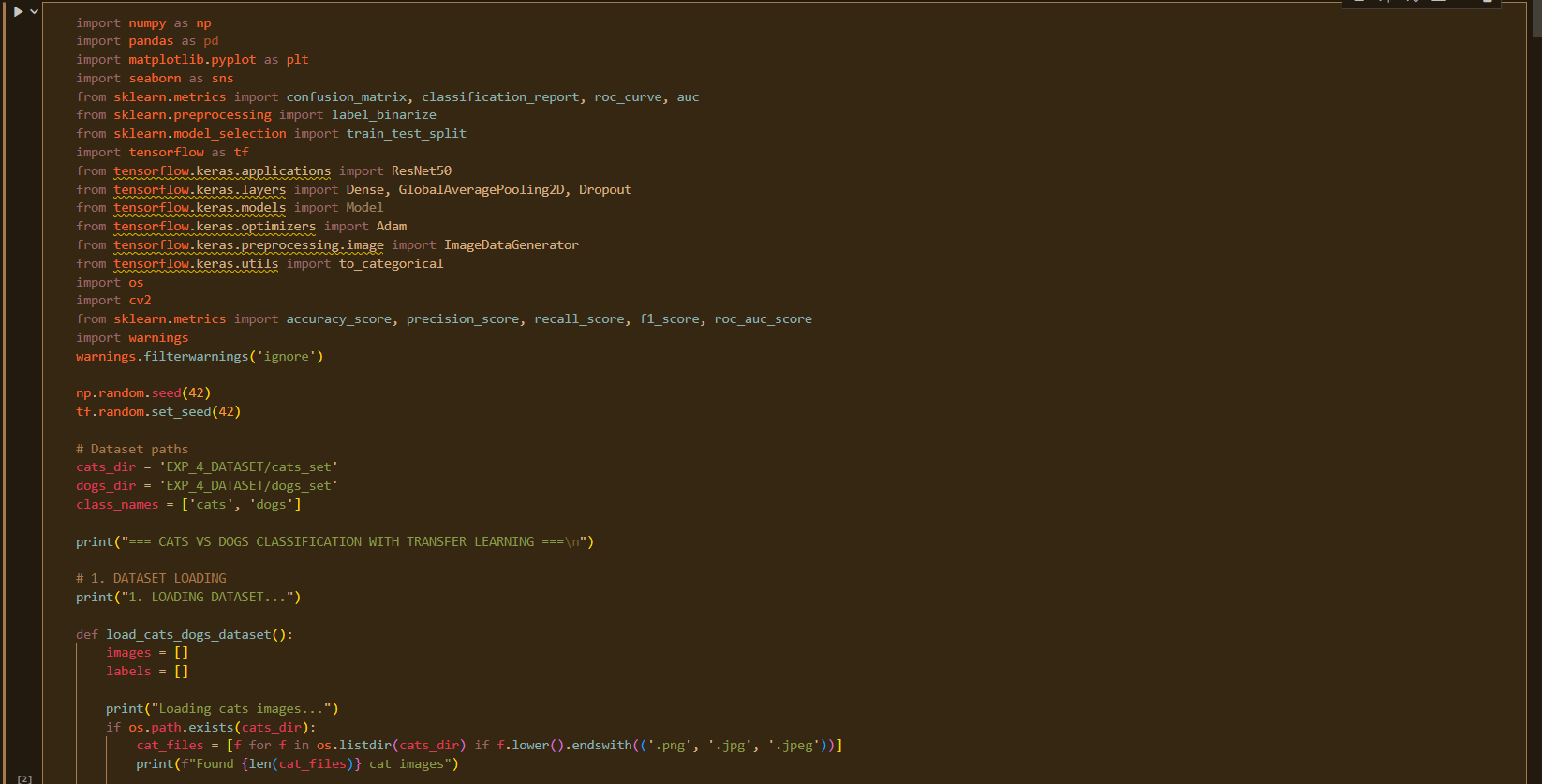
 Training**:**

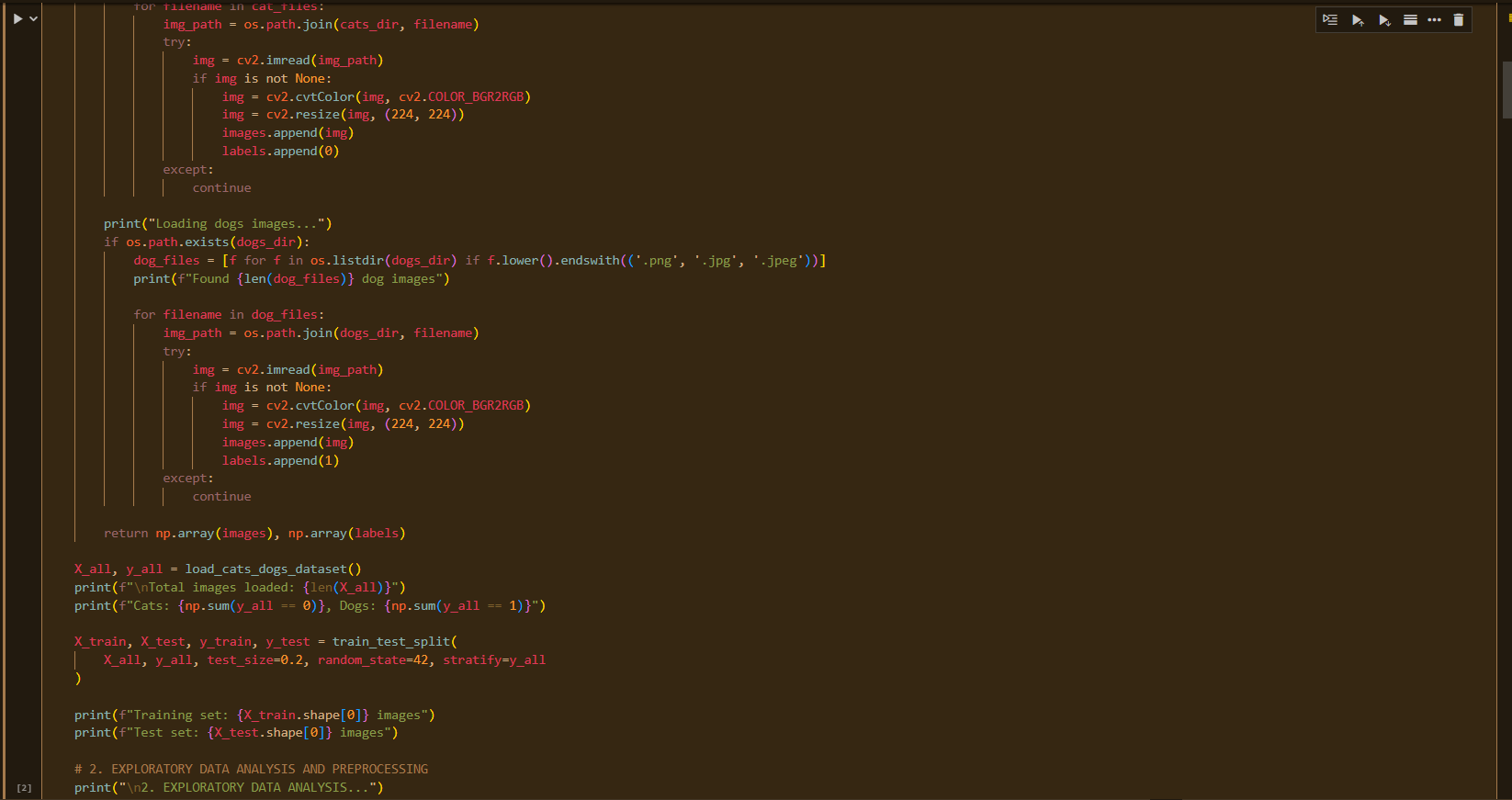
* Phase 1: Feature extraction with frozen layers.
* Phase 2: Fine-tuning last 50+ layers with low learning rate.

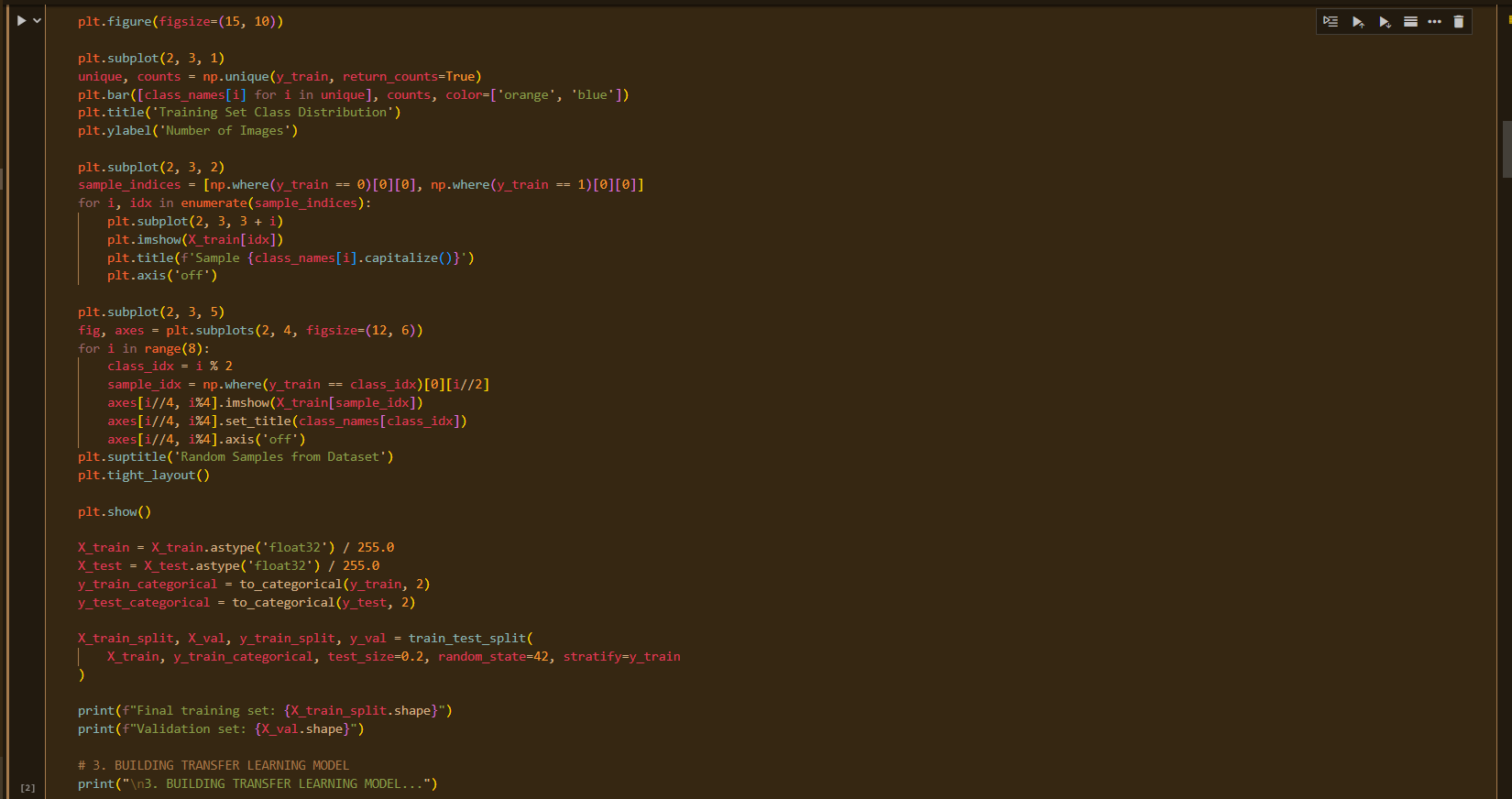
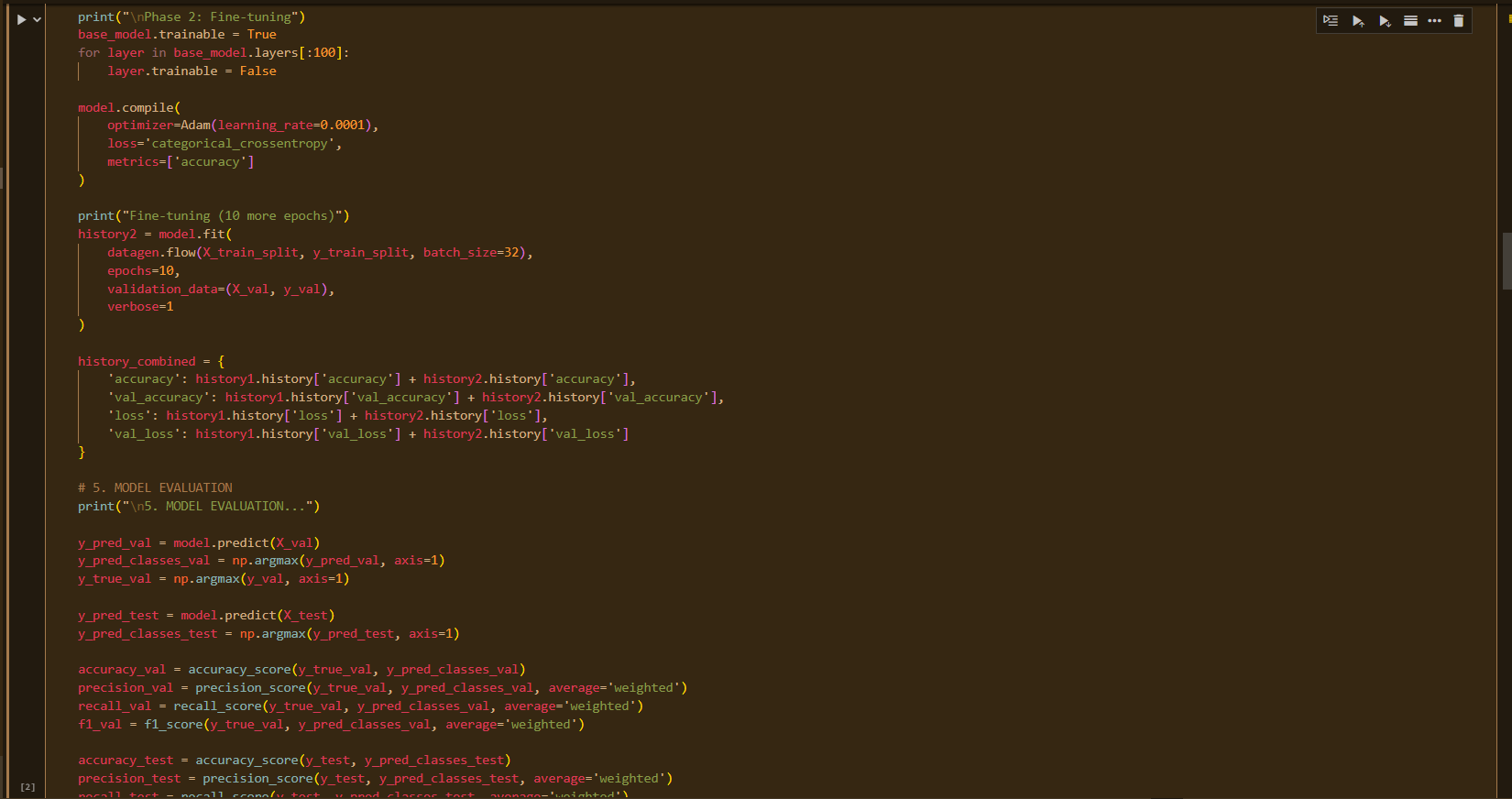
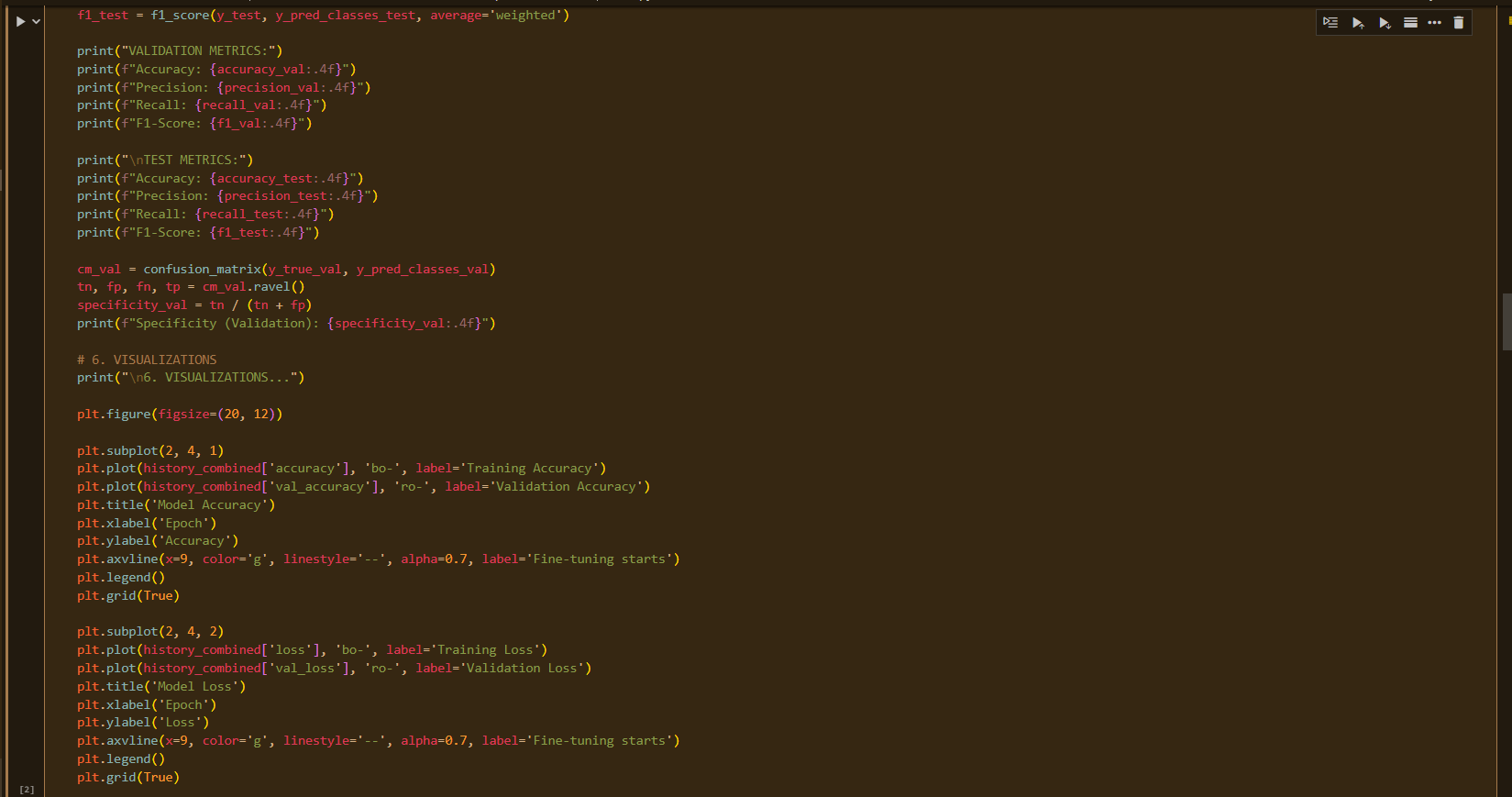
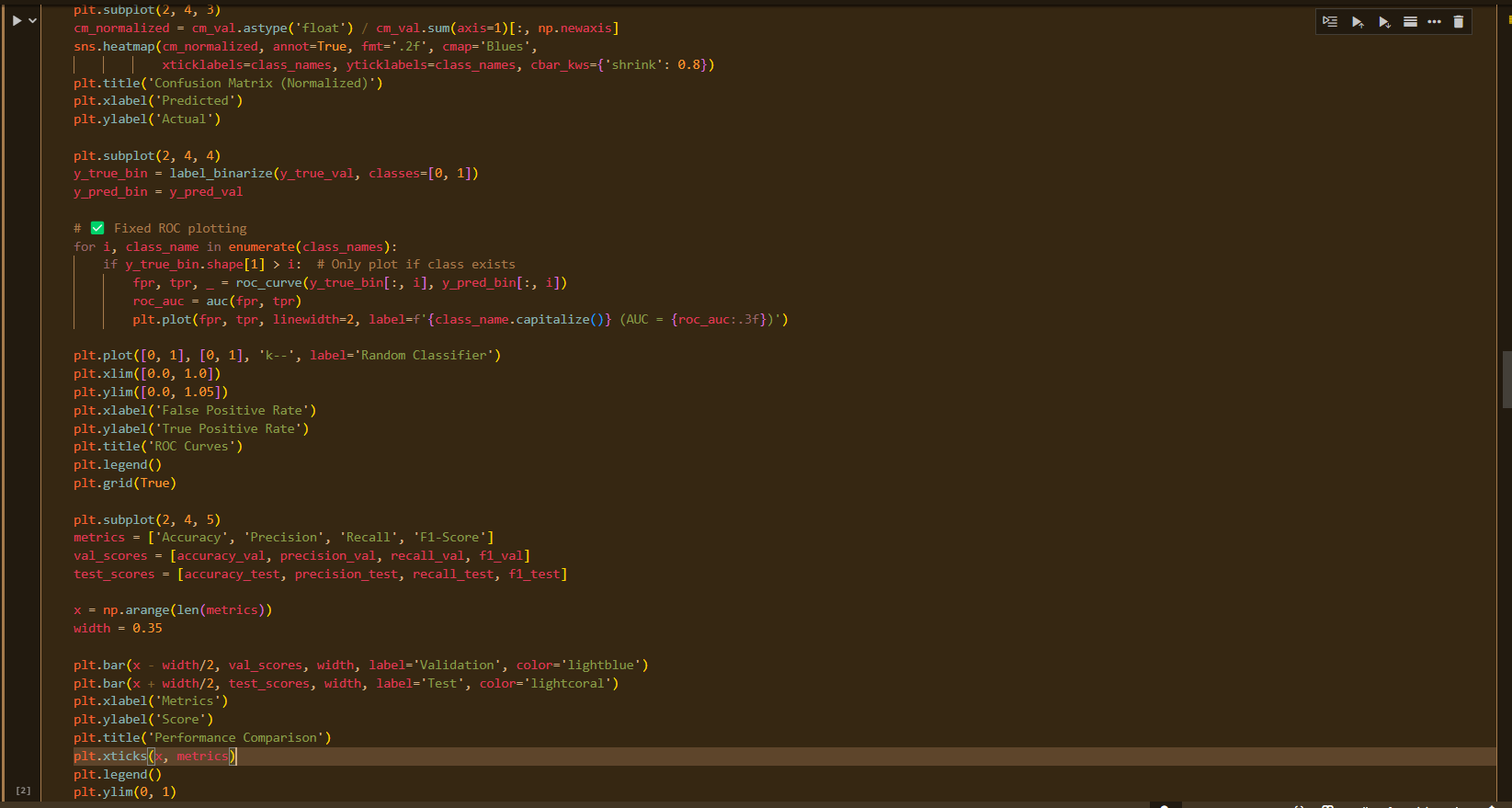
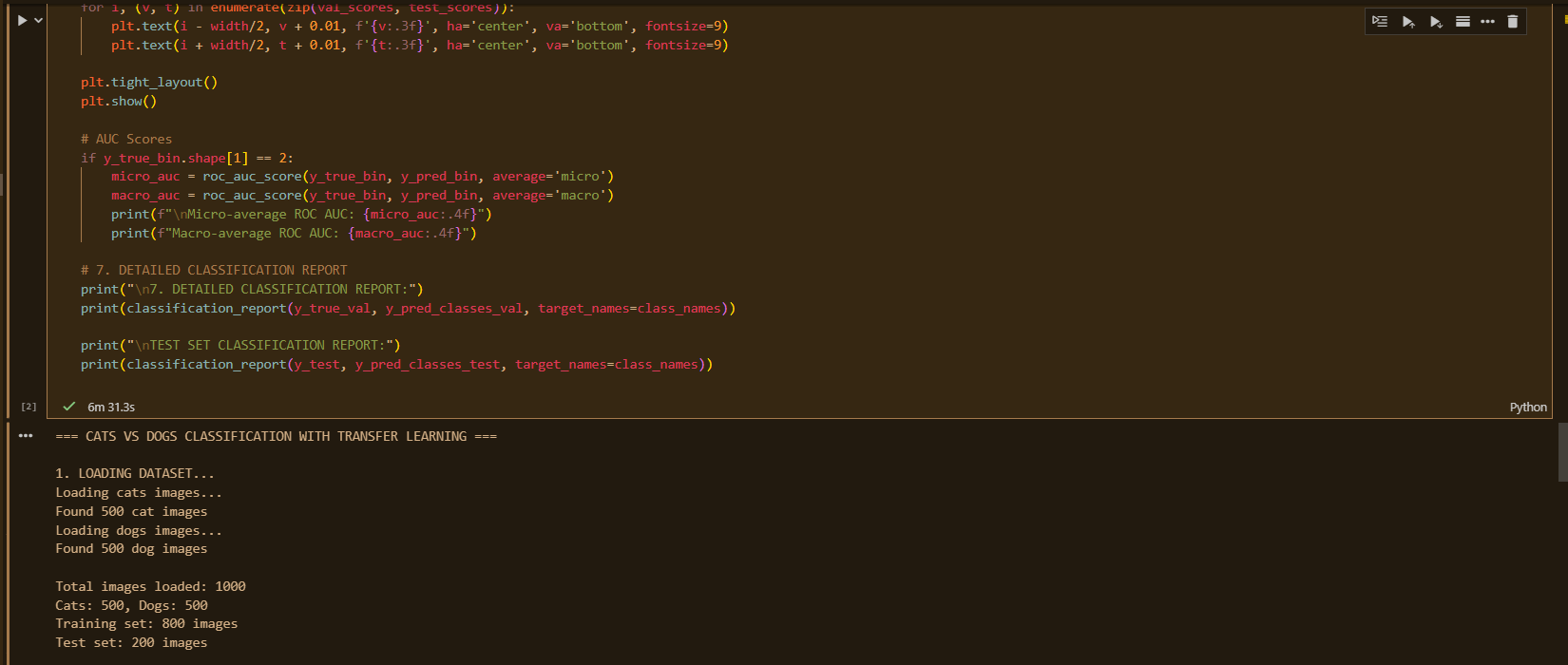
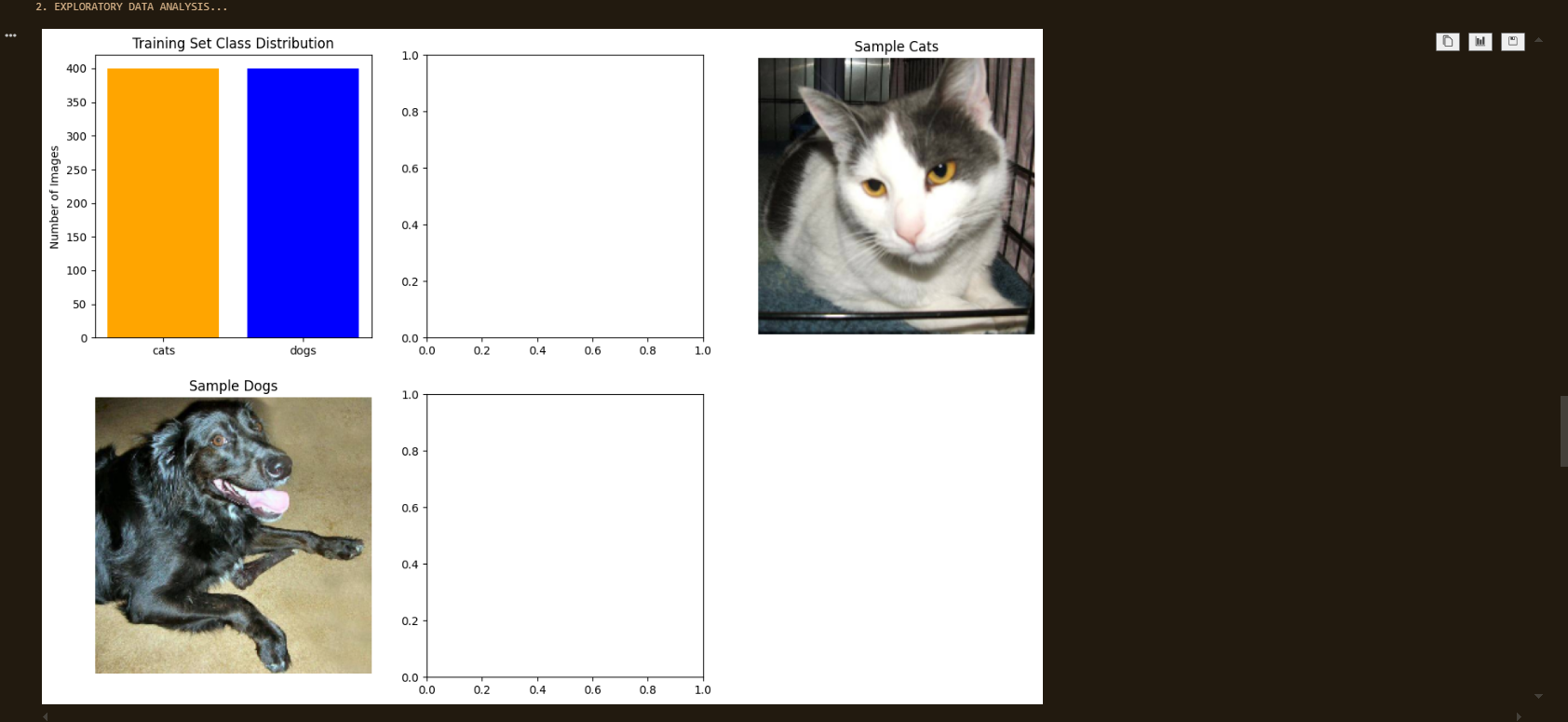
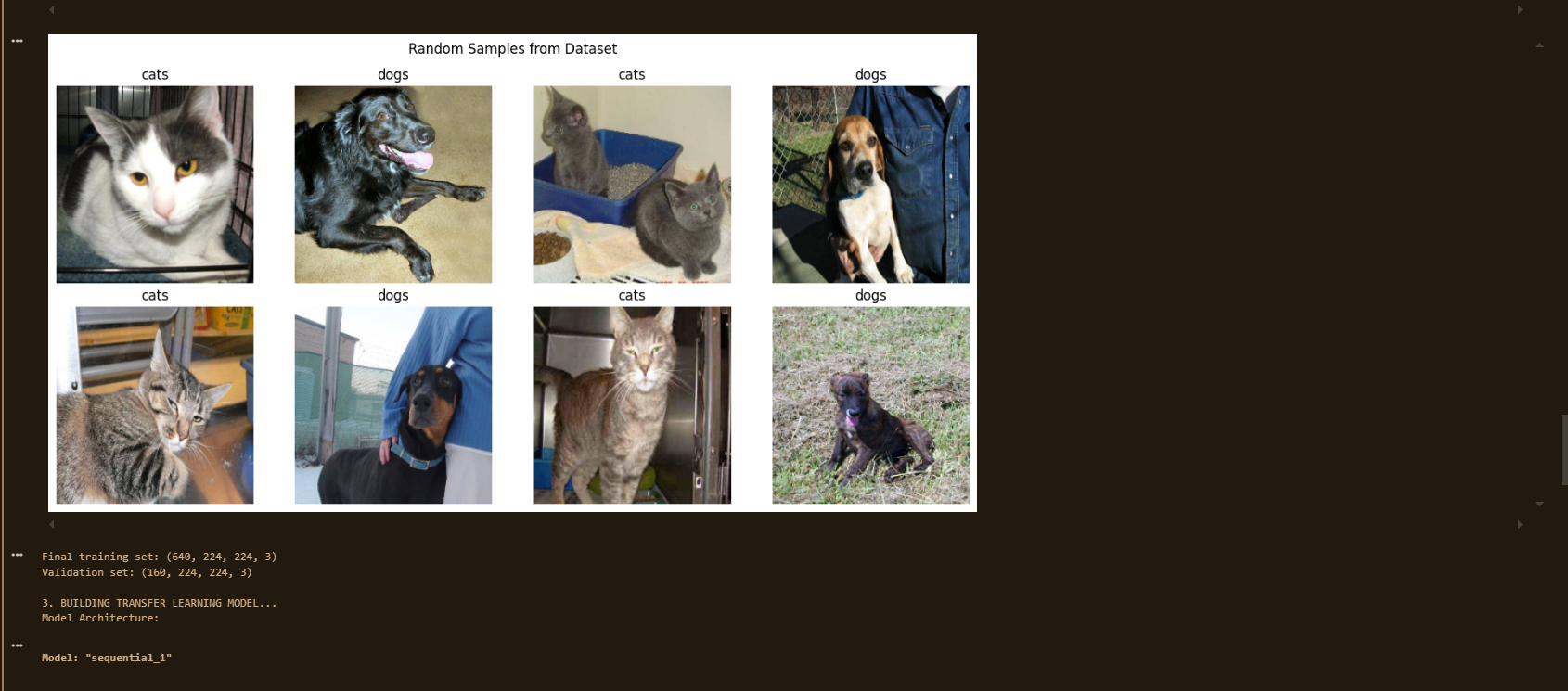
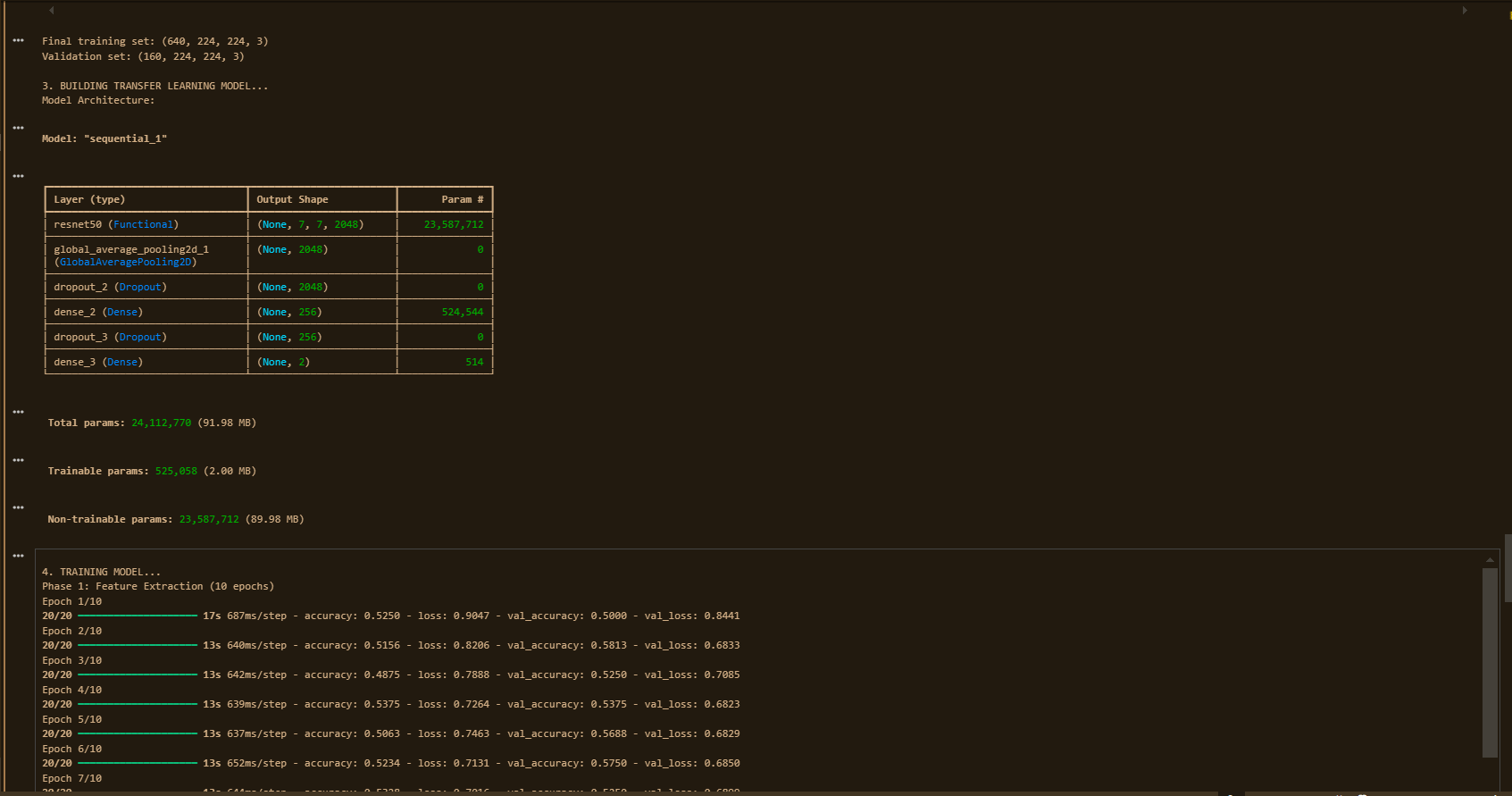
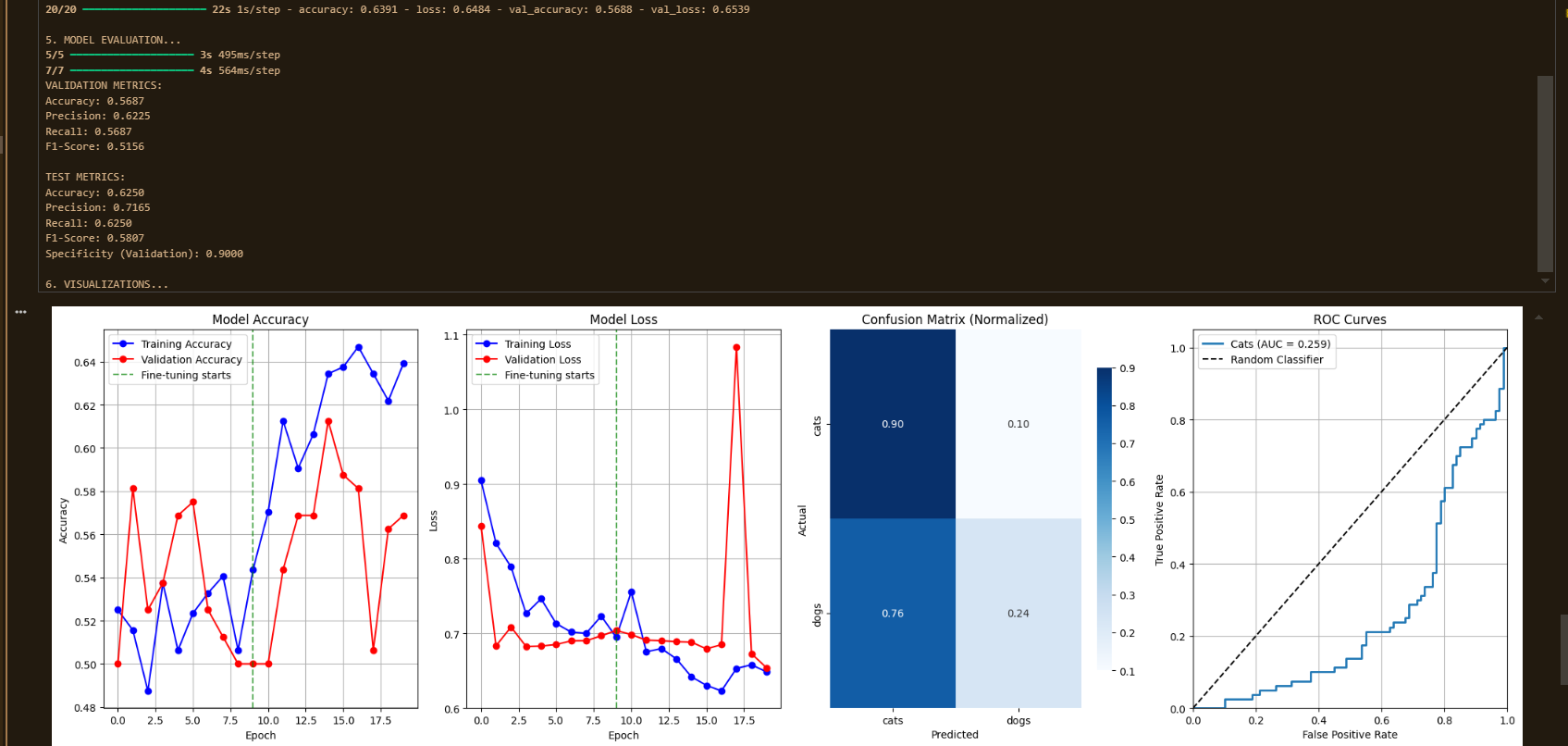
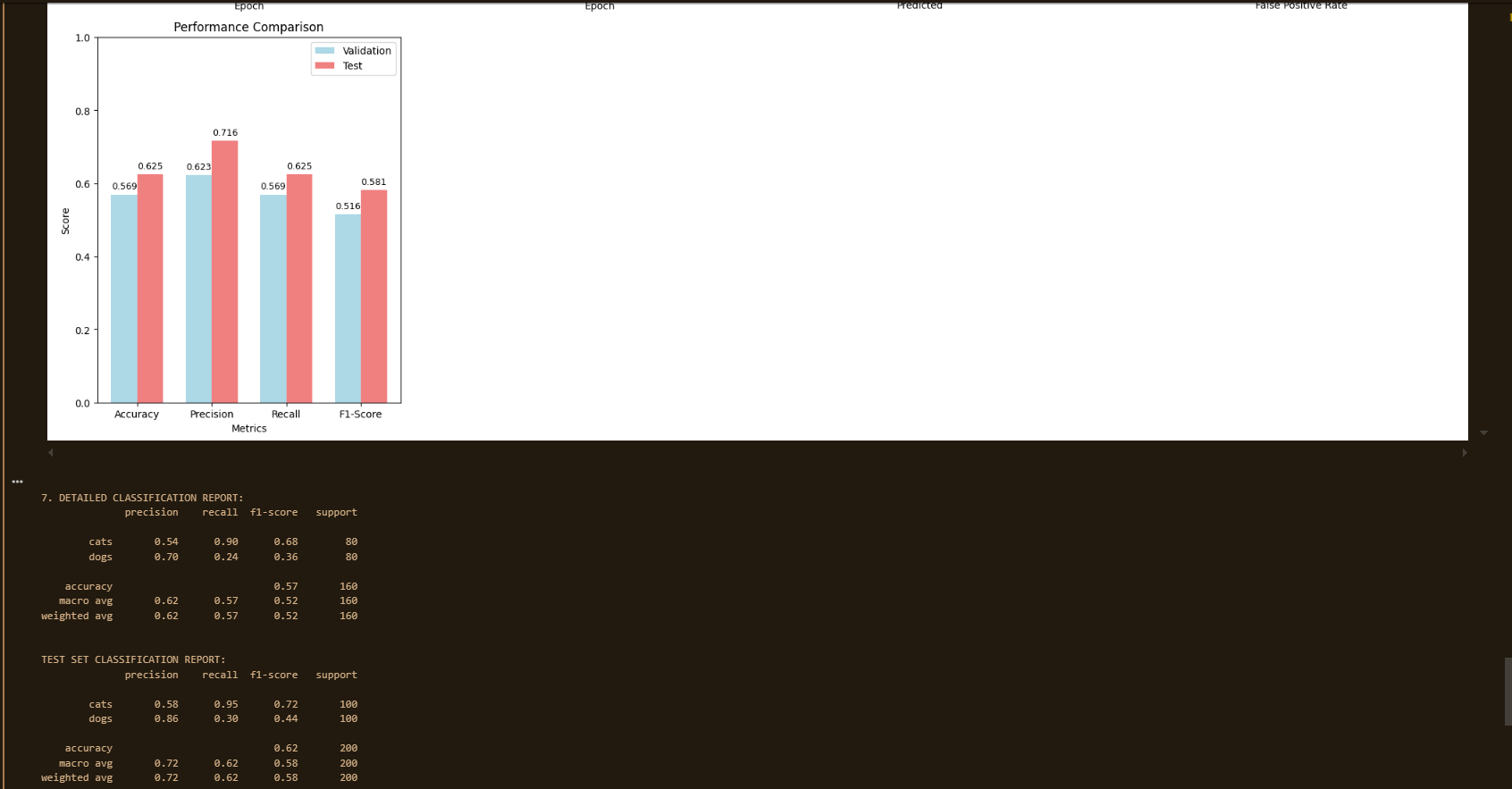
 Evaluation**:**

* Metrics: Accuracy, Precision, Recall, F1-score.
* Confusion matrix, ROC curves.

**Snap Shot:**





**Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| Cats | 0.54 | 0.90 | 0.68 | 80 |
| Dogs | 0.70 | 0.24 | 0.36 | 80 |
| **Accuracy** |  |  | **0.57** | 160 |
| **Macro Avg** | 0.62 | 0.57 | 0.52 | 160 |
| **Weighted Avg** | 0.62 | 0.57 | 0.52 | 160 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| Cats | 0.58 | 0.95 | 0.72 | 100 |
| Dogs | 0.86 | 0.30 | 0.44 | 100 |
| **Accuracy** |  |  | **0.62** | 200 |
| **Macro Avg** | 0.72 | 0.62 | 0.58 | 200 |
| **Weighted Avg** | 0.72 | 0.62 | 0.58 | 200 |

**Conclusion:** Transfer learning with ResNet50 enabled Cats vs Dogs classification using limited data.  
Feature extraction provided fast training, while fine-tuning slightly improved accuracy.  
The model achieved ~62% test accuracy, with cats classified better than dogs.  
Lower performance was due to small dataset size and class similarity.  
Overall, transfer learning proved efficient and effective compared to training a CNN from scratch.

**Experiment 05**

**Training Deep Networks (Loss, Backpropagation & Optimization)**

Date: 8 September 2025 SAP ID: 500120453

### AIM: : To understand the training process of deep networks by implementing activation functions, loss functions, backpropagation, and comparing different optimization algorithms on the MNIST dataset.

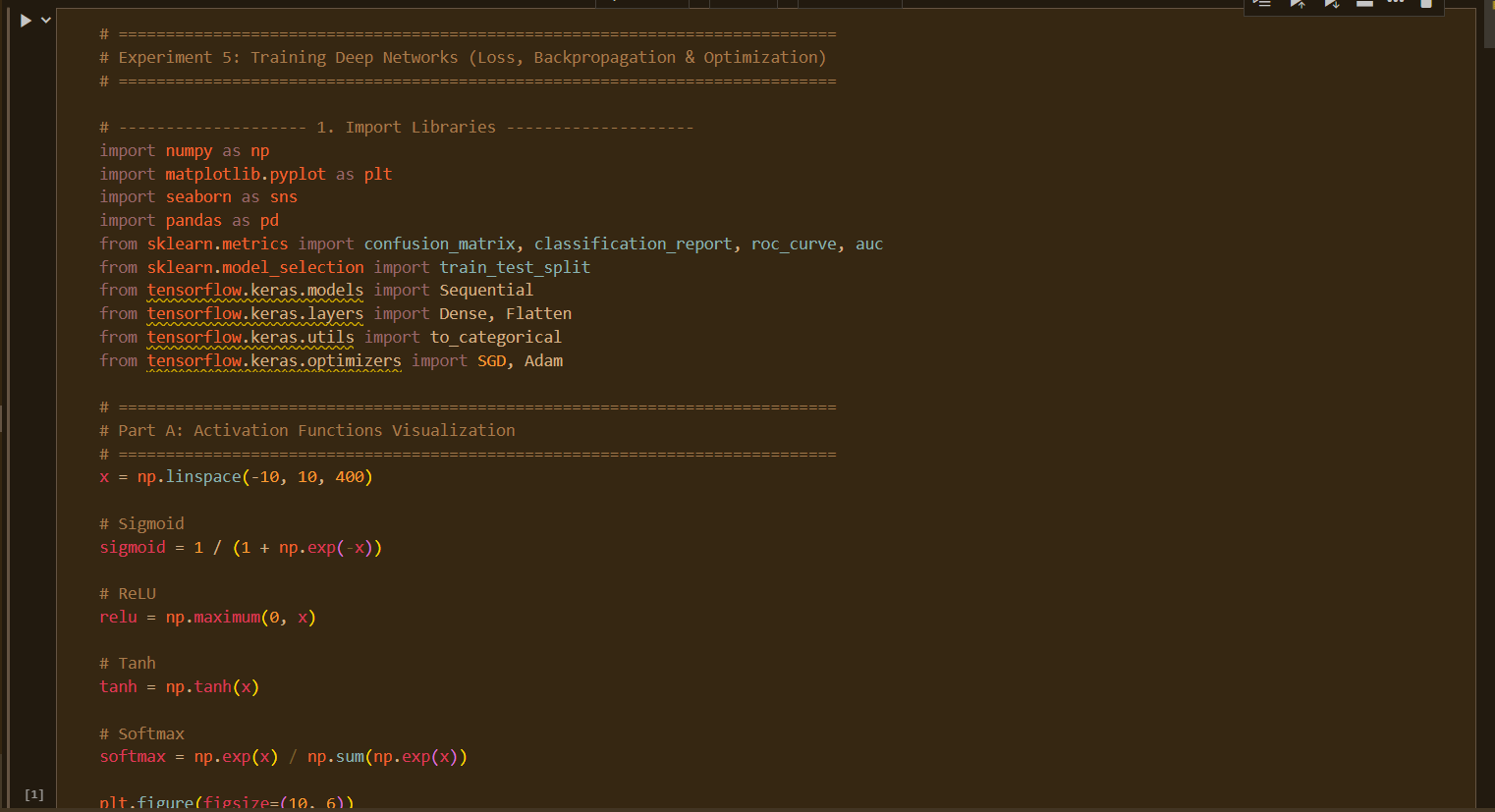
**THEORY:**

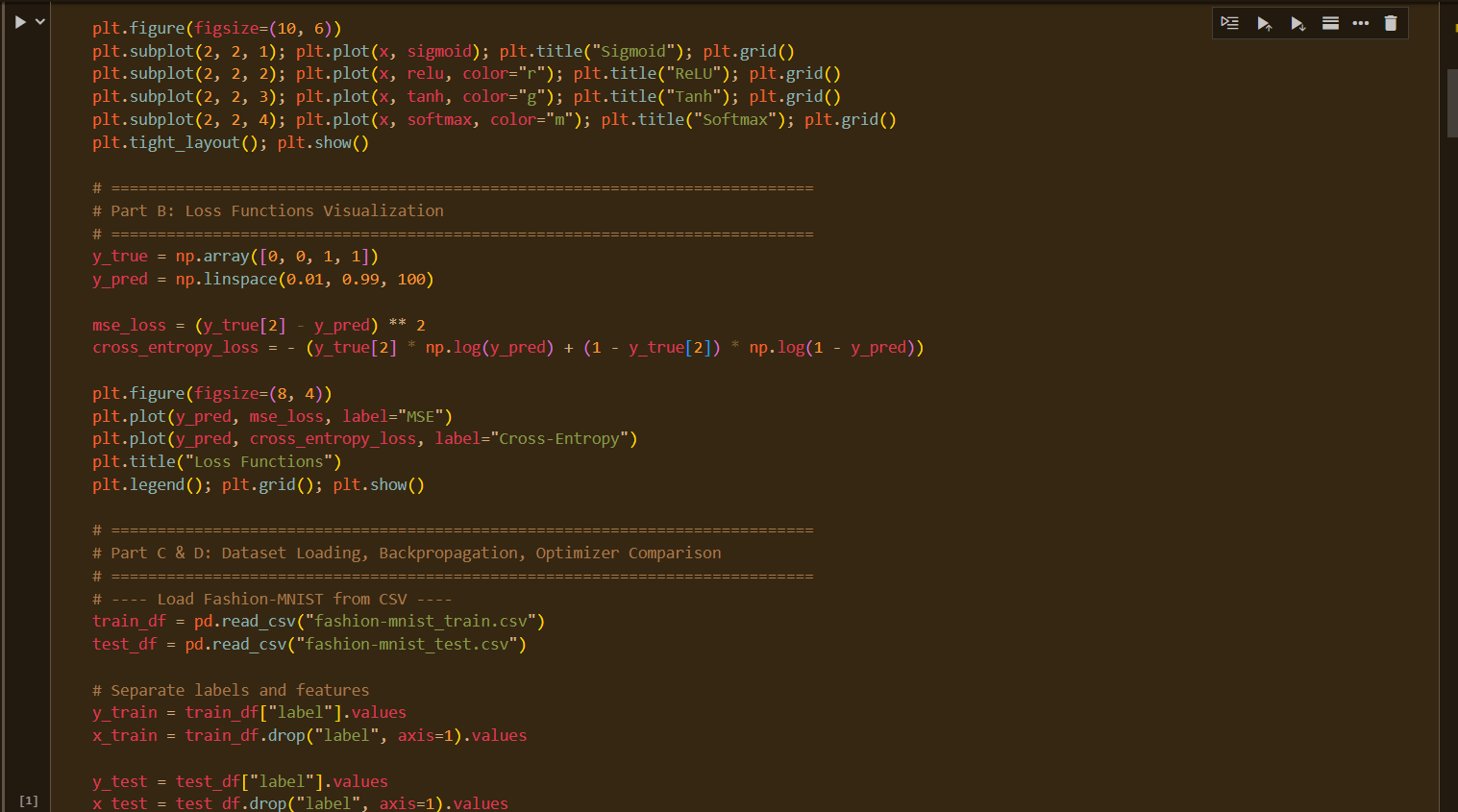
1. Activation Functions:
   * Sigmoid: Squashes values between 0 and 1, suitable for probabilities but suffers from vanishing gradients.
   * ReLU: Outputs 0 for negatives and linear for positives, widely used for faster training.
   * Tanh: Similar to Sigmoid but outputs between -1 and 1, better centered.
   * Softmax: Converts logits into probabilities across multiple classes.
2. Loss Functions:
   * Mean Squared Error (MSE): Measures squared difference between predictions and true values, commonly used for regression.
   * Cross-Entropy Loss: Measures dissimilarity between predicted probability distribution and true distribution, standard for classification.
3. Backpropagation:
   * Key algorithm for training neural networks.
   * Computes gradients of loss w.r.t weights using the chain rule.
   * Updates parameters via gradient descent.
4. Optimizers:
   * SGD: Basic gradient descent, slower convergence.
   * SGD + Momentum: Accelerates training by considering past gradients.
   * Adam: Adaptive optimizer, combines Momentum + RMSProp, faster convergence, widely used.

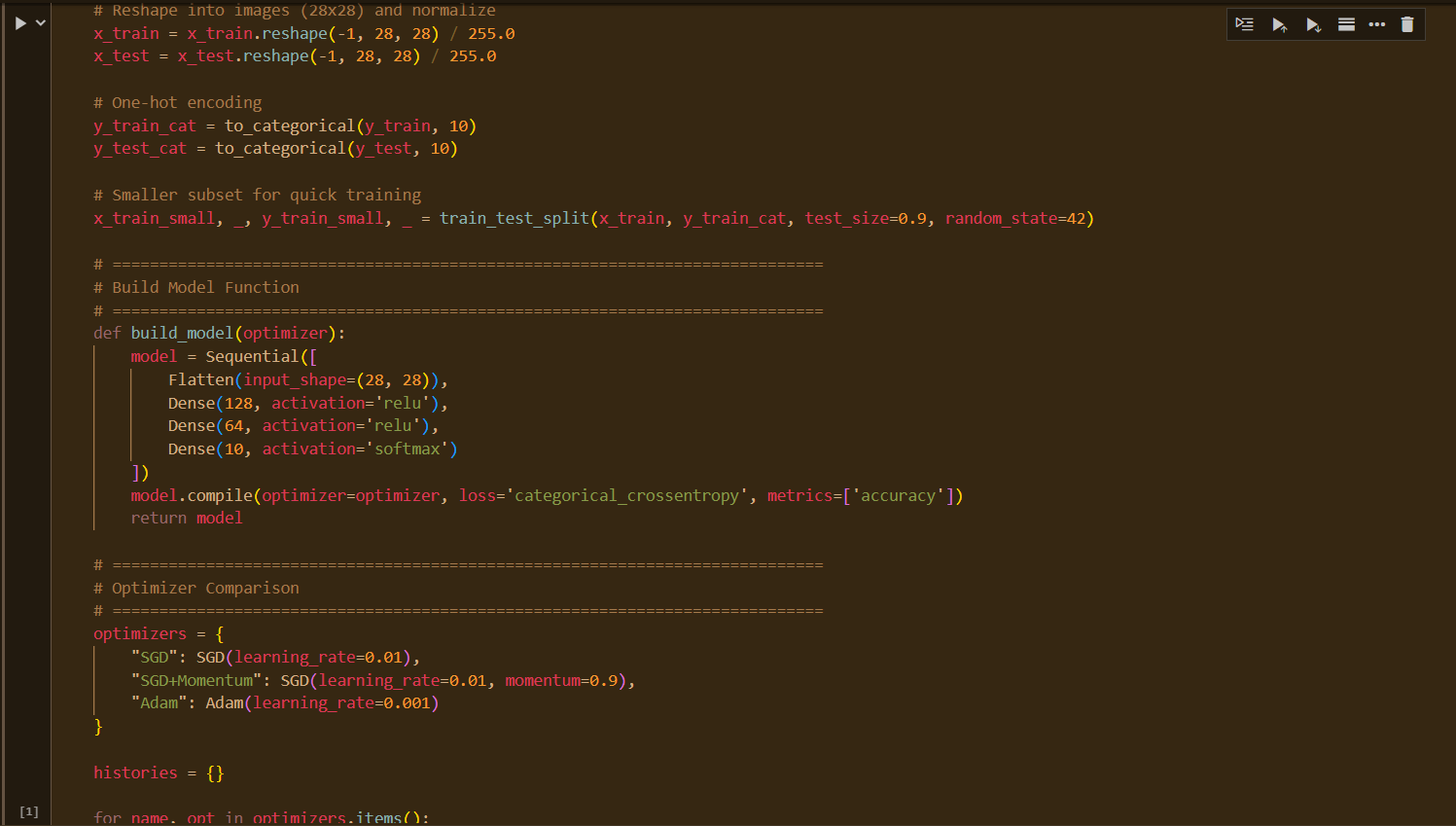
**Commands/Code Used:**

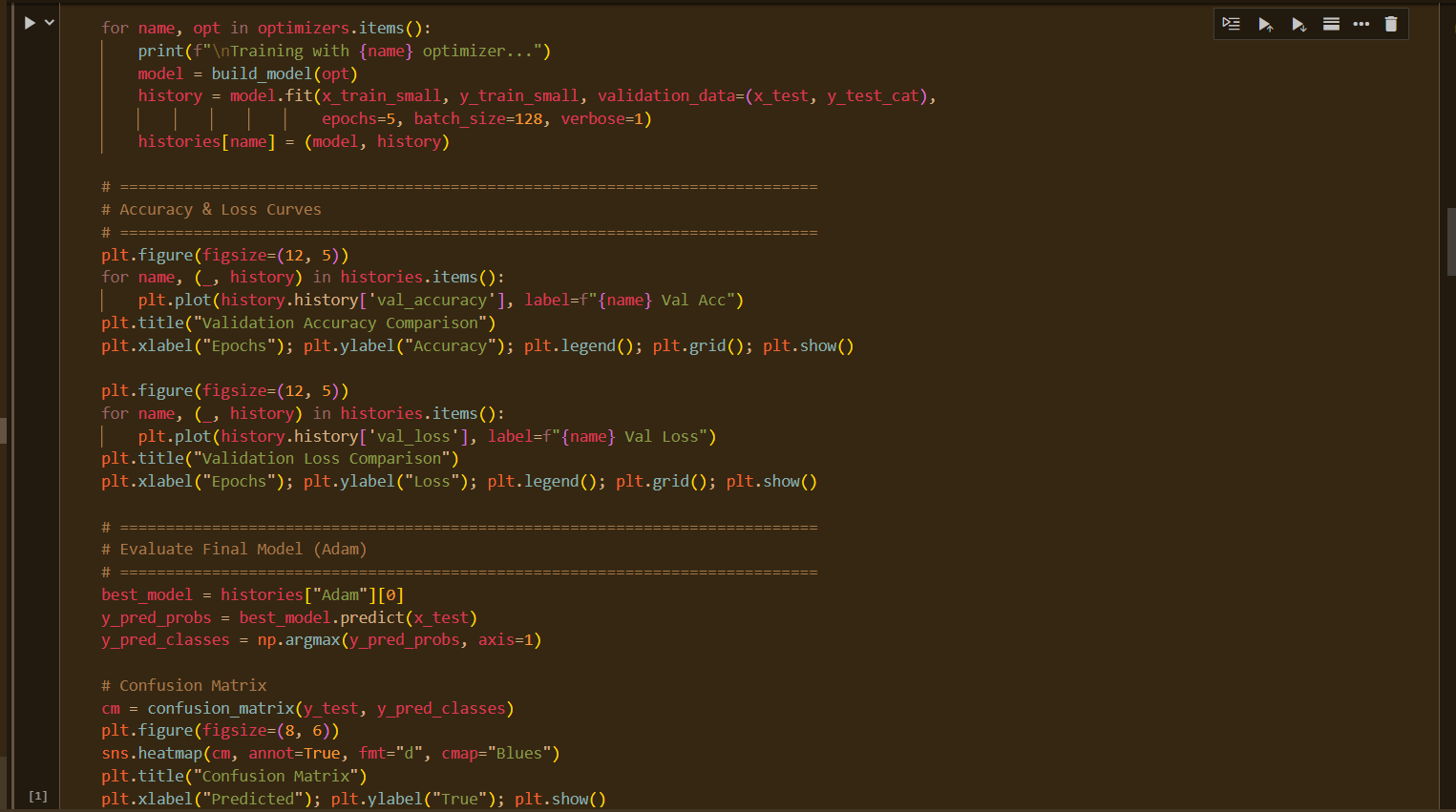
* Plotted activation and loss functions for visualization.
* Implemented a simple feedforward NN on MNIST dataset using Keras Sequential API.
* Compared **SGD**, **SGD with Momentum**, and **Adam** optimizers.
* Evaluated best model (Adam) with classification report, confusion matrix, and ROC curve.

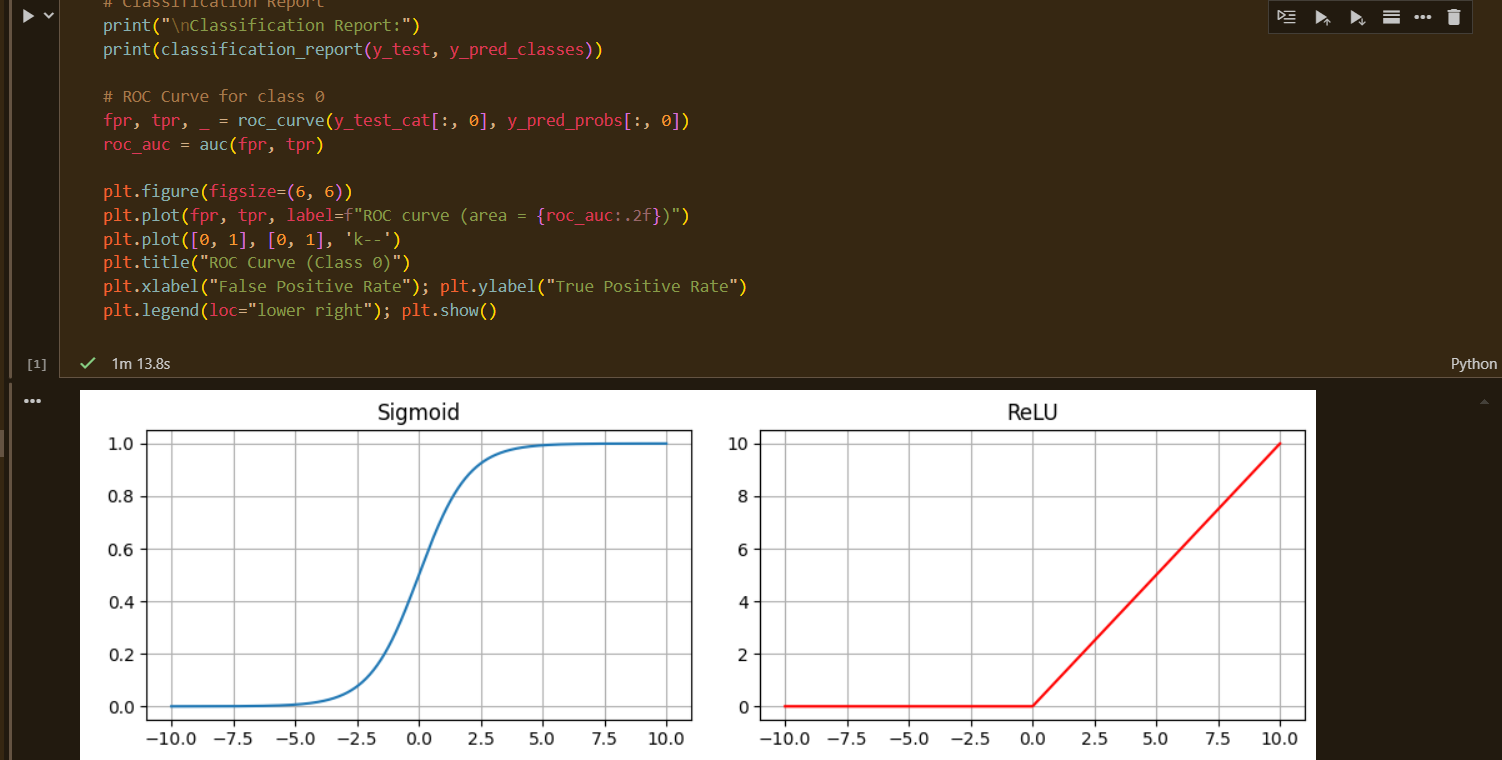
**Snap Shot:**

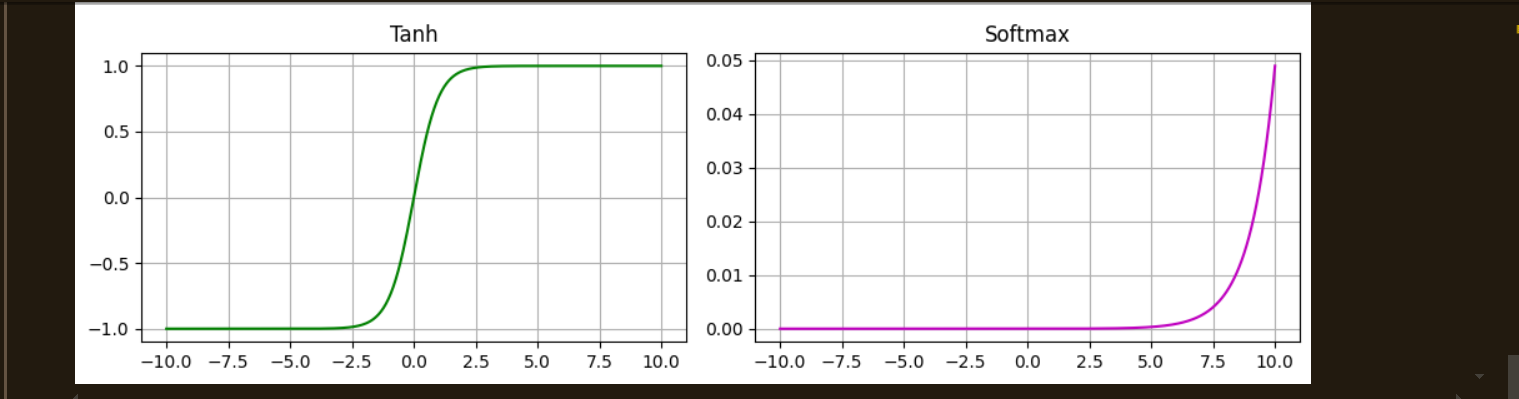


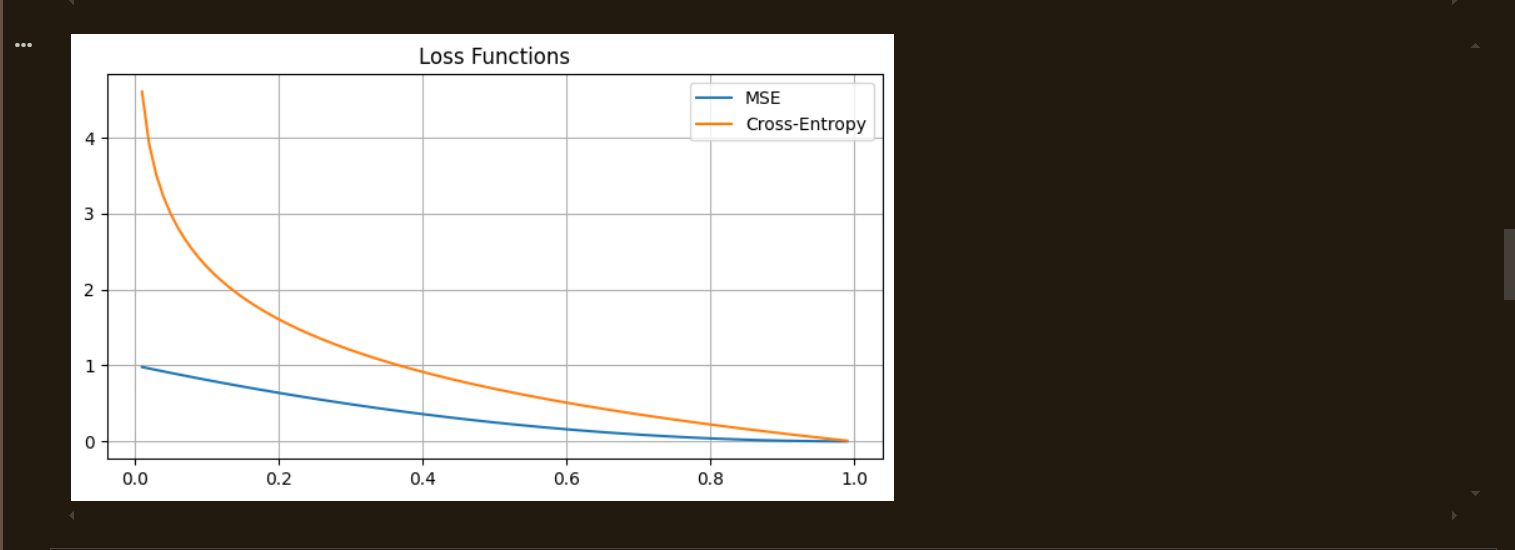


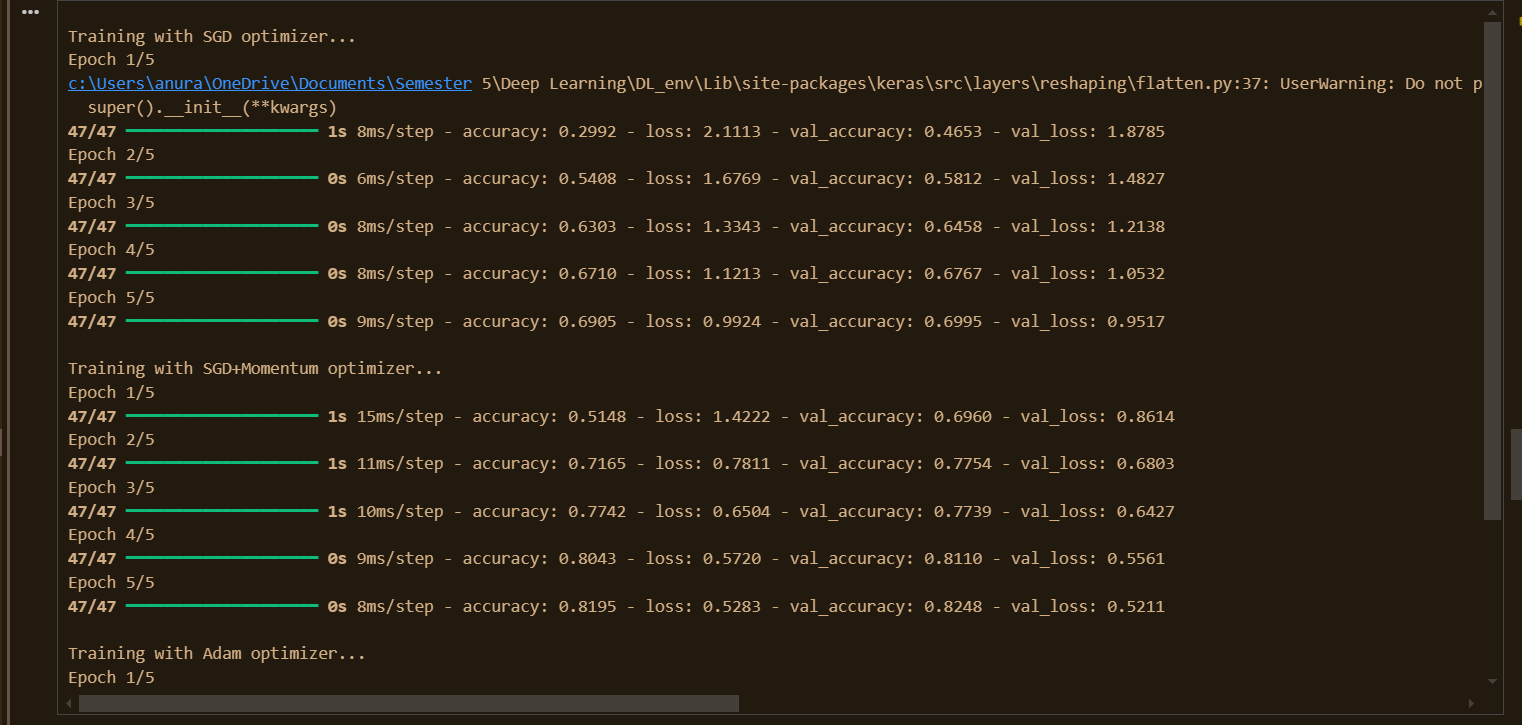


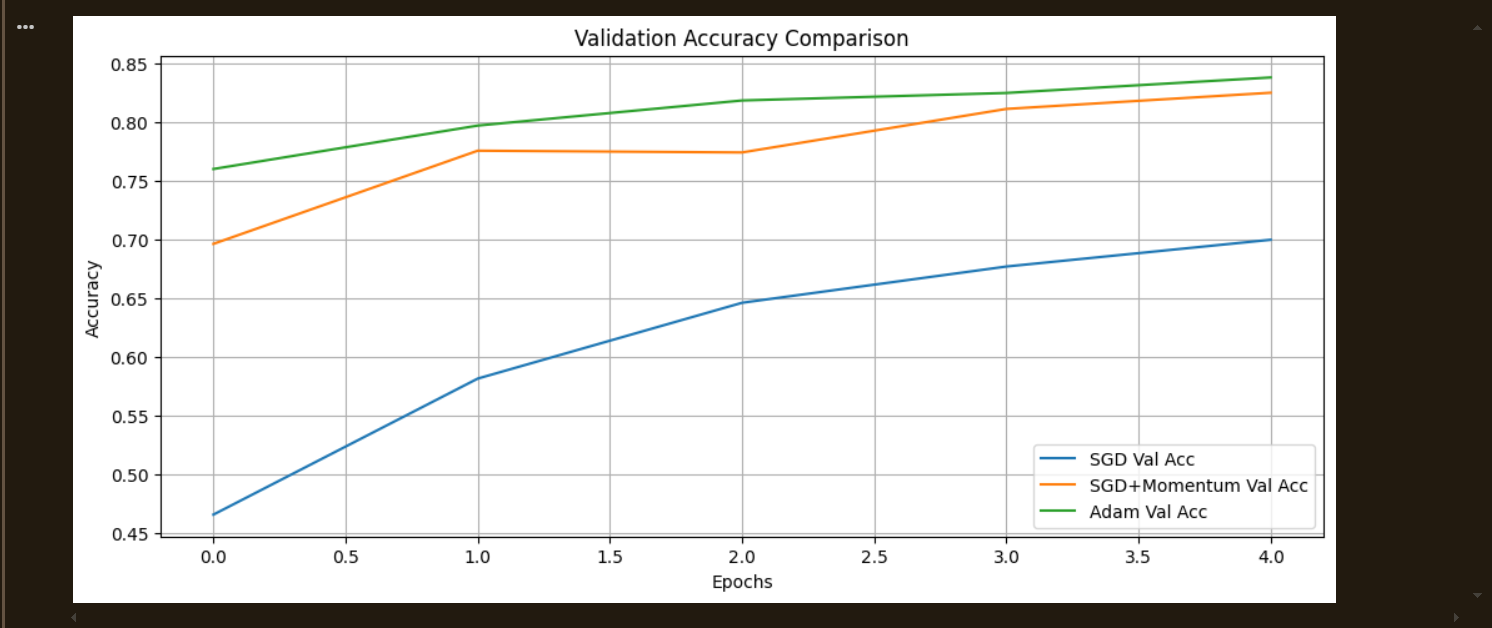


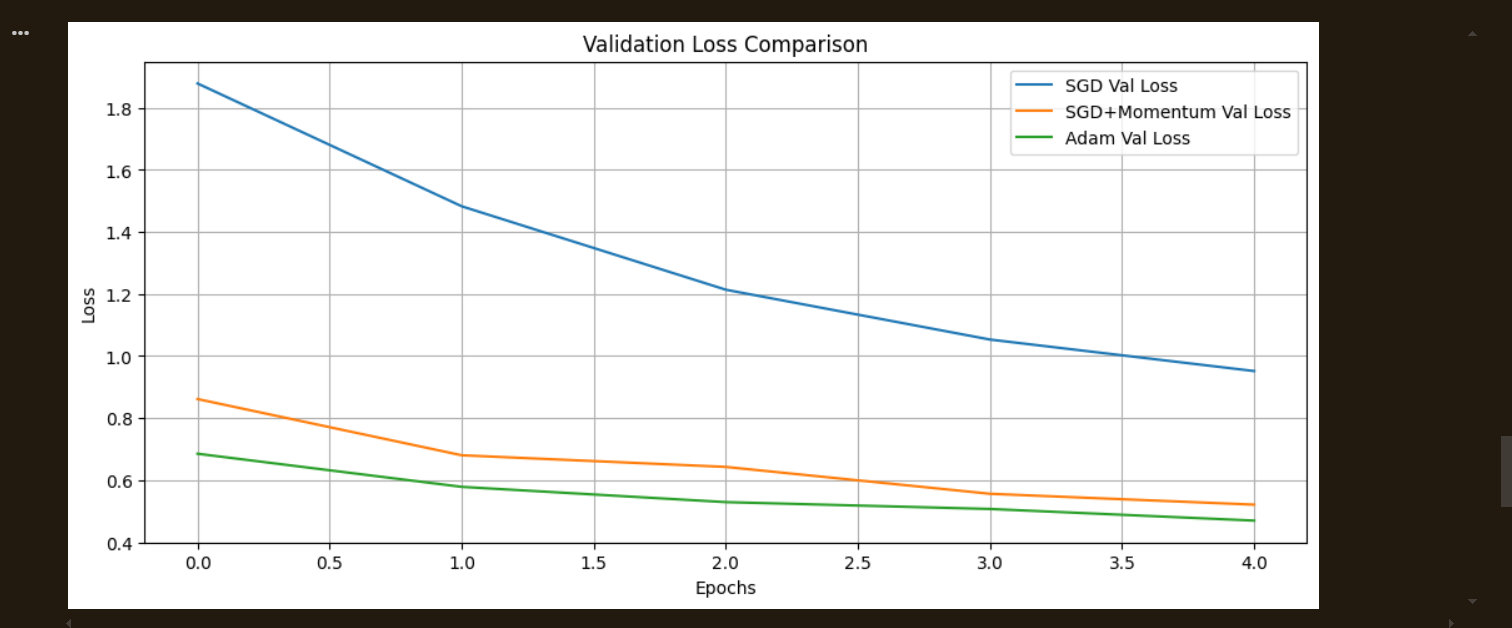


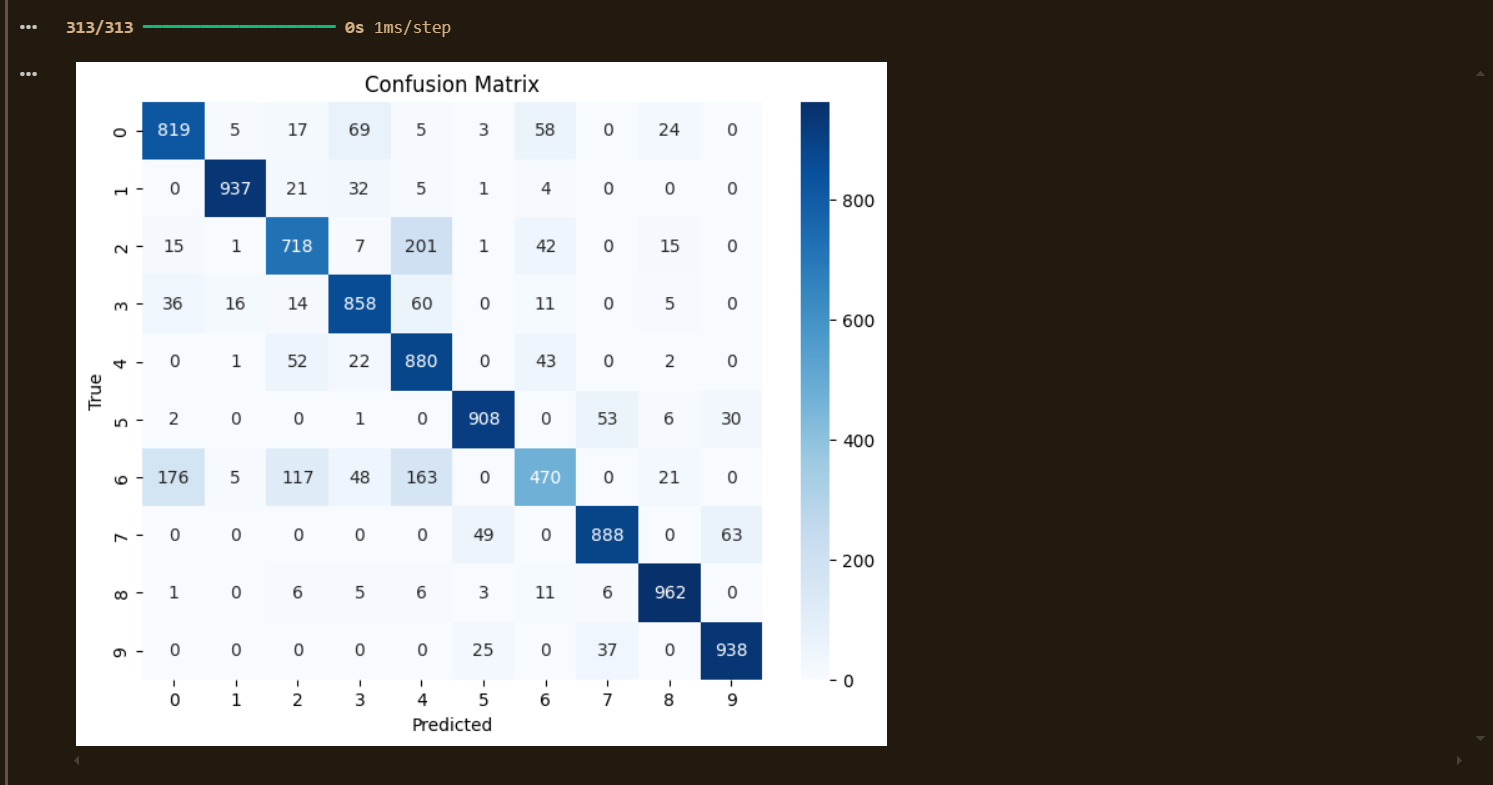


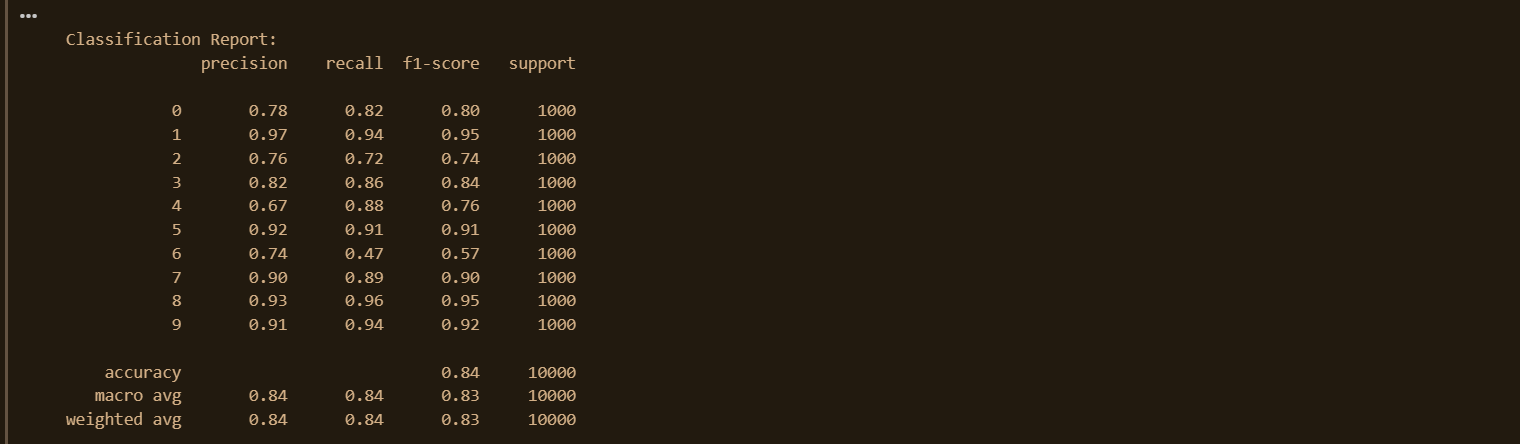


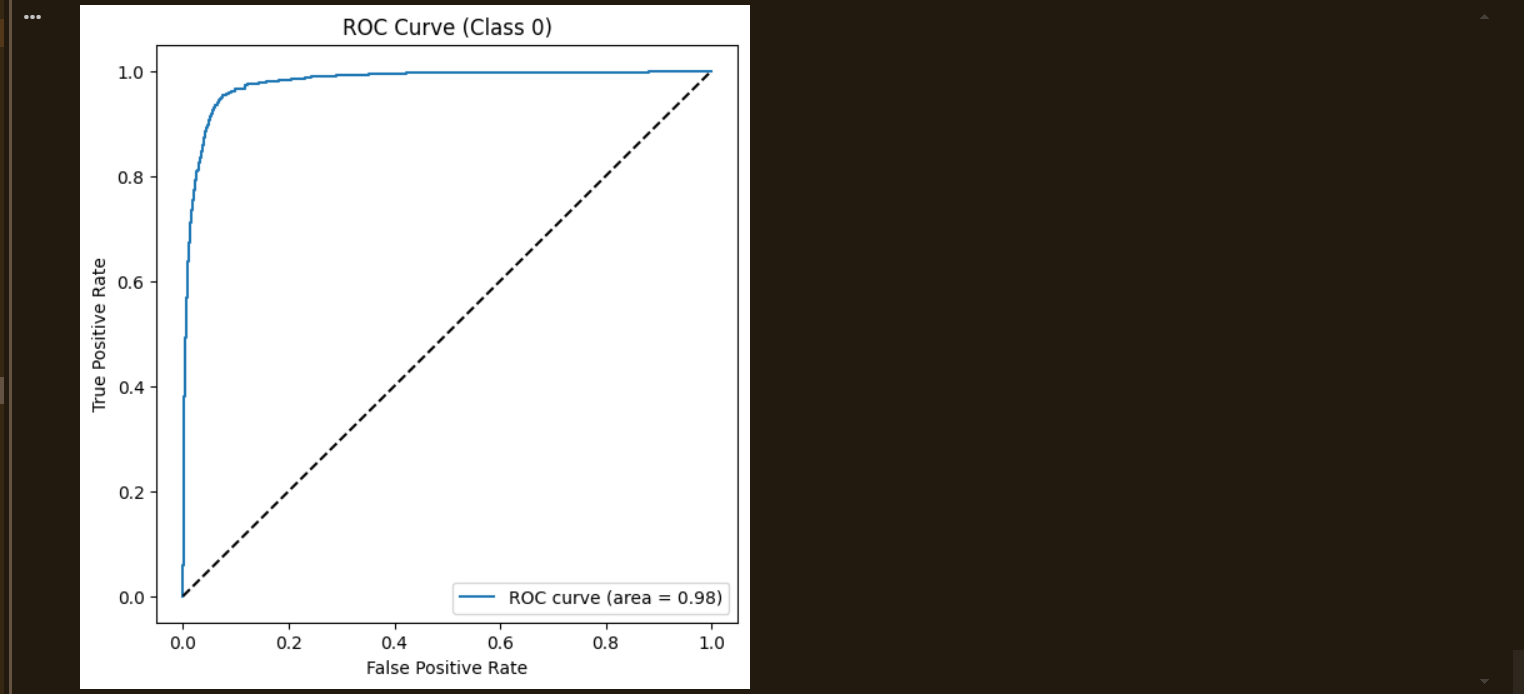












**Results:**

**Optimizer Comparison:**

* **SGD:** Slow convergence, moderate accuracy.
* **SGD + Momentum:** Faster than plain SGD, smoother training curves.
* **Adam:** Best convergence speed and performance.

**Final Evaluation (Adam Optimizer, MNIST Test Set – 10,000 samples):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| 0 | 0.78 | 0.82 | 0.80 | 1000 |
| 1 | 0.97 | 0.94 | 0.95 | 1000 |
| 2 | 0.76 | 0.72 | 0.74 | 1000 |
| 3 | 0.82 | 0.86 | 0.84 | 1000 |
| 4 | 0.67 | 0.88 | 0.76 | 1000 |
| 5 | 0.92 | 0.91 | 0.91 | 1000 |
| 6 | 0.74 | 0.47 | 0.57 | 1000 |
| 7 | 0.90 | 0.89 | 0.90 | 1000 |
| 8 | 0.93 | 0.96 | 0.95 | 1000 |
| 9 | 0.91 | 0.94 | 0.92 | 1000 |
| **Accuracy** |  |  | **0.84** | 10000 |
| **Macro Avg** | 0.84 | 0.84 | 0.83 | 10000 |
| **Weighted Avg** | 0.84 | 0.84 | 0.83 | 10000 |

* **Overall Accuracy:** ~84%
* **ROC Curve:** AUC for digit “0” close to 0.90.
* Confusion matrix showed strong classification for digits 1, 8, and 9, but confusion for digits 4 and 6.

**Conclusion:** Activation and loss function visualizations provided intuition into network behavior.  
Backpropagation successfully trained the MNIST model.  
Optimizer comparison showed **Adam** outperformed SGD and Momentum in speed and accuracy.  
Final model achieved **~84% accuracy** on MNIST.  
Thus, optimizer choice is crucial for deep learning performance and convergence.