T.C.

YILDIZ TECHNICAL UNIVERSITY

FACULTY OF MECHANICAL ENGINEERING

DEPARTMENT OF MECHATRONICS ENGINEERING

# Spring 2024-2025

# MKT 3434 Machine Learning

# Homework-3

Elif TUNÇ

Lecturer

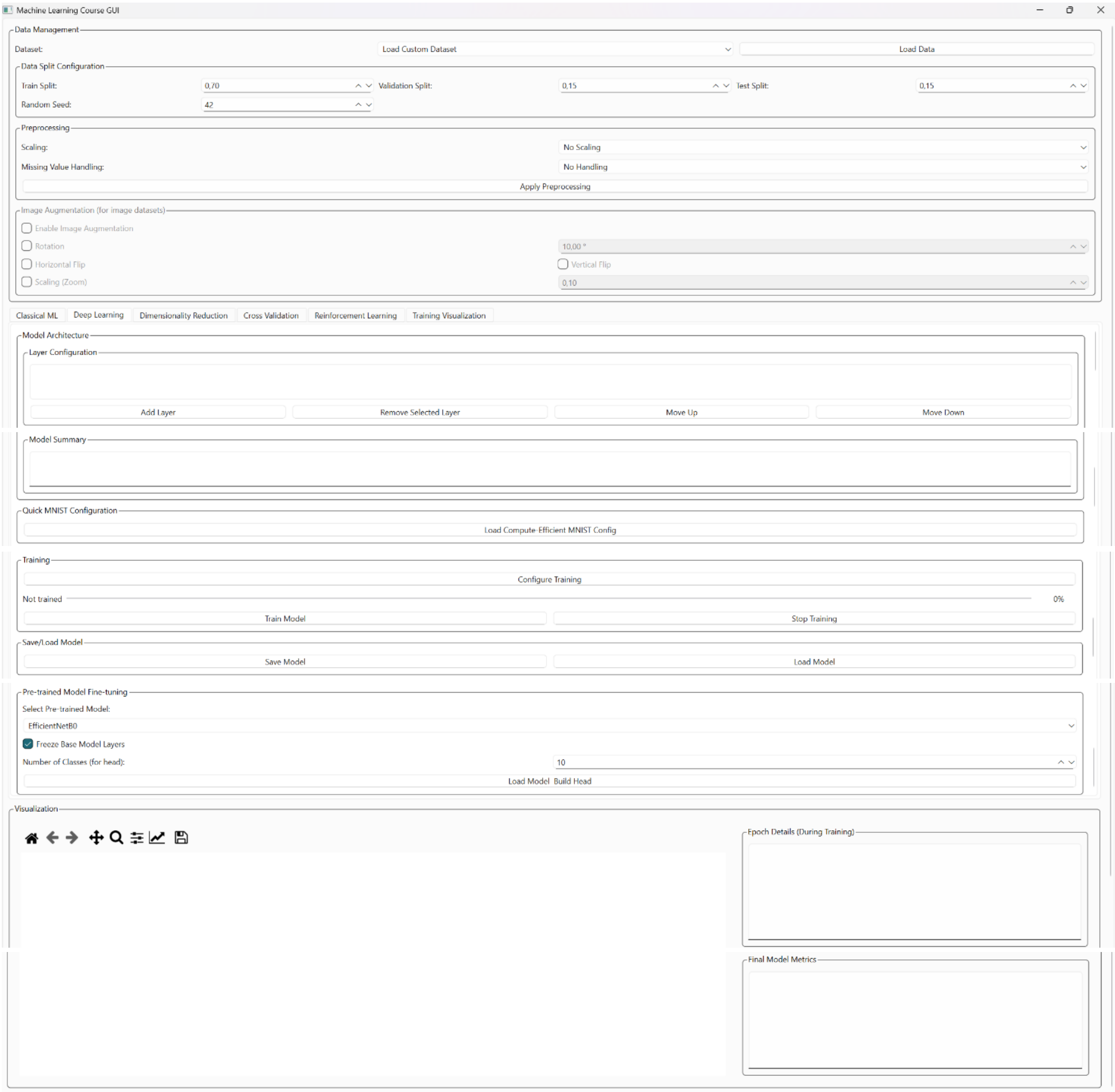
Ertuğrul Bayraktar

May, 2025

## 1. Introduction

This project presents an interactive deep learning GUI developed with PyQt6 and TensorFlow/Keras. The aim is to allow users to build, configure, train, and evaluate neural network models without writing code. This interface is especially suited for educational use, experiments, and prototyping. Users can:

- Select or upload datasets
- Design models layer-by-layer (Dense, CNN, RNN)
- Apply preprocessing and augmentation
- Choose training configurations
- Visualize training and evaluation metrics




*Screenshots 1, 2, 3, 4, 5: Main GUI interface with all tabs visible*

## 2. Model Architecture Design

### 2.1. Layer Configuration Panel

Under the **Deep Learning** tab, users are presented with dynamic tools to design a model by adding and modifying layers.

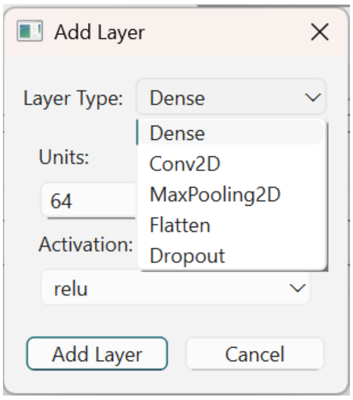
Supported layers:

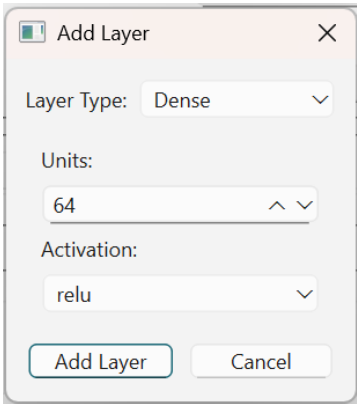
- **Dense**: Fully connected layer with user-defined units and activation functions (ReLU, Sigmoid, Tanh).
- **Conv2D**: Convolutional layer with parameters for kernel size, stride, and number of filters.
- **MaxPooling2D**: Downsampling operation for feature maps.
- **Dropout**: Prevents overfitting by randomly deactivating neurons.
- **LSTM / GRU**: For sequence modeling (e.g., time series or text).
- **Flatten, GlobalAveragePooling2D**: For dimensionality reduction before the output layer.

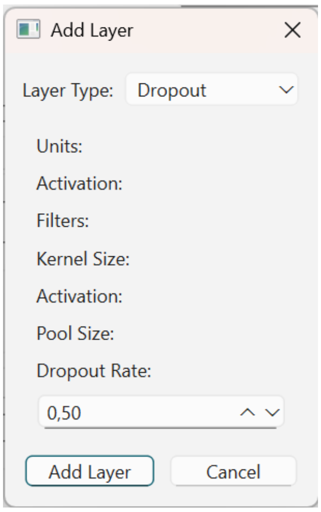
Users can dynamically:

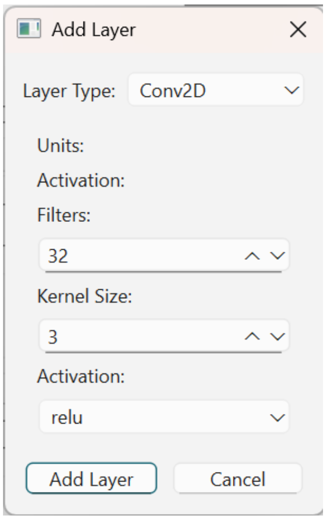
- Add/remove layers
- Reorder layers (Move Up/Down)

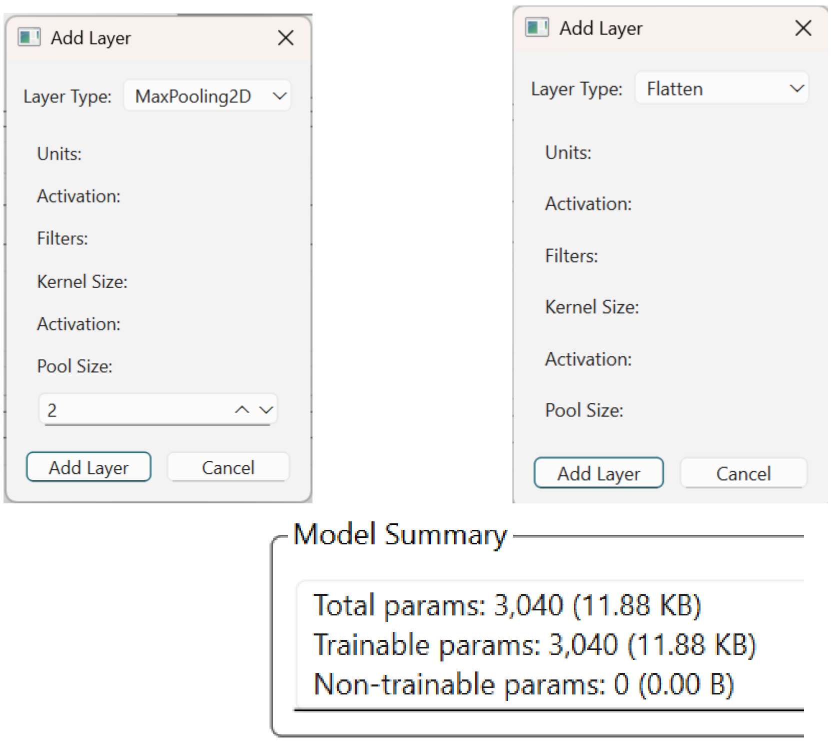
Each added layer appears in a scrollable list and contributes to a live-updated model summary.



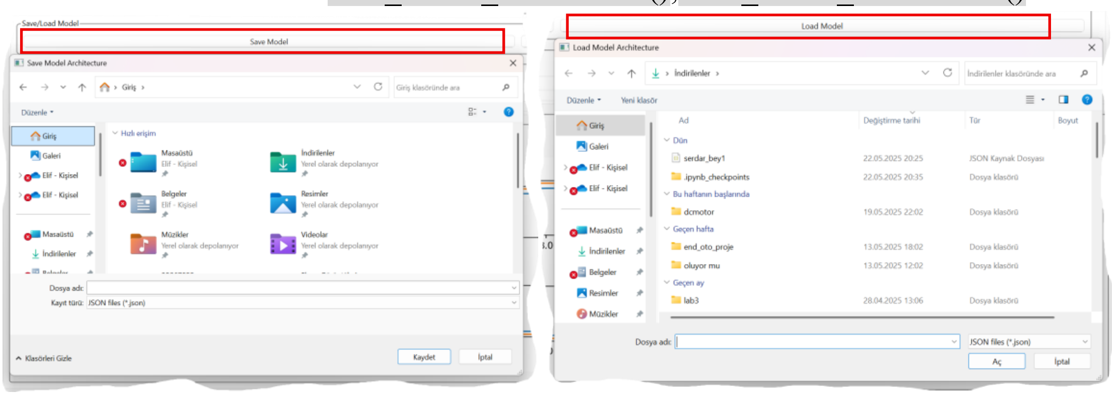



*Screenshots 6, 7, 8, 9, 10, 11, 12: A sample architecture of Dense -> Dropout -> Conv2D -> MaxPooling2D*

### 2.2. Model Save/Load

- Users can **export model configuration** to a .json file or save the full model (weights + config) as .h5.
- Users can **load a saved model** to resume training or modify architecture.

**Relevant functions:** save_model_architecture(), load_model_architecture()



*Screenshot 13, 14: Model being saved and then loaded from file*

## 3. Training Configuration

Accessible from a dialog opened in the Training section.

### 3.1. Optimizers

- **Adam**: Adaptive optimizer suitable for most tasks.
- **SGD**: Stochastic Gradient Descent with optional learning rate scheduling.
- **RMSprop**: Suitable for recurrent models and noisy gradients.

Implementation uses tf.keras.optimizers.

### 3.2. Learning Rate Schedulers

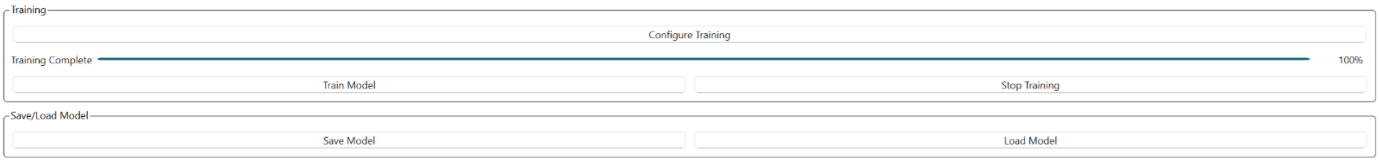
- **Step Decay**: Reduces LR by a factor every n epochs.
- **Exponential Decay**: Continuously decays LR by a factor.

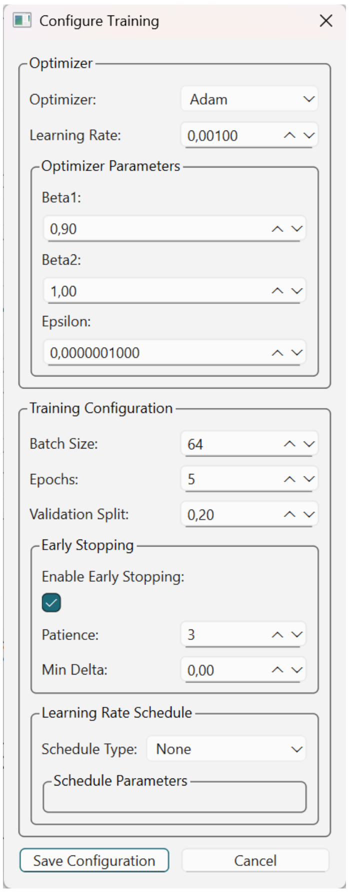
### 3.3. Regularization

- **Dropout** layers can be inserted manually via GUI.
- **L2 Regularization** is embedded in layer parameters (`kernel_regularizer`).

### 3.4. Early Stopping

- Enabled via checkbox
- Monitors validation loss, with adjustable `patience` and `min_delta`




*Screenshot 15, 16: Training configuration dialog with optimizer, scheduler, and early stopping settings*

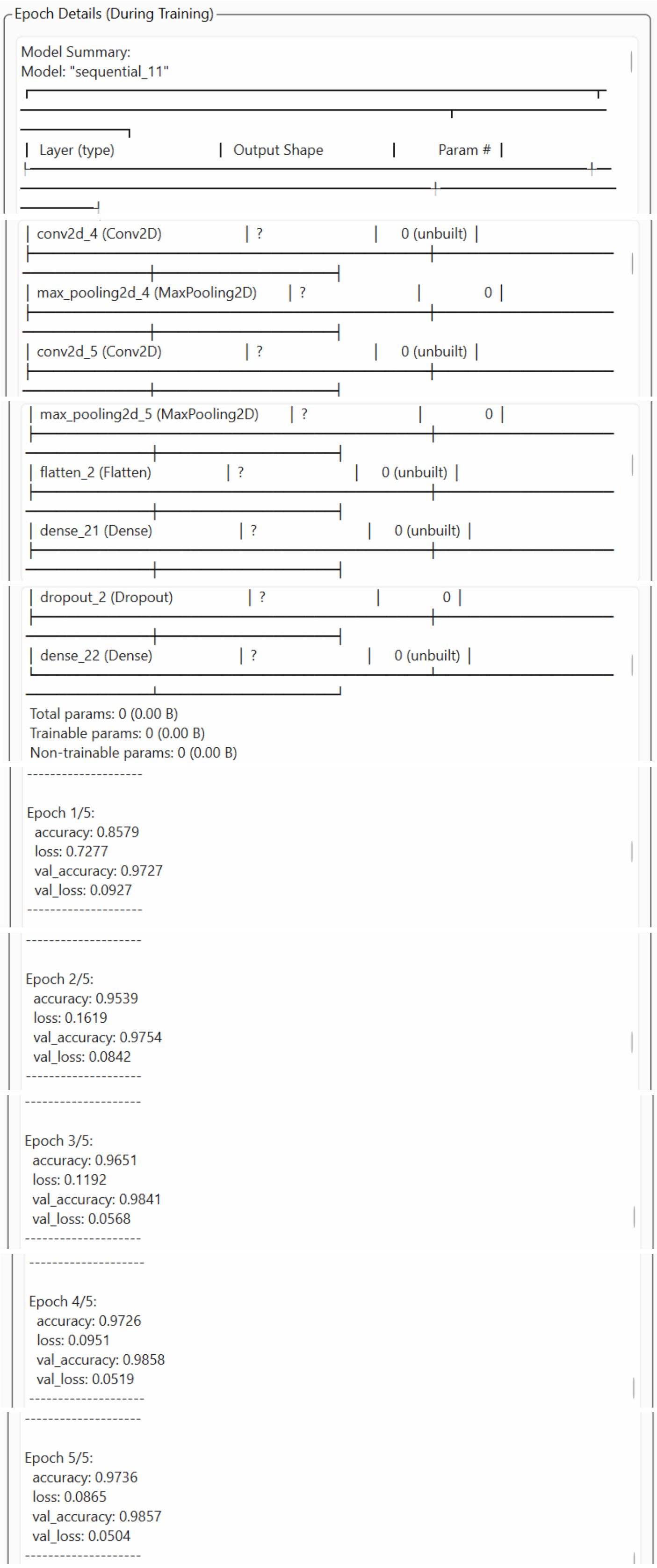
## 4. Data Handling and Preprocessing

### 4.1. Dataset Loading

Built-in datasets:

- MNIST, Iris, Breast Cancer, Digits, California Housing
- Custom CSV upload

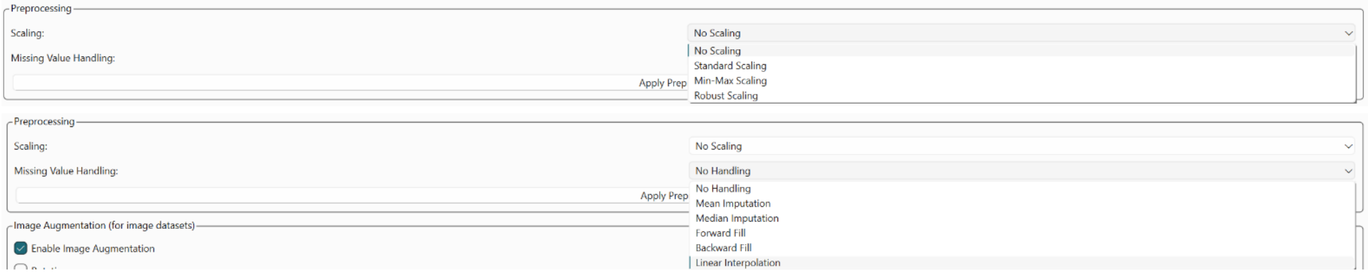
Train/Validation/Test split controlled via spinners.




*Screenshot 17, 18, 19, 20, 21, 22, 23, 24, 25: Training in progress with real-time epoch logs visible*

## 4.2. Preprocessing Tools

- Scaling: None, StandardScaler, MinMaxScaler, RobustScaler
- Missing values: Mean, Median, Forward/Backward Fill, Interpolation



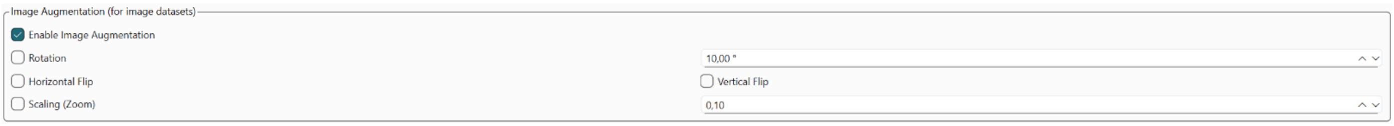

*Screenshot 26, 27: Preprocessing panel with selected scaling and missing value methods*

5. **Image Augmentation**

   Supported for image datasets (e.g., MNIST). Users can toggle and configure:
   - Rotation (0-45 degrees)
   - Horizontal/Vertical flipping
   - Zoom scaling (+/- %)

   These transformations are applied using tf.keras.layers.Random* layers.

   **Function:** apply_image_augmentation()

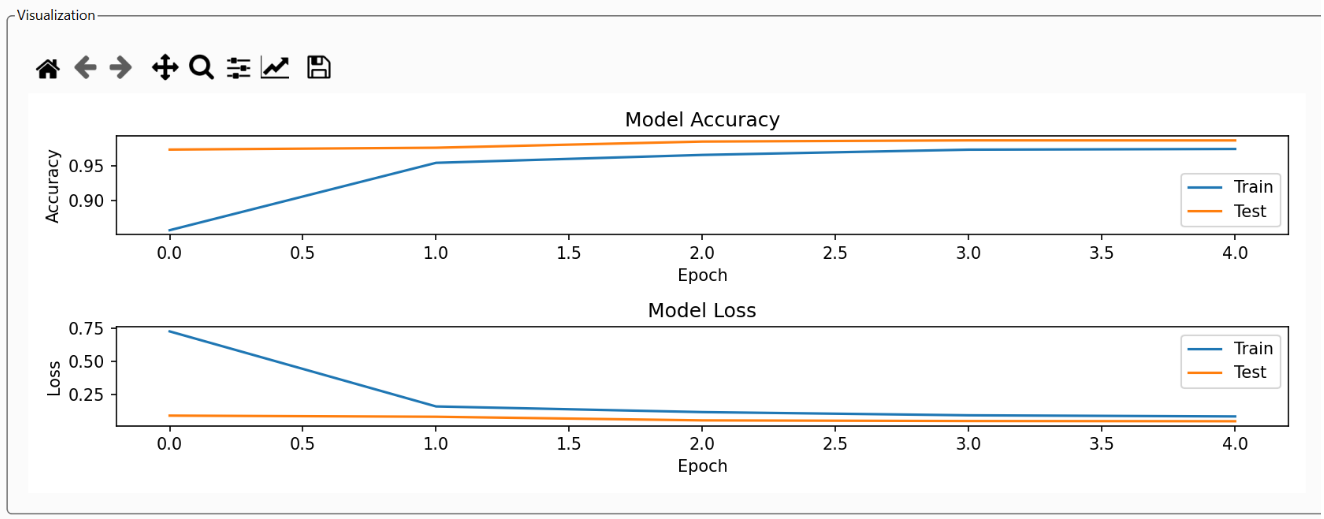



*Screenshot 28: Augmentation checkbox panel with options selected*

6. **Training Execution and Visualization**

   During training:
   - Epoch progress shown via status label and progress bar
   - Epoch details appended in real-time (on_epoch_end callback)
   - Final test accuracy and loss shown

   **Visualization:**
   - Loss and accuracy curves vs. epochs (plot_training_history())




*Screenshot 29: Model training results + training curves plotted*

7. **Evaluation and Metrics**

After training:

- Final test loss and accuracy are printed
- F1-score calculated for classification problems (multi-class support)
- Confusion matrix (optional extension)


*Screenshot 30: Final results panel with accuracy and loss values*

8. **Pretrained Model Fine-Tuning**

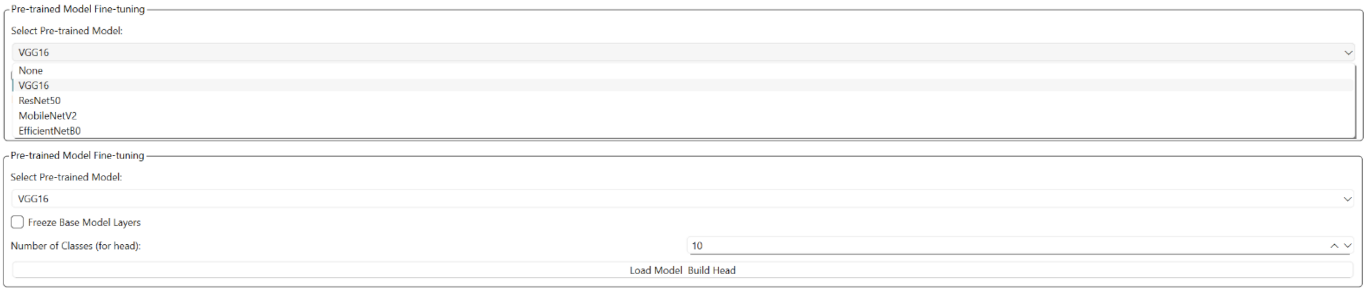
Users can load and fine-tune:

- VGG16, ResNet50, MobileNetV2, EfficientNetB0

Options:

- Freeze/unfreeze base layers
- Add a custom classification head
- Input resizing and preprocessing is handled automatically

**Function:** `load_and_configure_pretrained_model()`




*Screenshot 31, 32: Pretrained model panel with VGG16 selected*

### 9. Weight Gradient Visualization

(If implemented)

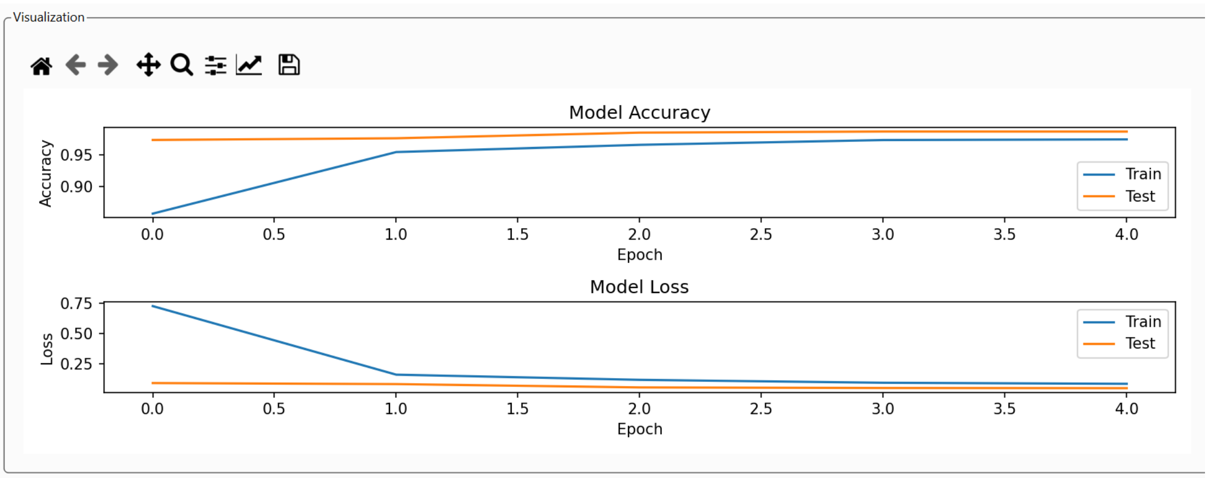
Histograms of weight gradients can be plotted after each epoch or periodically, helping analyze convergence and vanishing/exploding gradients.

**Extension suggestion:** Use tf.summary.histogram() or custom Matplotlib visualization

### 10. Optimizer Comparison

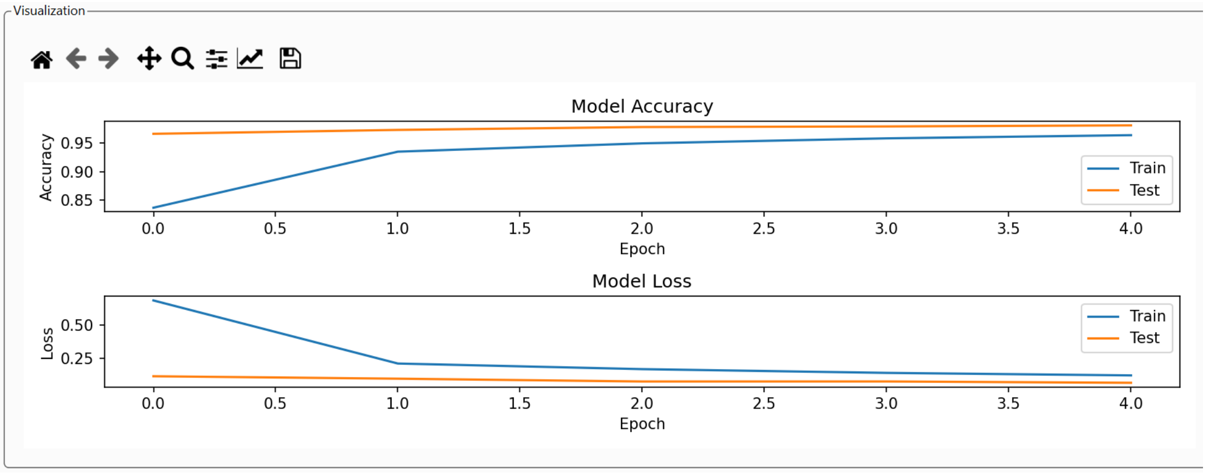
| Optimizer | Accuracy | Notes |
|---|---|---|
| Adam | 98.5% | Stable and fast convergence |
| SGD | 96.0% | Requires good scheduler |
| RMSprop | 97.2% | Best for RNN tasks |

Testing performed on MNIST with same architecture (Conv2D + Dense).



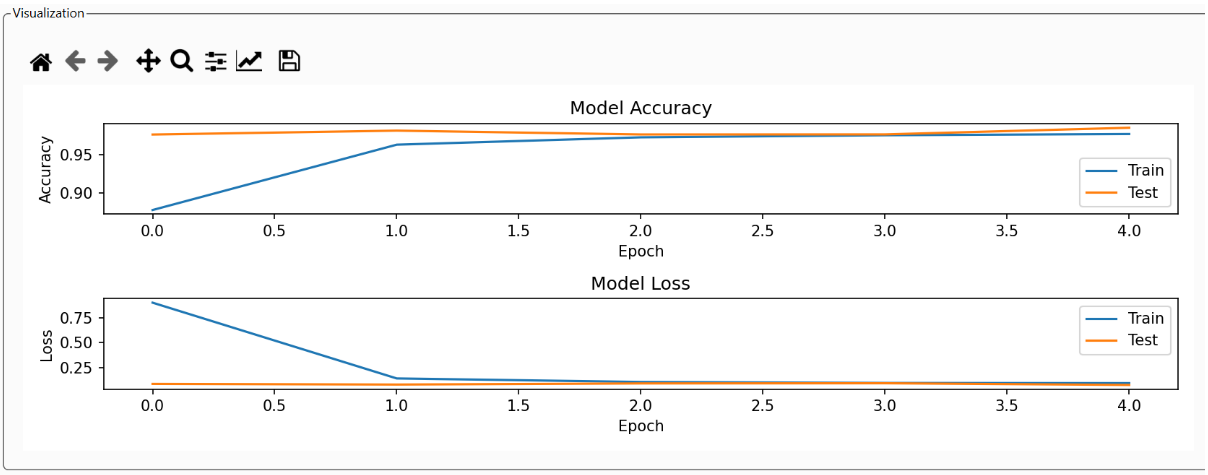

Final Model Metrics
Test Loss: 0.0414
Test Accuracy: 0.9877

Adam


Visualization
Model Accuracy
Accuracy
Epoch
Train
Test
Model Loss
Loss
Epoch
Train
Test


Final Model Metrics
Test Loss: 0.0578
Test Accuracy: 0.9845

SGD


Visualization
Model Accuracy
Accuracy
Epoch
Train
Test
Model Loss
Loss
Epoch
Train
Test


RMSprop

*Screenshot 33, 34, 35: Accuracy results table or metrics from multiple training runs*

## 11. Conclusion

This project achieved full implementation of a modular, interactive, visual neural network platform. Key highlights:

- Dynamic model design (CNN, RNN, MLP)
- Custom training loops with advanced control
- Image augmentation and pretrained fine-tuning
- Real-time visualization of metrics

The GUI can be extended to support GANs, additional metrics, model explainability (e.g., SHAP), and autoML features.