



YTU

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

The screenshot displays the 'Machine Learning Course GUI' window. It features a top navigation bar with tabs: 'Classical ML', 'Deep Learning' (selected), 'Dimensionality Reduction', 'Cross Validation', 'Reinforcement Learning', and 'Training Visualization'. The main interface is divided into several sections:

- Data Management:** Includes a 'Dataset' dropdown, a 'Load Custom Dataset' button, and a 'Load Data' button. Below this is a 'Data Split Configuration' section with input fields for 'Train Split' (0.70), 'Validation Split' (0.15), 'Test Split' (0.15), and 'Random Seed' (42).
- Preprocessing:** Contains 'Scaling' (No Scaling), 'Missing Value Handling' (No Handling), and an 'Apply Preprocessing' button.
- Image Augmentation (for image datasets):** Includes checkboxes for 'Enable Image Augmentation', 'Rotation', 'Horizontal Flip', and 'Scaling (Zoom)'. It also has input fields for '10.00°' and '0.10'.
- Model Architecture:** Features a 'Layer Configuration' section with 'Add Layer', 'Remove Selected Layer', 'Move Up', and 'Move Down' buttons.
- Model Summary:** A section for displaying model details.
- Quick MNIST Configuration:** Includes a 'Load Compute Efficient MNIST Config' button.
- Training:** Contains a 'Configure Training' button, a 'Not trained' status indicator, and 'Train Model' and 'Stop Training' buttons.
- Save/Load Model:** Includes 'Save Model' and 'Load Model' buttons.
- Pre-trained Model Fine-tuning:** Features a 'Select Pre-trained Model' dropdown (EfficientNetB0), a 'Freeze Base Model Layers' checkbox, and a 'Number of Classes (for head)' input field (10). It also has 'Load Model' and 'Build Head' buttons.
- Visualization:** Includes a toolbar with icons for home, back, forward, zoom, pan, and save. It also has sections for 'Epoch Details (During Training)' and 'Final Model 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.

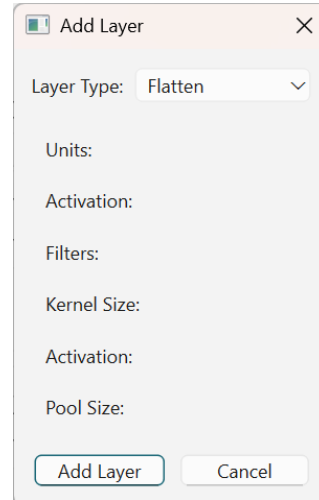
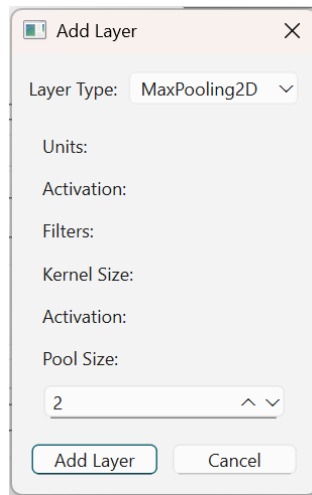
Users can dynamically:

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

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

The image displays four screenshots of the 'Add Layer' dialog box, illustrating the configuration options for different layer types:

- Top Left (Dense):** Layer Type: Dense. Units: 64. Activation: relu. Buttons: Add Layer, Cancel.
- Top Right (Dense):** Layer Type: Dense. Units: 64. Activation: relu. Buttons: Add Layer, Cancel.
- Bottom Left (Dropout):** Layer Type: Dropout. Units: (disabled). Activation: (disabled). Filters: (disabled). Kernel Size: (disabled). Pool Size: (disabled). Dropout Rate: 0,50. Buttons: Add Layer, Cancel.
- Bottom Right (Conv2D):** Layer Type: Conv2D. Units: (disabled). Activation: (disabled). Filters: 32. Kernel Size: 3. Pool Size: (disabled). Dropout Rate: (disabled). Buttons: Add Layer, Cancel.



Model Summary

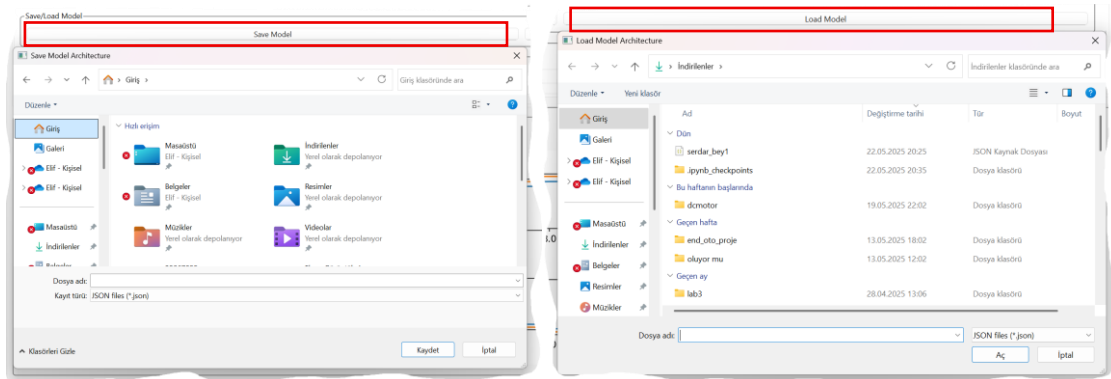
Total params: 3,040 (11.88 KB)
 Trainable params: 3,040 (11.88 KB)
 Non-trainable params: 0 (0.00 B)

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

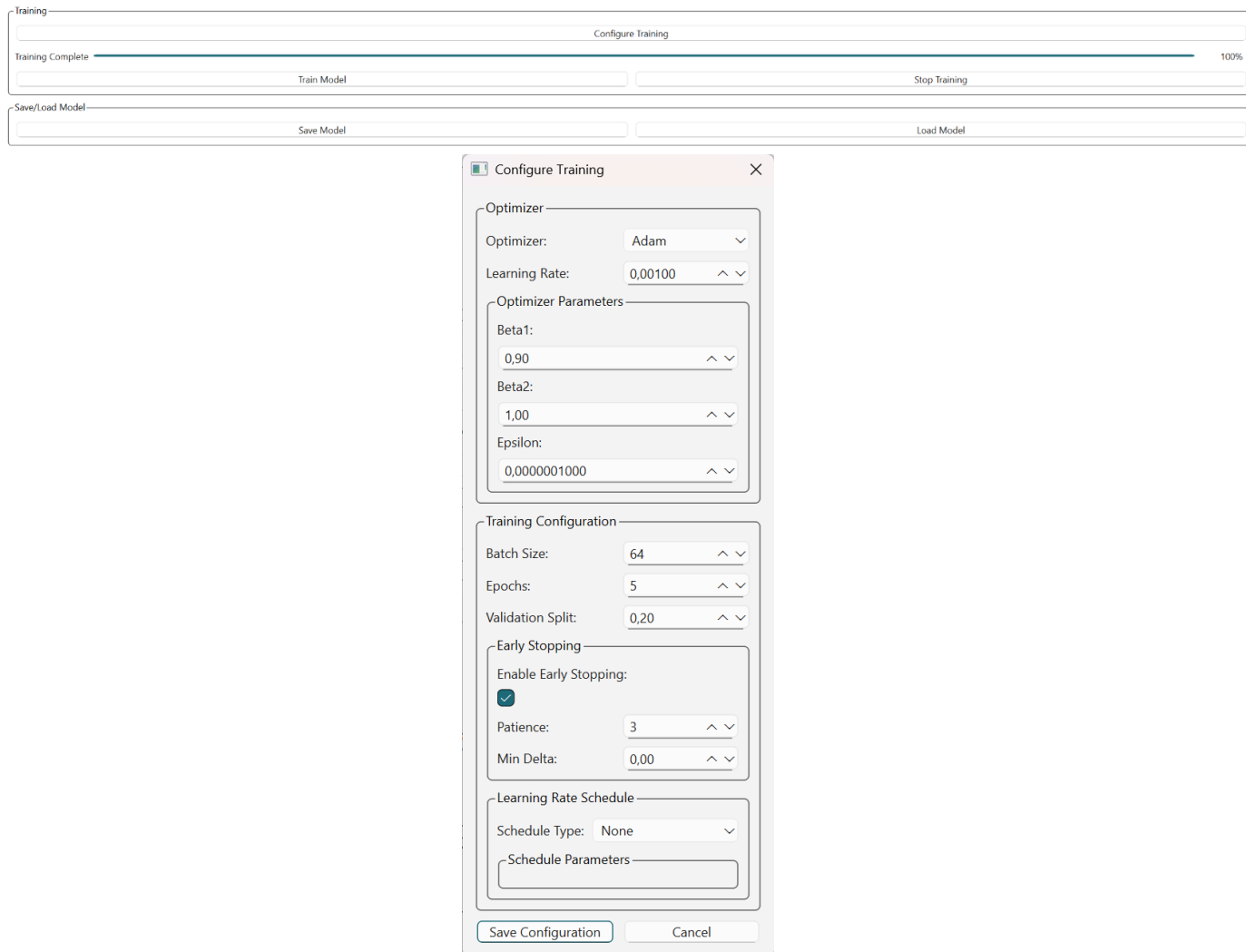
- **Step Decay:** Reduces LR by a factor every n epochs.
- **Exponential Decay:** Continuously decays LR by a factor.

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.

Epoch Details (During Training)		
Model Summary: Model: "sequential_11"		
Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	?	0 (unbuilt)
max_pooling2d_4 (MaxPooling2D)	?	0
conv2d_5 (Conv2D)	?	0 (unbuilt)
max_pooling2d_5 (MaxPooling2D)	?	0
flatten_2 (Flatten)	?	0 (unbuilt)
dense_21 (Dense)	?	0 (unbuilt)
dropout_2 (Dropout)	?	0
dense_22 (Dense)	?	0 (unbuilt)
Total params: 0 (0.00 B) Trainable params: 0 (0.00 B) Non-trainable params: 0 (0.00 B)		
----- Epoch 1/5: accuracy: 0.8579 loss: 0.7277 val_accuracy: 0.9727 val_loss: 0.0927 -----		
----- Epoch 2/5: accuracy: 0.9539 loss: 0.1619 val_accuracy: 0.9754 val_loss: 0.0842 -----		
----- Epoch 3/5: accuracy: 0.9651 loss: 0.1192 val_accuracy: 0.9841 val_loss: 0.0568 -----		
----- Epoch 4/5: accuracy: 0.9726 loss: 0.0951 val_accuracy: 0.9858 val_loss: 0.0519 -----		
----- Epoch 5/5: accuracy: 0.9736 loss: 0.0865 val_accuracy: 0.9857 val_loss: 0.0504 -----		

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

Preprocessing

Scaling: No Scaling

Missing Value Handling: No Handling

Apply Prep

Preprocessing

Scaling: Standard Scaling

Missing Value Handling: Mean Imputation

Apply Prep

Image Augmentation (for image datasets)

☒ Enable Image Augmentation

Rotation: 10.00 °

Horizontal Flip: ☐

Vertical Flip: ☐

Scaling (Zoom): 0.10

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()`

Image Augmentation (for image datasets)

☒ Enable Image Augmentation

☐ Rotation

☐ Horizontal Flip

☐ Scaling (Zoom)

10.00 °

☐ Vertical Flip

0.10

Screenshot 28: Augmentation checkbox panel with options selected

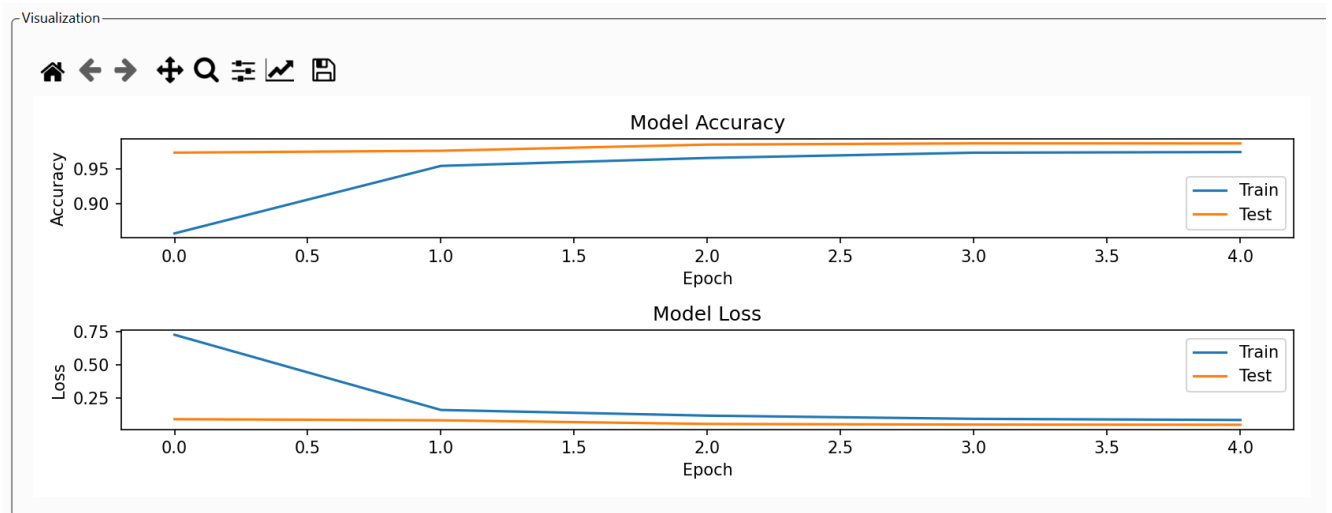
6. Training Execution and Visualization

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()`

Pre-trained Model Fine-tuning

Select Pre-trained Model:

VGG16

None

VGG16

ResNet50

MobileNetV2

EfficientNetB0

Pre-trained Model Fine-tuning

Select Pre-trained Model:

VGG16

☐ Freeze Base Model Layers

Number of Classes (for head): 10

Load Model Build Head

Screenshot 31, 32: Pretrained model panel with VGG16 selected

9. Weight Gradient Visualization

(If implemented)

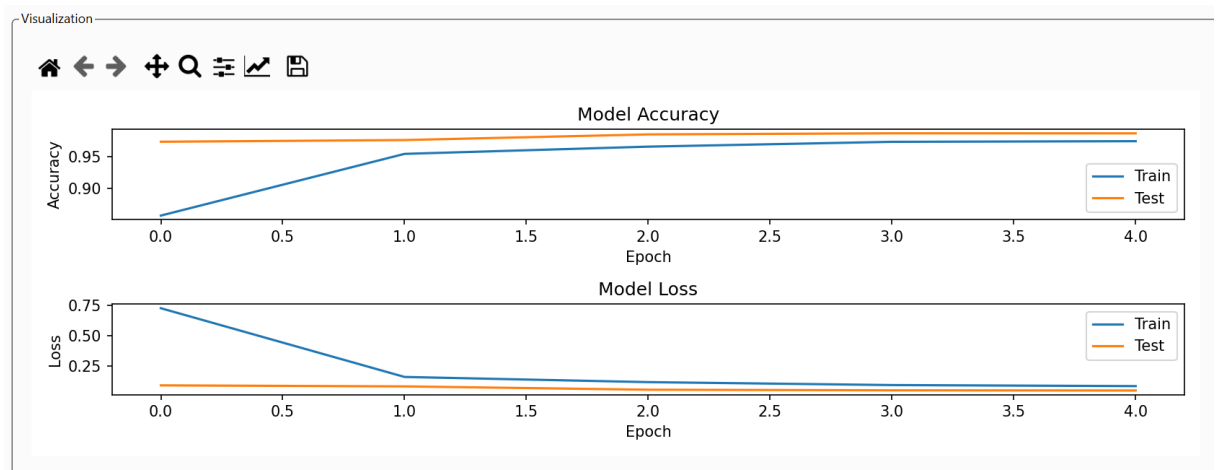
Histograms of weight gradients can be plotted after each epoch or periodically, helping analyze convergence and vanishing/exploding gradients.

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).

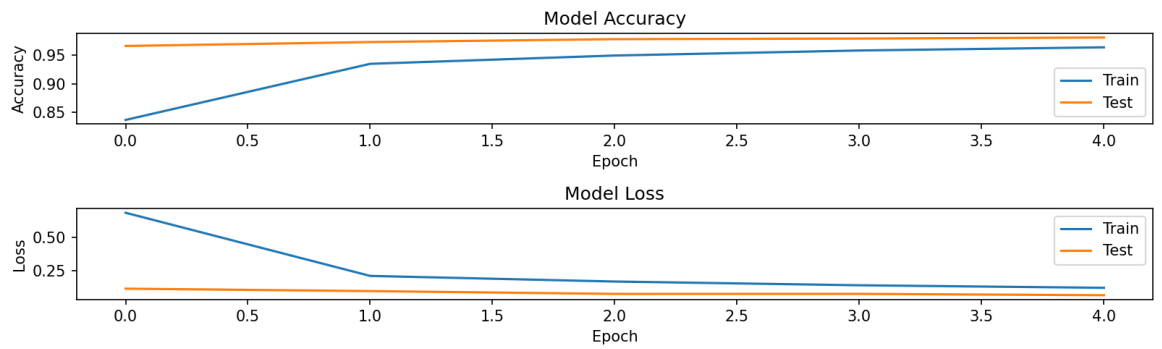
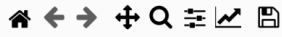


Final Model Metrics

Test Loss: 0.0414
Test Accuracy: 0.9877

Adam

Visualization

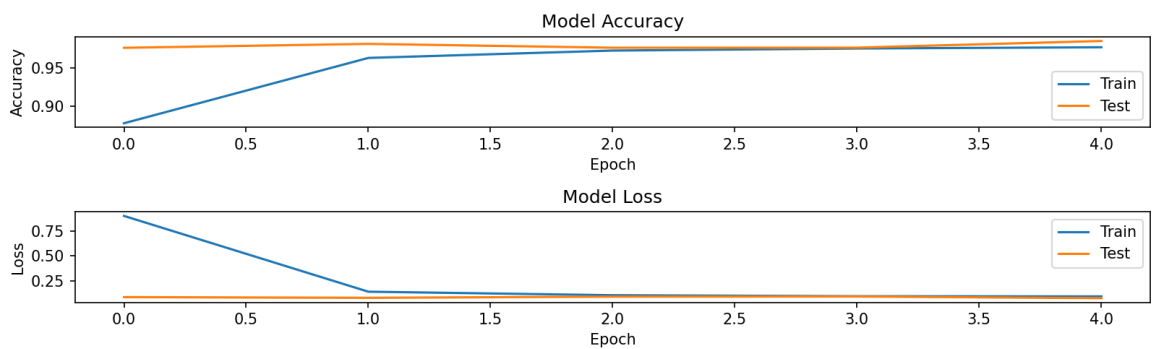
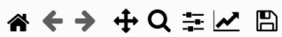


Final Model Metrics

Test Loss: 0.0578
Test Accuracy: 0.9845

SGD

Visualization





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.