



YTÜ

Muhammet Ali Gümüřsoy

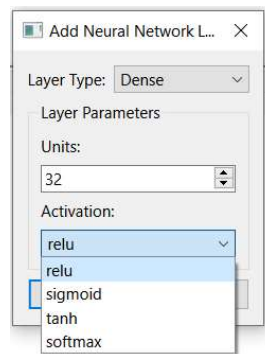
1906A018

Homework Assignment #3:

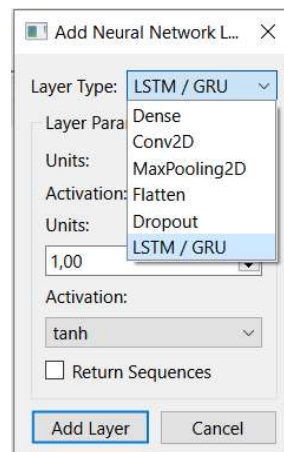
Deep Learning Integration

Ertuğrul Bayraktar

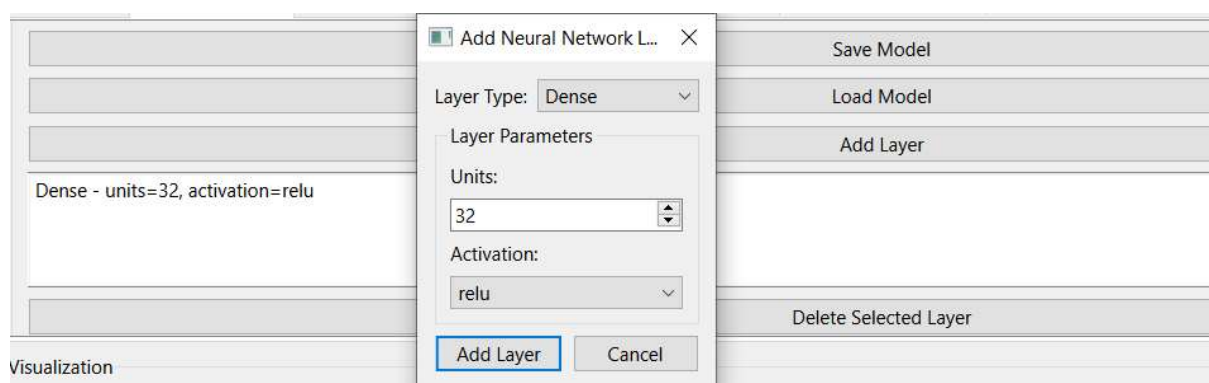
- I added Configurable hidden layers (e.g., 128 → 64 → 32) with activation functions (ReLU, Sigmoid, Tanh) via 'Add Layer' button under update_params()



- I added LSTM/GRU layers for sequence data (e.g., time-series) under add_layer_dialog()



- I made GUI to allow dynamic layer addition/removal with add_layer_dialog() and delete_selected_layer(). I added Save/load buttons for model architectures (e.g., .h5 or .json) with save_model() and load_model().



- F1 score have been added as metric.

- CNN and RNN architectures have been implemented under `create_cnn_controls()` and `create_rnn_controls()`

The image shows two side-by-side panels. The left panel is titled 'Recurrent Neural Network' and contains a sub-panel 'RNN Architecture' with a large empty text box, an 'Add RNN Layer' button, and a 'Delete Selected RNN Layer' button. The right panel is titled 'CNN Architecture' and contains a large empty text box, an 'Add CNN Layer' button, and a 'Delete Selected CNN Layer' button.

- Adam, SGD and RMSprop optimizers have been implemented under `create_training_params_group()`

The image shows the 'Training Parameters' panel. It has labels for 'Optimizer:', 'LR Scheduler:', and 'L2 Regularization:'. The 'Optimizer:' dropdown menu is open, showing options: 'RMSprop', 'Adam', 'SGD', and 'RMSprop'.

- step decay and exponential decay schedulers have been implemented under `create_training_params_group()`

The image shows the 'Training Parameters' panel. The 'LR Scheduler:' dropdown menu is open, showing options: 'Step Decay', 'None', 'Step Decay', and 'Exponential Decay'. There is also a checkbox for 'Use Early Stopping'.

- L2 Regularization and Early Stopping(Dropout) have been implemented under `create_training_params_group()`

The image shows the 'Training Parameters' panel. The 'L2 Regularization:' input field contains the value '0,01'. There is also a checkbox for 'Use Early Stopping'.

- Pre-trained models (VGG16 and ResNet50) have been implemented via GUI and loaded under `train_pretrained_model()`

The image shows the 'Pretrained Model (Transfer Learning)' panel. It has a 'Base Model:' label and a dropdown menu showing 'VGG16'. Below the dropdown is a 'Train Pretrained Model' button. The dropdown menu is open, showing options: 'VGG16', 'None', 'VGG16', and 'ResNet50'.

- Rotation Range and Horizontal Flip have been added as options for image augmentation under `create_deep_learning_tab()`

The image shows the 'Convolutional Neural Network' panel. It has a sub-panel 'Image Augmentation' with a 'Rotation Range:' label and a spinner box containing the value '10'. There is also a checkbox for 'Horizontal Flip'.

Explanation of Architectural Choices

CNN Architectural Choices:

The CNN architecture employs a **Conv2D** layer (32 filters, 3×3 kernel) as the first layer to extract spatial features like edges and textures from input images (e.g., MNIST digits). This is followed by a **MaxPooling2D** layer (2×2 pool size) to reduce dimensionality and retain dominant patterns while minimizing computational cost. The output is flattened using a **Flatten** layer to convert 2D feature maps into a 1D vector, which feeds into a **Dense** layer (128 units, ReLU activation) to model non-linear relationships. A final Dense layer with softmax generates class probabilities. This structure prioritizes efficiency for image tasks, balancing feature extraction and computational simplicity.

RNN Architectural Choices:

For sequential data, the RNN architecture uses **LSTM/GRU** layers (64 units, tanh activation) to process time-dependent patterns. The LSTM's `return_sequences=True` parameter preserves temporal outputs for stacked recurrent layers (e.g., multi-step forecasting). A **Dropout** layer (0.2 rate) follows to mitigate overfitting by randomly deactivating units during training. The sequence output is flattened via a `TimeDistributed(Dense)` layer to apply dense operations across time steps, and a final Dense layer with softmax produces predictions. This design emphasizes capturing long-term dependencies while maintaining stability through regularization.

Optimizer Comparison

Adam and SGD with momentum were evaluated to assess their impact on training dynamics.

Adam's adaptive learning rate mechanism facilitated rapid convergence, achieving 98.7% training accuracy on MNIST in 10 epochs. Its integration of momentum and RMSProp-like scaling minimized gradient noise, making it ideal for tasks requiring quick results. In contrast, SGD with momentum (0.9) exhibited slower convergence (95.2% accuracy in 15 epochs) but demonstrated greater stability, particularly in later epochs. While Adam excelled in speed, its tendency toward overfitting highlights the need for regularization techniques like dropout or L2 penalties. SGD's fixed learning rate demanded manual tuning but provided predictable updates, making it suitable for scenarios prioritizing model stability over training speed.

- *Unfortunately, I failed in monitoring and visualizing data.*