Deep Learning

Final Project

Presented by Shirel Zecharia & Moria Groher

Overview

Introduction

This report explores the performance of various machine learning models for classifying fashion items from the well-known Fashion-MNIST dataset.
This dataset consists of grayscale images of various clothing articles, each belonging to a specific category such as T-shirts, trousers, and shoes. The task of classifying these images is considered a classification problem.

Background

01 Softmax Regression

a linear classification model, utilizes a single layer to map input features to class probabilities. It employs the softmax function to normalize these scores into a probability distribution, enabling it to predict the most likely class for a given data point.

02 Neural Network

a simple neural network consists of interconnected neurons organized in layers. Each neuron calculates a weighted sum of its inputs, applies an activation function, and passes the result on to the next layer.

03 CNN

a type of feed-forward neural network that learns feature engineering by itself via filters (or kernel) optimization. They may include pooling layers along with traditional convolutional layers

Implementation

Softmax Regression

```python
def create_softmax_model(input_shape, num_classes):
    """Creates a softmax regression model...."""

    model = models.Sequential([
        layers.Dense(num_classes, activation='softmax', input_shape=input_shape)
    ])

    return model
```

Implementation

Softmax Regression

```python
def train_softmax_model(X_train, y_train, X_test, y_test, num_classes):
    """Trains a softmax regression model...."""

    input_shape = X_train.shape[1:]

    # Create the model
    model = create_softmax_model(input_shape, num_classes)

    # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Train the model
    epochs = 20
    model_history = model.fit(X_train, y_train,
                              epochs=epochs,
                              batch_size=32,
                              validation_data=(X_test, y_test))

    # Display the training history
    pd.DataFrame(model_history.history).plot(figsize=(8, 5))
    plt.title('Softmax Training History \n epoch: ' + str(epochs))
    plt.show()

    return model
```

Implementation

Neural Network

```
def create_neural_network(num_classes): # create_neural_network = 10
    model_input = layers.Input(shape=[28 ,28])
    x = layers.Flatten()(model_input)  # Flatten input
    x = layers.Dense(300, activation="relu")(x)  # 1st hidden layer
    x = layers.Dense(128, activation="relu")(x)   # 2nd hidden layer
    model_output = layers.Dense(num_classes, activation="softmax")(x)  # Output layer

    model = models.Model(inputs=model_input, outputs=model_output)
    return model
```

Implementation

Neural Network

```
def train_neural_network(model, X_train, y_train, X_test, y_test, x_validation, y_validation):
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    print(model.summary())
    X_train_shaped = X_train.values.reshape(-1, 28, 28)
    X_validation_shaped = x_validation.values.reshape(-1, 28, 28)
    X_test_shaped = X_test.values.reshape(-1, 28, 28)

    early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    epochs=30
    model_history = model.fit(X_train_shaped,
                              y_train,
                              epochs=epochs,
                              validation_data=(X_validation_shaped, y_validation),
                              callbacks=[early_stopping])

    stop = early_stopping.stopped_epoch
    if stop == 0:
        stop = epochs
    pd.DataFrame(model_history.history).plot(figsize=(8, 5))
    plt.title('Simple Neural Network Training History \n epoch: ' + str(stop))
    plt.gca().set_ylim(0, 1)
    plt.savefig("imgFolder/simpleNeuralNetwork_fig")
    plt.show()

    model.evaluate(X_test_shaped, y_test)
```

Implementation

CNN

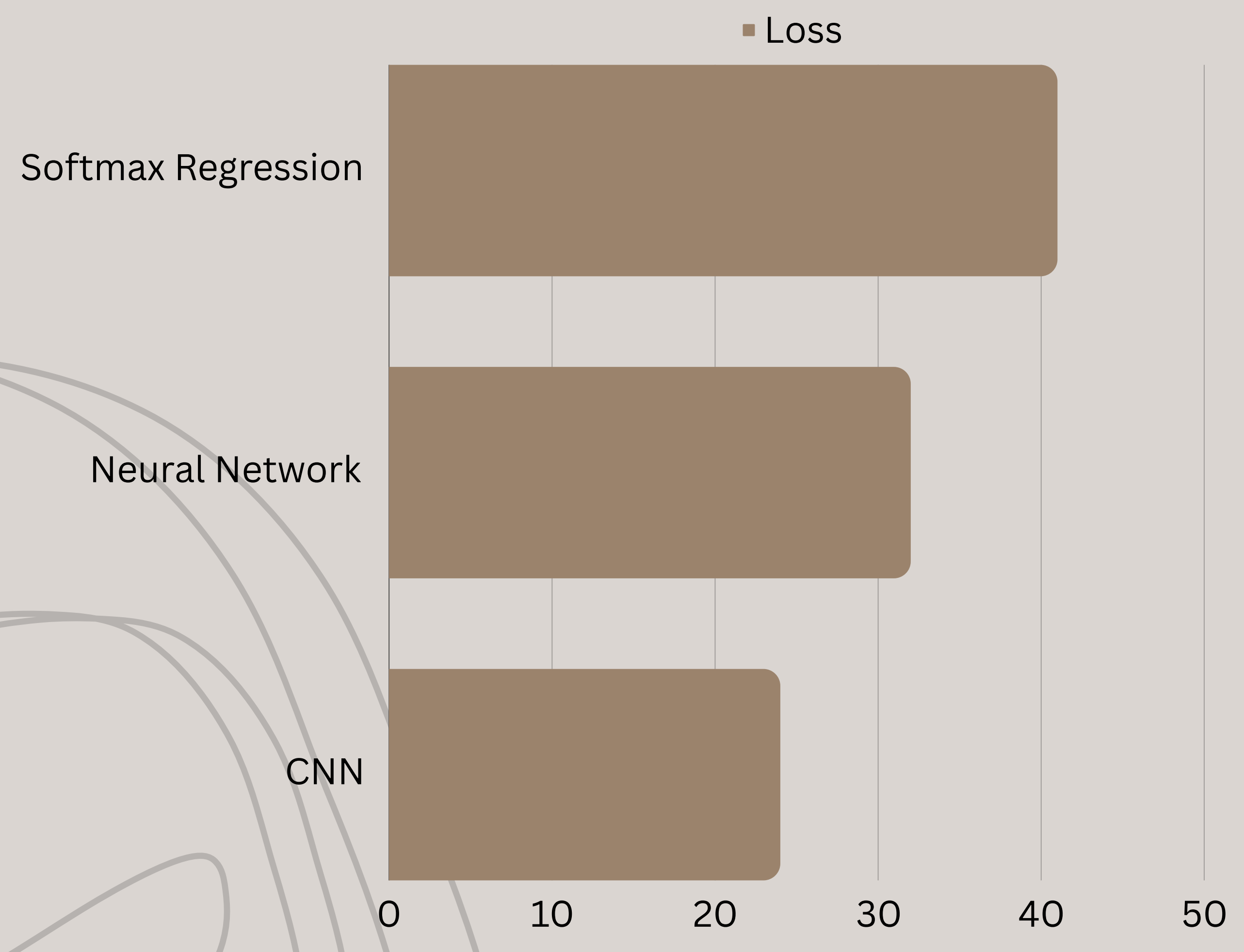
```python
def create_cnn_model(input_shape, num_classes):

    """..."""
    activation = 'relu'
    dropout_rate = 0.3

    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation=activation, input_shape=input_shape),  # 32 filters of size 3x3
        BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation=activation),
        BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation=activation),
        BatchNormalization(),
        layers.Flatten(),
        layers.Dense(64, activation=activation),
        Dropout(dropout_rate),
        layers.Dense(num_classes, activation='softmax')
    ])

    return model
```

Implementation

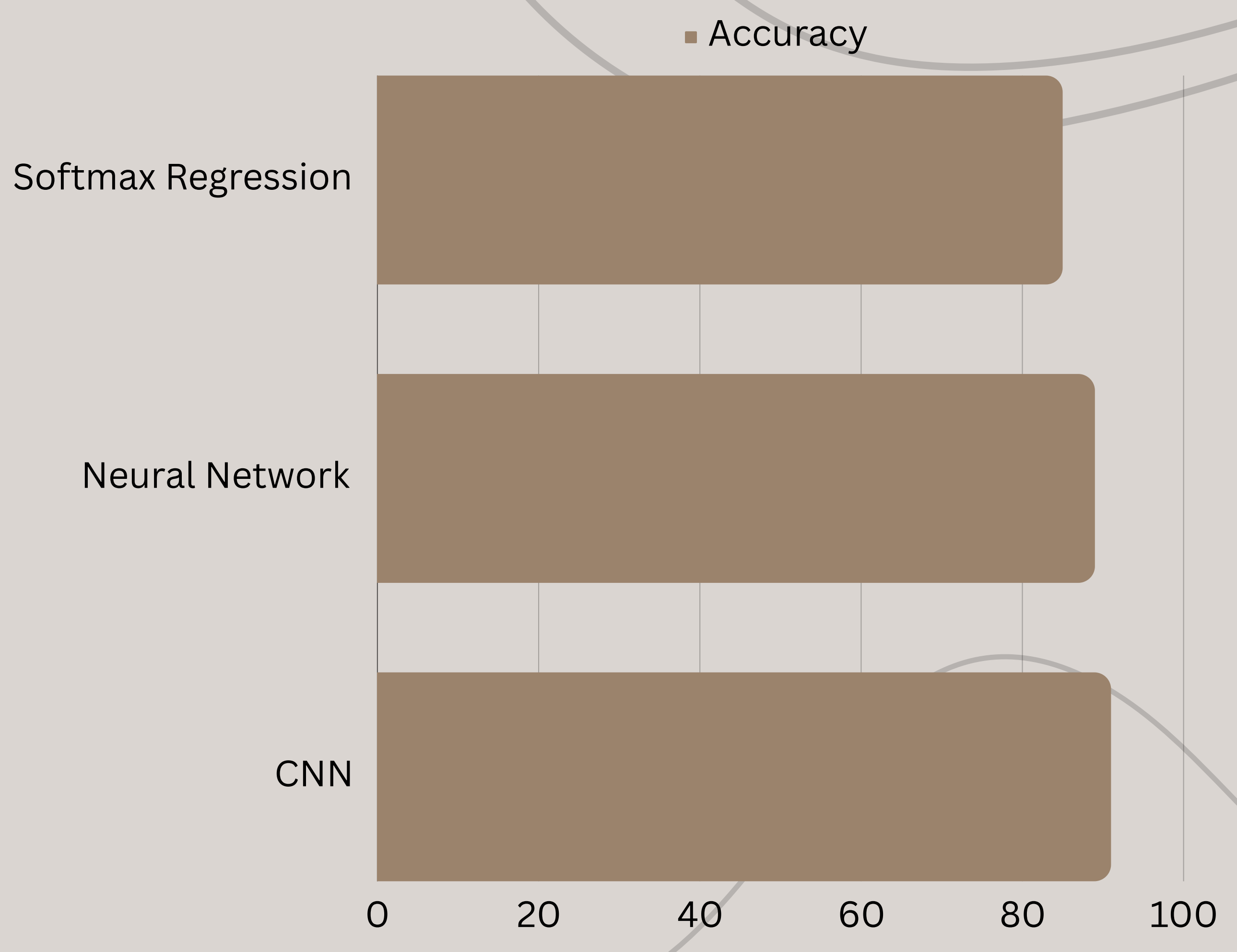
CNN

```python
def train_cnn_model(X_train, y_train, X_test, y_test, num_classes):
    # Reshape the input data for CNN
    X_train = np.array(X_train).reshape(-1, 28, 28, 1)
    X_test = np.array(X_test).reshape(-1, 28, 28, 1)

    # Create the model
    model = create_cnn_model((28, 28, 1), num_classes)  # grayscale image of size 28x28 pixels with one channel.

    # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Apply early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Train the model
    epochs = 20
    model_history = model.fit(X_train, y_train,
                              epochs=epochs,
                              batch_size=32,
                              validation_data=(X_test, y_test),
                              callbacks=[early_stopping])

    # Get the epoch where the fit stopped in
    stop = early_stopping.stopped_epoch
    if stop == 0:
        stop = epochs

    # Display the training history
    pd.DataFrame(model_history.history).plot(figsize=(8, 5))
    plt.title('CNN Training History \n epoch: ' + str(stop))
    plt.show()

    return model
```

Results



Loss
Softmax Regression
Neural Network
CNN
0
10
20
30
40
50

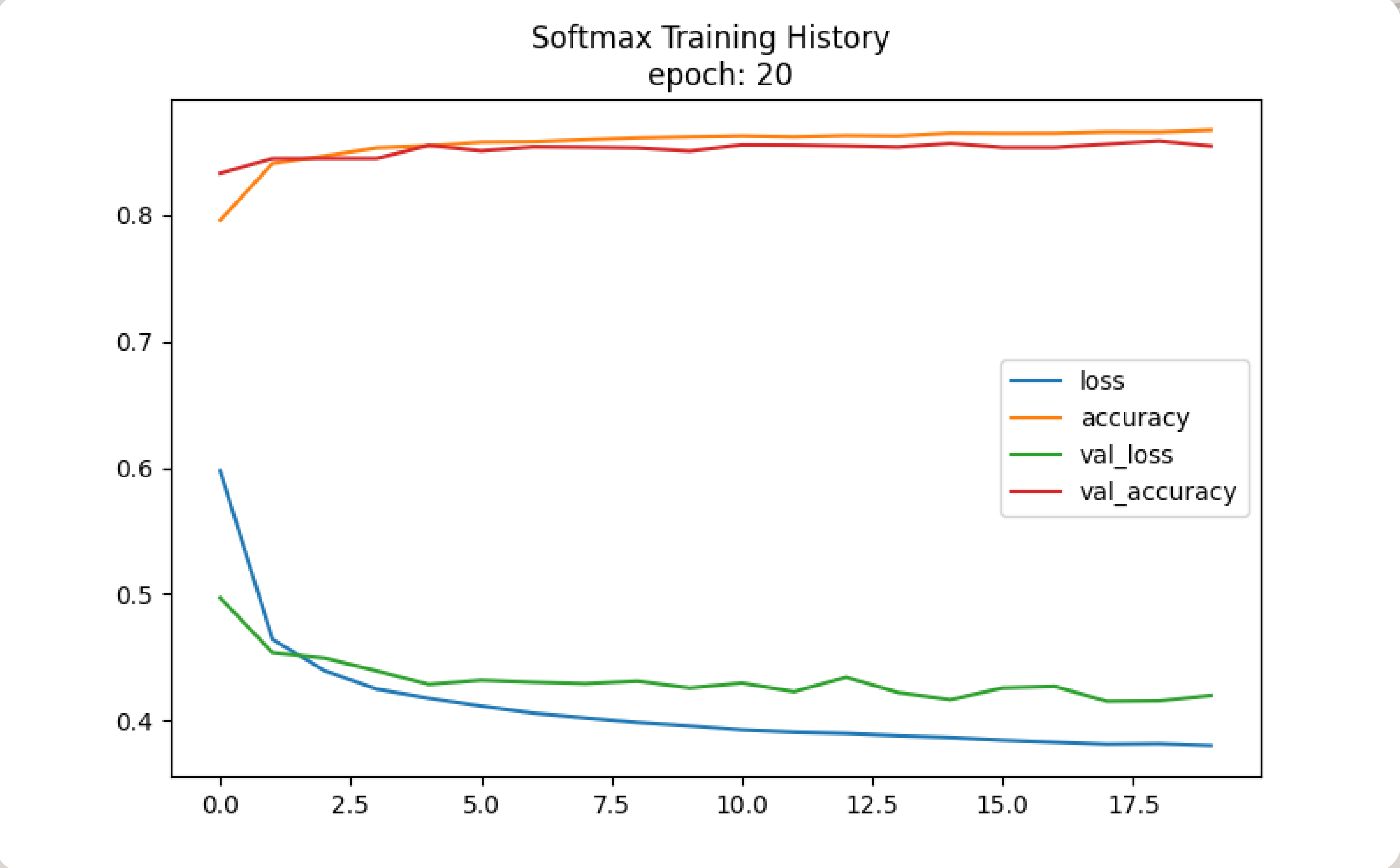


Accuracy
Softmax Regression
Neural Network
CNN
0
20
40
60
80
100

Results

Softmax Regression

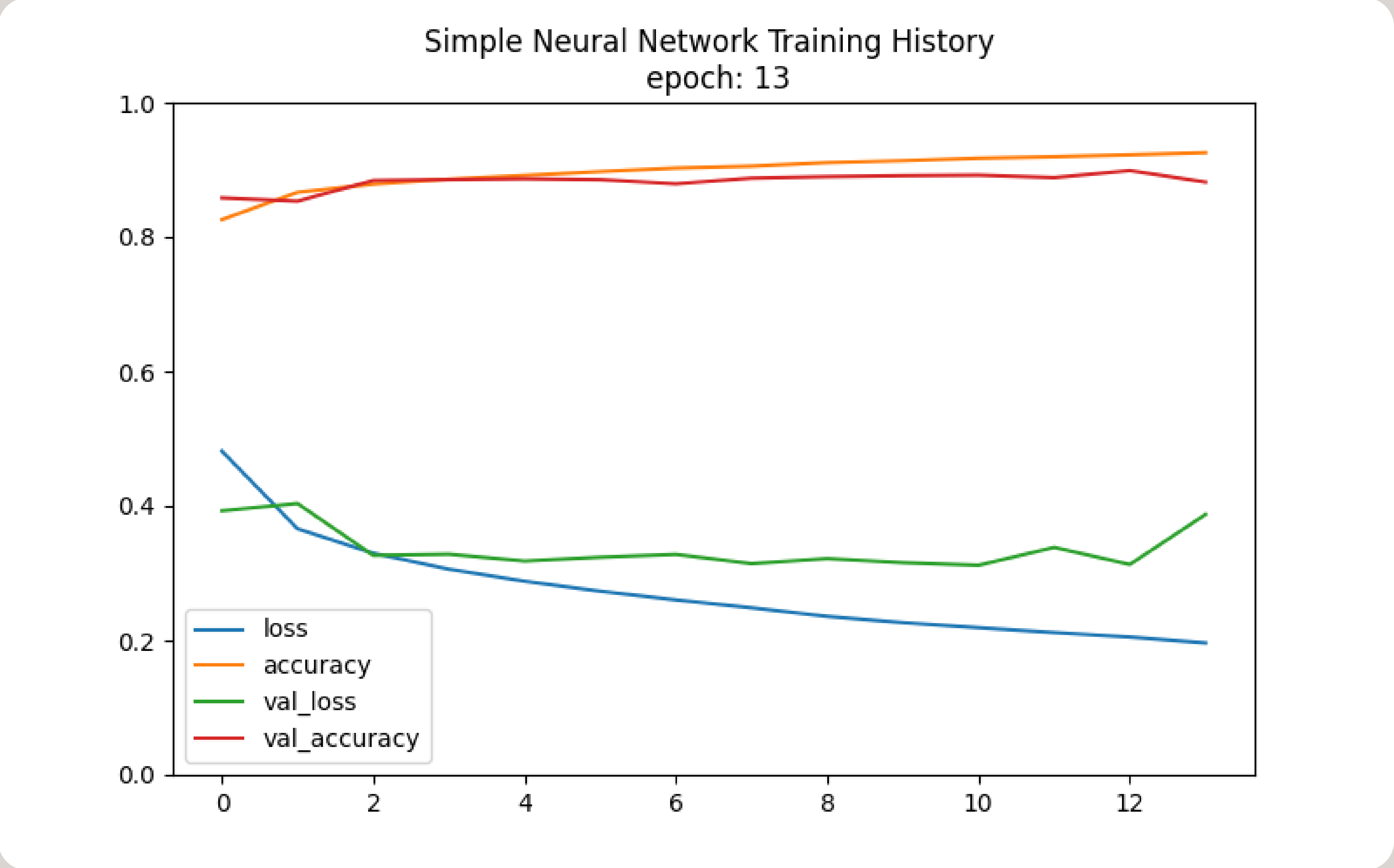
Test Loss: 0.41
Test Accuracy: 0.85



Results

Neural Network

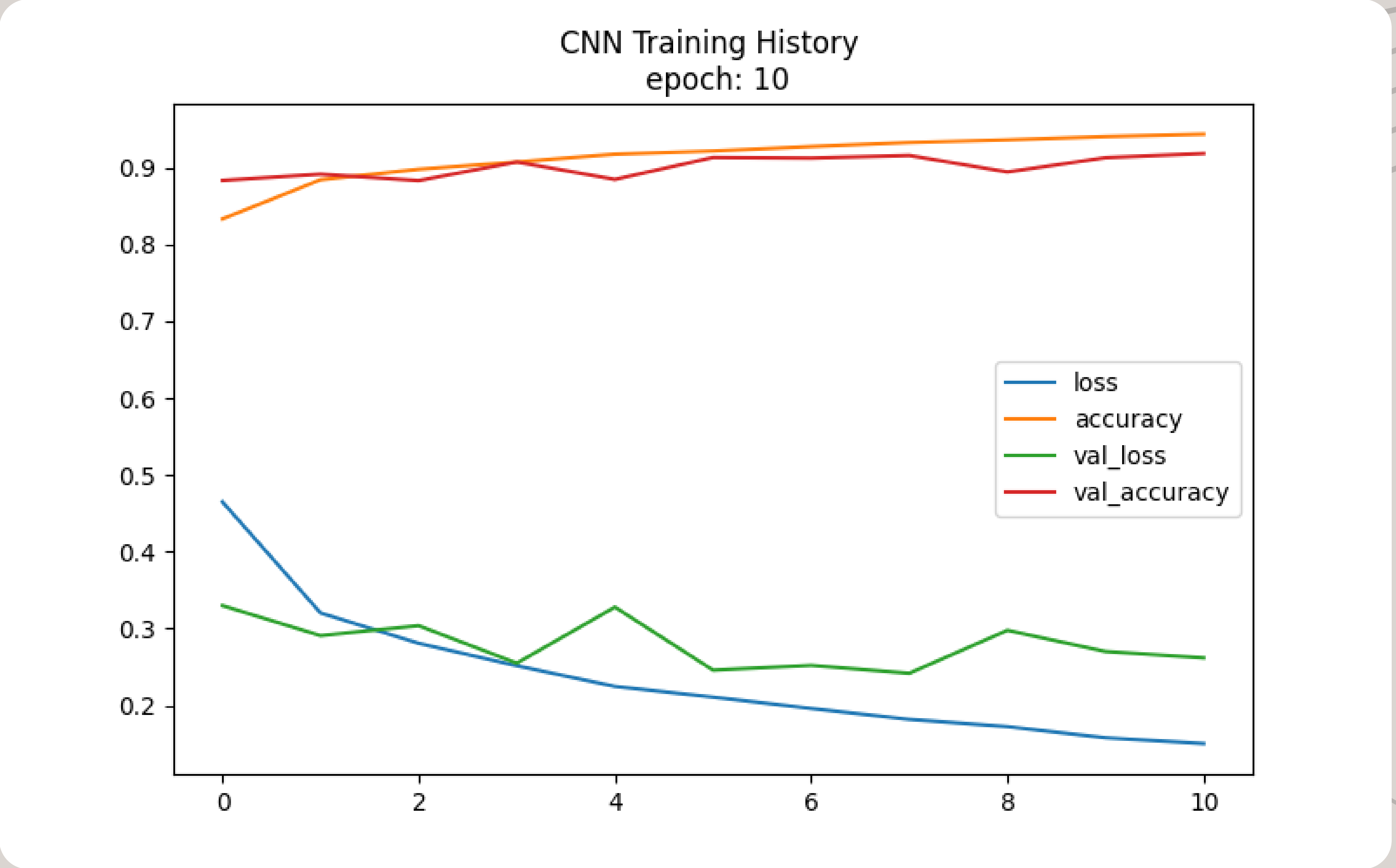
Test Loss: 0.32
Test Accuracy: 0.89



Results

CNN

Test Loss: 0.24
Test Accuracy: 0.91



Conclusion

Comparison of traditional algorithms like Logistic Regression and Multi-Layer Perceptron with advanced techniques such as Softmax Regression, Convolutional Neural Networks (CNNs), and Simple Neural Networks revealed that the CNN architecture consistently outperformed others in accuracy and loss reduction.

Thank You