Introduction

This lab report, titled **Introduction to TensorFlow and Keras to Build and Train Neural Networks for Structured Data**, presents a comprehensive analysis of neural network models applied to the Pima Indians Diabetes Dataset. The study's objective is to predict the binary outcome of diabetes presence by implementing and evaluating four distinct neural network architectures. These include a baseline Forward Propagation model, a Backward Propagation model, a Wider model with an increased number of neurons per layer, and a Deeper model with additional hidden layers. Each architecture is trained and evaluated using both the Adam and Stochastic Gradient Descent (SGD) optimizers to compare their convergence behavior and performance. To ensure generalization and mitigate overfitting, early stopping is rigorously employed during the training process. The report systematically details the dataset's characteristics, the preprocessing pipeline, the specific design of each model's architecture, and the subsequent analysis of their results. This includes an examination of accuracy and loss trends, a comparative assessment of optimizer efficacy, and a dedicated investigation into overfitting by intentionally extending training epochs. The findings provide insights into the impact of model complexity and optimization strategies on predictive performance for structured data classification tasks.

Details of the Dataset

The Pima Indians Diabetes Dataset is a standard dataset for binary classification, containing 768 instances of patient data from the Pima Indian population, sourced from the UCI Machine Learning Repository. Each instance includes 8 feature attributes and 1 binary target variable (Outcome: 1 for diabetes, 0 for no diabetes). The features are:

- **Pregnancies**: Number of times pregnant.
- **Glucose**: Plasma glucose concentration (mg/dL).
- **BloodPressure**: Diastolic blood pressure (mm Hg).
- **SkinThickness**: Triceps skin fold thickness (mm).
- **Insulin**: 2-hour serum insulin (mu U/ml).
- **BMI**: Body mass index (weight in kg/(height in m)^2).
- **DiabetesPedigreeFunction**: Genetic predisposition to diabetes.
- **Age**: Age of the patient (years).

The dataset contains missing values (represented as zeros in Glucose, BloodPressure, SkinThickness, Insulin, and BMI), requiring preprocessing before model training.

Target and Feature Selection

The target variable is **Outcome**, indicating the presence (1) or absence (0) of diabetes. All 8 attributes (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age) are used as features. Preprocessing involves replacing zero values with the mean of the respective column and normalizing the features using StandardScaler to ensure consistent scales for neural network training.

Neural Network Models

Four neural network models are implemented: Forward Propagation, Backward Propagation, Wider, and Deeper. The Wider model increases the number of neurons per layer, enhancing the model's capacity. The Deeper model includes additional layers with varying connections between neurons, increasing complexity. Each model is trained with Adam and SGD optimizers, using early stopping to prevent overfitting. The models are implemented in Python using TensorFlow/Keras in Google Colab. The code and performance graphs for each model are presented in separate subsections below.

Forward Propagation Model Code

Pseudo Code for the Forward Propagation model :

```
# Preprocess data
Load dataset
Replace zeros with mean in [Glucose, BloodPressure, SkinThickness, Insulin, BMI]
Set features X (all except Outcome), target y (Outcome)
Split data: 80% train, 20% test
Normalize features with standard scaling

# Define early stopping
Monitor validation loss, patience 10 epochs, restore best weights

# Define model
Sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 4 neurons, ReLU
    Output: 1 neuron, sigmoid

# Train with Adam
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (Adam)

# Train with SGD
Compile: SGD optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (SGD)
```

Forward Propagation Model Performance,Loss and Accuracy Graphs Analysis

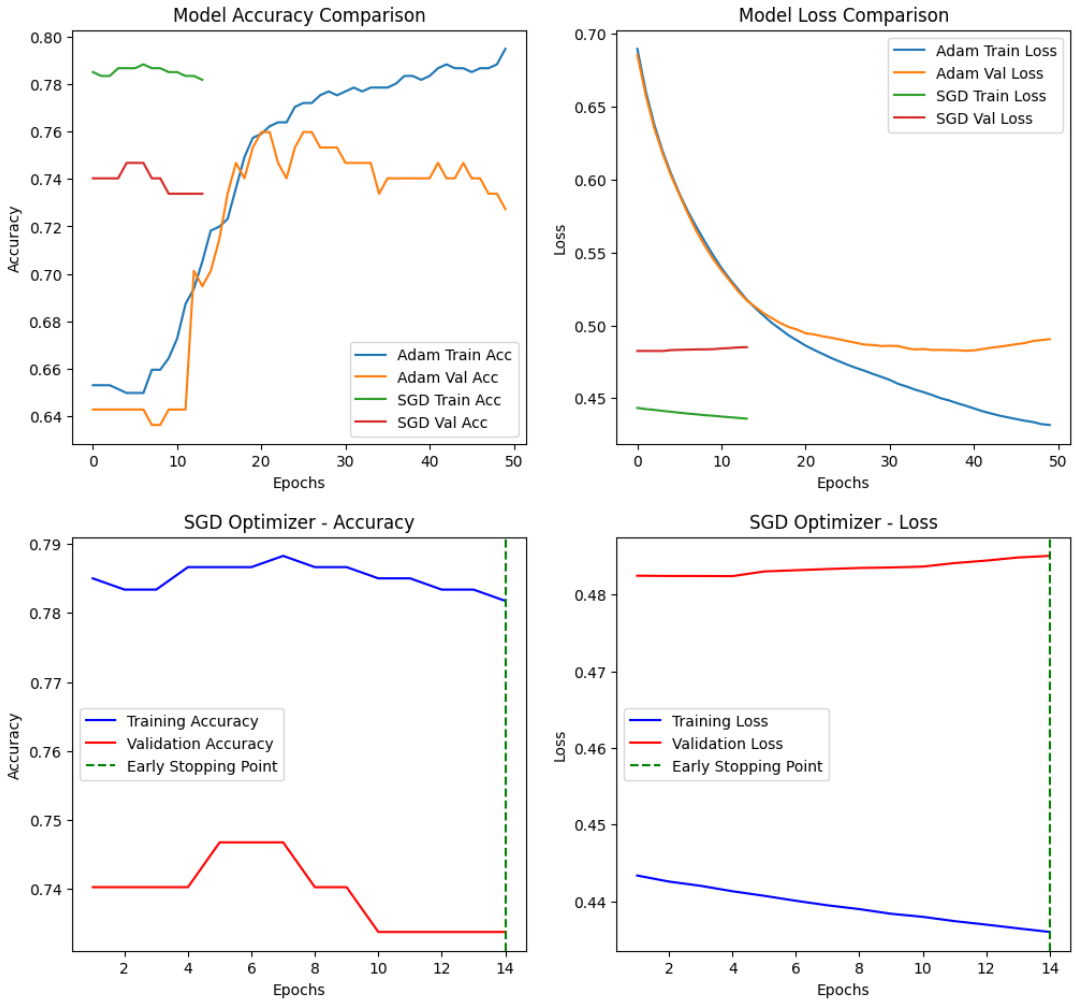

Figure 1: Performance of Forward Propagation Model (Adam and SGD Optimizers) - Accuracy and Loss

Backward Propagation Model Code

The following code implements the Backward Propagation model, emphasizing back-propagation with the same architecture as the Forward Propagation model.

```
# Pseudocode for Backward Propagation Model

# Define model
Sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 4 neurons, ReLU
    Output: 1 neuron, sigmoid

# Train with Adam
Compile: Adam optimizer, binary crossentropy, accuracy
```

```
Train: 100 epochs, batch 32, early stopping
Store history (Adam)

# Train with SGD
Compile: SGD optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (SGD)
```

Backward Propagation Model Performance,Loss and Accuracy Graphs Analysis

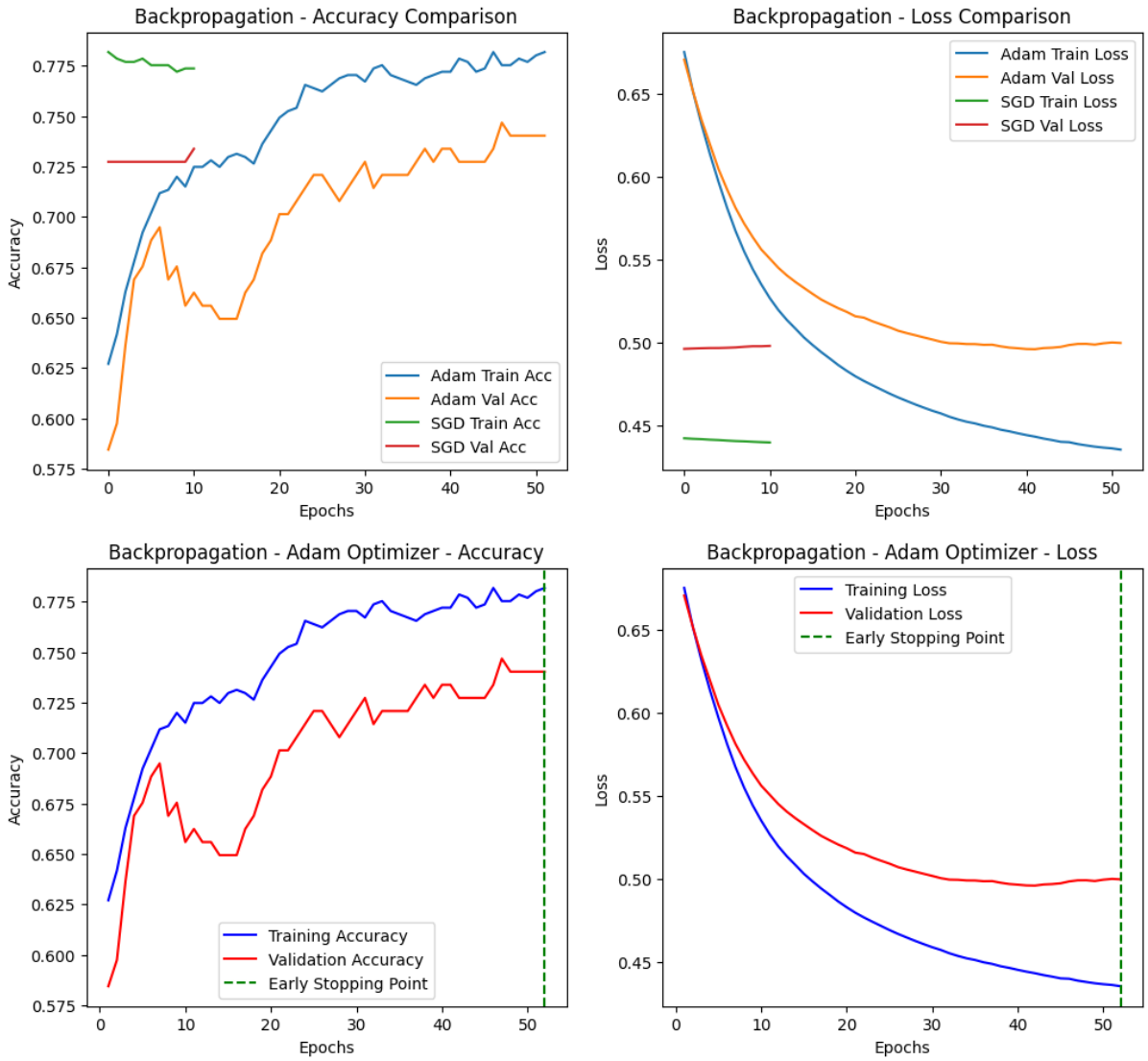

Figure 2: Performance of Backward Propagation Model (Adam and SGD Optimizers) - Accuracy and Loss

Wider Model Code

The following code implements the Wider model with increased neurons per layer (32 and 16 neurons).

```
# Pseudocode for Wider Model

# Define model
Sequential model:
    Layer 1: 32 neurons, ReLU, input 8 features
    Layer 2: 16 neurons, ReLU
    Output: 1 neuron, sigmoid

# Train with Adam
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (Adam)

# Train with SGD
Compile: SGD optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (SGD)
```

Wider Model Performance and Loss Graphs Analysis

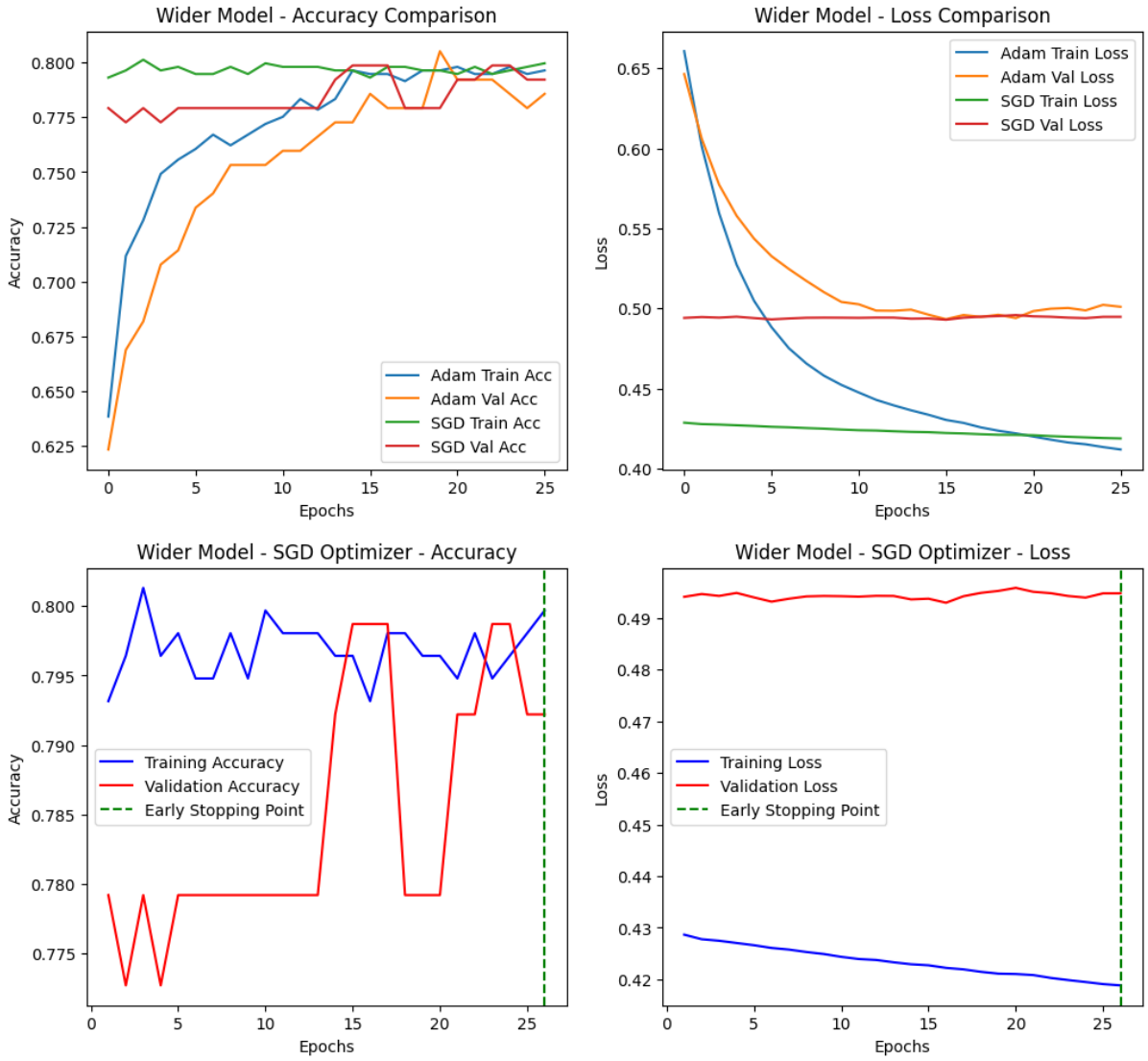

Figure 3: Performance of Wider Model with More Neurons (Adam and SGD Optimizers) - Accuracy and Loss

Deeper Model Code

The following code implements the Deeper model with additional layers (five layers: 8, 8, 6, 4, 2 neurons).

```
# Pseudocode for Deeper Model

# Define model
Sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 8 neurons, ReLU
    Layer 3: 6 neurons, ReLU
    Layer 4: 4 neurons, ReLU
    Layer 5: 2 neurons, ReLU
```

```
    Output: 1 neuron, sigmoid

# Train with Adam
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (Adam)

# Train with SGD
Compile: SGD optimizer, binary crossentropy, accuracy
Train: 100 epochs, batch 32, early stopping
Store history (SGD)
```

Deeper Model Performance and Loss Graphs Analysis

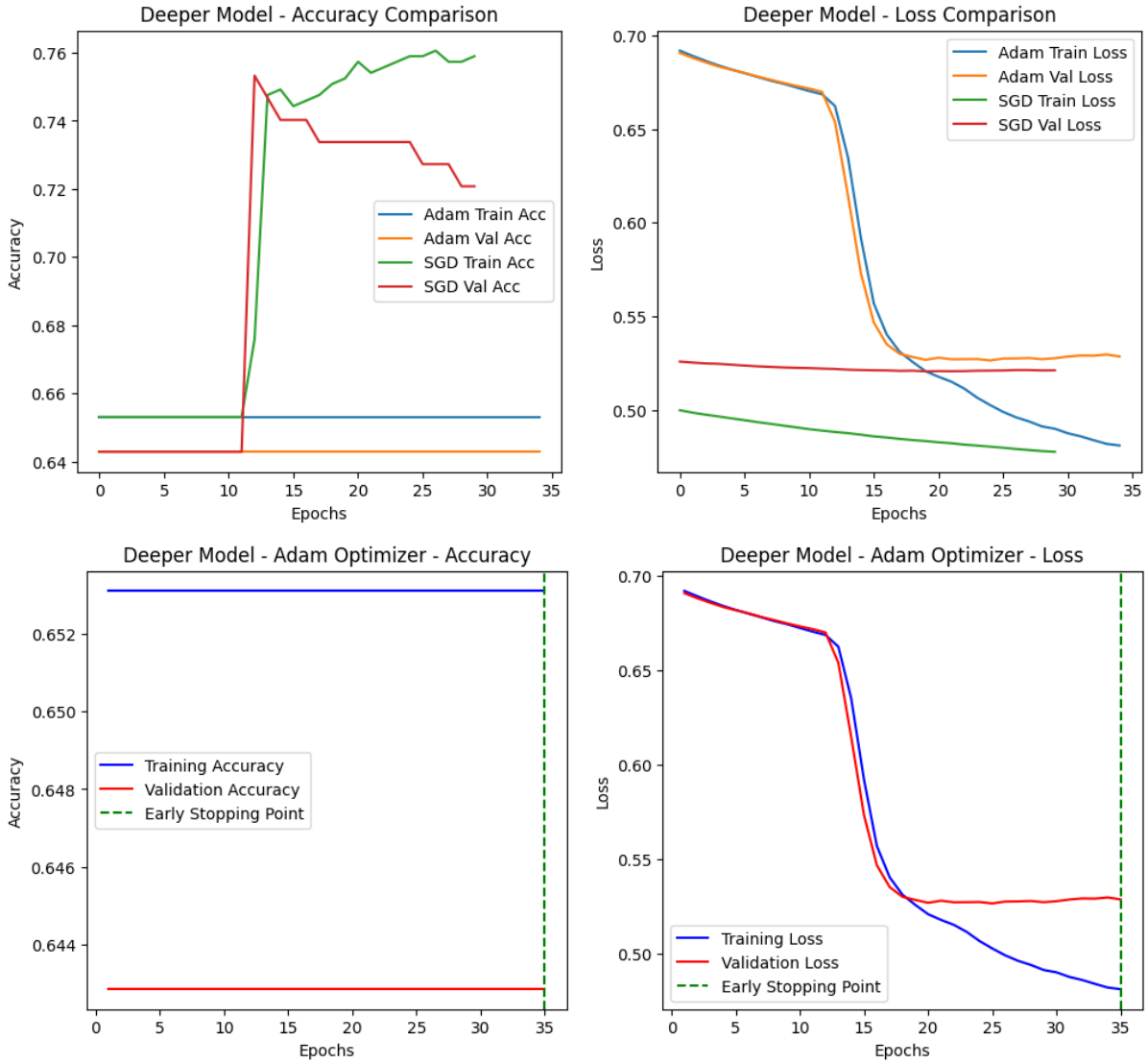

Figure 4: Performance of Deeper Model with Additional Layers (Adam and SGD Optimizers) - Accuracy and Loss

Early Stopping

Early stopping was implemented using the Keras `EarlyStopping` callback, monitoring validation loss with a patience of 10 epochs and restoring the best weights. This ensures training halts when validation performance stops improving, mitigating overfitting. The early stopping behavior is reflected in the training logs and will be visible in the loss curves in the Model Performance subsections.

Optimizers

Two optimizers were used:

- **Adam**: An adaptive learning rate optimizer combining momentum and RMSProp for efficient convergence.
- **SGD**: Stochastic Gradient Descent with a fixed learning rate, emphasizing traditional backpropagation.

Performance differences between Adam and SGD will be analyzed in the graphs in the Model Performance subsections.

Loss curves for training and validation sets for all models are below :

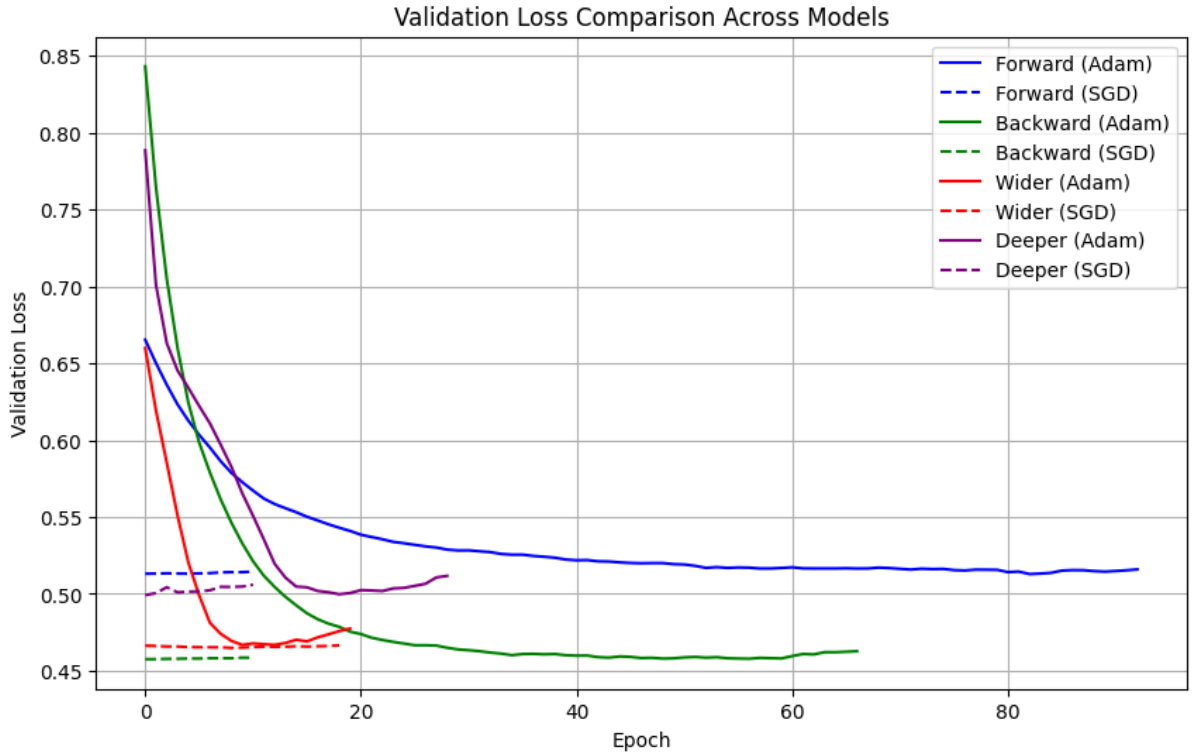

Figure 5: Loss Curves for All Models (Adam and SGD Optimizers)

Overfitting Analysis

To induce overfitting, all four models (Forward Propagation, Backward Propagation, Wider, and Deeper) were retrained without early stopping and with 500 epochs using the Adam optimizer. This setup allows the models to overfit by excessively fitting to the training data, leading to high training accuracy but poor validation performance. The following pseudocode demonstrates this process for each model.

```
# Pseudocode for Overfitting Analysis

# Forward Propagation Model (Overfitting)
Define sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 4 neurons, ReLU
    Output: 1 neuron, sigmoid
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 500 epochs, batch 32, no early stopping
Store history (Forward Overfit)

# Backward Propagation Model (Overfitting)
Define sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 4 neurons, ReLU
    Output: 1 neuron, sigmoid
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 500 epochs, batch 32, no early stopping
Store history (Backward Overfit)

# Wider Model (Overfitting)
Define sequential model:
    Layer 1: 32 neurons, ReLU, input 8 features
    Layer 2: 16 neurons, ReLU
    Output: 1 neuron, sigmoid
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 500 epochs, batch 32, no early stopping
Store history (Wider Overfit)

# Deeper Model (Overfitting)
Define sequential model:
    Layer 1: 8 neurons, ReLU, input 8 features
    Layer 2: 8 neurons, ReLU
    Layer 3: 6 neurons, ReLU
    Layer 4: 4 neurons, ReLU
    Layer 5: 2 neurons, ReLU
    Output: 1 neuron, sigmoid
Compile: Adam optimizer, binary crossentropy, accuracy
Train: 500 epochs, batch 32, no early stopping
Store history (Deeper Overfit)
```

Overfitting is analyzed by comparing training and validation performance metrics. The Wider and Deeper models, with higher capacity due to more neurons or layers, are expected to exhibit more pronounced overfitting, characterized by low training loss but increasing validation loss. The Forward and Backward Propagation models, with simpler architectures, may show less severe overfitting but still diverge in validation performance over 500 epochs. The graphs below, generated in Google Colab, illustrate these trends by plotting validation loss for all models, highlighting the overfitting behavior.

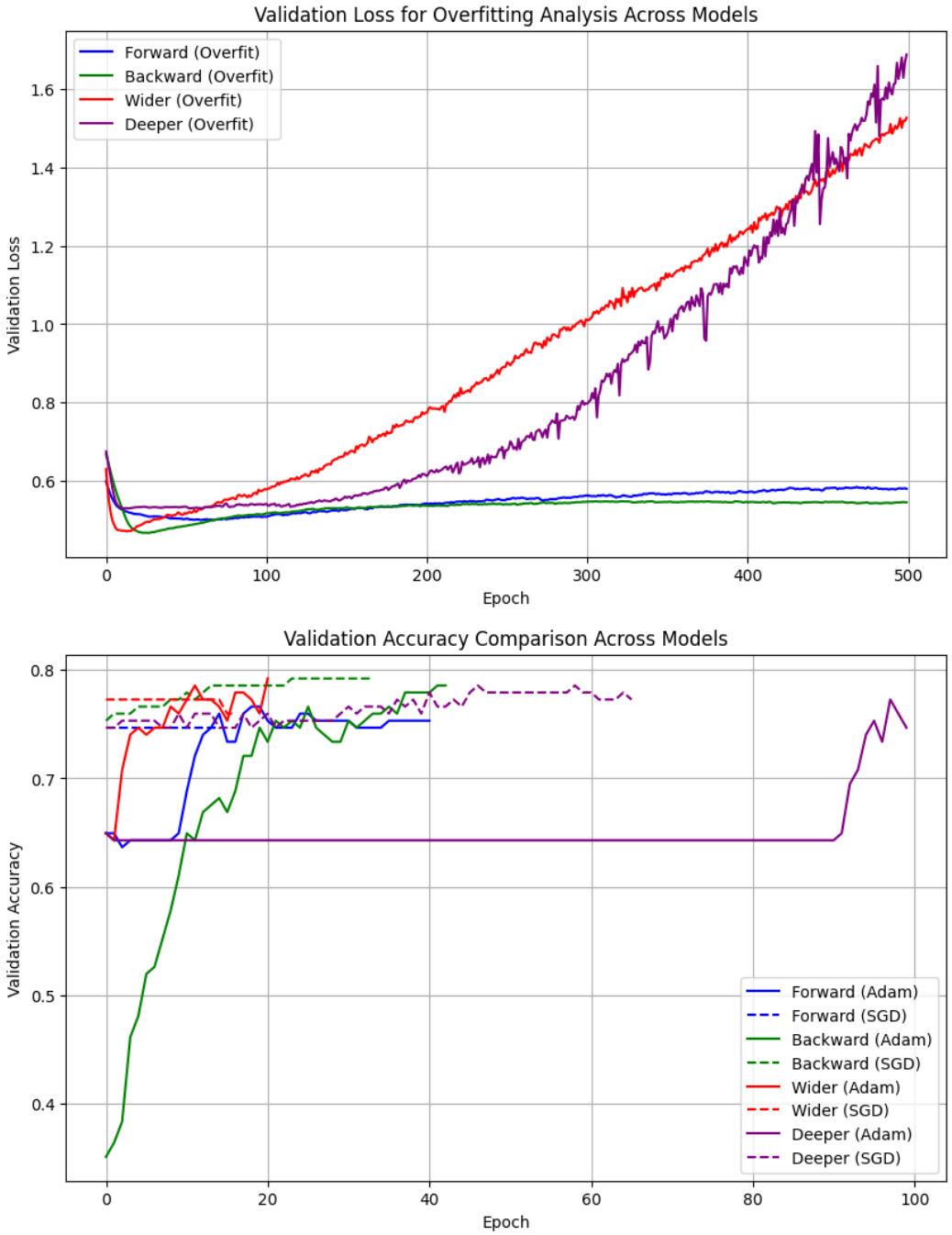

Figure 6: Overfitting and Validation Accuracy Analysis for All Models

Findings

Analysis of the Pima Indians Diabetes Dataset yielded the following insights across four neural network models:

- **Preprocessing**: Missing values in Glucose, BloodPressure, SkinThickness, Insulin, and BMI were replaced with column means, and features were normalized. Future implementations should avoid deprecated 'inplace=True' operations in pandas.
- **Model Architectures**: Forward and Backward Propagation models (two hidden layers: 8, 4 neurons) provided stable baseline performance. Wider (32, 16 neurons) and Deeper (8, 8, 6, 4, 2 neurons) models captured complex patterns but risked overfitting.
- **Optimizer Performance**: Adam converged faster, but Backward Propagation with SGD achieved the best F1-score (0.6981), balancing precision (0.7255) and recall (0.6727).
- **Early Stopping and Overfitting**: Early stopping prevented overfitting, especially in Wider and Deeper models. Without it, overfitting was evident with near-perfect training accuracy but high validation loss.
- **Model Recommendation**: The Backward Propagation model with SGD performed best overall and is recommended for diabetes prediction. Wider and Deeper models were competitive but prone to overfitting.
- **Dataset Limitations**: Small size (768 samples) and class imbalance (35% positive cases) limited recall. Future improvements could include oversampling or class weighting.