Neural Network for Diabetes Prediction

- **Name:** Nkechi Rosemary Ogbonna
- **Student ID:** 24145555
- **E-mail:** nkechi.ogbonna@mail.bcu.ac.uk

- **Main Code Link:** https://colab.research.google.com/drive/1U0cpcbB3lg-3SpwV_2cfVZ5_wl7CMOHC?usp=sharing
- **Blog Link** https://colab.research.google.com/drive/1ir5md6BmB3PmuKFBnkKZQmv78gKp5P20?usp=sharing

Understanding the Dataset

The **Pima Indians Diabetes Dataset** contains medical records with eight key features such as glucose levels, blood pressure, BMI, and age, along with a binary outcome indicating diabetes status.

Initial Observations

- **Feature Names**: were generic placeholders (A1, A2, etc.), making them unclear.
- **Missing Values** initially seemed absent, but zero values in columns like **Glucose, Blood Pressure, and BMI** were medically implausible and treated as missing data.
- **Class Imbalance**: showed more **non-diabetic** cases, which could bias predictions.
- **Outliers** were detected but not handled in this case.

Column Renaming

To enhance clarity, column names were updated to reflect their medical meaning.

```
# Rename columns for better readability
df.rename(columns={
    'A1': 'Pregnancies',
    'A2': 'Glucose',
    'A3': 'BloodPressure',
    'A4': 'SkinThickness',
    'A5': 'Insulin',
    'A6': 'BMI',
    'A7': 'DiabetesPedigreeFunction',
    'A8': 'Age',
    'class': 'Outcome'
}, inplace=True)
```

Data Preprocessing

1. **Handling Missing Values**

Certain columns contained medically impossible zero values (e.g., Glucose, Blood Pressure, and BMI), as a person cannot have zero blood pressure or glucose levels. These were treated as missing values (NaN) and replaced with the median to maintain the original data distribution.

Affected Columns & Zero Counts:

- Glucose: 5
- BloodPressure: 32
- SkinThickness: 196
- Insulin: 322
- BMI: 9

Replacing these values helped prevent biased learning while preserving dataset integrity.

```
zero_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
for column in zero_columns:
    df[column] = df[column].replace(0, np.nan)  # Replace zero with NaN
```

After introducing missing values, The median value of each column was used for imputation to maintain the statistical distribution.

```
for column in df.columns:
    if df[column].isnull().sum() > 0:
        df[column].fillna(df[column].median(), inplace=True)
```

Why this approach?

- Using median imputation preserves the distribution better than mean imputation, especially in skewed datasets.
- Keeping missing values untreated could lead to biased learning by the model.

2. Handling Class Imbalance

A class imbalance was observed, meaning more **non-diabetic** than **diabetic** cases. The model might predict the majority class more often. **SMOTE (Synthetic Minority Over-sampling Technique)** was applied to balance the dataset.

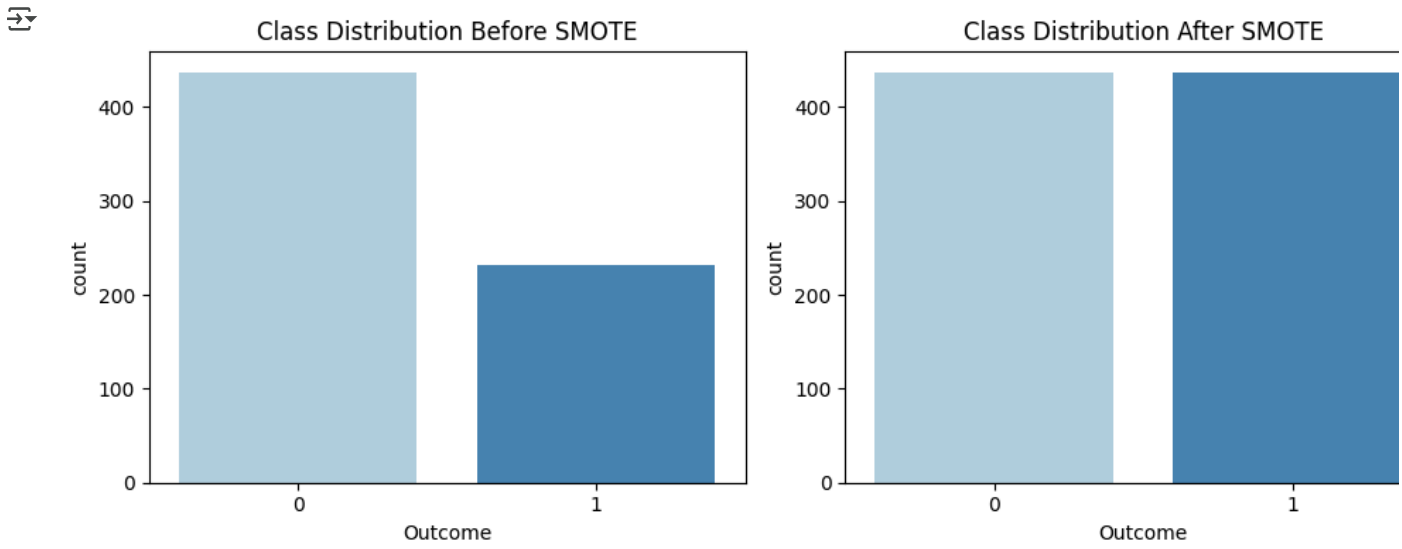
Why SMOOTH?

Instead of duplicating existing minority class instances, SMOTE generates synthetic samples to balance the dataset.

This prevents the model from being biased toward the majority class, improving overall fairness.

Class Distribution Before and After SMOTE:

```
smote = SMOTE(random_state=100)
X_resampled, y_resampled = smote.fit_resample(X, y)
```



Feature Scaling

Since the dataset contained different numerical scales, Standardization was applied to ensure all features contribute equally:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test
```

Why standardization?

- Features like Glucose and BMI had different magnitudes. Scaling ensures no feature dominates training.
- Deep learning models perform better when inputs have zero mean and unit variance.

Model Architecture

A simple yet effective Feedforward Neural Network (FNN) was designed with the following architecture::

```
Input Layer → Dense(32, ReLU) → Dense(64, ReLU) → Dense(1, Sigmoid)
```

Why this architecture?

- Two hidden layers provided sufficient complexity without overfitting.
- L2 regularization (0.009) prevented excessive weight memorization.
- A learning rate of 0.0005 ensured stable gradient updates.

Training & Optimization Summary

Early stopping (patience=15) prevented overfitting, while **ReduceLROnPlateau** fine-tuned the learning rate when progress slowed. The model achieved 80% accuracy, with balanced precision and recall (0.80), ensuring fair predictions. Regularization and adaptive learning rates played a key role in maintaining performance.

```
history = model.fit(
    X_train_scaled, y_train,
    epochs=60,
    validation_data=(X_test_scaled, y_test),
    callbacks=[
        EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=8, min_lr=1e-6)
    ]
)
```

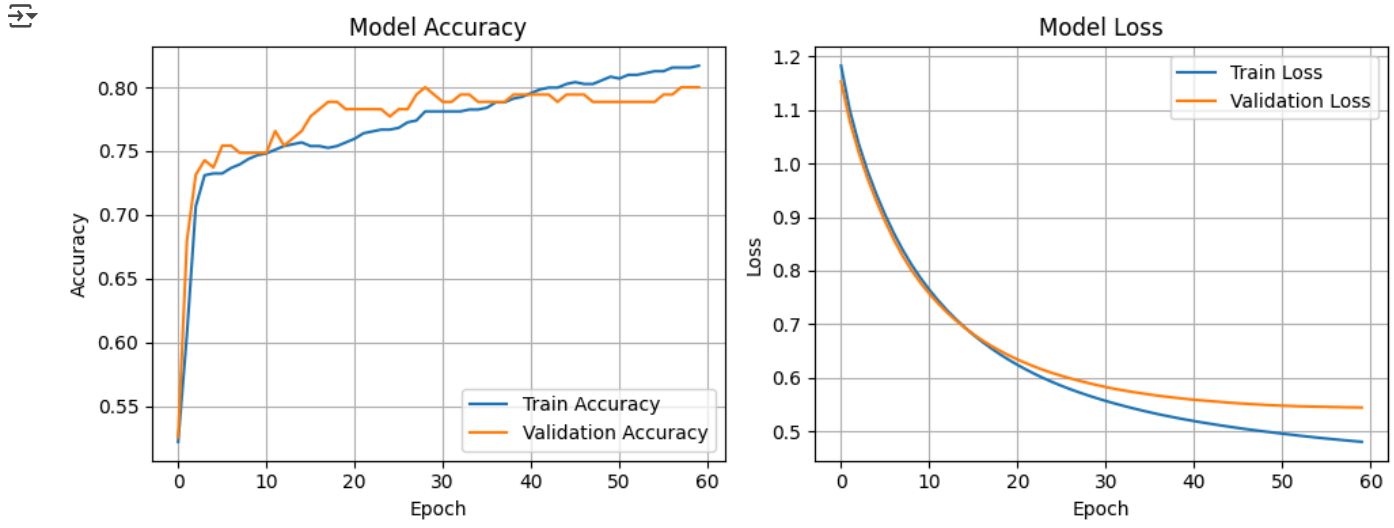
Why These Strategies?

- **Early stopping (patience=15)** prevented overfitting by stopping training when validation loss stops improving..
- **ReduceLROnPlateau** dynamically adjusted learning rates when improvement plateaued.

Model Performance

Accuracy and Loss Curves

```
# Plot Accuracy and Loss
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Final Model Evaluation

Model Accuracy:

```
# Evaluate Model
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"\nTest Accuracy: {accuracy * 100:.2f}%")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7724 - loss: 0.5889
```

```
Test Accuracy: 80.00%
```

Inbaised Prediction Counts:

```
y_pred = (model.predict(X_test_scaled[:len(test_df)]) > 0.5).astype(int).flatten()
print(f"Prediction Summary counts:\n{pd.DataFrame(y_pred).value_counts()}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Prediction Summary counts:
```

	count
1	51
0	49

dtype: int64

Experimentation & Hyperparameter Tuning

Throughout training, several experiments were conducted by adjusting hyperparameters such as **learning rate, L2 regularization, and number of epochs**. The objective was to **balance underfitting and overfitting** while achieving the best generalization.

Experiment	Hidden Layers	Units Per Layer	L2 Regularization	Learning Rate	Epochs	Train Accuracy	Validation Accuracy	Observation
Exp 1	2	32, 64	0.005	0.001	50	85%	74%	Overfitting
Exp 2	2	32, 64	0.01	0.0005	60	82%	78%	Balanced Model
Exp 3	3	64, 128, 64	0.001	0.0005	60	90%	72%	Overfitting
Exp 4	2	32, 64	0.009	0.0005	60	81%	80%	Best Performance

Key Observations

- **Higher L2 regularization** (0.009) effectively reduced overfitting while maintaining strong generalization.
- **A lower learning rate (0.0005)** helped the model converge smoothly.
- **Training beyond 60 epochs** provided minimal gains while increasing computation time.