جامعة الملك سعود
King Saud University

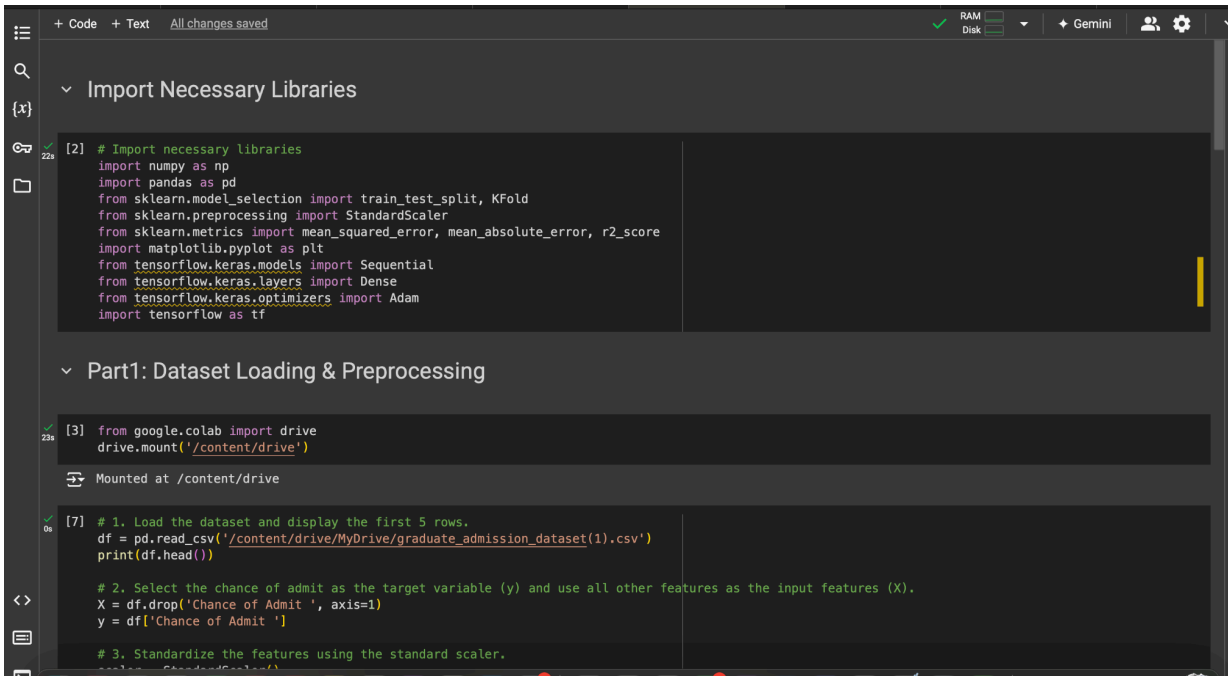
# Machine Learning HW3

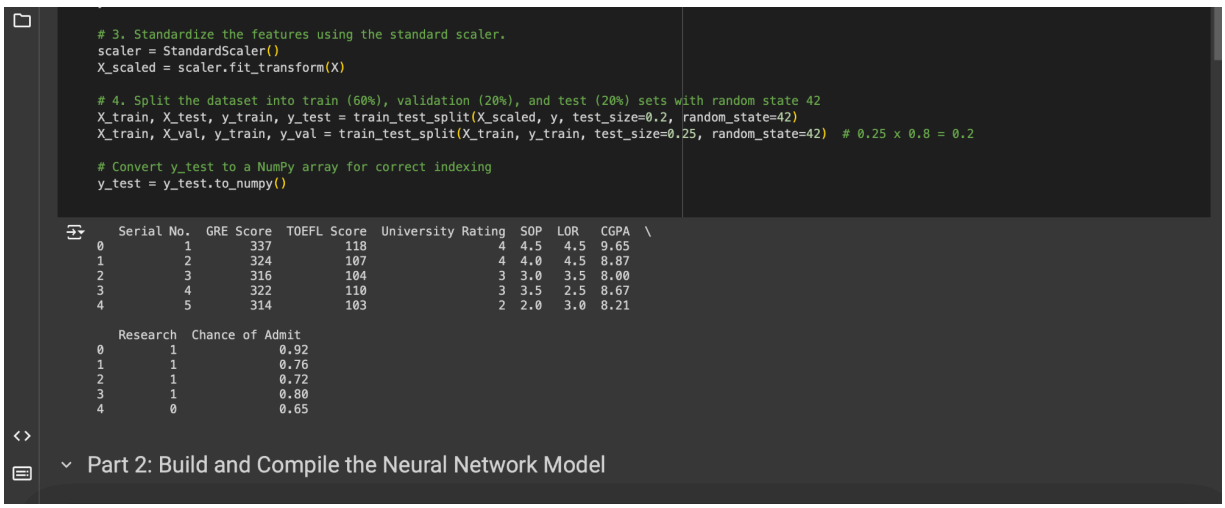
Yara Alfouzan
443203087

## DATA SET DESCRIPTION:

The  Graduate Admission Dataset contains information about applicants to graduate programs. It includes various features that are typically considered during the admission process(GRE Scores,TOEFL Scores,University Rating,Statement of Purpose (SOP),Letter of Recommendation (LOR) Strength,Undergraduate GPA,Research Experience,Chance of Admit) , in this homework we will implement and train a neural network for a regression task using Keras using these various features to predict the "Chance of Admit."

## Clear screenshots of the code, evaluations and outputs:



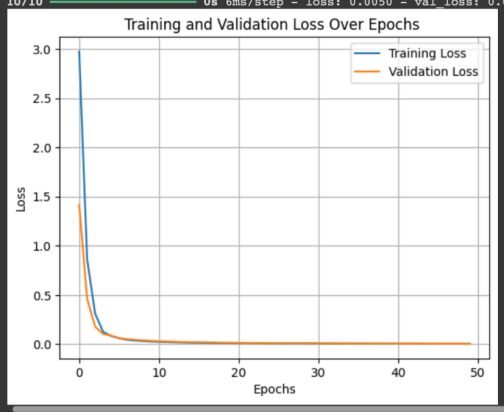
## Part 2: Build and Compile the Neural Network Model

```python
# 1. Complete the function build_model to build a simple feedforward neural network with the following architecture:
# • Input layer: Based on the number of features in the dataset.
# • Hidden layer: 10 neurons with the ReLU activation function.
# • Output layer: A single neuron with a linear activation function for regression.

def build_model():
    model = Sequential()
    model.add(Dense(10, activation='relu',input_dim=X_train.shape[1] ))
    model.add(Dense(1, activation='linear'))


    # 2. Compile the model using Mean Squared Error (MSE) as the loss function and the Adam optimizer with a learning rate of 0.01.

    model.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=0.01))
    return model

# 3. Train the model using the training set and validate on the validation set with 50 epochs and batch size 32.
model = build_model()
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))

# Plot training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```

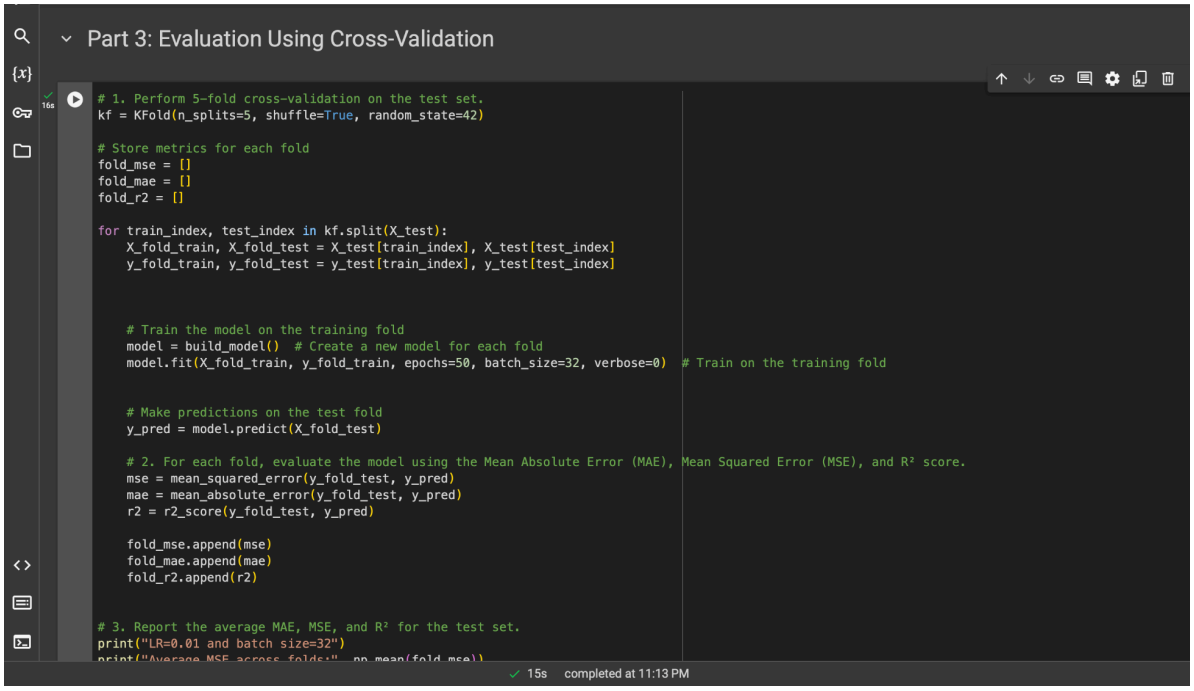
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.4430 - val_loss: 0.2122
```

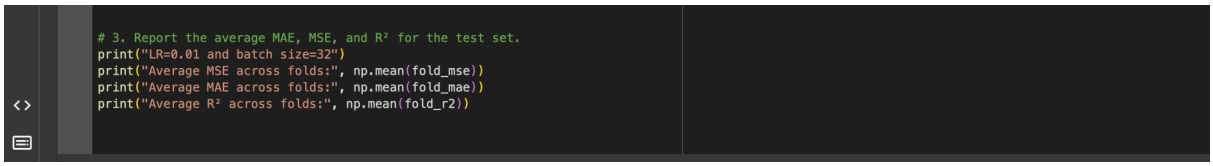
+ Code + Text All changes saved

```
Epoch 47/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0055 - val_loss: 0.0064
Epoch 48/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0052 - val_loss: 0.0062
Epoch 49/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0047 - val_loss: 0.0061
Epoch 50/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0050 - val_loss: 0.0060
```




## Part 3: Evaluation Using Cross-Validation



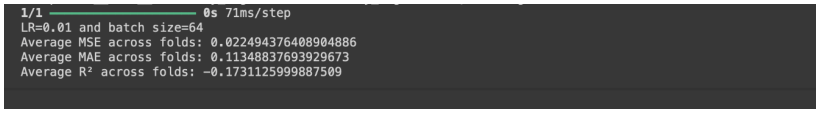


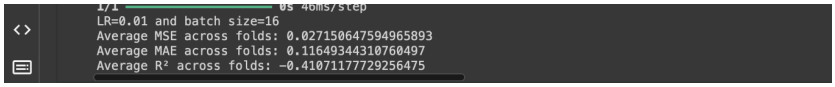
## A table displaying printed results (e.g., model parameters and/or predicted values):

## MSE , MAE ,AND R^2:

## Here are some of the outputs at several LR and batch sizes specified in the printed statement

Learning Rate 0.01:



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
LR=0.01 and batch size=32
Average MSE across folds: 0.015125461536795535
Average MAE across folds: 0.10042175749540327
Average R² across folds: 0.22646966545138278
```

## Learning Rate 0.001

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
LR=0.001 and batch size=64
Average MSE across folds: 0.6290488938318151
Average MAE across folds: 0.6137113001346588
Average R² across folds: -34.098611630351236
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
LR=0.001 and batch size=32
Average MSE across folds: 0.23625051766685906
Average MAE across folds: 0.393665089815855
Average R² across folds: -11.475281351952836
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
LR=0.001 and batch size=16
Average MSE across folds: 0.16972428482415342
Average MAE across folds: 0.326173982706666
Average R² across folds: -7.441868100188134
```

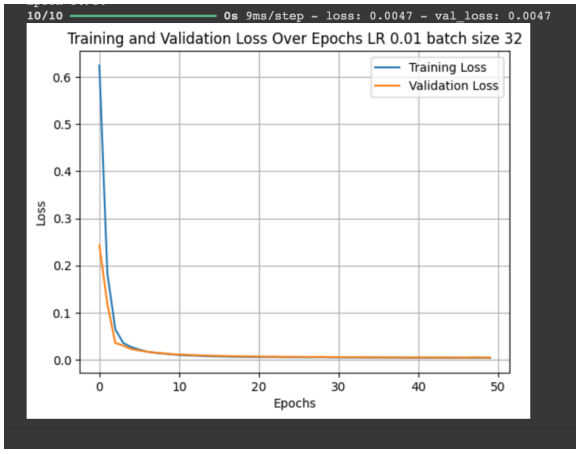
## Learning Rate 0.1:

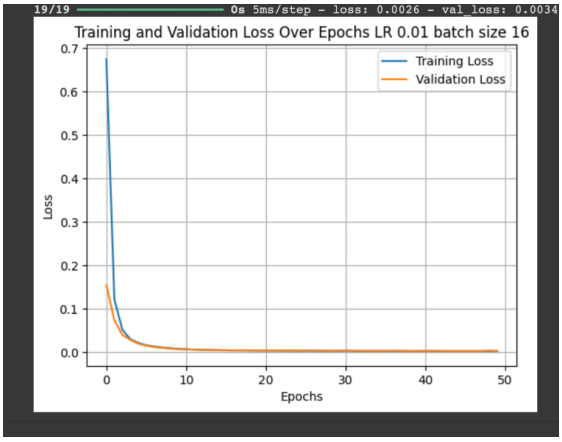
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
LR=0.1 and batch size=16
Average MSE across folds: 0.006227111004786278
Average MAE across folds: 0.060076577568054l9
Average R² across folds: 0.6817006173976485
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
LR=0.1 and batch size=32
Average MSE across folds: 0.005704663499543468
Average MAE across folds: 0.056855614328384395
Average R² across folds: 0.7104786662381615
```

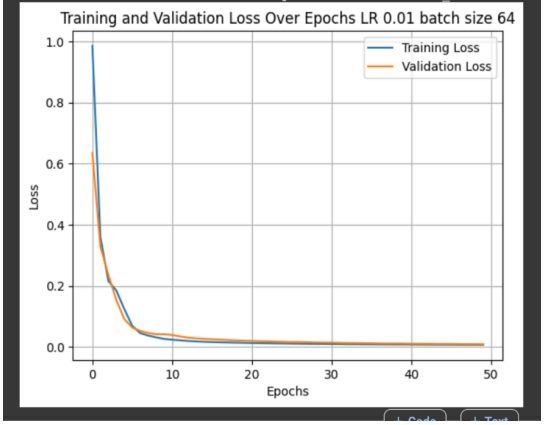
```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
LR=0.1 and batch size=64
Average MSE across folds: 0.010539361052626284
Average MAE across folds: 0.07542648433446883
Average R² across folds: 0.5026643365211059
```

## Some graph outputs using several learning rate and batch sizes as specified in the title of the graph:

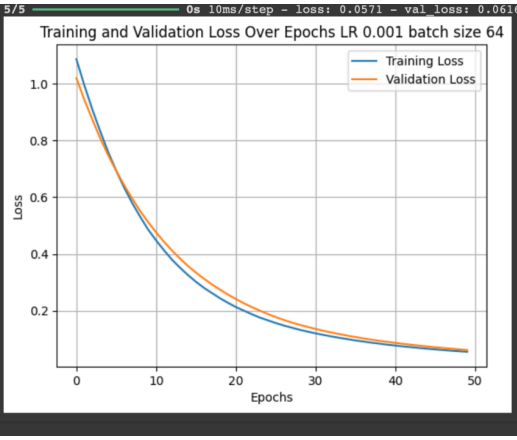


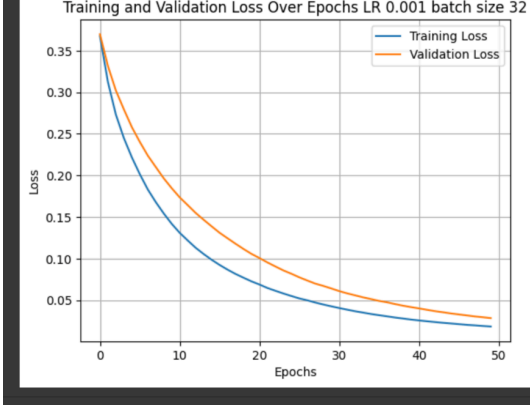
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0072 - val_loss: 0.0090



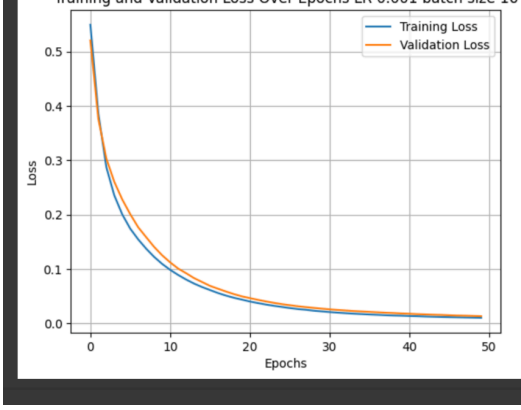

5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0571 - val_loss: 0.0616



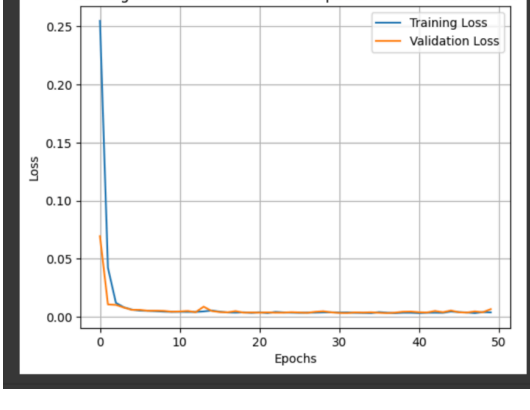

10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0162 - val_loss: 0.0284



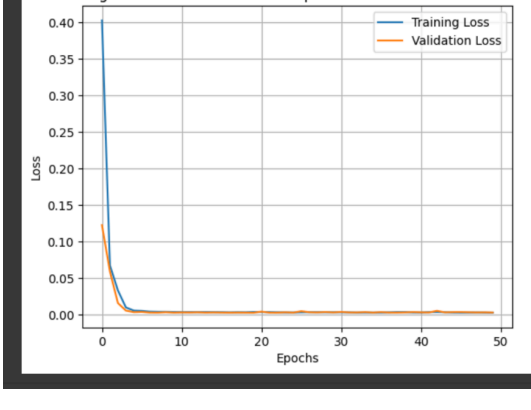

19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0105 - val_loss: 0.0132

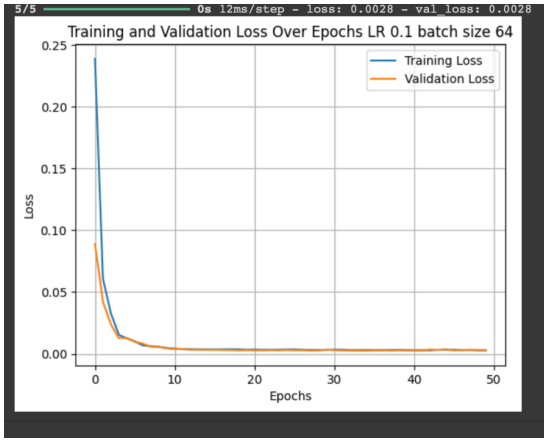



19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0044 - val_loss: 0.0066




10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0026 - val_loss: 0.0031






## Part 3:

## 1.Experiment with different learning rates (e.g., 0.001, 0.01, 0.1) and compare the effect on convergence.

Since the outputs are slightly different with every run due to that when a neural network is created, the weights of the neurons are initialized randomly. These weights determine how the network processes inputs and learns over time. Even slight differences in weight initialization can lead to different learning dynamics and, consequently, different results. A few other reasons also affect the variability of the outputs such as data shuffling.

### Therefore in general :

### Low LR :

Leads to slower convergence, but it may help the model find a more precise local minimum. If too small, it might take too long to converge or get stuck in local minima.

### High LR :

Leads to faster convergence initially, but the risk of overshooting the optimal minimum is higher, which can cause the model to diverge.

In the **screenshot graph results** presented earlier if **we fix batch size to 32** and compare we can see that a learning rate of **0.1** seems to converge the fastest , A learning rate of **0.01** converges almost the same as 0.1 but slightly slower, and a learning rate of **0.001** presented slow convergence.

## 2. Experiment with different batch sizes (e.g., 16, 32, 64) and evaluate the performance.

From the results presented in the screenshots, batch size 32 seemed to give the best performance **since it gave lowest MAE, MSE and highest R^2** when the LR was 0.1 and 0.01 but batch size 16 performed slightly better on LR 0.001, batch size 64 was the worst in LR 0.1 and 0.001 and second worst at 0.01; **in general small batch sizes** might offer better generalization but with more variability in loss per epoch. It can sometimes improve performance metrics due to better exploration of the loss surface.**Larger batch sizes** are more likely to lead to poorer generalization, and potential increases in MAE and MSE and a lower $R^2$ on the test set.larger batch size **converges faster** but the model could get stuck in local minima.
In our case batch size 32 seems to be the most appropriate.

**With LR fixed to 0.01:**
As we can see in this example the **best performance** was presented by **batch size 32**.

| **Batch size** | **MAE** | **MSE** | **R^2** |
|---|---|---|---|
| 16 | 0.116 | 0.027 | -0.410 |
| 32 | 0.100 | 0.015 | 0.226 |
| 64 | 0.113 | 0.022 | -0.173 |