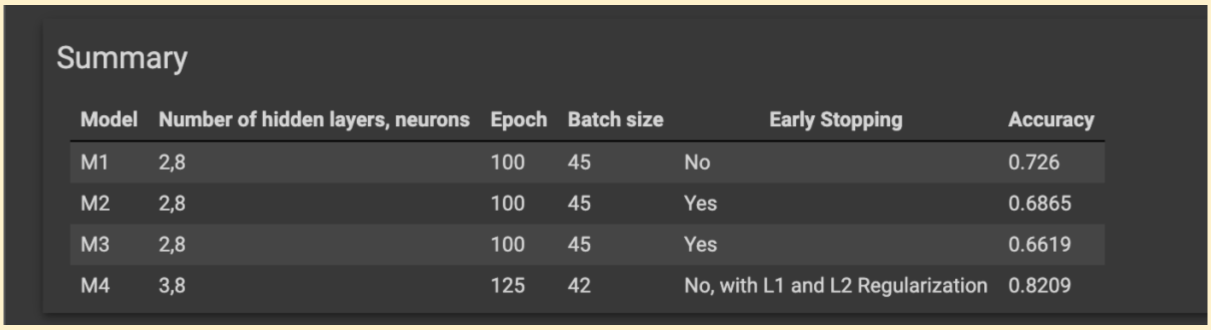
Deep Learning for Diabetes Prediction

Deep learning algorithms can effectively predict the likelihood of diabetes by leveraging large datasets to train neural networks, unlocking the full potential of AI to support healthcare professionals in making informed decisions. This blog illustrates the practical application of deep learning in predicting diabetes, using a dataset comprising 668 training entries and 100 testing entries, which can be accessed via the provided Colab notebook.

To ensure the dataset is devoid of errors, exploratory data analysis is carried out after importing the necessary libraries and loading the training set. A count plot is generated to visually represent the distribution of positive and negative diabetes cases. For evaluation purposes, the training data is divided into training and validation sets before model training.

The initial model, consisting of two layers and eight neurons, utilizes the RELU activation function for the hidden layers. The model is compiled with the Adam optimizer and binary_crossentropy loss function, trained using a batch size of 60 and 50 epochs.

To optimize the model, the number of neurons is increased and early stopping is employed to halt training when accuracy ceases to improve. L1 and L2 regularization are also applied, which result in significant improvement in accuracy.
Evaluation.



Summary

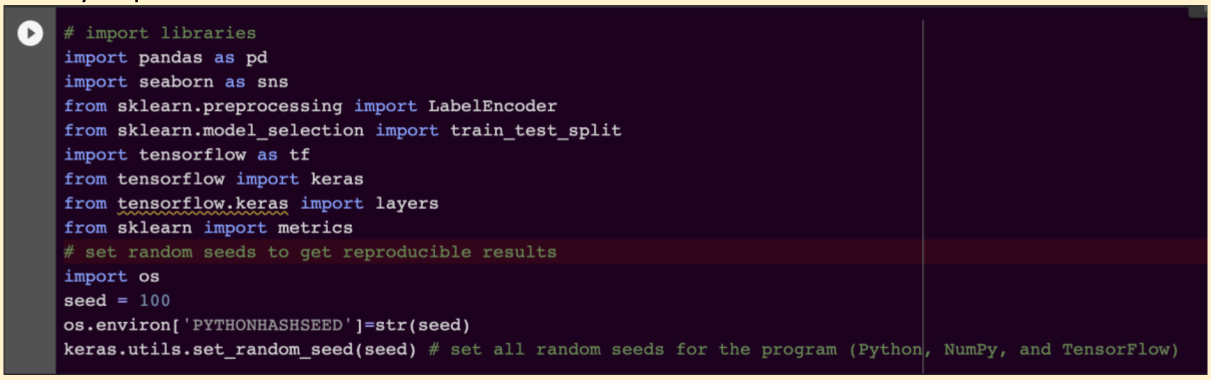
Model	Number of hidden layers, neurons	Epoch	Batch size	Early Stopping	Accuracy
M1	2,8	100	45	No	0.726
M2	2,8	100	45	Yes	0.6865
M3	2,8	100	45	Yes	0.6619
M4	3,8	125	42	No, with L1 and L2 Regularization	0.8209

Observations

- Early stopping did not increase the accuracy. It is suspected that early stopping caused overfitting on the train data which caused poor predictions on the validation data.
- Regularization, with adding more layers improved the accuracy.
- The best model used the relu activation function.
- The best accuracy achieved was 0.8209

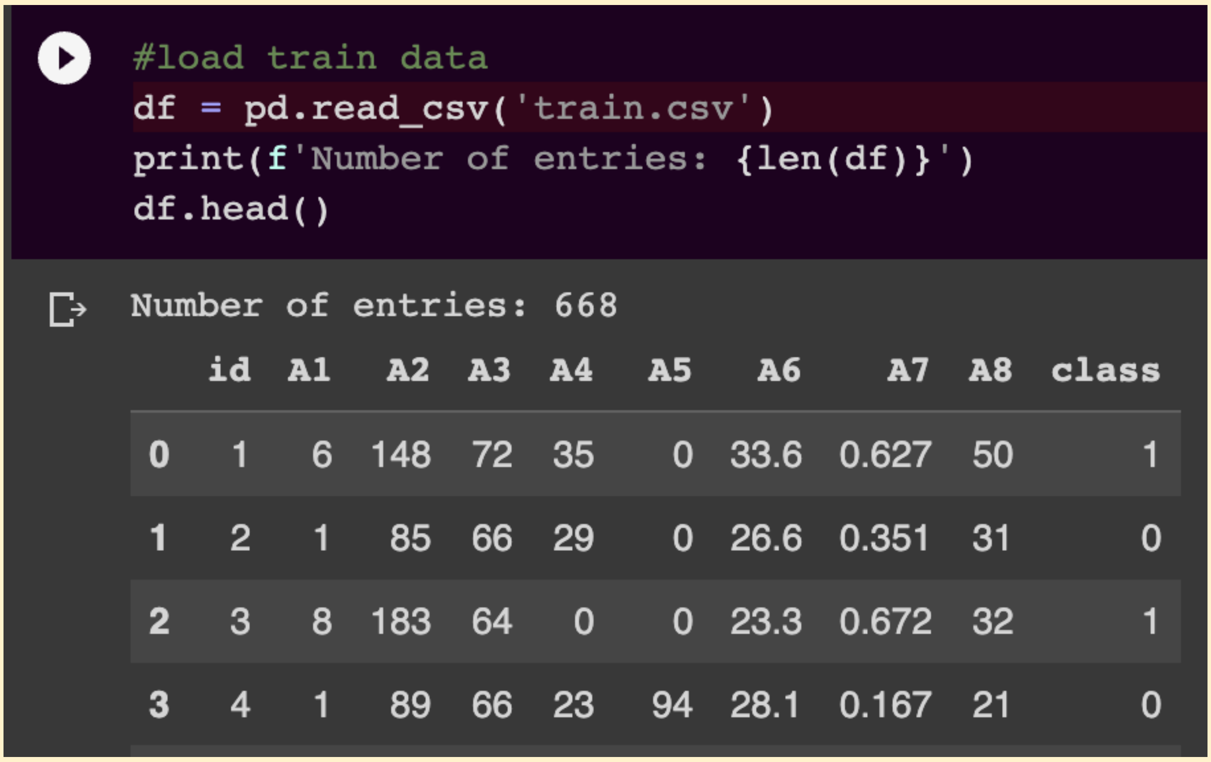
Visuals illustrating the process are shown below.

1. Library imports



```
# import libraries
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn import metrics
# set random seeds to get reproducible results
import os
seed = 100
os.environ['PYTHONHASHSEED']=str(seed)
keras.utils.set_random_seed(seed) # set all random seeds for the program (Python, NumPy, and TensorFlow)
```

2. Loading data

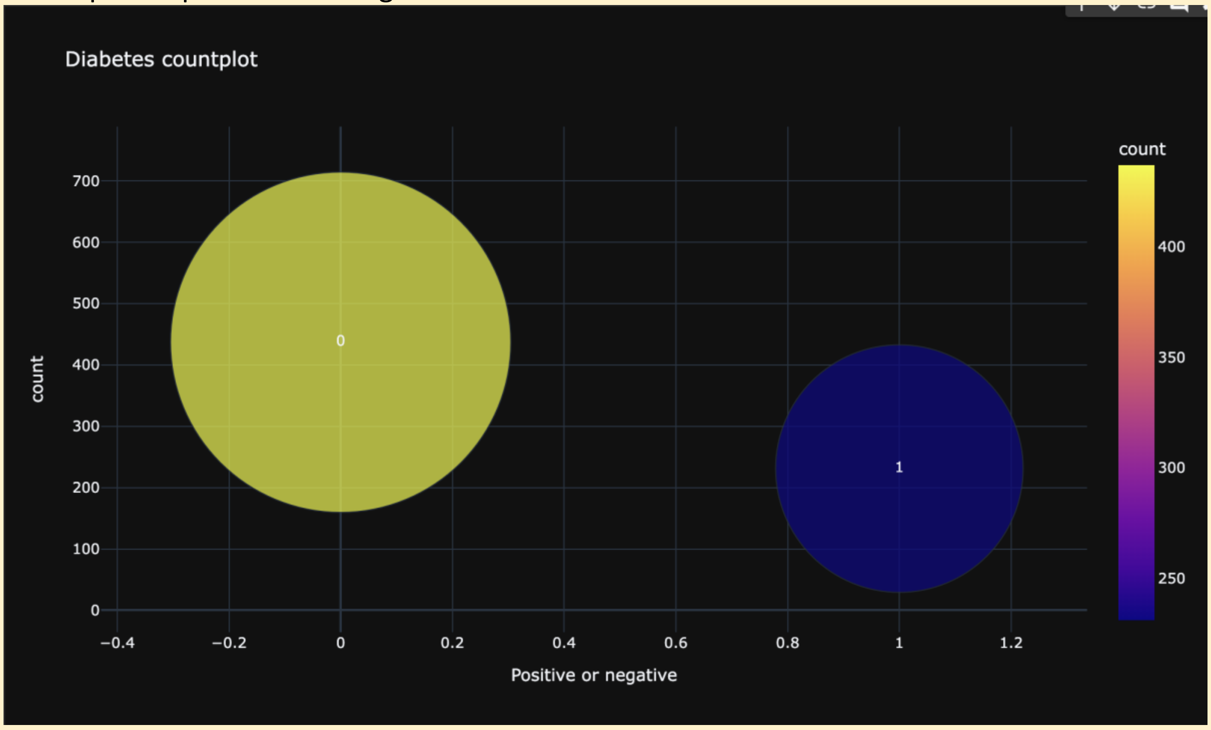


```
#load train data
df = pd.read_csv('train.csv')
print(f'Number of entries: {len(df)}')
df.head()
```

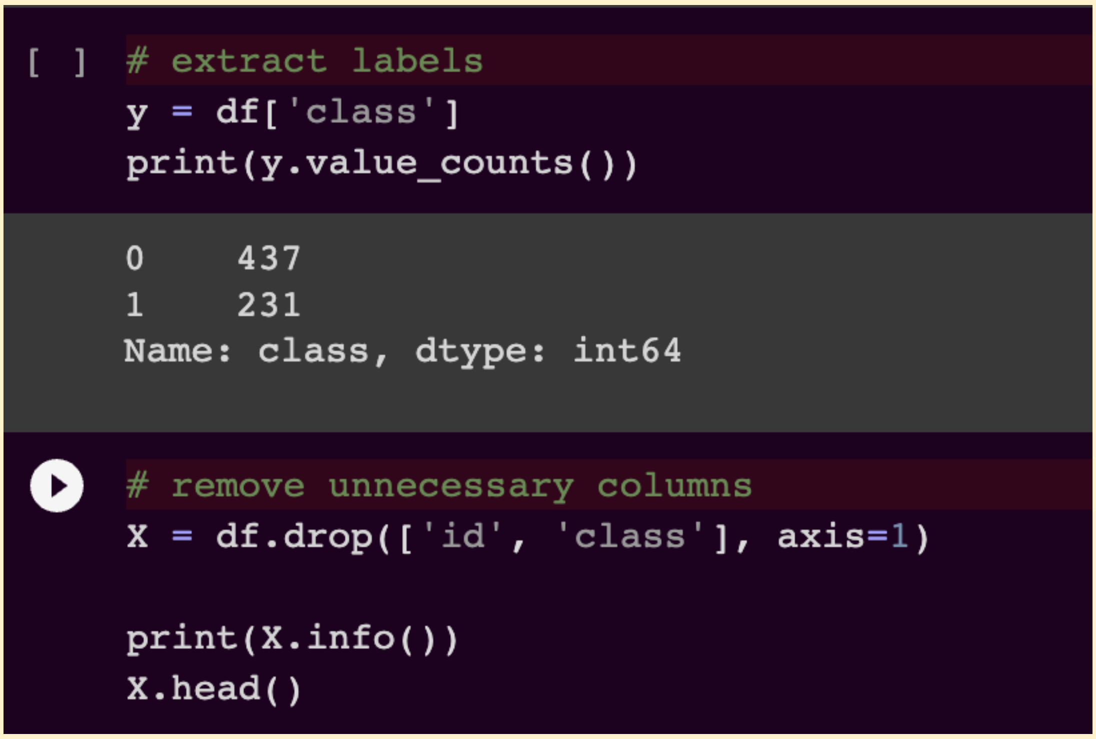
```
Number of entries: 668
```

	id	A1	A2	A3	A4	A5	A6	A7	A8	class
0	1	6	148	72	35	0	33.6	0.627	50	1
1	2	1	85	66	29	0	26.6	0.351	31	0
2	3	8	183	64	0	0	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0

3. Count plot of positive and negative cases



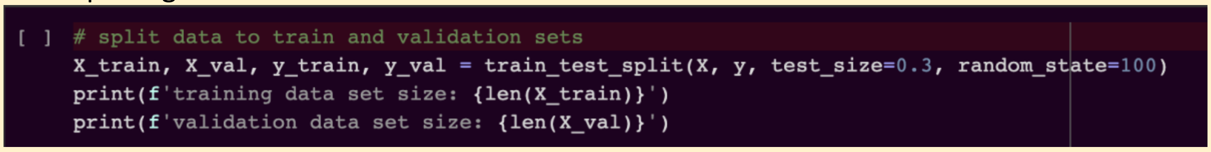
4. Label extractions

5. 

```
[ ] # extract labels
    y = df['class']
    print(y.value_counts())

    0    437
    1    231
    Name: class, dtype: int64
```

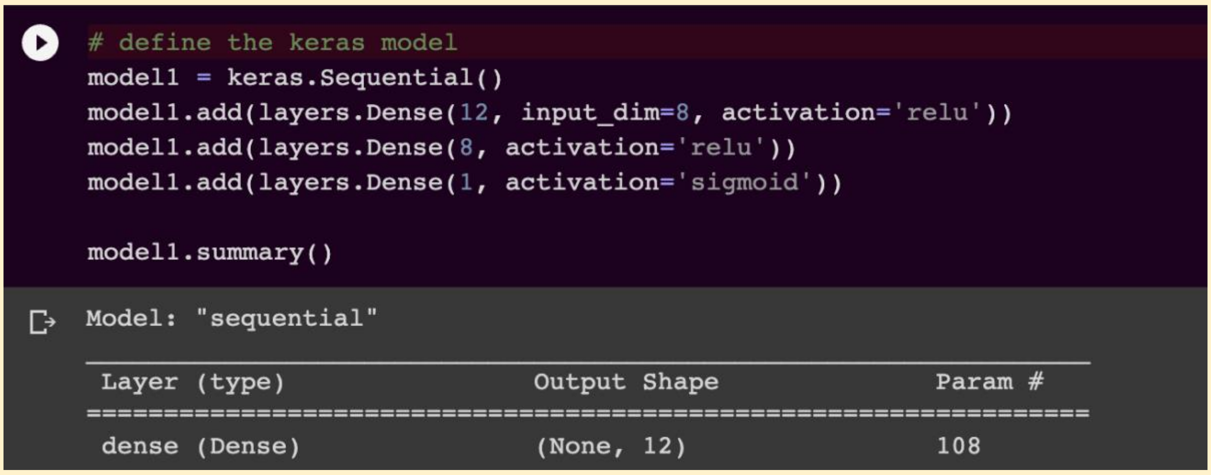
```
    # remove unnecessary columns
    X = df.drop(['id', 'class'], axis=1)

    print(X.info())
    X.head()
```

6. Data Splitting



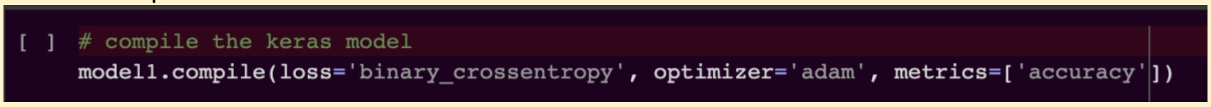
```
[ ] # split data to train and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=100)
    print(f'training data set size: {len(X_train)}')
    print(f'validation data set size: {len(X_val)}')
```

7. Model definition



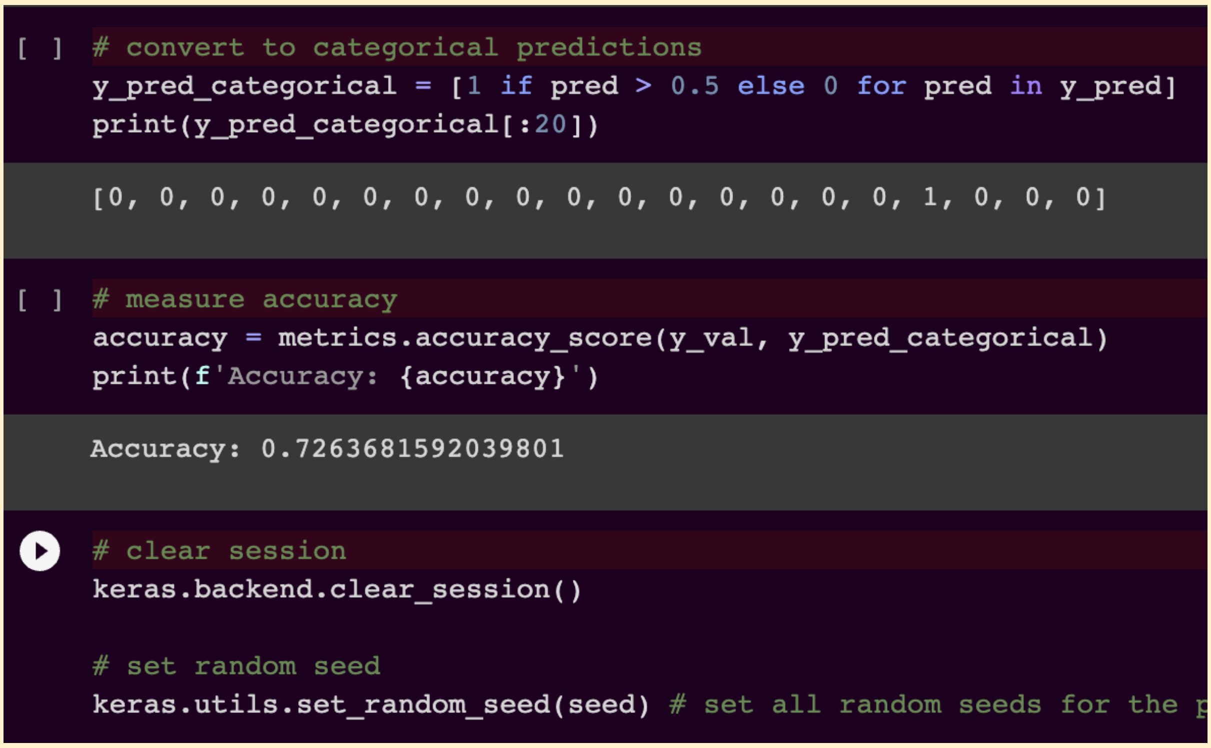
```
    # define the keras model
    model1 = keras.Sequential()
    model1.add(layers.Dense(12, input_dim=8, activation='relu'))
    model1.add(layers.Dense(8, activation='relu'))
    model1.add(layers.Dense(1, activation='sigmoid'))

    model1.summary()

    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     dense (Dense)               (None, 12)                108
```

8. Model compilation



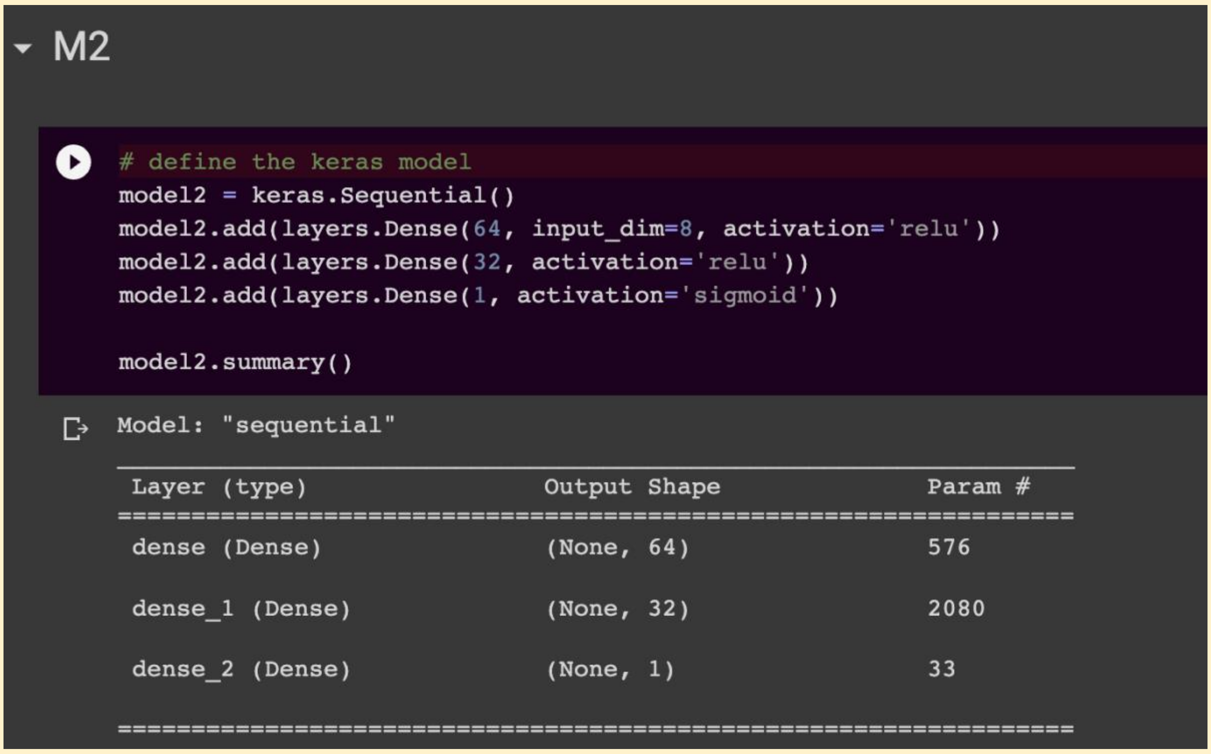
```
[ ] # compile the keras model
    model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

9. Evaluation



```
[ ] # convert to categorical predictions
    y_pred_categorical = [1 if pred > 0.5 else 0 for pred in y_pred]
    print(y_pred_categorical[:20])

    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

[ ] # measure accuracy
    accuracy = metrics.accuracy_score(y_val, y_pred_categorical)
    print(f'Accuracy: {accuracy}')

    Accuracy: 0.7263681592039801

    # clear session
    keras.backend.clear_session()

    # set random seed
    keras.utils.set_random_seed(seed) # set all random seeds for the p
```

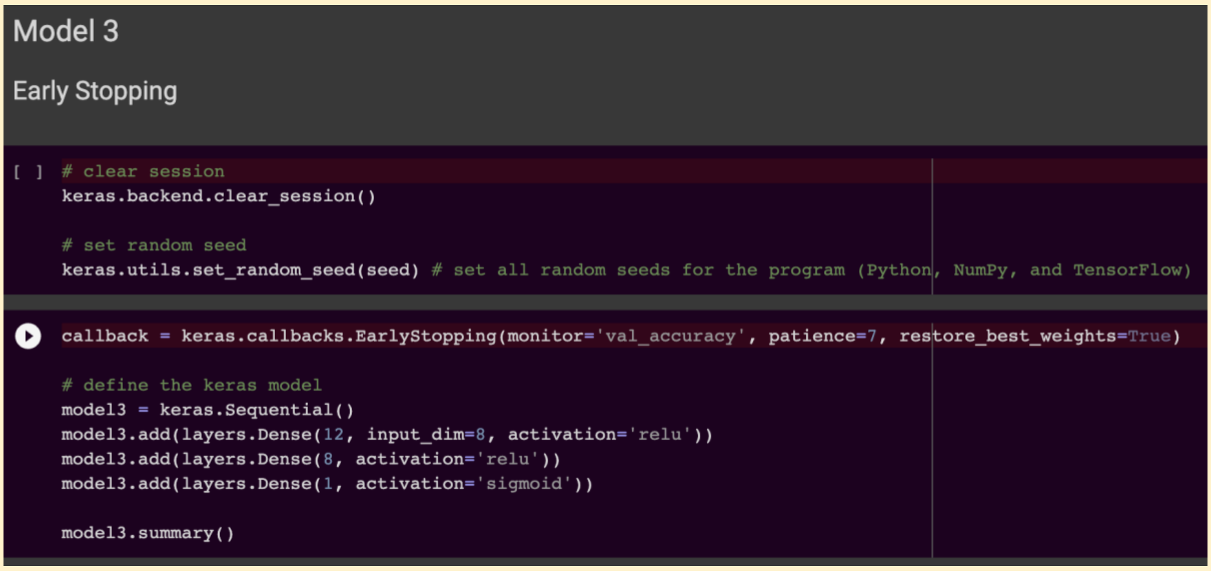
M2



M2

```
# define the keras model
model2 = keras.Sequential()
model2.add(layers.Dense(64, input_dim=8, activation='relu'))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))

model2.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                576

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
```

M3



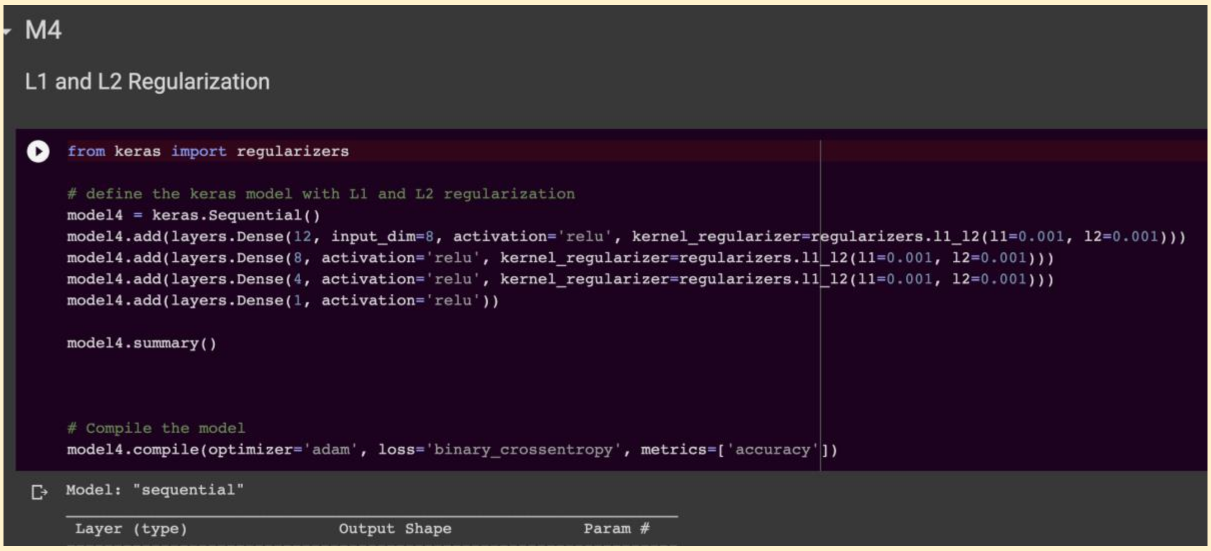
Model 3

Early Stopping

```
# clear session
keras.backend.clear_session()

# set random seed
keras.utils.set_random_seed(seed) # set all random seeds for the program (Python, NumPy, and TensorFlow)
```

```
callback = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=7, restore_best_weights=True)

# define the keras model
model3 = keras.Sequential()
model3.add(layers.Dense(12, input_dim=8, activation='relu'))
model3.add(layers.Dense(8, activation='relu'))
model3.add(layers.Dense(1, activation='sigmoid'))

model3.summary()
```

M4



M4

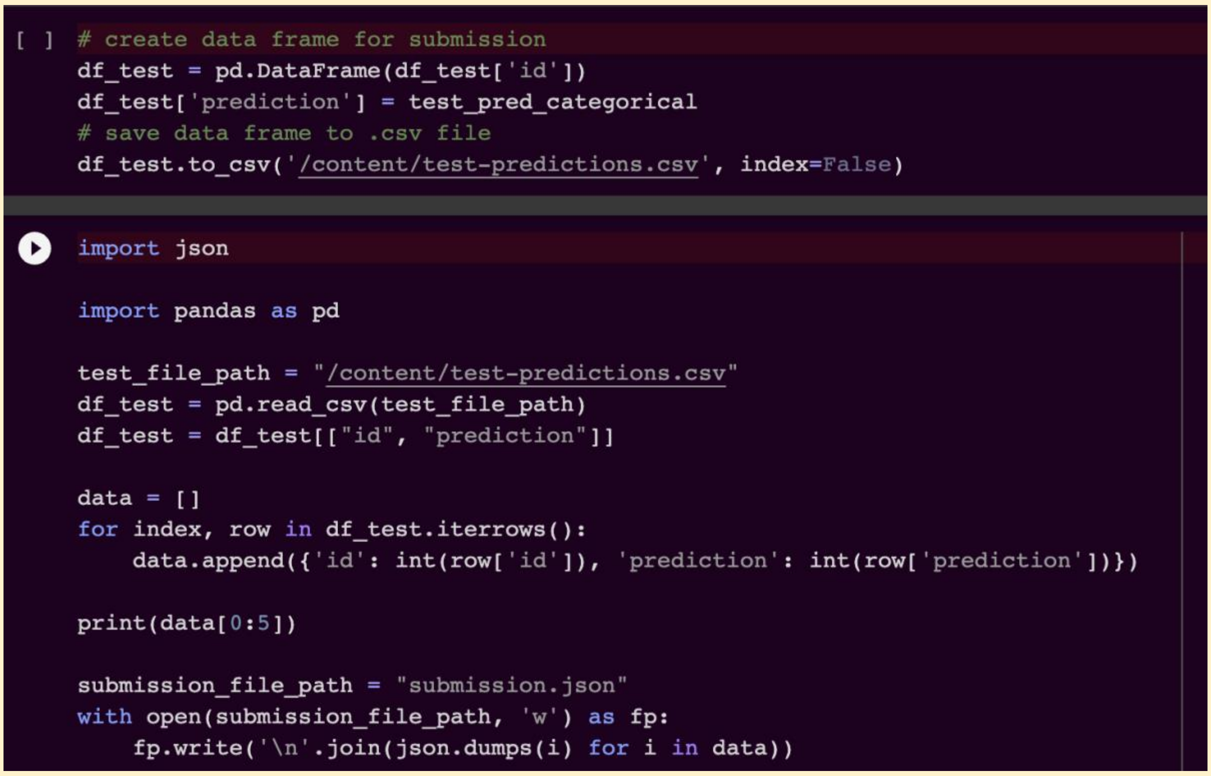
L1 and L2 Regularization

```
from keras import regularizers

# define the keras model with L1 and L2 regularization
model4 = keras.Sequential()
model4.add(layers.Dense(12, input_dim=8, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001)))
model4.add(layers.Dense(8, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001)))
model4.add(layers.Dense(4, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001)))
model4.add(layers.Dense(1, activation='relu'))

model4.summary()



# Compile the model
model4.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

Final Predictions

Model m4 is used for making final predictions.



```
[ ] # create data frame for submission
    df_test = pd.DataFrame(df_test['id'])
    df_test['prediction'] = test_pred_categorical
    # save data frame to .csv file
    df_test.to_csv('/content/test-predictions.csv', index=False)
```

```
import json

import pandas as pd

test_file_path = "/content/test-predictions.csv"
df_test = pd.read_csv(test_file_path)
df_test = df_test[["id", "prediction"]]

data = []
for index, row in df_test.iterrows():
    data.append({'id': int(row['id']), 'prediction': int(row['prediction'])})

print(data[0:5])

submission_file_path = "submission.json"
with open(submission_file_path, 'w') as fp:
    fp.write('\n'.join(json.dumps(i) for i in data))
```

Through this blog, we have explored variety of techniques for improving model accuracy when using neural networks.