Name: Borris A. Esplanada

Section: CPE32S1

Instructions:

- Choose any dataset applicable to either a classification problem or a regression problem.
- Explain your datasets and the problem being addressed.
- Show evidence that you can do the following:
    1. Save a model in HDF5 format
    2. Save a model and load the model in a JSON format
    3. Save a model and load the model in a YAML format
    4. Checkpoint Neural Network Model Improvements
    5. Checkpoint Best Neural Network Model only
    6. Load a saved Neural Network model
    7. Visualize Model Training History in Keras
    8. Show the application of Dropout Regularization
    9. Show the application of Dropout on the visible layer
    10. Show the application of Dropout on the hidden layer
    11. Show the application of a time-based learning rate schedule
    12. Show the application of a drop-based learning rate schedule
- Submit the link to your Google Colab (make sure that it is accessible to me) NOTE:

Submit a well-prepared notebook for your report Include conclusion or learning

```
from google.colab import drive
drive.mount('/content/drive')
```

Classification Problem

Title: Personal Loan Modeling

Link: https://www.kaggle.com/datasets/teertha/personal-loan-modeling

Explain your datasets and the problem being addressed.

The Personal Loan Modeling dataset contains information about customers of a bank, including their personal and financial details, as well as whether or not they accepted a personal loan offer in the past. The dataset consists of 5,000 rows and 14 columns.

The problem being addressed is whether or not a customer will accept a personal loan offer. This is a binary classification problem, where the goal is to predict whether a customer will accept the loan offer or not based on their personal and financial information.

This is important to banks and other financial institutions because they want to make targeted marketing efforts towards customers who are most likely to accept personal loan offers, so they can increase their chances of making a profit.

By using this dataset, banks can gain insights into what factors may influence a customer's decision to accept a personal loan offer and adjust their marketing strategy accordingly.

For classification, do the following:

- Create a base model
- Evaluate the model with k-fold cross validation
- Improve the accuracy of your model by applying additional hidden layers

Double-click (or enter) to edit

```
path = "/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv"
df = pd.read_csv(path)
```

1. Save a model in HDF5 format

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
```

```
pip install h5py
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: h5py in /usr/local/lib/python3.9/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.9/dist-packages (from h5py) (1.22.4)
```

```
# load dataset

df = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")
X = df.drop(['ID', 'ZIP Code', 'Personal Loan'], axis=1)
Y = df['Personal Loan']
X_train, Y_train = X[:3500], Y[:3500]
X_test, Y_test = X[3500:], Y[3500:]
```

```
df
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	4996	29	3	40	92697	1	1.9	3	0	0	0	0	1
4996	4997	30	4	15	92037	4	0.4	1	85	0	0	0	1
4997	4998	63	39	24	93023	2	0.3	3	0	0	0	0	0
4998	4999	65	40	49	90034	3	0.5	2	0	0	0	0	1
4999	5000	28	4	83	92612	3	0.8	1	0	0	0	0	1

5000 rows × 14 columns

```
model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

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

```
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
```

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=50, batch_size=32, callbacks=[checkpoint])
```

```
Epoch 1/50
110/110 [==============================] - 1s 5ms/step - loss: 7.2153 - accuracy: 0.6114 - val_loss: 0.5688 - val_accuracy: 0.8873
Epoch 2/50
110/110 [==============================] - 0s 3ms/step - loss: 0.6340 - accuracy: 0.8617 - val_loss: 0.4424 - val_accuracy: 0.8847
Epoch 3/50
110/110 [==============================] - 0s 4ms/step - loss: 0.5052 - accuracy: 0.8546 - val_loss: 0.3747 - val_accuracy: 0.8973
Epoch 4/50
110/110 [==============================] - 1s 5ms/step - loss: 0.4369 - accuracy: 0.8629 - val_loss: 0.3286 - val_accuracy: 0.8860
Epoch 5/50
110/110 [==============================] - 1s 5ms/step - loss: 0.3845 - accuracy: 0.8689 - val_loss: 0.2987 - val_accuracy: 0.8787
Epoch 6/50
110/110 [==============================] - 0s 4ms/step - loss: 0.3404 - accuracy: 0.8737 - val_loss: 0.2612 - val_accuracy: 0.9073
Epoch 7/50
```

```
110/110 [==============================] - 1s 5ms/step - loss: 0.3087 - accuracy: 0.8829 - val_loss: 0.2502 - val_accuracy: 0.9027
Epoch 8/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2868 - accuracy: 0.8874 - val_loss: 0.2353 - val_accuracy: 0.9133
Epoch 9/50
110/110 [==============================] - 1s 5ms/step - loss: 0.2736 - accuracy: 0.8914 - val_loss: 0.2241 - val_accuracy: 0.9147
Epoch 10/50
110/110 [==============================] - 1s 5ms/step - loss: 0.2612 - accuracy: 0.8906 - val_loss: 0.2231 - val_accuracy: 0.9127
Epoch 11/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2541 - accuracy: 0.8940 - val_loss: 0.2262 - val_accuracy: 0.9040
Epoch 12/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2508 - accuracy: 0.8931 - val_loss: 0.2072 - val_accuracy: 0.9140
Epoch 13/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2400 - accuracy: 0.8943 - val_loss: 0.2177 - val_accuracy: 0.9147
Epoch 14/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2325 - accuracy: 0.8960 - val_loss: 0.2041 - val_accuracy: 0.9127
Epoch 15/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2288 - accuracy: 0.8989 - val_loss: 0.2074 - val_accuracy: 0.9127
Epoch 16/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2238 - accuracy: 0.9017 - val_loss: 0.1998 - val_accuracy: 0.9140
Epoch 17/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2163 - accuracy: 0.9043 - val_loss: 0.1896 - val_accuracy: 0.9160
Epoch 18/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2154 - accuracy: 0.9026 - val_loss: 0.1900 - val_accuracy: 0.9120
Epoch 19/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2073 - accuracy: 0.9049 - val_loss: 0.1847 - val_accuracy: 0.9120
Epoch 20/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2020 - accuracy: 0.9094 - val_loss: 0.1840 - val_accuracy: 0.9120
Epoch 21/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2006 - accuracy: 0.9134 - val_loss: 0.1781 - val_accuracy: 0.9213
Epoch 22/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1943 - accuracy: 0.9143 - val_loss: 0.1780 - val_accuracy: 0.9220
Epoch 23/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1942 - accuracy: 0.9143 - val_loss: 0.1776 - val_accuracy: 0.9180
Epoch 24/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1881 - accuracy: 0.9211 - val_loss: 0.1664 - val_accuracy: 0.9260
Epoch 25/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1830 - accuracy: 0.9234 - val_loss: 0.1675 - val_accuracy: 0.9193
Epoch 26/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1809 - accuracy: 0.9249 - val_loss: 0.1662 - val_accuracy: 0.9293
Epoch 27/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1767 - accuracy: 0.9257 - val_loss: 0.1615 - val_accuracy: 0.9240
Epoch 28/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1778 - accuracy: 0.9271 - val_loss: 0.1735 - val_accuracy: 0.9140
Epoch 29/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1748 - accuracy: 0.9266 - val_loss: 0.1559 - val_accuracy: 0.9200
```

```
score = model.evaluate(X_test, Y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
47/47 [==============================] - 0s 2ms/step - loss: 0.1198 - accuracy: 0.9547
Test loss: 0.11981695890426636
Test accuracy: 0.9546666741371155
```

```
best_model = load_model('best_model.h5')
```

2. Save a model and load the model in a JSON format

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint


# load dataset

df = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")
X = df.drop(['ID', 'ZIP Code', 'Personal Loan'], axis=1)
Y = df['Personal Loan']
X_train, Y_train = X[:3500], Y[:3500]
X_test, Y_test = X[3500:], Y[3500:]


model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

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


checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)


history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=50, batch_size=32, callbacks=[checkpoint])
```

```
Epoch 1/50
110/110 [==============================] - 1s 5ms/step - loss: 6.0605 - accuracy: 0.5246 - val_loss: 0.6234 - val_accuracy: 0.8140
Epoch 2/50
110/110 [==============================] - 0s 3ms/step - loss: 0.4586 - accuracy: 0.8514 - val_loss: 0.2937 - val_accuracy: 0.8893
Epoch 3/50
110/110 [==============================] - 0s 3ms/step - loss: 0.3118 - accuracy: 0.8854 - val_loss: 0.2502 - val_accuracy: 0.9033
Epoch 4/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2890 - accuracy: 0.8874 - val_loss: 0.2378 - val_accuracy: 0.8993
Epoch 5/50
110/110 [==============================] - 1s 8ms/step - loss: 0.2669 - accuracy: 0.8909 - val_loss: 0.2219 - val_accuracy: 0.9027
Epoch 6/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2640 - accuracy: 0.8929 - val_loss: 0.2255 - val_accuracy: 0.8980
Epoch 7/50
110/110 [==============================] - 1s 8ms/step - loss: 0.2409 - accuracy: 0.9009 - val_loss: 0.2124 - val_accuracy: 0.9060
Epoch 8/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2419 - accuracy: 0.9006 - val_loss: 0.2522 - val_accuracy: 0.9040
Epoch 9/50
110/110 [==============================] - 1s 8ms/step - loss: 0.2328 - accuracy: 0.9020 - val_loss: 0.1942 - val_accuracy: 0.9127
Epoch 10/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2241 - accuracy: 0.9060 - val_loss: 0.1910 - val_accuracy: 0.9180
Epoch 11/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2123 - accuracy: 0.9046 - val_loss: 0.2009 - val_accuracy: 0.9107
Epoch 12/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2104 - accuracy: 0.9089 - val_loss: 0.1793 - val_accuracy: 0.9200
Epoch 13/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2012 - accuracy: 0.9109 - val_loss: 0.1965 - val_accuracy: 0.9100
Epoch 14/50
110/110 [==============================] - 1s 5ms/step - loss: 0.2035 - accuracy: 0.9120 - val_loss: 0.1816 - val_accuracy: 0.9160
Epoch 15/50
110/110 [==============================] - 1s 5ms/step - loss: 0.2049 - accuracy: 0.9166 - val_loss: 0.1900 - val_accuracy: 0.9200
Epoch 16/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2003 - accuracy: 0.9186 - val_loss: 0.1794 - val_accuracy: 0.9187
Epoch 17/50
110/110 [==============================] - 1s 5ms/step - loss: 0.1955 - accuracy: 0.9200 - val_loss: 0.1719 - val_accuracy: 0.9200
Epoch 18/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1888 - accuracy: 0.9217 - val_loss: 0.1709 - val_accuracy: 0.9220
Epoch 19/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1786 - accuracy: 0.9246 - val_loss: 0.1694 - val_accuracy: 0.9293
Epoch 20/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1785 - accuracy: 0.9254 - val_loss: 0.2090 - val_accuracy: 0.9233
Epoch 21/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1999 - accuracy: 0.9166 - val_loss: 0.1802 - val_accuracy: 0.9260
Epoch 22/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1785 - accuracy: 0.9263 - val_loss: 0.1526 - val_accuracy: 0.9340
Epoch 23/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1674 - accuracy: 0.9331 - val_loss: 0.1717 - val_accuracy: 0.9293
Epoch 24/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1685 - accuracy: 0.9314 - val_loss: 0.1511 - val_accuracy: 0.9393
Epoch 25/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1657 - accuracy: 0.9320 - val_loss: 0.2600 - val_accuracy: 0.9313
Epoch 26/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1723 - accuracy: 0.9306 - val_loss: 0.1604 - val_accuracy: 0.9320
Epoch 27/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1708 - accuracy: 0.9317 - val_loss: 0.1538 - val_accuracy: 0.9380
Epoch 28/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1699 - accuracy: 0.9360 - val_loss: 0.1888 - val_accuracy: 0.9260
Epoch 29/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1585 - accuracy: 0.9369 - val_loss: 0.1449 - val_accuracy: 0.9387
```

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)



from keras.models import model_from_json

with open('model.json', 'r') as json_file:
    loaded_model_json = json_file.read()

loaded_model = model_from_json(loaded_model_json)
```

```
loaded_model.load_weights('best_model.h5')
```

3. Save a model and load the model in a YAML format

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint



# load dataset

df = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")
X = df.drop(['ID', 'ZIP Code', 'Personal Loan'], axis=1)
Y = df['Personal Loan']
X_train, Y_train = X[:3500], Y[:3500]
X_test, Y_test = X[3500:], Y[3500:]



model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))



model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])



checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)



history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=50, batch_size=32, callbacks=[checkpoint])
```

```
Epoch 1/50
110/110 [==============================] - 2s 7ms/step - loss: 0.7679 - accuracy: 0.7974 - val_loss: 0.2642 - val_accuracy: 0.8940
Epoch 2/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2828 - accuracy: 0.8860 - val_loss: 0.2199 - val_accuracy: 0.9013
Epoch 3/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2518 - accuracy: 0.8894 - val_loss: 0.2284 - val_accuracy: 0.9087
Epoch 4/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2373 - accuracy: 0.8997 - val_loss: 0.1913 - val_accuracy: 0.9087
Epoch 5/50
110/110 [==============================] - 1s 4ms/step - loss: 0.2241 - accuracy: 0.8969 - val_loss: 0.2013 - val_accuracy: 0.9020
Epoch 6/50
110/110 [==============================] - 1s 5ms/step - loss: 0.2184 - accuracy: 0.9014 - val_loss: 0.1754 - val_accuracy: 0.9220
Epoch 7/50
110/110 [==============================] - 0s 4ms/step - loss: 0.2103 - accuracy: 0.9031 - val_loss: 0.1719 - val_accuracy: 0.9220
Epoch 8/50
110/110 [==============================] - 0s 3ms/step - loss: 0.2028 - accuracy: 0.9097 - val_loss: 0.1773 - val_accuracy: 0.9227
Epoch 9/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1969 - accuracy: 0.9180 - val_loss: 0.1772 - val_accuracy: 0.9207
Epoch 10/50
110/110 [==============================] - 0s 4ms/step - loss: 0.1912 - accuracy: 0.9171 - val_loss: 0.1597 - val_accuracy: 0.9247
Epoch 11/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1884 - accuracy: 0.9234 - val_loss: 0.1641 - val_accuracy: 0.9320
Epoch 12/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1813 - accuracy: 0.9269 - val_loss: 0.1792 - val_accuracy: 0.9187
Epoch 13/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1839 - accuracy: 0.9229 - val_loss: 0.1451 - val_accuracy: 0.9420
Epoch 14/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1750 - accuracy: 0.9283 - val_loss: 0.1534 - val_accuracy: 0.9427
Epoch 15/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1674 - accuracy: 0.9340 - val_loss: 0.1359 - val_accuracy: 0.9447
Epoch 16/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1643 - accuracy: 0.9383 - val_loss: 0.1600 - val_accuracy: 0.9293
Epoch 17/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1624 - accuracy: 0.9329 - val_loss: 0.1541 - val_accuracy: 0.9440
Epoch 18/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1645 - accuracy: 0.9346 - val_loss: 0.1275 - val_accuracy: 0.9427
Epoch 19/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1568 - accuracy: 0.9417 - val_loss: 0.1433 - val_accuracy: 0.9487
Epoch 20/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1547 - accuracy: 0.9429 - val_loss: 0.1240 - val_accuracy: 0.9593
```

```
Epoch 21/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1501 - accuracy: 0.9423 - val_loss: 0.1232 - val_accuracy: 0.9560
Epoch 22/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1489 - accuracy: 0.9391 - val_loss: 0.1256 - val_accuracy: 0.9607
Epoch 23/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1469 - accuracy: 0.9446 - val_loss: 0.1221 - val_accuracy: 0.9533
Epoch 24/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1402 - accuracy: 0.9466 - val_loss: 0.1169 - val_accuracy: 0.9440
Epoch 25/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1412 - accuracy: 0.9446 - val_loss: 0.1271 - val_accuracy: 0.9547
Epoch 26/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1413 - accuracy: 0.9466 - val_loss: 0.1125 - val_accuracy: 0.9607
Epoch 27/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1346 - accuracy: 0.9480 - val_loss: 0.1117 - val_accuracy: 0.9480
Epoch 28/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1310 - accuracy: 0.9506 - val_loss: 0.1088 - val_accuracy: 0.9547
Epoch 29/50
110/110 [==============================] - 0s 3ms/step - loss: 0.1311 - accuracy: 0.9489 - val_loss: 0.1202 - val_accuracy: 0.9473
```

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

```
from keras.models import model_from_json

with open("model.json", "r") as json_file:
    loaded_model_json = json_file.read()
loaded_model = model_from_json(loaded_model_json)
```

```
loaded_model.load_weights('best_model.h5')
```

5. Checkpoint Best Neural Network Model only

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load data
df = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split data into input and output variables
X = df.drop(['Personal Loan'], axis=1)
y = df['Personal Loan']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Scale data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define model architecture
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(1, activation='sigmoid'))

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

# Define callback to save best model based on validation accuracy
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True, mode='max')

# Train model with callback
model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test), callbacks=[checkpoint])
```

```
Epoch 1/100
125/125 [==============================] - 1s 4ms/step - loss: 0.6026 - accuracy: 0.6862 - val_loss: 0.3695 - val_accuracy: 0.8920
```

```
Epoch 2/100
125/125 [==============================] - 0s 3ms/step - loss: 0.2611 - accuracy: 0.9275 - val_loss: 0.2513 - val_accuracy: 0.9100
Epoch 3/100
125/125 [==============================] - 0s 3ms/step - loss: 0.1809 - accuracy: 0.9420 - val_loss: 0.2083 - val_accuracy: 0.9220
Epoch 4/100
125/125 [==============================] - 0s 3ms/step - loss: 0.1478 - accuracy: 0.9482 - val_loss: 0.1855 - val_accuracy: 0.9300
Epoch 5/100
125/125 [==============================] - 0s 2ms/step - loss: 0.1284 - accuracy: 0.9542 - val_loss: 0.1712 - val_accuracy: 0.9350
Epoch 6/100
125/125 [==============================] - 0s 3ms/step - loss: 0.1151 - accuracy: 0.9592 - val_loss: 0.1596 - val_accuracy: 0.9370
Epoch 7/100
125/125 [==============================] - 0s 2ms/step - loss: 0.1053 - accuracy: 0.9628 - val_loss: 0.1486 - val_accuracy: 0.9430
Epoch 8/100
125/125 [==============================] - 0s 2ms/step - loss: 0.0976 - accuracy: 0.9635 - val_loss: 0.1419 - val_accuracy: 0.9470
Epoch 9/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0913 - accuracy: 0.9668 - val_loss: 0.1357 - val_accuracy: 0.9480
Epoch 10/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0861 - accuracy: 0.9685 - val_loss: 0.1317 - val_accuracy: 0.9500
Epoch 11/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0818 - accuracy: 0.9705 - val_loss: 0.1264 - val_accuracy: 0.9520
Epoch 12/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0780 - accuracy: 0.9712 - val_loss: 0.1249 - val_accuracy: 0.9570
Epoch 13/100
125/125 [==============================] - 0s 2ms/step - loss: 0.0747 - accuracy: 0.9740 - val_loss: 0.1235 - val_accuracy: 0.9570
Epoch 14/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0715 - accuracy: 0.9750 - val_loss: 0.1171 - val_accuracy: 0.9580
Epoch 15/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0692 - accuracy: 0.9753 - val_loss: 0.1174 - val_accuracy: 0.9590
Epoch 16/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0668 - accuracy: 0.9765 - val_loss: 0.1139 - val_accuracy: 0.9590
Epoch 17/100
125/125 [==============================] - 0s 4ms/step - loss: 0.0649 - accuracy: 0.9770 - val_loss: 0.1114 - val_accuracy: 0.9610
Epoch 18/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0631 - accuracy: 0.9772 - val_loss: 0.1092 - val_accuracy: 0.9610
Epoch 19/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0617 - accuracy: 0.9780 - val_loss: 0.1080 - val_accuracy: 0.9620
Epoch 20/100
125/125 [==============================] - 0s 4ms/step - loss: 0.0601 - accuracy: 0.9795 - val_loss: 0.1096 - val_accuracy: 0.9630
Epoch 21/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0588 - accuracy: 0.9800 - val_loss: 0.1078 - val_accuracy: 0.9630
Epoch 22/100
125/125 [==============================] - 0s 4ms/step - loss: 0.0576 - accuracy: 0.9812 - val_loss: 0.1063 - val_accuracy: 0.9650
Epoch 23/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0563 - accuracy: 0.9820 - val_loss: 0.1039 - val_accuracy: 0.9640
Epoch 24/100
125/125 [==============================] - 1s 4ms/step - loss: 0.0553 - accuracy: 0.9827 - val_loss: 0.1044 - val_accuracy: 0.9650
Epoch 25/100
125/125 [==============================] - 0s 4ms/step - loss: 0.0544 - accuracy: 0.9840 - val_loss: 0.1042 - val_accuracy: 0.9650
Epoch 26/100
125/125 [==============================] - 0s 2ms/step - loss: 0.0535 - accuracy: 0.9830 - val_loss: 0.1032 - val_accuracy: 0.9650
Epoch 27/100
125/125 [==============================] - 0s 3ms/step - loss: 0.0526 - accuracy: 0.9820 - val_loss: 0.1006 - val_accuracy: 0.9650
Epoch 28/100
125/125 [==============================] - 0s 2ms/step - loss: 0.0519 - accuracy: 0.9840 - val_loss: 0.1018 - val_accuracy: 0.9650
Epoch 29/100
125/125 [==============================] - 0s 2ms/step - loss: 0.0510 - accuracy: 0.9847 - val_loss: 0.0993 - val_accuracy: 0.9660
```

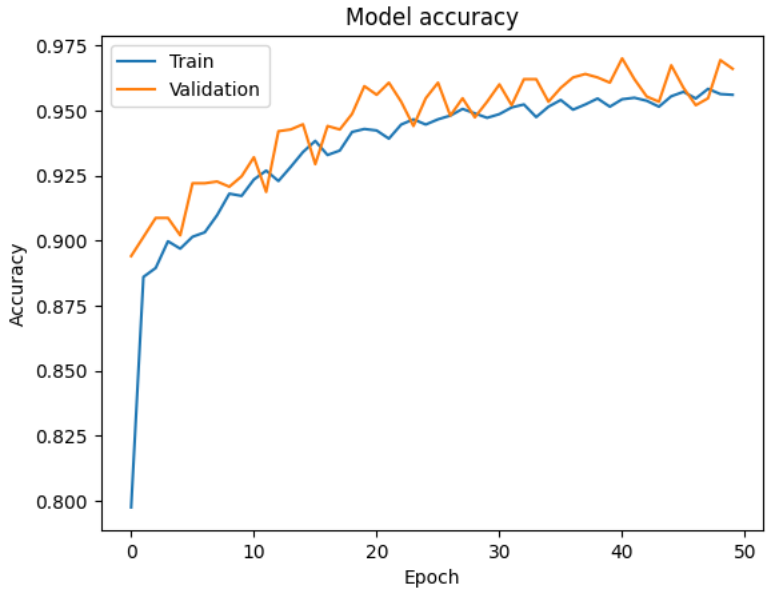
7. Visualize Model Training History in Keras

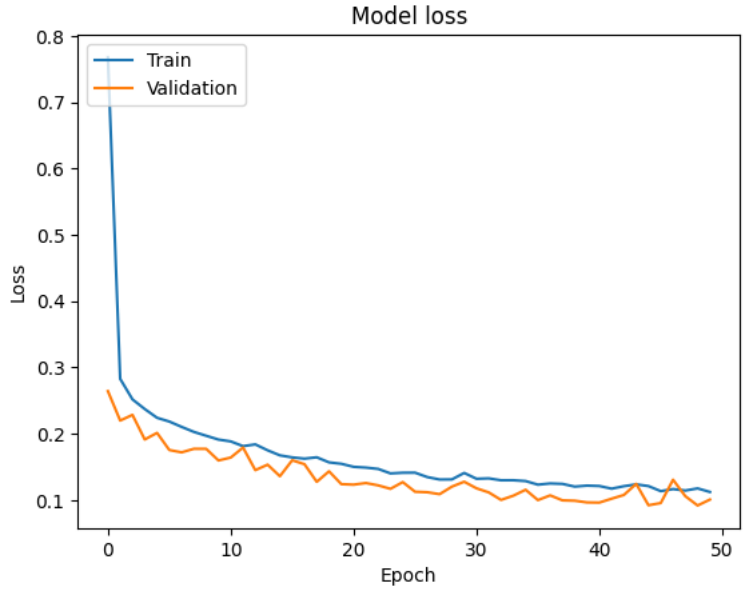
8. Show the application of Dropout Regularization

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split the data into input features and labels
X = data.drop(['ID', 'Personal Loan'], axis=1)
y = data['Personal Loan']

# Build the model
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=10))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Plot the training and validation accuracy over time
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot the training and validation loss over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

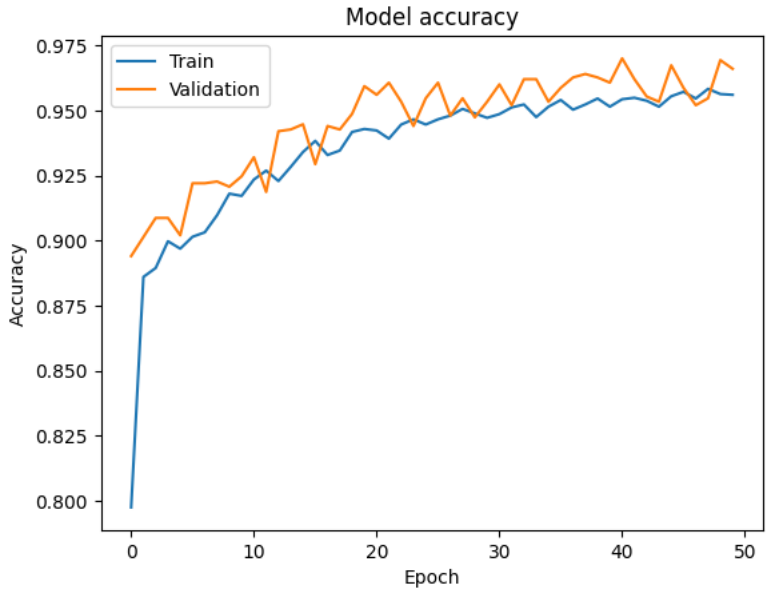
[1]

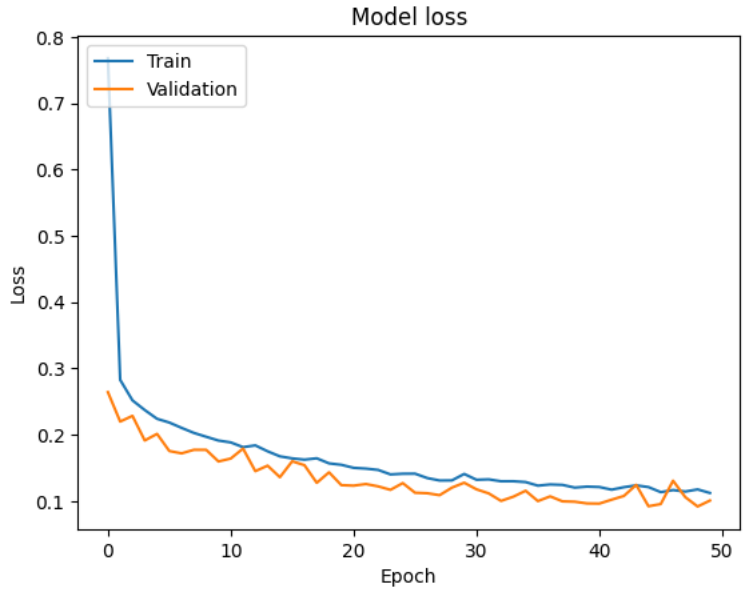




9. Show the application of Dropout on the visible layer

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split the data into input features and labels
X = data.drop(['ID', 'Personal Loan'], axis=1)
y = data['Personal Loan']

# Build the model with Dropout on the visible layer
model = Sequential()
model.add(Dropout(0.2, input_shape=(10,))) # Dropout layer on the input layer with 20% dropout rate
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Plot the training and validation accuracy over time
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot the training and validation loss over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

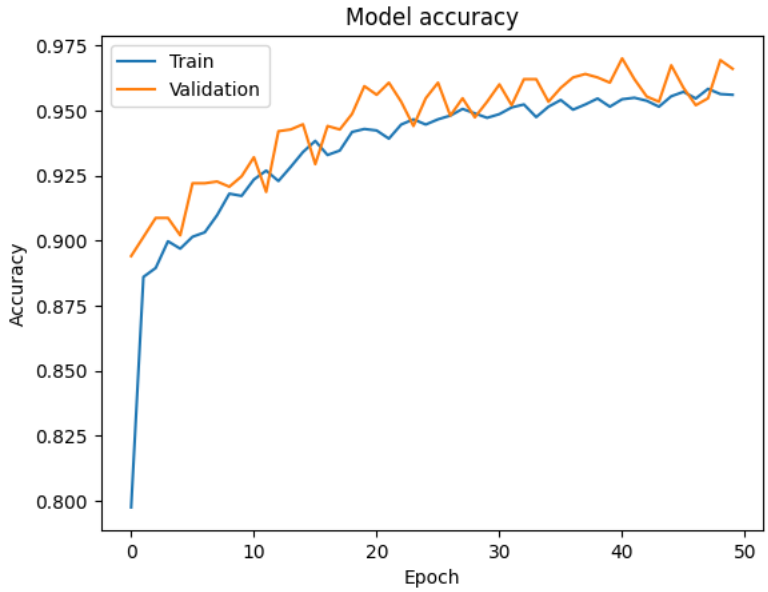


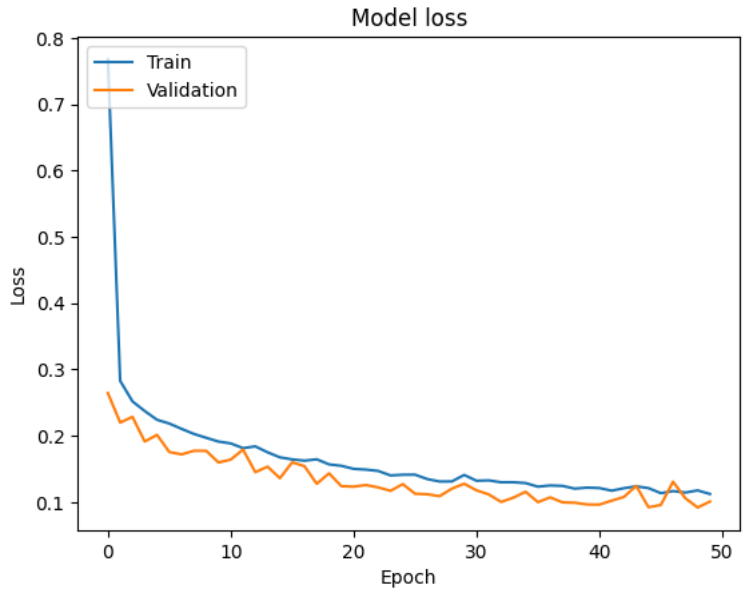


10. Show the application of Dropout on the hidden layer

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split the data into input features and labels
X = data.drop(['ID', 'Personal Loan'], axis=1)
y = data['Personal Loan']

# Build the model with Dropout on the hidden layer
model = Sequential()
model.add(Dense(16, input_shape=(10,), activation='relu'))
model.add(Dropout(0.2)) # Dropout layer on the first hidden layer with 20% dropout rate
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# Plot the training and validation accuracy over time
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot the training and validation loss over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



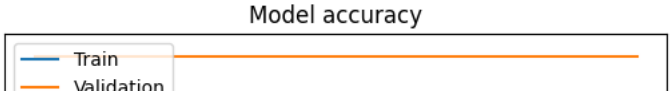


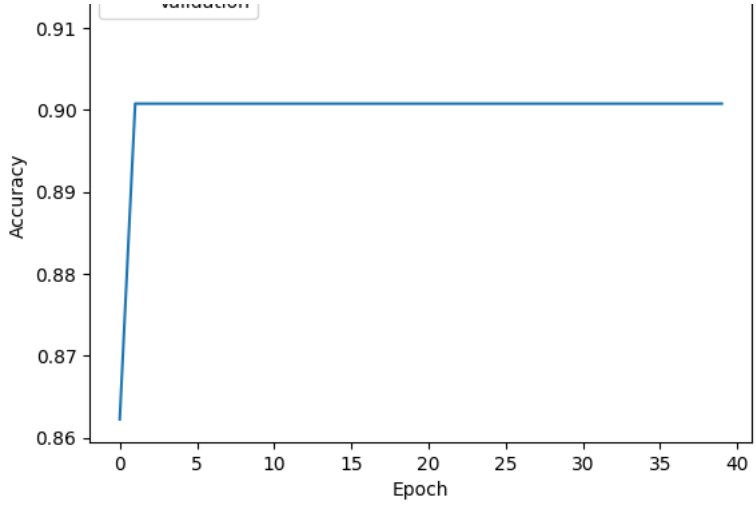
11. Show the application of a time-based learning rate schedule

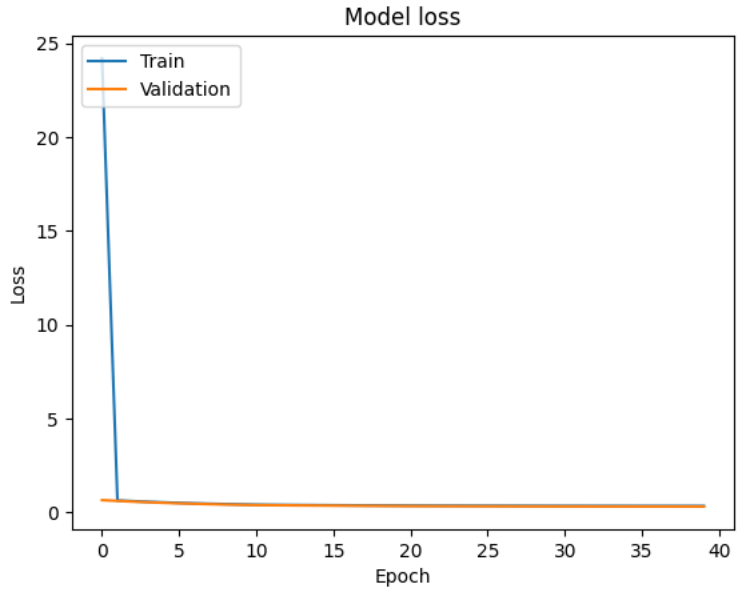
```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.callbacks import LearningRateScheduler
import matplotlib.pyplot as plt
import math

# Load the data
data = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split the data into input features and labels
X = data.drop(['ID', 'Personal Loan'], axis=1)
y = data['Personal Loan']

# Define the learning rate schedule
def lr_schedule(epoch):
    lr = 0.001
    if epoch > 10:
        lr = lr / 2
    if epoch > 20:
        lr = lr / 2
    if epoch > 30:
        lr = lr / 2
    return lr

# Build the model
model = Sequential()
model.add(Dense(16, input_shape=(12,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model with a time-based learning rate schedule
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

# Define the callbacks
lr_scheduler = LearningRateScheduler(lr_schedule)

# Train the model
history = model.fit(X, y, epochs=40, validation_split=0.2, callbacks=[lr_scheduler])

# Plot the training and validation accuracy over time
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot the training and validation loss over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/40
125/125 [==============================] - 5s 9ms/step - loss: 24.2029 - accuracy: 0.8622 - val_loss: 0.6503 - val_accuracy: 0.9170 - lr
Epoch 2/40
125/125 [==============================] - 1s 8ms/step - loss: 0.6319 - accuracy: 0.9007 - val_loss: 0.6085 - val_accuracy: 0.9170 - lr:
Epoch 3/40
125/125 [==============================] - 1s 9ms/step - loss: 0.5936 - accuracy: 0.9007 - val_loss: 0.5704 - val_accuracy: 0.9170 - lr:
Epoch 4/40
125/125 [==============================] - 1s 7ms/step - loss: 0.5592 - accuracy: 0.9007 - val_loss: 0.5361 - val_accuracy: 0.9170 - lr:
Epoch 5/40
125/125 [==============================] - 1s 7ms/step - loss: 0.5286 - accuracy: 0.9007 - val_loss: 0.5057 - val_accuracy: 0.9170 - lr:
Epoch 6/40
125/125 [==============================] - 1s 9ms/step - loss: 0.5015 - accuracy: 0.9007 - val_loss: 0.4787 - val_accuracy: 0.9170 - lr:
Epoch 7/40
125/125 [==============================] - 0s 3ms/step - loss: 0.4777 - accuracy: 0.9007 - val_loss: 0.4550 - val_accuracy: 0.9170 - lr:
Epoch 8/40
125/125 [==============================] - 0s 3ms/step - loss: 0.4568 - accuracy: 0.9007 - val_loss: 0.4338 - val_accuracy: 0.9170 - lr:
Epoch 9/40
125/125 [==============================] - 0s 3ms/step - loss: 0.4384 - accuracy: 0.9007 - val_loss: 0.4154 - val_accuracy: 0.9170 - lr:
Epoch 10/40
125/125 [==============================] - 0s 3ms/step - loss: 0.4224 - accuracy: 0.9007 - val_loss: 0.3990 - val_accuracy: 0.9170 - lr:
Epoch 11/40
125/125 [==============================] - 0s 3ms/step - loss: 0.4084 - accuracy: 0.9007 - val_loss: 0.3847 - val_accuracy: 0.9170 - lr:
Epoch 12/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3990 - accuracy: 0.9007 - val_loss: 0.3781 - val_accuracy: 0.9170 - lr:
Epoch 13/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3933 - accuracy: 0.9007 - val_loss: 0.3719 - val_accuracy: 0.9170 - lr:
Epoch 14/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3878 - accuracy: 0.9007 - val_loss: 0.3659 - val_accuracy: 0.9170 - lr:
Epoch 15/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3826 - accuracy: 0.9007 - val_loss: 0.3603 - val_accuracy: 0.9170 - lr:
Epoch 16/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3777 - accuracy: 0.9007 - val_loss: 0.3549 - val_accuracy: 0.9170 - lr:
Epoch 17/40
125/125 [==============================] - 1s 5ms/step - loss: 0.3731 - accuracy: 0.9007 - val_loss: 0.3499 - val_accuracy: 0.9170 - lr:
Epoch 18/40
125/125 [==============================] - 1s 5ms/step - loss: 0.3688 - accuracy: 0.9007 - val_loss: 0.3451 - val_accuracy: 0.9170 - lr:
Epoch 19/40
125/125 [==============================] - 1s 5ms/step - loss: 0.3648 - accuracy: 0.9007 - val_loss: 0.3407 - val_accuracy: 0.9170 - lr:
Epoch 20/40
125/125 [==============================] - 1s 4ms/step - loss: 0.3610 - accuracy: 0.9007 - val_loss: 0.3364 - val_accuracy: 0.9170 - lr:
Epoch 21/40
125/125 [==============================] - 1s 5ms/step - loss: 0.3575 - accuracy: 0.9007 - val_loss: 0.3325 - val_accuracy: 0.9170 - lr:
Epoch 22/40
125/125 [==============================] - 1s 4ms/step - loss: 0.3551 - accuracy: 0.9007 - val_loss: 0.3306 - val_accuracy: 0.9170 - lr:
Epoch 23/40
125/125 [==============================] - 1s 5ms/step - loss: 0.3535 - accuracy: 0.9007 - val_loss: 0.3288 - val_accuracy: 0.9170 - lr:
Epoch 24/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3519 - accuracy: 0.9007 - val_loss: 0.3270 - val_accuracy: 0.9170 - lr:
Epoch 25/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3504 - accuracy: 0.9007 - val_loss: 0.3252 - val_accuracy: 0.9170 - lr:
Epoch 26/40
125/125 [==============================] - 0s 2ms/step - loss: 0.3490 - accuracy: 0.9007 - val_loss: 0.3235 - val_accuracy: 0.9170 - lr:
Epoch 27/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3475 - accuracy: 0.9007 - val_loss: 0.3218 - val_accuracy: 0.9170 - lr:
Epoch 28/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3462 - accuracy: 0.9007 - val_loss: 0.3201 - val_accuracy: 0.9170 - lr:
Epoch 29/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3448 - accuracy: 0.9007 - val_loss: 0.3185 - val_accuracy: 0.9170 - lr:
Epoch 30/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3436 - accuracy: 0.9007 - val_loss: 0.3169 - val_accuracy: 0.9170 - lr:
Epoch 31/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3423 - accuracy: 0.9007 - val_loss: 0.3154 - val_accuracy: 0.9170 - lr:
Epoch 32/40
125/125 [==============================] - 0s 2ms/step - loss: 0.3414 - accuracy: 0.9007 - val_loss: 0.3147 - val_accuracy: 0.9170 - lr:
Epoch 33/40
125/125 [==============================] - 0s 2ms/step - loss: 0.3408 - accuracy: 0.9007 - val_loss: 0.3140 - val_accuracy: 0.9170 - lr:
Epoch 34/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3402 - accuracy: 0.9007 - val_loss: 0.3133 - val_accuracy: 0.9170 - lr:
Epoch 35/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3397 - accuracy: 0.9007 - val_loss: 0.3126 - val_accuracy: 0.9170 - lr:
Epoch 36/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3391 - accuracy: 0.9007 - val_loss: 0.3118 - val_accuracy: 0.9170 - lr:
Epoch 37/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3385 - accuracy: 0.9007 - val_loss: 0.3111 - val_accuracy: 0.9170 - lr:
Epoch 38/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3380 - accuracy: 0.9007 - val_loss: 0.3104 - val_accuracy: 0.9170 - lr:
Epoch 39/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3374 - accuracy: 0.9007 - val_loss: 0.3098 - val_accuracy: 0.9170 - lr:
Epoch 40/40
125/125 [==============================] - 0s 3ms/step - loss: 0.3369 - accuracy: 0.9007 - val_loss: 0.3091 - val_accuracy: 0.9170 - lr:
```







12. Show the application of a drop-based learning rate schedule

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler
import matplotlib.pyplot as plt
import math

# Load the data
data = pd.read_csv("/content/drive/MyDrive/3rdYear/CPE019/hoa8.1/Bank_Personal_Loan_Modelling.csv")

# Split the data into input features and labels
X = data.drop(['ID', 'Personal Loan'], axis=1)
y = data['Personal Loan']

# Define the learning rate schedule
def lr_schedule(epoch):
    lr = 0.001
    drop = 0.5
    epochs_drop = 5
    if epoch > 10:
        lr = lr * math.pow(drop, math.floor((1+epoch-10)/epochs_drop))
    return lr

# Build the model
model = Sequential()
model.add(Dense(16, input_shape=(12,), activation='relu'))
model.add(Dense(8, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))

# Compile the model with a drop-based learning rate schedule
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

# Define the callbacks
lr_scheduler = LearningRateScheduler(lr_schedule)

# Train the model
history = model.fit(X, y, epochs=40, validation_split=0.2, callbacks=[lr_scheduler])

# Plot the training and validation accuracy over time
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot the training and validation loss over time
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/40
125/125 [==============================] - 3s 4ms/step - loss: 3196.9702 - accuracy: 0.5458 - val_loss: 45.8032 - val_accuracy: 0.9170 -
Epoch 2/40
125/125 [==============================] - 0s 3ms/step - loss: 8.0490 - accuracy: 0.8420 - val_loss: 1.2898 - val_accuracy: 0.8970 - lr:
Epoch 3/40
125/125 [==============================] - 0s 3ms/step - loss: 2.5605 - accuracy: 0.8535 - val_loss: 1.5877 - val_accuracy: 0.9170 - lr:
Epoch 4/40
125/125 [==============================] - 0s 3ms/step - loss: 1.3680 - accuracy: 0.8700 - val_loss: 1.9138 - val_accuracy: 0.9170 - lr:
Epoch 5/40
125/125 [==============================] - 0s 3ms/step - loss: 1.7831 - accuracy: 0.8673 - val_loss: 0.6100 - val_accuracy: 0.8980 - lr:
Epoch 6/40
125/125 [==============================] - 0s 3ms/step - loss: 4.4124 - accuracy: 0.8475 - val_loss: 0.8029 - val_accuracy: 0.9090 - lr:
Epoch 7/40
125/125 [==============================] - 0s 3ms/step - loss: 2.5519 - accuracy: 0.8637 - val_loss: 3.5749 - val_accuracy: 0.9170 - lr:
Epoch 8/40
125/125 [==============================] - 0s 3ms/step - loss: 5.5449 - accuracy: 0.8397 - val_loss: 0.9763 - val_accuracy: 0.8780 - lr:
Epoch 9/40
125/125 [==============================] - 0s 2ms/step - loss: 6.9446 - accuracy: 0.8338 - val_loss: 8.2487 - val_accuracy: 0.9170 - lr:
Epoch 10/40
125/125 [==============================] - 1s 4ms/step - loss: 3.4636 - accuracy: 0.8668 - val_loss: 1.1397 - val_accuracy: 0.9120 - lr:
Epoch 11/40
125/125 [==============================] - 1s 5ms/step - loss: 4.8479 - accuracy: 0.8602 - val_loss: 0.9943 - val_accuracy: 0.8870 - lr:
Epoch 12/40
125/125 [==============================] - 1s 4ms/step - loss: 3.8565 - accuracy: 0.8590 - val_loss: 2.0855 - val_accuracy: 0.9190 - lr:
Epoch 13/40
125/125 [==============================] - 1s 5ms/step - loss: 2.3516 - accuracy: 0.8773 - val_loss: 0.8393 - val_accuracy: 0.8950 - lr:
Epoch 14/40
125/125 [==============================] - 1s 5ms/step - loss: 3.4542 - accuracy: 0.8565 - val_loss: 1.7404 - val_accuracy: 0.9190 - lr:
Epoch 15/40
125/125 [==============================] - 1s 4ms/step - loss: 1.6706 - accuracy: 0.8838 - val_loss: 2.8684 - val_accuracy: 0.9170 - lr:
Epoch 16/40
125/125 [==============================] - 0s 4ms/step - loss: 2.0516 - accuracy: 0.8792 - val_loss: 0.9111 - val_accuracy: 0.8920 - lr:
Epoch 17/40
125/125 [==============================] - 1s 5ms/step - loss: 1.2722 - accuracy: 0.8860 - val_loss: 1.1221 - val_accuracy: 0.9140 - lr:
Epoch 18/40
125/125 [==============================] - 0s 3ms/step - loss: 1.1730 - accuracy: 0.8900 - val_loss: 1.7495 - val_accuracy: 0.9170 - lr:
Epoch 19/40
125/125 [==============================] - 0s 3ms/step - loss: 1.5972 - accuracy: 0.8745 - val_loss: 0.8525 - val_accuracy: 0.8910 - lr:
Epoch 20/40
125/125 [==============================] - 0s 3ms/step - loss: 1.3755 - accuracy: 0.8820 - val_loss: 1.0138 - val_accuracy: 0.9100 - lr:
Epoch 21/40
125/125 [==============================] - 0s 3ms/step - loss: 1.0015 - accuracy: 0.8972 - val_loss: 0.8071 - val_accuracy: 0.8910 - lr:
Epoch 22/40
125/125 [==============================] - 0s 3ms/step - loss: 1.3950 - accuracy: 0.8808 - val_loss: 0.7500 - val_accuracy: 0.8990 - lr:
Epoch 23/40
125/125 [==============================] - 0s 3ms/step - loss: 1.1030 - accuracy: 0.8867 - val_loss: 0.7190 - val_accuracy: 0.9000 - lr:
Epoch 24/40
125/125 [==============================] - 0s 3ms/step - loss: 1.0250 - accuracy: 0.8930 - val_loss: 0.6961 - val_accuracy: 0.9050 - lr:
Epoch 25/40
125/125 [==============================] - 0s 3ms/step - loss: 0.9000 - accuracy: 0.8938 - val_loss: 0.7401 - val_accuracy: 0.9000 - lr:
Epoch 26/40
```