Name: Santiago, John Loyd C. Santiago
Course and Section: CPE 019 - CPE32S3
Date of Submission: April 19, 2024
Instructor: Engr. Roman Richard

1.Choose any dataset applicable to either a classification problem or a regression problem.

link of the google colab: https://colab.research.google.com/drive/10xpEaqUR58LoJKK06yP6W10_N3nO55ts?usp=sharing

link of the dataset: https://archive.ics.uci.edu/dataset/850/raisin

link of the google drive: https://drive.google.com/drive/folders/1kFHEHuDS9-wMfvQZ8ymkUTDByFjRuExo?usp=sharing

Instructions:

1. Choose any dataset applicable to either a classification problem or a regression problem.
2. Explain your datasets and the problem being addressed.
3. Show evidence that you can do the following:

- Save a model in HDF5 format
- Save a model and load the model in a JSON format
- Save a model and load the model in a YAML format
- Checkpoint Neural Network Model Improvements
- Checkpoint Best Neural Network Model only
- Load a saved Neural Network model
- Visualize Model Training History in Keras
- Show the application of Dropout Regularization
- Show the application of Dropout on the visible layer
- Show the application of Dropout on the hidden layer
- Show the application of a time-based learning rate schedule
- Show the application of a drop-based learning rate schedule

4. Submit the link to your Google Colab (make sure that it is accessible to me)

2.Explain your datasets and the problem being addressed.

My dataset is about two kinds of raisin kecimen and besni this two kinds of raisin is from turkey. the dataset have a total of 900 data from the 2 kinds of raisin the kecimen and besni the dataset also have 7 features Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Extent and Perimeter.

3.Show evidence that you can do the following:

Save a model in HDF5 format

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
pip install h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (3.9.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from h5py) (1.23.5)
```

```
!pip install tensorflow==2.12.0
```

```
Requirement already satisfied: tensorflow==2.12.0 in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (24.3.25)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.62.1)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (3.9.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.4.26)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (18.1.1)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (2.12.3)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (2.12.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (4.11.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.36.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.12.0) (0.43.0)
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow==2.12.0) (0.2.0)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow==2.12.0) (1.11.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.0.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.2.2)
```

I install the 2.12.0 version of the tensorflow because the 2.15.0 version isn't working for the model yaml

```
import numpy as np
import pandas as pd
import os
from tensorflow.keras.models import Sequential, model_from_json, model_from_yaml
from sklearn.preprocessing import StandardScaler
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split
```

```
dataset = "/content/Raisin_Dataset.xlsx"
R = pd.read_excel(dataset)
```

```
R.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             900 non-null    int64
 1   MajorAxisLength  900 non-null    float64
 2   MinorAxisLength  900 non-null    float64
 3   Eccentricity     900 non-null    float64
 4   ConvexArea       900 non-null    int64
 5   Extent           900 non-null    float64
 6   Perimeter        900 non-null    float64
 7   Class            900 non-null    object
dtypes: float64(5), int64(2), object(1)
memory usage: 56.4+ KB
```

```
R.dtypes
```

```
Area                 int64
MajorAxisLength    float64
MinorAxisLength    float64
Eccentricity       float64
ConvexArea           int64
Extent             float64
Perimeter          float64
Class               object
dtype: object
```

```
R.head(1000)
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.040	Kecimen
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.786	Kecimen
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.575	Kecimen
3	45928	286.540559	208.760042	0.684989	47336	0.699599	844.162	Kecimen
4	79408	352.190770	290.827533	0.564011	81463	0.792772	1073.251	Kecimen
...	...	...	...	...	...	...	...	...
895	83248	430.077308	247.838695	0.817263	85839	0.668793	1129.072	Besni
896	87350	440.735698	259.293149	0.808629	90899	0.636476	1214.252	Besni
897	99657	431.706981	298.837323	0.721684	106264	0.741099	1292.828	Besni
898	93523	476.344094	254.176054	0.845739	97653	0.658798	1258.548	Besni
899	85609	512.081774	215.271976	0.907345	89197	0.632020	1272.862	Besni

900 rows × 8 columns

Next steps: View recommended plots

```
R.describe()
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter
count	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000
mean	87804.127778	430.929950	254.488133	0.781542	91186.090000	0.699508	1165.906636
std	39002.111390	116.035121	49.988902	0.090318	40769.290132	0.053468	273.764315
min	25387.000000	225.629541	143.710872	0.348730	26139.000000	0.379856	619.074000
25%	59348.000000	345.442898	219.111126	0.741766	61513.250000	0.670869	966.410750
50%	78902.000000	407.803951	247.848409	0.798846	81651.000000	0.707367	1119.509000
75%	105028.250000	494.187014	279.888575	0.842571	108375.750000	0.734991	1308.389750
max	235047.000000	997.291941	492.275279	0.962124	278217.000000	0.835455	2697.753000

- 0 = Kecimen
- 1 = Besni

```
R['Class'] = R['Class'].apply(lambda toLabel: 0 if toLabel == 'Kecimen' else 1)
```

```
count = R['Class'].value_counts()
print(count)
```

```
Class
0    450
1    450
Name: count, dtype: int64
```

```
np.random.seed(7)
```

```
x = R.iloc[:, :-1].values
y = R['Class'].values
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
normalizer = StandardScaler()
x_train_n = normalizer.fit_transform(x_train)
x_test_n = normalizer.transform(x_test)
```

Creating the model for JSON

```
model = Sequential()
model.add(Dense(16, input_dim=7, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

Save a model and load the model in a JSON format

Save a model in HDF5 format

```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(x_train_n, y_train, epochs=150, batch_size=10, verbose=0)
# evaluate the model
scores = model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("/content/drive/MyDrive/Assignment 8.1 : Saving Models/model.json/model.weights.h5")
print("Saved model to disk")
```

```
accuracy: 89.58%
Saved model to disk
```

```
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("/content/drive/MyDrive/Assignment 8.1 : Saving Models/model.json/model.weights.h5")
print("Loaded model from disk")


# evaluate loaded model on test data
loaded_model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
score = loaded_model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 89.58%
```

Creating the model for YAML

Save a model and load the model in a YAML format

Save a model in HDF5 format

```
    model = Sequential()
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
```

```
model.fit(x_train_n, y_train, epochs=150, batch_size=10, verbose=0)
# evaluate the model
scores = model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# serialize model to YAML
model_yaml = model.to_json()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)
# serialize weights to HDF5
model.save_weights("/content/drive/MyDrive/Assignment 8.1 : Saving Models/model_yaml.h5")
print("Saved model to disk")
```

```
accuracy: 89.58%
Saved model to disk
```

```
# load YAML and create model
yaml_file = open('model.yaml', 'r')
loaded_model_yaml = yaml_file.read()
yaml_file.close()
loaded_model = model_from_json(loaded_model_yaml)
# load weights into new model
loaded_model.load_weights("/content/drive/MyDrive/Assignment 8.1 : Saving Models/model_yaml.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
score = loaded_model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 89.58%
```

Save a Keras Model

```
from numpy import loadtxt
# create model
model = Sequential()
model.add(Dense(16, input_dim=7, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(x_train_n, y_train, epochs=150, batch_size=10, verbose=0)
# evaluate the model
scores = model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# save model and architecture to single file
model.save("/content/drive/MyDrive/Assignment 8.1 : Saving Models/model.h5")
print("Saved model to disk")
```

```
accuracy: 88.06%
Saved model to disk
```

Checkpoint Neural Network Model Improvements

```
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
# create model
model = Sequential()
model.add(Dense(16, input_dim=7, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# checkpoint
filepath="/content/drive/MyDrive/Assignment 8.1 : Saving Models/weights-improvement-{epoch:02d}-{val_accuracy:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

# Fit the model
model.fit(x_train_n, y_train, validation_split=0.33, epochs=150, batch_size=10, callbacks=callbacks_list, verbose=0)
```

```
Epoch 122: val_accuracy did not improve from 0.88235

Epoch 123: val_accuracy did not improve from 0.88235

Epoch 124: val_accuracy did not improve from 0.88235

Epoch 125: val_accuracy did not improve from 0.88235

Epoch 126: val_accuracy did not improve from 0.88235

Epoch 127: val_accuracy did not improve from 0.88235

Epoch 128: val_accuracy did not improve from 0.88235

Epoch 129: val_accuracy did not improve from 0.88235

Epoch 130: val_accuracy did not improve from 0.88235

Epoch 131: val_accuracy did not improve from 0.88235

Epoch 132: val_accuracy did not improve from 0.88235

Epoch 133: val_accuracy did not improve from 0.88235

Epoch 134: val_accuracy did not improve from 0.88235

Epoch 135: val_accuracy did not improve from 0.88235

Epoch 136: val_accuracy did not improve from 0.88235

Epoch 137: val_accuracy did not improve from 0.88235

Epoch 138: val_accuracy did not improve from 0.88235

Epoch 139: val_accuracy did not improve from 0.88235

Epoch 140: val_accuracy did not improve from 0.88235

Epoch 141: val_accuracy did not improve from 0.88235

Epoch 142: val_accuracy did not improve from 0.88235

Epoch 143: val_accuracy did not improve from 0.88235

Epoch 144: val_accuracy did not improve from 0.88235

Epoch 145: val_accuracy did not improve from 0.88235

Epoch 146: val_accuracy did not improve from 0.88235

Epoch 147: val_accuracy did not improve from 0.88235

Epoch 148: val_accuracy did not improve from 0.88235

Epoch 149: val_accuracy did not improve from 0.88235

Epoch 150: val_accuracy did not improve from 0.88235
<keras.callbacks.History at 0x7c77f7373be0>
```

Checkpoint Best Neural Network Model only

```
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
```

```
# create model
model = Sequential()
model.add(Dense(16, input_dim=7, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# checkpoint
filepath="/content/drive/MyDrive/Assignment 8.1 : Saving Models/weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
# Fit the model
model.fit(x_train_n, y_train, validation_split=0.33, epochs=150, batch_size=10, callbacks=callbacks_list, verbose=0)
```

```
Epoch 134: val_accuracy did not improve from 0.88235

Epoch 135: val_accuracy did not improve from 0.88235

Epoch 136: val_accuracy did not improve from 0.88235

Epoch 137: val_accuracy did not improve from 0.88235

Epoch 138: val_accuracy did not improve from 0.88235

Epoch 139: val_accuracy did not improve from 0.88235

Epoch 140: val_accuracy did not improve from 0.88235

Epoch 141: val_accuracy did not improve from 0.88235

Epoch 142: val_accuracy did not improve from 0.88235

Epoch 143: val_accuracy did not improve from 0.88235

Epoch 144: val_accuracy did not improve from 0.88235

Epoch 145: val_accuracy did not improve from 0.88235

Epoch 146: val_accuracy did not improve from 0.88235

Epoch 147: val_accuracy did not improve from 0.88235

Epoch 148: val_accuracy did not improve from 0.88235

Epoch 149: val_accuracy did not improve from 0.88235

Epoch 150: val_accuracy did not improve from 0.88235
<keras.callbacks.History at 0x7c77f716e0e0>
```

Load a saved Neural Network model

```
# create model
model = Sequential()
model.add(Dense(16, input_dim=7, kernel_initializer = 'uniform' , activation= 'relu' ))
model.add(Dense(8, kernel_initializer= 'uniform' , activation= 'relu' ))
model.add(Dense(1, kernel_initializer= 'uniform' , activation= 'sigmoid' ))
# load weights
model.load_weights("/content/drive/MyDrive/Assignment 8.1 : Saving Models/weights.best.hdf5")
# Compile model (required to make predictions)
model.compile(loss= 'binary_crossentropy' , optimizer= 'adam' , metrics=[ 'accuracy' ])
print("Created model and loaded weights from file")
```

```
Created model and loaded weights from file
```

```
# estimate accuracy on whole dataset using loaded weights
scores = model.evaluate(x_train_n, y_train, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

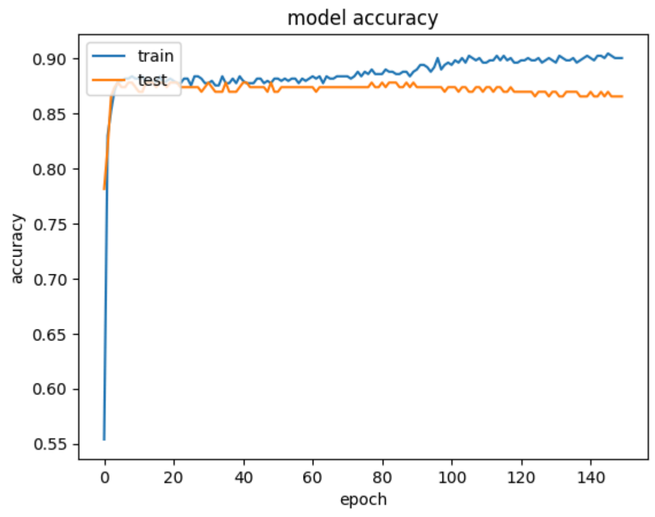
```
accuracy: 87.78%
```

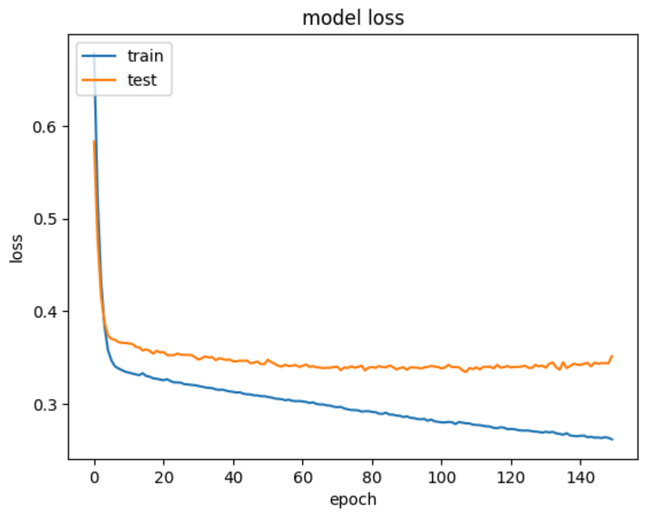
Visualize Model Training History in Keras

```
# create model
model = Sequential()
model.add(Dense(16, input_dim=7, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
history = model.fit(x_train_n, y_train, validation_split=0.33, epochs=150, batch_size=10, verbose=0)
```

```
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





Show the application of Dropout Regularization

```
!pip install scikeras
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.13.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (3.2.1)
Requirement already satisfied: scikit-learn>=1.4.2 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.4.2)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (1.23.5)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (13.7.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (3.9.0)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.11.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.2.0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (3.4.0)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras>=3.2.0->scikeras) (4.11.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras) (2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->scikeras) (0.1.2)
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from scikeras.wrappers import KerasClassifier
```

```
# baseline
def create_baseline():
    # create model
  model = Sequential()
  model.add(Dense(16, input_dim=7, activation='relu'))
  model.add(Dense(8, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
    # Compile model
  sgd = SGD(learning_rate=0.01, momentum=0.8)
  model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
  return model

estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(model=create_baseline, epochs=300, batch_size=16, verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x79c352abf7f0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.fun
WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x79c352abf7f0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.fun
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Baseline: 87.92% (2.85%)
```

Show the application of Dropout on the visible layer

```
from keras.layers import Dropout
from keras.constraints import MaxNorm
from keras.optimizers import SGD
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
# dropout in the input layer with weight constraint
def create_model():
    # create model
    model = Sequential()
    model.add(Dropout(0.2, input_shape=(7,)))
    model.add(Dense(60, activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dense(30, activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dense(1, activation='sigmoid'))
  # Compile model
    sgd = SGD(learning_rate=0.1, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model
```

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(model=create_model, epochs=300, batch_size=16, verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold)
print("Visible: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropout.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Visible: 83.61% (3.50%)
```

Show the application of Dropout on the hidden layer

```
from keras.layers import Dense
from keras.layers import Dropout
from keras.constraints import MaxNorm
from keras.optimizers import SGD
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
# dropout in hidden layers with weight constraint
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(60, input_shape=(7,), activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(30, activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    sgd = SGD(learning_rate=0.1, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model
```

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(model=create_model, epochs=300, batch_size=16, verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold)
print("Visible: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:86: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Visible: 85.97% (4.58%)
```

Show the application of a time-based learning rate schedule

```
# Time Based Learning Rate Decay
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
# tf.keras.optimizers.legacy.SGD
from keras.optimizers.legacy import SGD
from sklearn.preprocessing import LabelEncoder
```

```
# create model
model = Sequential()
model.add(Dense(34, input_shape=(7,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
epochs = 50
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(learning_rate=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
# Fit the model
model.fit(x_train_n, y_train, validation_split=0.33, epochs=epochs, batch_size=28, verbose=2)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:34: UserWarning: Argument `decay` is no longer supported and will be ignored.
  warnings.warn(
18/18 - 1s - 69ms/step - accuracy: 0.7448 - loss: 0.4889 - val_accuracy: 0.8697 - val_loss: 0.3907
Epoch 2/50
18/18 - 0s - 21ms/step - accuracy: 0.8734 - loss: 0.3504 - val_accuracy: 0.8739 - val_loss: 0.3773
Epoch 3/50
18/18 - 0s - 7ms/step - accuracy: 0.8776 - loss: 0.3448 - val_accuracy: 0.8697 - val_loss: 0.3785
Epoch 4/50
18/18 - 0s - 8ms/step - accuracy: 0.8817 - loss: 0.3422 - val_accuracy: 0.8697 - val_loss: 0.3758
Epoch 5/50
18/18 - 0s - 16ms/step - accuracy: 0.8755 - loss: 0.3426 - val_accuracy: 0.8697 - val_loss: 0.3709
Epoch 6/50
18/18 - 0s - 8ms/step - accuracy: 0.8797 - loss: 0.3374 - val_accuracy: 0.8782 - val_loss: 0.3762
Epoch 7/50
18/18 - 0s - 7ms/step - accuracy: 0.8693 - loss: 0.3386 - val_accuracy: 0.8697 - val_loss: 0.3695
Epoch 8/50
18/18 - 0s - 8ms/step - accuracy: 0.8672 - loss: 0.3388 - val_accuracy: 0.8739 - val_loss: 0.3715
Epoch 9/50
18/18 - 0s - 7ms/step - accuracy: 0.8714 - loss: 0.3424 - val_accuracy: 0.8739 - val_loss: 0.3686
Epoch 10/50
18/18 - 0s - 8ms/step - accuracy: 0.8734 - loss: 0.3368 - val_accuracy: 0.8782 - val_loss: 0.3659
Epoch 11/50
18/18 - 0s - 17ms/step - accuracy: 0.8672 - loss: 0.3353 - val_accuracy: 0.8655 - val_loss: 0.4043
Epoch 12/50
18/18 - 0s - 14ms/step - accuracy: 0.8693 - loss: 0.3377 - val_accuracy: 0.8824 - val_loss: 0.3688
Epoch 13/50
18/18 - 0s - 8ms/step - accuracy: 0.8734 - loss: 0.3353 - val_accuracy: 0.8697 - val_loss: 0.3704
Epoch 14/50
18/18 - 0s - 8ms/step - accuracy: 0.8651 - loss: 0.3342 - val_accuracy: 0.8739 - val_loss: 0.3806
Epoch 15/50
18/18 - 0s - 8ms/step - accuracy: 0.8714 - loss: 0.3336 - val_accuracy: 0.8782 - val_loss: 0.3618
Epoch 16/50
18/18 - 0s - 8ms/step - accuracy: 0.8693 - loss: 0.3309 - val_accuracy: 0.8697 - val_loss: 0.3755
Epoch 17/50
18/18 - 0s - 5ms/step - accuracy: 0.8714 - loss: 0.3347 - val_accuracy: 0.8782 - val_loss: 0.3604
Epoch 18/50
18/18 - 0s - 5ms/step - accuracy: 0.8714 - loss: 0.3326 - val_accuracy: 0.8739 - val_loss: 0.3666
Epoch 19/50
18/18 - 0s - 5ms/step - accuracy: 0.8693 - loss: 0.3375 - val_accuracy: 0.8739 - val_loss: 0.3516
Epoch 20/50
18/18 - 0s - 5ms/step - accuracy: 0.8672 - loss: 0.3369 - val_accuracy: 0.8782 - val_loss: 0.3530
Epoch 21/50
18/18 - 0s - 5ms/step - accuracy: 0.8631 - loss: 0.3375 - val_accuracy: 0.8655 - val_loss: 0.3738
Epoch 22/50
18/18 - 0s - 7ms/step - accuracy: 0.8651 - loss: 0.3331 - val_accuracy: 0.8697 - val_loss: 0.3615
Epoch 23/50
18/18 - 0s - 5ms/step - accuracy: 0.8734 - loss: 0.3286 - val_accuracy: 0.8782 - val_loss: 0.3560
Epoch 24/50
18/18 - 0s - 9ms/step - accuracy: 0.8755 - loss: 0.3263 - val_accuracy: 0.8782 - val_loss: 0.3524
Epoch 25/50
18/18 - 0s - 5ms/step - accuracy: 0.8693 - loss: 0.3273 - val_accuracy: 0.8655 - val_loss: 0.3654
Epoch 26/50
18/18 - 0s - 6ms/step - accuracy: 0.8755 - loss: 0.3265 - val_accuracy: 0.8655 - val_loss: 0.3614
Epoch 27/50
18/18 - 0s - 5ms/step - accuracy: 0.8734 - loss: 0.3343 - val_accuracy: 0.8739 - val_loss: 0.3489
Epoch 28/50
18/18 - 0s - 8ms/step - accuracy: 0.8797 - loss: 0.3251 - val_accuracy: 0.8739 - val_loss: 0.3532
```

Show the application of a drop-based learning rate schedule

```
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from sklearn.preprocessing import LabelEncoder
from keras.callbacks import LearningRateScheduler

# learning rate schedule
def step_decay(epoch):
    initial_lrate = 0.1
    drop = 0.5
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate
```

```
# create model
model = Sequential()
model.add(Dense(34, input_shape=(7,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
sgd = SGD(learning_rate=0.0, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
# learning schedule callback
lrate = LearningRateScheduler(step_decay)
callbacks_list = [lrate]
# Fit the model
model.fit(x_train_n, y_train, validation_split=0.33, epochs=50, batch_size=28, callbacks=callbacks_list, verbose=2)
```

```
Epoch 1/50
18/18 - 1s - 75ms/step - accuracy: 0.8299 - loss: 0.4394 - val_accuracy: 0.8613 - val_loss: 0.4112 - learning_rate: 0.1000
Epoch 2/50
18/18 - 0s - 10ms/step - accuracy: 0.8693 - loss: 0.3833 - val_accuracy: 0.8403 - val_loss: 0.3733 - learning_rate: 0.1000
Epoch 3/50
18/18 - 0s - 8ms/step - accuracy: 0.8237 - loss: 0.3935 - val_accuracy: 0.8655 - val_loss: 0.4012 - learning_rate: 0.1000
Epoch 4/50
18/18 - 0s - 6ms/step - accuracy: 0.8714 - loss: 0.3559 - val_accuracy: 0.8655 - val_loss: 0.3953 - learning_rate: 0.1000
Epoch 5/50
18/18 - 0s - 7ms/step - accuracy: 0.8589 - loss: 0.3564 - val_accuracy: 0.8739 - val_loss: 0.3812 - learning_rate: 0.1000
Epoch 6/50
18/18 - 0s - 7ms/step - accuracy: 0.8672 - loss: 0.3465 - val_accuracy: 0.8739 - val_loss: 0.3543 - learning_rate: 0.1000
Epoch 7/50
18/18 - 0s - 5ms/step - accuracy: 0.8693 - loss: 0.3545 - val_accuracy: 0.8697 - val_loss: 0.3563 - learning_rate: 0.1000
Epoch 8/50
18/18 - 0s - 5ms/step - accuracy: 0.8693 - loss: 0.3442 - val_accuracy: 0.8361 - val_loss: 0.4383 - learning_rate: 0.1000
Epoch 9/50
18/18 - 0s - 8ms/step - accuracy: 0.8610 - loss: 0.3739 - val_accuracy: 0.8739 - val_loss: 0.3578 - learning_rate: 0.1000
Epoch 10/50
18/18 - 0s - 7ms/step - accuracy: 0.8734 - loss: 0.3346 - val_accuracy: 0.8655 - val_loss: 0.3652 - learning_rate: 0.0500
Epoch 11/50
18/18 - 0s - 5ms/step - accuracy: 0.8672 - loss: 0.3334 - val_accuracy: 0.8655 - val_loss: 0.3671 - learning_rate: 0.0500
Epoch 12/50
18/18 - 0s - 8ms/step - accuracy: 0.8714 - loss: 0.3274 - val_accuracy: 0.8739 - val_loss: 0.3682 - learning_rate: 0.0500
Epoch 13/50
18/18 - 0s - 6ms/step - accuracy: 0.8755 - loss: 0.3321 - val_accuracy: 0.8782 - val_loss: 0.3553 - learning_rate: 0.0500
Epoch 14/50
18/18 - 0s - 7ms/step - accuracy: 0.8693 - loss: 0.3276 - val_accuracy: 0.8739 - val_loss: 0.3618 - learning_rate: 0.0500
Epoch 15/50
18/18 - 0s - 8ms/step - accuracy: 0.8734 - loss: 0.3276 - val_accuracy: 0.8782 - val_loss: 0.3626 - learning_rate: 0.0500
Epoch 16/50
18/18 - 0s - 7ms/step - accuracy: 0.8734 - loss: 0.3304 - val_accuracy: 0.8782 - val_loss: 0.3575 - learning_rate: 0.0500
Epoch 17/50
18/18 - 0s - 6ms/step - accuracy: 0.8776 - loss: 0.3269 - val_accuracy: 0.8782 - val_loss: 0.3533 - learning_rate: 0.0500
Epoch 18/50
18/18 - 0s - 7ms/step - accuracy: 0.8734 - loss: 0.3260 - val_accuracy: 0.8824 - val_loss: 0.3549 - learning_rate: 0.0500
Epoch 19/50
18/18 - 0s - 6ms/step - accuracy: 0.8734 - loss: 0.3244 - val_accuracy: 0.8739 - val_loss: 0.3567 - learning_rate: 0.0500
Epoch 20/50
18/18 - 0s - 5ms/step - accuracy: 0.8651 - loss: 0.3250 - val_accuracy: 0.8613 - val_loss: 0.3604 - learning_rate: 0.0250
Epoch 21/50
18/18 - 0s - 7ms/step - accuracy: 0.8734 - loss: 0.3210 - val_accuracy: 0.8782 - val_loss: 0.3522 - learning_rate: 0.0250
Epoch 22/50
18/18 - 0s - 8ms/step - accuracy: 0.8755 - loss: 0.3231 - val_accuracy: 0.8824 - val_loss: 0.3513 - learning_rate: 0.0250
Epoch 23/50
18/18 - 0s - 7ms/step - accuracy: 0.8755 - loss: 0.3265 - val_accuracy: 0.8739 - val_loss: 0.3603 - learning_rate: 0.0250
Epoch 24/50
18/18 - 0s - 5ms/step - accuracy: 0.8797 - loss: 0.3192 - val_accuracy: 0.8782 - val_loss: 0.3471 - learning_rate: 0.0250
Epoch 25/50
18/18 - 0s - 5ms/step - accuracy: 0.8797 - loss: 0.3205 - val_accuracy: 0.8697 - val_loss: 0.3582 - learning_rate: 0.0250
Epoch 26/50
```

```
18/18 - 0s - 8ms/step - accuracy: 0.8776 - loss: 0.3202 - val_accuracy: 0.8782 - val_loss: 0.3492 - learning_rate: 0.0250
Epoch 27/50
18/18 - 0s - 5ms/step - accuracy: 0.8755 - loss: 0.3206 - val_accuracy: 0.8782 - val_loss: 0.3529 - learning_rate: 0.0250
Epoch 28/50
18/18 - 0s - 5ms/step - accuracy: 0.8734 - loss: 0.3205 - val_accuracy: 0.8782 - val_loss: 0.3463 - learning_rate: 0.0250
Epoch 29/50
18/18 - 0s - 5ms/step - accuracy: 0.8776 - loss: 0.3179 - val_accuracy: 0.8739 - val_loss: 0.3529 - learning_rate: 0.0250
```

4.Submit the link to your Google Colab (make sure that it is accessible to me)

The link is in the top part of the activity

Conclusion and Learning

I concluded that with this activity I was able to learn about how to save models in different format like HDF5 format, JSON format and YAML format I also learn how to do chechpoints for neural network model performance and best neural netwok model only. also I was able to load a save neural network model then load it. lastly I was able to show some of the application like Dropout Regularization, Dropout on the visible layer,Dropout on the hidden layer, application of a time-based learning rate schedule and application of a drop-based learning rate schedule thats all of the things that I did on this assignment