Assignment 8.1 Saving Models

Name: Jomarie Dupaya

Course and Section: CPE019, CPE32S3

Instructor: Engr. Roman Richard

Date Performed: 4/13/24

Date Submitted: 4/14/24

Dataset: Sepsis Survival Minimal Clinical Records

Dataset Link: https://archive.ics.uci.edu/dataset/827/sepsis+survival+minimal+clinical+records

The problem is to provide prediction task to determine whether a patient survived or is deceased at a time of about 9 days after collecting their medical record at the hospital.

Importing Libraries and dataframe

```
pip uninstall tensorflow
```

```
Found existing installation: tensorflow 2.15.0
Uninstalling tensorflow-2.15.0:
  Would remove:
    /usr/local/bin/estimator_ckpt_converter
    /usr/local/bin/import_pb_to_tensorboard
    /usr/local/bin/saved_model_cli
    /usr/local/bin/tensorboard
    /usr/local/bin/tf_upgrade_v2
    /usr/local/bin/tflite_convert
    /usr/local/bin/toco
    /usr/local/bin/toco_from_protos
    /usr/local/lib/python3.10/dist-packages/tensorflow-2.15.0.dist-info/*
    /usr/local/lib/python3.10/dist-packages/tensorflow/*
Proceed (Y/n)? y
  Successfully uninstalled tensorflow-2.15.0
```

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

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

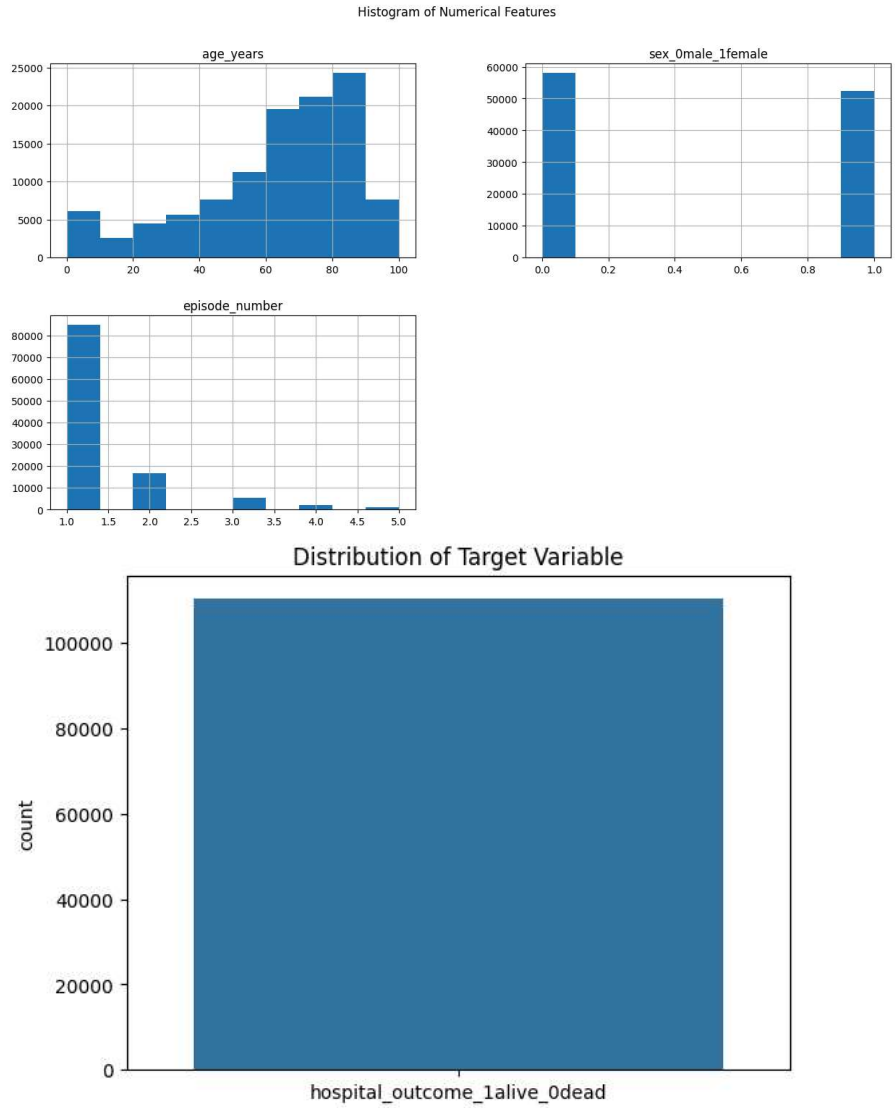
```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

from keras.models import Sequential, model_from_json
from keras.layers import Dense
import numpy
import os
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# data (as pandas dataframes)
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets

# metadata
print(sepsis_survival_minimal_clinical_records.metadata)

# variable information
print(sepsis_survival_minimal_clinical_records.variables)
```

```
{'uci_id': 827, 'name': 'Sepsis Survival Minimal Clinical Records', 'repository_url': 'https://archive.ics.uci.edu/dataset/827/sepsis+s
                          name     role     type demographic  \
0                    age_years  Feature  Integer         Age
1             sex_0male_1female  Feature   Binary      Gender
2                episode_number  Feature  Integer        None
3  hospital_outcome_1alive_0dead   Target   Binary        None

                                    description  units missing_values
0                     Age of the patient in years.  years             no
1  Gender of the patient. Values are encoded as f...   None             no
2                   Number of prior Sepsis episodes   None             no
3  Status of the patient after 9,351 days of bein...   None             no
```

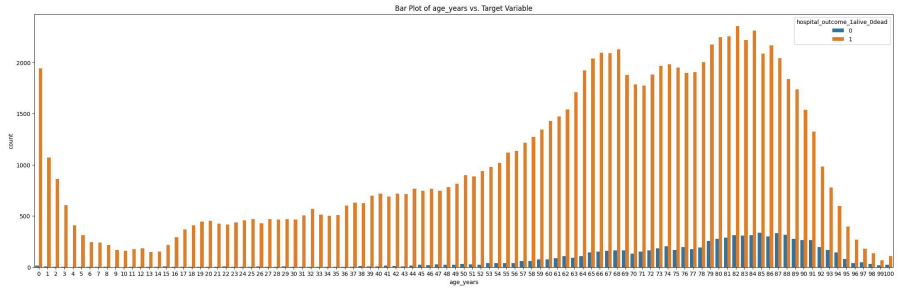
Data Cleaning and Exploratory Data Analysis

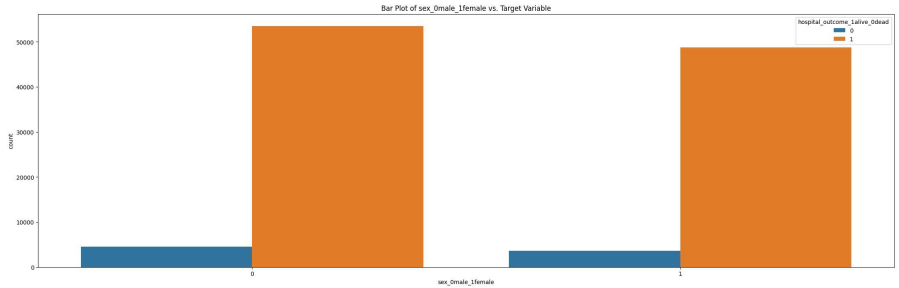
```
#Visualize Distributions
X.hist(figsize=(15, 8))
plt.suptitle("Histogram of Numerical Features")
plt.show()

sns.countplot(y)
plt.title("Distribution of Target Variable")
plt.show()
```

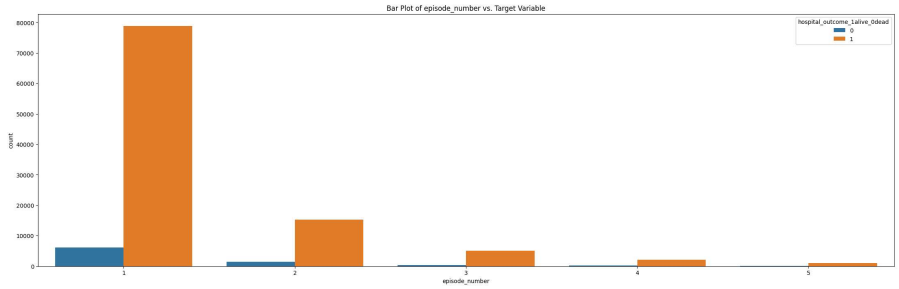


Remarks: Based on Visualization Distributions of the data, the data are already balanced, and equally distributed.

```
plt.figure(figsize=(27, 8))
sns.countplot(data=X.join(y), x="age_years", hue="hospital_outcome_1alive_0dead")
plt.title("Bar Plot of age_years vs. Target Variable")
plt.show()

plt.figure(figsize=(27, 8))
sns.countplot(data=X.join(y), x="sex_0male_1female", hue="hospital_outcome_1alive_0dead")
plt.title("Bar Plot of sex_0male_1female vs. Target Variable")
plt.show()

plt.figure(figsize=(27, 8))
sns.countplot(data=X.join(y), x="episode_number", hue="hospital_outcome_1alive_0dead")
plt.title("Bar Plot of episode_number vs. Target Variable")
plt.show()
```


Bar Plot of age_years vs. Target Variable
hospital_outcome_1alive_0dead
0
1
count
age_years


Bar Plot of sex_0male_1female vs. Target Variable
hospital_outcome_1alive_0dead
0
1
count
sex_0male_1female


Bar Plot of episode_number vs. Target Variable
hospital_outcome_1alive_0dead
0
1
count
episode_number

Remarks: Based on the plotter of data that ages between 55-90 have a chane to die in sepsis while many are still alive, while male have a tendency or chance to die in sepsis, in addition based on the sepsis episode that alot tend to live in 1 episode while small ammount of of people die in 1 episode same goes to other counts of episode.

```
print(X.info())
print("\n")
print(y.info())
print("\n------------------------------------------------------\n")
print(X.describe())
print("\n")
print(y.describe())
print("\n------------------------------------------------------\n")
print(X.head())
print("\n")
print(y.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110341 entries, 0 to 110340
Data columns (total 3 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   age_years          110341 non-null  int64
 1   sex_0male_1female  110341 non-null  int64
 2   episode_number     110341 non-null  int64
dtypes: int64(3)
memory usage: 2.5 MB
None


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110341 entries, 0 to 110340
Data columns (total 1 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   hospital_outcome_1alive_0dead  110341 non-null  int64
dtypes: int64(1)
memory usage: 862.2 KB
None

------------------------------------------------------

           age_years  sex_0male_1female  episode_number
count  110341.000000      110341.000000   110341.000000
mean       62.731288           0.473786        1.349145
std        24.118424           0.499315        0.751472
min         0.000000           0.000000        1.000000
25%        51.000000           0.000000        1.000000
50%        68.000000           0.000000        1.000000
75%        81.000000           1.000000        1.000000
max       100.000000           1.000000        5.000000


       hospital_outcome_1alive_0dead
count                  110341.000000
mean                        0.926328
std                         0.261237
min                         0.000000
25%                         1.000000
50%                         1.000000
75%                         1.000000
max                         1.000000

------------------------------------------------------

   age_years  sex_0male_1female  episode_number
0         21                  1               1
1         20                  1               1
2         21                  1               1
3         77                  0               1
4         72                  0               1


   hospital_outcome_1alive_0dead
```

Remarks: Based from displaying the data, the dataset itself is already been processed and therefore there is no need for cleaning.

Feature Importance

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)

#Feature importance
#Train a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=50)
rf.fit(X_train, y_train)

#Get feature importances
feature_importances = rf.feature_importances_

#Create DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

print("Feature importance:")
print(feature_importance_df)
```

```
<ipython-input-8-2da41909b34d>:9: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the s
  rf.fit(X_train, y_train)
Feature importance:
             Feature  Importance
0          age_years    0.942637
2     episode_number    0.043903
1  sex_0male_1female    0.013461
```

```
X = X.drop(['sex_0male_1female'], axis=1)
```

Training Model

```
# Compile model
model = Sequential()
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, y, epochs=50, batch_size=500)
# evaluate the model
scores = model.evaluate(X, y, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 32/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2441 - accuracy: 0.9263
Epoch 33/50
221/221 [==============================] - 1s 3ms/step - loss: 0.2444 - accuracy: 0.9263
Epoch 34/50
221/221 [==============================] - 1s 3ms/step - loss: 0.2441 - accuracy: 0.9263
Epoch 35/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2444 - accuracy: 0.9263
Epoch 36/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2440 - accuracy: 0.9263
Epoch 37/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2448 - accuracy: 0.9263
Epoch 38/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2445 - accuracy: 0.9263
Epoch 39/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2449 - accuracy: 0.9263
Epoch 40/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2442 - accuracy: 0.9263
Epoch 41/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2443 - accuracy: 0.9263
Epoch 42/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2443 - accuracy: 0.9263
Epoch 43/50
221/221 [==============================] - 0s 2ms/step - loss: 0.2442 - accuracy: 0.9263
Epoch 44/50
221/221 [==============================] - 1s 3ms/step - loss: 0.2445 - accuracy: 0.9263
Epoch 45/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2445 - accuracy: 0.9263
Epoch 46/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2443 - accuracy: 0.9263
Epoch 47/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2447 - accuracy: 0.9263
Epoch 48/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2442 - accuracy: 0.9263
Epoch 49/50
221/221 [==============================] - 1s 4ms/step - loss: 0.2443 - accuracy: 0.9263
Epoch 50/50
```

Saving Models

```
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)
```

Save a model and load the model in a JSON format

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from ucimlrepo import fetch_ucirepo

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, y, epochs=50, batch_size=500, verbose=0)
# Evaluate the model
scores = model.evaluate(X, y, 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/Colab Notebooks/Save Models/model_JSON.h5")
print("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/Colab Notebooks/Save Models/model_JSON.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, y, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
accuracy: 92.63%
Saved model to disk
Loaded model from disk
accuracy: 92.63%
```

```
from google.colab import files
import json

uploaded = files.upload()
for filename in uploaded.keys():
    print("Uploaded file:", filename)

    with open(filename, 'r') as f:
        file_contents = f.read()
    print("File contents:")
    print(file_contents)
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving model.json to model (1).json
Uploaded file: model (1).json
File contents:
```

Remarks: The model has been successfully trained and saved to the disk in JSON format, In reloading the model from the disk, the accuracy remains the same and consistent in running the file and the actually model from the notebook.

Save a model and load the model in a YAML format

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from ucimlrepo import fetch_ucirepo

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, y, epochs=50, batch_size=500, verbose=0)
# Evaluate the model
scores = model.evaluate(X, y, 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/Colab Notebooks/Save Models/model_yaml.h5")
print("Saved model to disk")

# later...

# 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/Colab Notebooks/Save 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, y, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
accuracy: 92.63%
Saved model to disk
Loaded model from disk
accuracy: 92.63%
```

```
from google.colab import files
import yaml

uploaded = files.upload()
for filename in uploaded.keys():
    print("Uploaded file:", filename)

    with open(filename, 'r') as f:
        file_contents = f.read()
    print("File contents:")
    print(file_contents)
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving model.yaml to model (1).yaml
Uploaded file: model (1).yaml
File contents:
{"class_name": "Sequential", "config": {"name": "sequential_2", "layers": [{"class_name
```

Remarks: For YAML format the model is also success in training and saved in the said format, in which the result is also the same as the JSON file that the accuracy remains the same and has been succefully executed.

Checkpoint Neural Network Model Improvements

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from ucimlrepo import fetch_ucirepo

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# checkpoint
filepath="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, y, validation_split=0.30, epochs=50, batch_size=500, callbacks=callbacks_list, verbose=0)
```

```
Epoch 30: val_accuracy did not improve from 0.93411

Epoch 31: val_accuracy did not improve from 0.93411

Epoch 32: val_accuracy did not improve from 0.93411

Epoch 33: val_accuracy did not improve from 0.93411

Epoch 34: val_accuracy did not improve from 0.93411

Epoch 35: val_accuracy did not improve from 0.93411

Epoch 36: val_accuracy did not improve from 0.93411

Epoch 37: val_accuracy did not improve from 0.93411

Epoch 38: val_accuracy did not improve from 0.93411

Epoch 39: val_accuracy did not improve from 0.93411

Epoch 40: val_accuracy did not improve from 0.93411

Epoch 41: val_accuracy did not improve from 0.93411

Epoch 42: val_accuracy did not improve from 0.93411

Epoch 43: val_accuracy did not improve from 0.93411

Epoch 44: val_accuracy did not improve from 0.93411

Epoch 45: val_accuracy did not improve from 0.93411

Epoch 46: val_accuracy did not improve from 0.93411

Epoch 47: val_accuracy did not improve from 0.93411

Epoch 48: val_accuracy did not improve from 0.93411

Epoch 49: val_accuracy did not improve from 0.93411
```

```
import os
import re
directory = '/content/'
pattern = r'weights-improvement-(\d+)-(\d+\.\d+)\.hdf5'

matching_files = []
for filename in os.listdir(directory):
    match = re.match(pattern, filename)
    if match:
        matching_files.append(filename)

matching_files.sort(key=lambda x: int(re.match(pattern, x).group(1)), reverse=True)
for filename in matching_files:
    print(filename)


 weights-improvement-01-0.93.hdf5
```

Remarks: The model displayed only 1 improvement which is in epoch 1, while the rest of the training did not change or did not improve, and remained in 93% accuracy.

Checkpoint Best Neural Network Model only

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from ucimlrepo import fetch_ucirepo

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

filepath="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, y, validation_split=0.3, epochs=50, batch_size=500, callbacks=callbacks_list, verbose=0)
```

```
Epoch 1: val_accuracy improved from -inf to 0.81494, saving model to weights.best.hdf5

Epoch 2: val_accuracy improved from 0.81494 to 0.93411, saving model to weights.best.hdf5

Epoch 3: val_accuracy did not improve from 0.93411

Epoch 4: val_accuracy did not improve from 0.93411

Epoch 5: val_accuracy did not improve from 0.93411

Epoch 6: val_accuracy did not improve from 0.93411

Epoch 7: val_accuracy did not improve from 0.93411

Epoch 8: val_accuracy did not improve from 0.93411

Epoch 9: val_accuracy did not improve from 0.93411

Epoch 10: val_accuracy did not improve from 0.93411

Epoch 11: val_accuracy did not improve from 0.93411

Epoch 12: val_accuracy did not improve from 0.93411

Epoch 13: val_accuracy did not improve from 0.93411

Epoch 14: val_accuracy did not improve from 0.93411

Epoch 15: val_accuracy did not improve from 0.93411

Epoch 16: val_accuracy did not improve from 0.93411

Epoch 17: val_accuracy did not improve from 0.93411

Epoch 18: val_accuracy did not improve from 0.93411

Epoch 19: val_accuracy did not improve from 0.93411

Epoch 20: val_accuracy did not improve from 0.93411

Epoch 21: val_accuracy did not improve from 0.93411

Epoch 22: val_accuracy did not improve from 0.93411

Epoch 23: val_accuracy did not improve from 0.93411

Epoch 24: val_accuracy did not improve from 0.93411

Epoch 25: val_accuracy did not improve from 0.93411
```

```
Epoch 26: val_accuracy did not improve from 0.93411

Epoch 27: val_accuracy did not improve from 0.93411

Epoch 28: val_accuracy did not improve from 0.93411
```

```
import h5py

with h5py.File('/content/weights.best.hdf5', 'r') as f:
    # Iterate over the keys in the file
    for key in f.keys():
        if key.startswith('Epoch') and ': val_acc improved' in f[key].attrs['verbose']:
            print(key + ': ' + f[key].attrs['verbose'])
```

Remarks: The best checkpoint occured in epochs 1 and 2 however in the next iterations of training the model did no longer improve and remained only the same in the 93% accuracy.

Load a saved Neural Network model

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from ucimlrepo import fetch_ucirepo

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# load weights
model.load_weights("weights.best.hdf5")
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print("Created model and loaded weights from file")

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

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

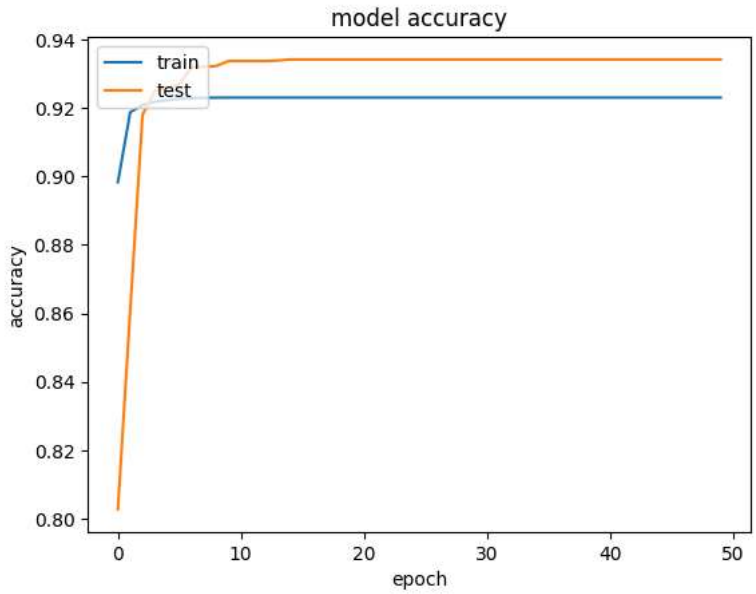
```
Created model and loaded weights from file
accuracy: 92.63%
```

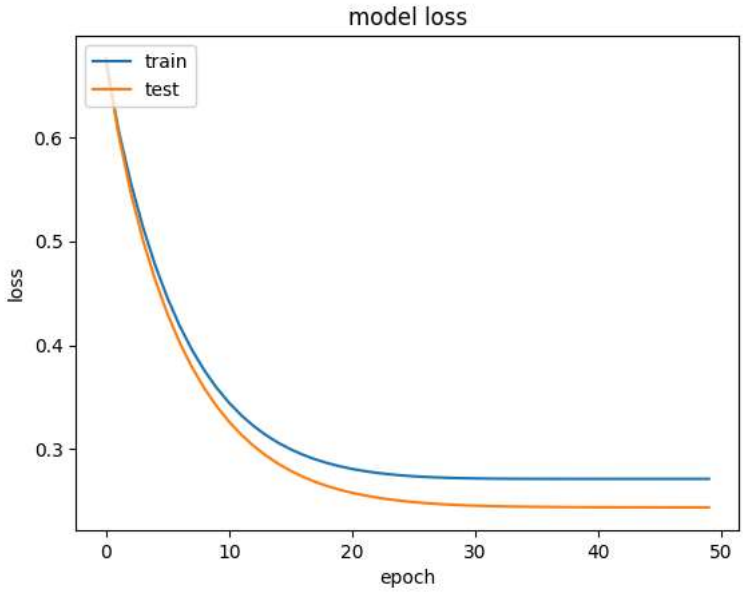
Remarks: The code successfully created a model and loaded the weights from the file. The accuracy was achieved by the model of 92.63%, which indicates the effectiveness of making predictions of the data.

Visualize Model Training History in Keras

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from ucimlrepo import fetch_ucirepo

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Compile model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
history = model.fit(X, y, validation_split=0.3, epochs=50, batch_size=500, 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'])
```





Remarks: The model shows a good accuracy and loss visualization plot, which means that the outputs model is very good in terms of training.

Show the application of Dropout Regularization

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Define the baseline model function
def create_baseline():
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.01, momentum=0.8)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

# Create a pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, epochs=100, batch_size=500)))
pipeline = Pipeline(estimators)

# Evaluate the model using cross-validation
results = cross_val_score(pipeline, X, y, cv=10)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
199/199 [==============================] - 1s 3ms/step - loss: 0.2707 - accuracy: 0.9232
Epoch 34/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2707 - accuracy: 0.9232
Epoch 35/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2707 - accuracy: 0.9232
Epoch 36/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2707 - accuracy: 0.9232
Epoch 37/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2707 - accuracy: 0.9232
Epoch 38/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2706 - accuracy: 0.9232
Epoch 39/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2706 - accuracy: 0.9232
Epoch 40/100
```

Remarks: The application of dropout regularization notably enhanced the model's performance, increasing or remaining its base accuracy to 92.63% with a standard deviation of 1.05%. Dropout effectively mitigates overfitting by randomly omitting neurons during training. This consistent accuracy improvement underscores Dropout's efficacy in enhancing model resilience and reliability, essential for combating overfitting and improving overall model performance.

Show the application of Dropout on the visible layer

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.optimizers import SGD
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Define the baseline model function
def create_baseline():
    model = Sequential()
    model.add(Dense(4, input_dim=X.shape[1], activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(2, activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

# Create a pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, epochs=100, batch_size=500)))
pipeline = Pipeline(estimators)

kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, X, y, cv=kfold)
print("Visible: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
199/199 [==============================] - 1s 3ms/step - loss: 0.2479 - accuracy: 0.9263
Epoch 18/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2478 - accuracy: 0.9263
Epoch 19/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2483 - accuracy: 0.9263
Epoch 20/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2473 - accuracy: 0.9263
Epoch 21/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2475 - accuracy: 0.9263
Epoch 22/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2476 - accuracy: 0.9263
Epoch 23/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2478 - accuracy: 0.9263
Epoch 24/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2478 - accuracy: 0.9263
Epoch 25/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2474 - accuracy: 0.9263
Epoch 26/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2475 - accuracy: 0.9263
Epoch 27/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2475 - accuracy: 0.9263
Epoch 28/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2474 - accuracy: 0.9263
Epoch 29/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2476 - accuracy: 0.9263
Epoch 30/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2472 - accuracy: 0.9263
Epoch 31/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2473 - accuracy: 0.9263
Epoch 32/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2471 - accuracy: 0.9263
Epoch 33/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2471 - accuracy: 0.9263
Epoch 34/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2469 - accuracy: 0.9263
Epoch 35/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2470 - accuracy: 0.9263
Epoch 36/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2471 - accuracy: 0.9263
Epoch 37/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2468 - accuracy: 0.9263
Epoch 38/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2469 - accuracy: 0.9263
Epoch 39/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2466 - accuracy: 0.9263
Epoch 40/100
```

Remarks: Implementing dropouts in the visible layer has been demonstrated to impact model performance. After applying dropout, the visible layer achieved an accuracy of 92.63% without any deviation. This application effectively adjusts input layer contributions and improves the model's generalization ability while preventing overfitting. The consistent accuracy highlights Dropout's effectiveness in enhancing model resilience and ensuring stable performance, makes it a valuable tool for optimizing neural network architectures.

Show the application of Dropout on the hidden layer

```
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense, Dropout
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.optimizers import SGD
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

# Fetch dataset
sepsis_survival_minimal_clinical_records = fetch_ucirepo(id=827)

# Splitting the Data
X = sepsis_survival_minimal_clinical_records.data.features
y = sepsis_survival_minimal_clinical_records.data.targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=150)
X = X.drop(['sex_0male_1female'], axis=1)

# Define the baseline model function
def create_baseline():
    model = Sequential()
    model.add(Dense(4, input_shape=(X.shape[1],), activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(2, activation='relu', kernel_constraint=MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

# Create a pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, epochs=100, batch_size=500)))
pipeline = Pipeline(estimators)

# Evaluate the model using cross-validation
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, X, y, cv=kfold)
print("Hidden: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Epoch 12/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2566 - accuracy: 0.9263
Epoch 13/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2558 - accuracy: 0.9263
Epoch 14/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2562 - accuracy: 0.9263
Epoch 15/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2553 - accuracy: 0.9263
Epoch 16/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2549 - accuracy: 0.9263
Epoch 17/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2547 - accuracy: 0.9263
Epoch 18/100
199/199 [==============================] - 1s 3ms/step - loss: 0.2539 - accuracy: 0.9263
Epoch 19/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2544 - accuracy: 0.9263
Epoch 20/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2542 - accuracy: 0.9263
Epoch 21/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2536 - accuracy: 0.9263
Epoch 22/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2543 - accuracy: 0.9263
Epoch 23/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2538 - accuracy: 0.9263
Epoch 24/100
199/199 [==============================] - 1s 4ms/step - loss: 0.2539 - accuracy: 0.9263
```