```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


drive.mount("/content/drive")
path = "/content/drive/MyDrive/Capstone/exercise_datasetV2.csv"
df = pd.read_csv(path)
print(df.head())
banyak_kategori = len(df.index)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
  Activity, Exercise or Sport (1 hour) Intensity Description  \
0          Cycling, mountain bike, bmx                   NaN
1  Cycling, <10 mph, leisure bicycling                   NaN
2             Cycling, >20 mph, racing                   NaN
3          Cycling, 10-11.9 mph, light                   NaN
4       Cycling, 12-13.9 mph, moderate                   NaN

   Duration (minutes)  Calories per kg
0                  60         1.750730
1                  60         0.823236
2                  60         3.294974
3                  60         1.234853
4                  60         1.647825
```

```
list_berat = []
for i in range(len(df.index)):
  list_berat.append(1)

df['berat'] =  list_berat
dict_df = {'Activity, Exercise or Sport (1 hour)' : [], 'Duration (minutes)': [], 'Calories per kg': [], 'berat' : []}
df_new = df
for index, row in df.iterrows():
  print(index)
  menit = row['Duration (minutes)']
  activity = row['Activity, Exercise or Sport (1 hour)']
  calories = row['Calories per kg']
  for i in range(1,menit):
    for j in range(2,101):
      new_calories = calories*1.0/60*i*j
      list_activity = dict_df.get('Activity, Exercise or Sport (1 hour)')
      list_duration = dict_df.get('Duration (minutes)')
      list_calories = dict_df.get('Calories per kg')
      list_berat = dict_df.get('berat')
      list_activity.append(activity)
      list_duration.append(i)
      list_calories.append(new_calories)
      list_berat.append(j)
      #new_row = pd.DataFrame({'Activity, Exercise or Sport (1 hour)' : [activity], 'Duration (minutes)': [i], 'Calories per k

df_curr = pd.DataFrame(dict_df)
df_new = pd.concat([df_curr, df_new.loc[:]]).reset_index(drop=True)
#df2 = pd.concat([new_row,df.loc[:]]).reset_index(drop=True)
print(df_new.head())
print(df_new.tail())
```

```
0
1
2
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
```

```
print(len(df_new.index))
print(df_new.describe())
print(df_new.dtypes)
df_new.rename(columns = {'Activity, Exercise or Sport (1 hour)':'activity', 'Duration (minutes)' : 'durasi' , 'Calories per kg
print(df_new.head())
```

```
1448816
       Duration (minutes)  Calories per kg         berat
count        1.448816e+06     1.448816e+06  1.448816e+06
mean         3.000514e+01     3.467251e+01  5.099144e+01
std          1.703246e+01     3.748635e+01  2.858243e+01
min          1.000000e+00     1.033558e-02  1.000000e+00
25%          1.500000e+01     8.237434e+00  2.600000e+01
50%          3.000000e+01     2.219663e+01  5.100000e+01
75%          4.500000e+01     4.774767e+01  7.600000e+01
max          6.000000e+01     3.644815e+02  1.000000e+02
Activity, Exercise or Sport (1 hour)     object
Duration (minutes)                        int64
Calories per kg                         float64
berat                                     int64
Intensity Description                    object
dtype: object
                      activity  durasi  calories  berat Intensity Description
0  Cycling, mountain bike, bmx       1  0.058358      2                    NaN
1  Cycling, mountain bike, bmx       1  0.087536      3                    NaN
2  Cycling, mountain bike, bmx       1  0.116715      4                    NaN
3  Cycling, mountain bike, bmx       1  0.145894      5                    NaN
4  Cycling, mountain bike, bmx       1  0.175073      6                    NaN
```

```
target = df['Activity, Exercise or Sport (1 hour)']
print(df_new.head())
numeric_feature_names = ['durasi', 'calories', 'berat']
```

```
numeric_features = df_new[numeric_feature_names]
numeric_features.head()
```

	activity	durasi	calories	berat	Intensity Description
0	Cycling, mountain bike, bmx	1	0.058358	2	NaN
1	Cycling, mountain bike, bmx	1	0.087536	3	NaN
2	Cycling, mountain bike, bmx	1	0.116715	4	NaN
3	Cycling, mountain bike, bmx	1	0.145894	5	NaN
4	Cycling, mountain bike, bmx	1	0.175073	6	NaN

	durasi	calories	berat
0	1	0.058358	2
1	1	0.087536	3
2	1	0.116715	4
3	1	0.145894	5
4	1	0.175073	6

```
"""
https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
https://www.tensorflow.org/tutorials/load_data/pandas_dataframe
https://regenerativetoday.com/a-step-by-step-tutorial-to-develop-a-multi-output-model-in-tensorflow/
"""
```

```
'\nhttps://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/\nhttps://www.tensc
a/pandas_dataframe\nhttps://regenerativetoday.com/a-step-by-step-tutorial-to-develop-a-multi-output-model-in-tensorflow/\
```

```
jumlah_class = len(df_new['activity'].value_counts())
print(jumlah_class)
```

```
248
```

```
df_new['activity'] = df_new['activity'].astype('category')
df_new['activity_category'] = df_new['activity'].cat.codes.astype('category')
print(df_new.head())
```

```
                        activity  durasi  calories  berat Intensity Description  \
0  Cycling, mountain bike, bmx       1  0.058358      2                    NaN
1  Cycling, mountain bike, bmx       1  0.087536      3                    NaN
2  Cycling, mountain bike, bmx       1  0.116715      4                    NaN
3  Cycling, mountain bike, bmx       1  0.145894      5                    NaN
4  Cycling, mountain bike, bmx       1  0.175073      6                    NaN

  activity_category
0                61
1                61
2                61
3                61
4                61
```

```
df_new_2 = df_new.drop(columns = ['activity', 'Intensity Description', 'activity_category'])
sc = StandardScaler()
x = pd.DataFrame(sc.fit_transform(df_new_2))
```

```
df_new_2['durasi'] = MinMaxScaler().fit_transform(np.array(df_new_2['durasi']).reshape(-1,1))
df_new_2['calories'] = MinMaxScaler().fit_transform(np.array(df_new_2['calories']).reshape(-1,1))
df_new_2['berat'] = MinMaxScaler().fit_transform(np.array(df_new_2['berat']).reshape(-1,1))
```

```
y = tf.keras.utils.to_categorical(df_new["activity_category"].values, num_classes=jumlah_class)

x_train, x_test, y_train, y_test = train_test_split(x.values, y, test_size=0.2)
```

```
print(x_train)
print(y_train)
print(x_test)
print(y_test)
```

```
[[-0.93968479 -0.63774467 -0.80439099 -0.9428579 ]
 [ 1.17392762 -0.63656415 -1.46913527  0.69142913]
 [ 0.88037034  0.4482544   0.80498987 -0.6215878 ]
 ...
 [-1.35066498 -0.52491909  0.94493604 -0.25841291]
 [-0.05901295  1.8469556   1.2598149  -1.08254056]
 [ 0.46939015  0.07684826 -0.66444483  0.35619076]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
```

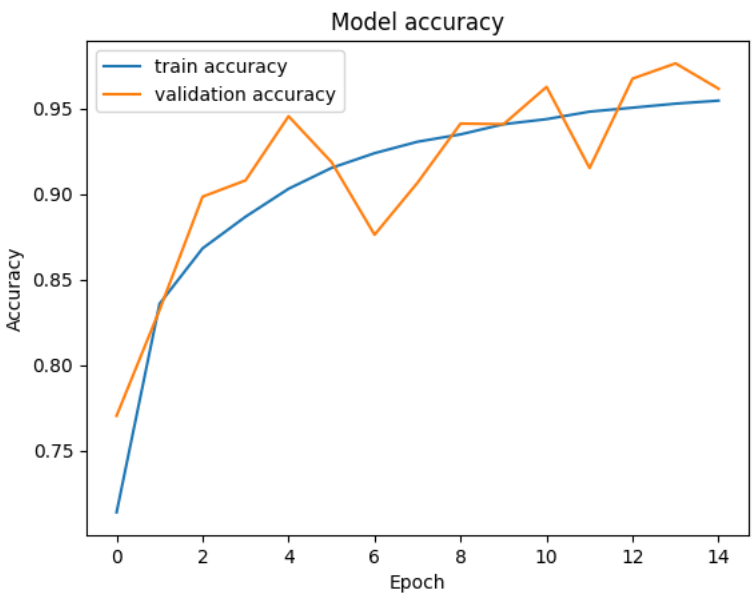
```
       [0. 0. 0. ... 0. 0. 0.]
       [0. 0. 0. ... 0. 0. 0.]]
      [[-0.93968479 -0.80195575 -0.66444483 -0.39809556]
       [-1.64422226 -0.89974037  0.1402456  -1.50158851]
       [ 0.41067869  0.11258537  0.59507063 -0.03492066]
       ...
       [ 0.52810161 -0.72492808 -1.53910835  1.45968372]
       [ 1.64361927  1.33594272  0.70003025  1.22222321]
       [ 0.29325578 -0.60787586 -0.62945829 -1.45968372]]
      [[0. 0. 0. ... 0. 0. 0.]
       [0. 0. 0. ... 0. 0. 0.]
       [0. 0. 0. ... 0. 0. 0.]
       ...
       [0. 0. 0. ... 0. 0. 0.]
       [0. 0. 0. ... 0. 0. 0.]
       [0. 0. 0. ... 0. 0. 0.]]
```

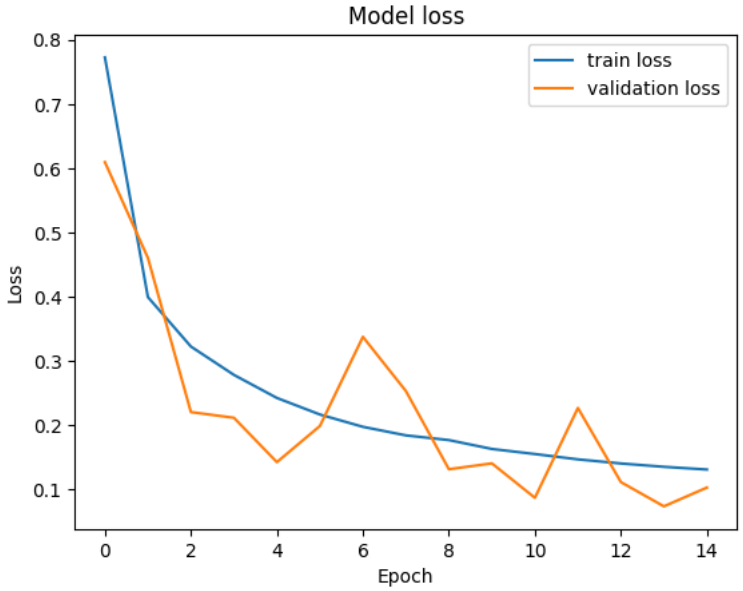
```
from keras.engine import sequential
def get_model():
    model =  tf.keras.Sequential([
        Dense(50, activation='relu'),
        Dense(50, activation='relu'),
        Dense(60, activation='relu'),
        Dense(70, activation='relu'),
        Dense(80, activation='relu'),
        Dense(90, activation='relu'),
        Dense(100, activation='relu'),
        Dense(banyak_kategori, activation='softmax')
    ])

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


my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]


model = get_model()



model_fit = model.fit(x_train,
                      y_train,
                      epochs = 15,
                      validation_data = (x_test, y_test))
```

```
Epoch 1/15
36221/36221 [==============================] - 155s 4ms/step - loss: 0.7726 - accuracy: 0.7139 - val_loss: 0.6095 - val_a
Epoch 2/15
36221/36221 [==============================] - 151s 4ms/step - loss: 0.3992 - accuracy: 0.8358 - val_loss: 0.4602 - val_a
Epoch 3/15
36221/36221 [==============================] - 146s 4ms/step - loss: 0.3222 - accuracy: 0.8681 - val_loss: 0.2201 - val_a
Epoch 4/15
36221/36221 [==============================] - 154s 4ms/step - loss: 0.2780 - accuracy: 0.8867 - val_loss: 0.2112 - val_a
Epoch 5/15
36221/36221 [==============================] - 151s 4ms/step - loss: 0.2421 - accuracy: 0.9030 - val_loss: 0.1421 - val_a
Epoch 6/15
36221/36221 [==============================] - 147s 4ms/step - loss: 0.2163 - accuracy: 0.9152 - val_loss: 0.1984 - val_a
Epoch 7/15
36221/36221 [==============================] - 145s 4ms/step - loss: 0.1971 - accuracy: 0.9238 - val_loss: 0.3373 - val_a
Epoch 8/15
36221/36221 [==============================] - 143s 4ms/step - loss: 0.1838 - accuracy: 0.9304 - val_loss: 0.2528 - val_a
Epoch 9/15
36221/36221 [==============================] - 149s 4ms/step - loss: 0.1764 - accuracy: 0.9347 - val_loss: 0.1310 - val_a
Epoch 10/15
36221/36221 [==============================] - 144s 4ms/step - loss: 0.1626 - accuracy: 0.9407 - val_loss: 0.1402 - val_a
Epoch 11/15
36221/36221 [==============================] - 149s 4ms/step - loss: 0.1547 - accuracy: 0.9436 - val_loss: 0.0865 - val_a
Epoch 12/15
36221/36221 [==============================] - 155s 4ms/step - loss: 0.1464 - accuracy: 0.9480 - val_loss: 0.2264 - val_a
Epoch 13/15
36221/36221 [==============================] - 156s 4ms/step - loss: 0.1400 - accuracy: 0.9503 - val_loss: 0.1110 - val_a
Epoch 14/15
36221/36221 [==============================] - 153s 4ms/step - loss: 0.1348 - accuracy: 0.9527 - val_loss: 0.0732 - val_a
Epoch 15/15
36221/36221 [==============================] - 161s 4ms/step - loss: 0.1306 - accuracy: 0.9545 - val_loss: 0.1025 - val_a
```

```
def plot_accuracy(history):

    plt.plot(history.history['accuracy'],label='train accuracy')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
```

```
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='best')
    plt.savefig('Accuracy_v1_model_inceptionv3')
    plt.show()

def plot_loss(history):

    plt.plot(history.history['loss'],label="train loss")
    plt.plot(history.history['val_loss'],label="validation loss")
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='best')
    plt.savefig('Loss_v1_model_inceptionv3')
    plt.show()

plot_accuracy(model_fit)
plot_loss(model_fit)
```






```
model.save('/content/drive/MyDrive/Capstone/model_exercise.h5')
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('/content/drive/MyDrive/Capstone/model_exercise.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be
```

```
from keras.models import model_from_json
# serialize model to json
json_model_exercise = model.to_json()
#save the model architecture to JSON file
with open('/content/drive/MyDrive/Capstone/exercise_model.json', 'w') as json_file:
    json_file.write(json_model_exercise)
```

```
predict_x = model.predict(x_test)
classes_x = np.argmax(predict_x,axis=1)
#y_pred_class = model.predict_classes(x_test)

y_pred = model.predict(x_test)
y_test_class = np.argmax(y_test, axis=1)
confusion_matrix = confusion_matrix(y_test_class, classes_x)
```

```
9056/9056 [==============================] - 15s 2ms/step
9056/9056 [==============================] - 18s 2ms/step
```

```
report = classification_report(y_test_class, classes_x, output_dict=True, zero_division=0)
print(report)

# Extract the metrics
precision = report['macro avg']['precision']
recall = report['macro avg']['recall']
f1_score = report['macro avg']['f1-score']
support = report['macro avg']['support']
accuracy = report['accuracy']

print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
print("support" , support)
```

```
{'0': {'precision': 0.8932330827067669, 'recall': 0.9761709120788825, 'f1-score': 0.9328621908127208, 'support': 1217}, '
accuracy: 0.9614306815201337
Precision: 0.9646650924389747
Recall: 0.9614497926575039
F1-score: 0.9609796760828131
support 289764
```

```
#print(type(confusion_matrix(y_test_class, classes_x)))
#print(y_test_class)
#print(y_test)
print(len(y_test_class))
print(len(classes_x))
print(len(np.unique(y_test_class)))
print(len(np.unique(classes_x)))
```

```
289764
289764
248
248
```

```
dict_activity = dict(enumerate(df_new['activity'].cat.categories))
df_new['activity_code'] = df_new['activity'].cat.codes
print(df_new['activity_code'])
print(dict_activity)
df_new['activity_reversed'] = df_new['activity_code'].map(dict_activity)
df_y_test_class = pd.DataFrame(y_test_class, columns = ['activity_class'])
df_y_test_class['activity_class_reversed'] = df_y_test_class['activity_class'].map(dict_activity)
print(df_y_test_class)
```

```
0           61
1           61
2           61
3           61
4           61
           ...
1448811     73
1448812     40
1448813    207
1448814    232
1448815    208
Name: activity_code, Length: 1448816, dtype: int16
{0: 'Aerobics, general', 1: 'Aerobics, high impact', 2: 'Aerobics, low impact', 3: 'Aerobics, step aerobics', 4: 'Archery
        activity_class                activity_class_reversed
0                   95                     Horseshoe pitching
```

```
1                  16                                Billiards
2                 121                                 Painting
3                 245   Whitewater rafting, kayaking, canoeing
4                 161    Running, training, pushing wheelchair
...               ...                                      ...
289759            114                Martial arts, tae kwan do
289760            222                          Walking 2.5 mph
289761            228                          Walking 5.0 mph
289762            211  Track and field (high jump, pole vault)
289763             19                                  Bowling

[289764 rows x 2 columns]
```

```
df_cm = pd.DataFrame(confusion_matrix(y_test_class, classes_x), columns=np.unique(y_test_class), index=np.unique(y_test_class)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'


f, ax = plt.subplots(figsize=(500, 500))
cmap = sns.cubehelix_palette(light=1, as_cmap=True)

sns.heatmap(df_cm, cbar=False, annot=True, cmap=cmap, square=True, fmt='.0f',
            annot_kws={'size': 10})
plt.title('Actuals vs Predicted')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-19d9437c6b04> in <cell line: 1>()
----> 1 df_cm = pd.DataFrame(confusion_matrix(y_test_class, classes_x), columns=np.unique(y_test_class), index=np.unique(
      2 df_cm.index.name = 'Actual'
      3 df_cm.columns.name = 'Predicted'
      4
      5

TypeError: 'numpy.ndarray' object is not callable
```

SEARCH STACK OVERFLOW