```
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.preprocessing import StandardScaler
```

```
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())
```

```
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
  Activity, Exercise or Sport (1 hour)  Duration (minutes)  Calories per kg  \
0          Cycling, mountain bike, bmx                   1         0.058358   
1          Cycling, mountain bike, bmx                   1         0.087536   
2          Cycling, mountain bike, bmx                   1         0.116715   
3          Cycling, mountain bike, bmx                   1         0.145894   
4          Cycling, mountain bike, bmx                   1         0.175073   

   berat Intensity Description  
0      2                   NaN  
1      3                   NaN  
2      4                   NaN  
3      5                   NaN  
4      6                   NaN  
            Activity, Exercise or Sport (1 hour)  Duration (minutes)  \
1448811                         General cleaning                  60   
1448812                        Cleaning, dusting                  60   
1448813                         Taking out trash                  60   
1448814            Walking, pushing a wheelchair                  60   
1448815  Teach physical education,exercise class                  60   

         Calories per kg  berat Intensity Description  
1448811         0.721008      1                   NaN  
1448812         0.515199      1                   NaN  
1448813         0.617427      1                   NaN  
1448814         0.823236      1                   NaN  
1448815         0.823236      1                   NaN  
```

```
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
4	1	0.175073	6

```
"""
def get_base_model():
  model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(banyak_kategori, activation = 'softmax')
  ])

  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-3),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model
"""
```

```
'\ndef get_base_model():\n  model = tf.keras.Sequential([\n    normalizer,\n    tf.keras.layers.Dense(10, activation='rel
se(10, activation='relu'),\n    tf.keras.layers.Dense(banyak_kategori, activation = 'softmax')\n  ])\n\n  model.compile(
Adam(learning_rate=2e-3),\n                loss='categorical_crossentropy',\n                metrics=['accuracy'])\n  ret
```

```
"""
y = df_new['activity']
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
"""
```

```
'\ny = df_new['activity']\nencoder = LabelEncoder()\nencoder.fit(y)\nencoded_Y = encoder.transform(y)\n# convert integers
hot encoded)\ndummy_y = np_utils.to_categorical(encoded_Y)\n'
```

```
#est = KerasClassifier(build_fn= get_base_model, epochs=200, batch_size=5, verbose=0)
```

```
#kfold = KFold(n_splits=5, shuffle=True)
```

```
"""
x = df_new[numeric_feature_names]

results = cross_val_score(est, x, dummy_y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
"""
```

```
'\nx = df_new[numeric_feature_names]\n\nresults = cross_val_score(est, x, dummy_y, cv=kfold)\nprint("Baseline: %.2f%% (%.
results.std()*100))\n'
```

```
"""
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.tens
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'])
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)
```

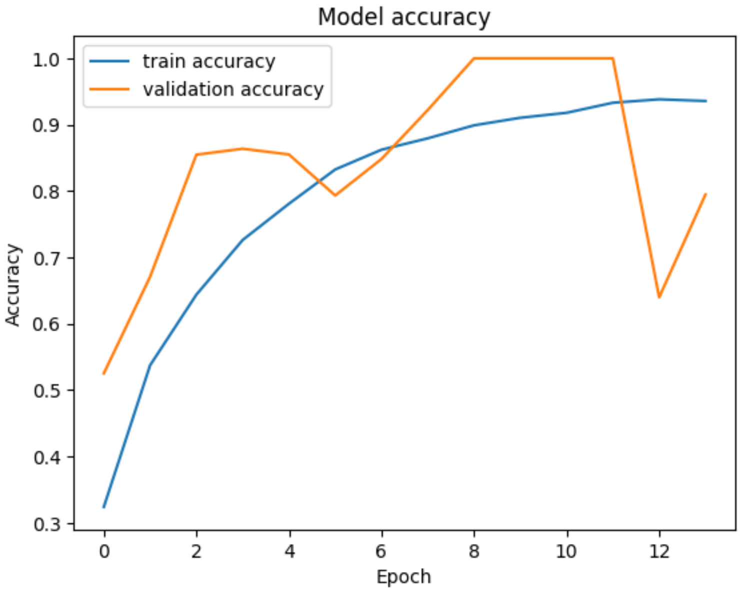
```
[[-0.35257023 -0.7249046  -0.87436408 -1.6692077 ]
 [-0.70483897 -0.16678584 -0.17463326 -0.04888893]
 [ 0.23454433  0.84364059  0.70003025  1.09650882]
 ...
 [ 0.35196724 -0.04787163  1.53970723  0.88698484]
 [ 1.40877344  0.24118937  0.28019176 -0.97079443]
 [-1.11581916 -0.62297401  0.83997641 -0.97079443]]
[[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.]]
[[ 0.88037034  0.7550756   0.59507063 -0.95682617]
 [ 1.70233072 -0.08758285 -0.69943137 -1.18031841]
 [-0.11772441  0.71616233  1.01490912  1.20825494]
 ...
 [-0.64612751 -0.68130778 -0.38455251 -1.40381066]
 [ 1.58490781 -0.47611016 -1.08428332  0.13269852]
 [-1.29195353 -0.68108143 -0.48951213 -1.12444535]]
[[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.]]
```

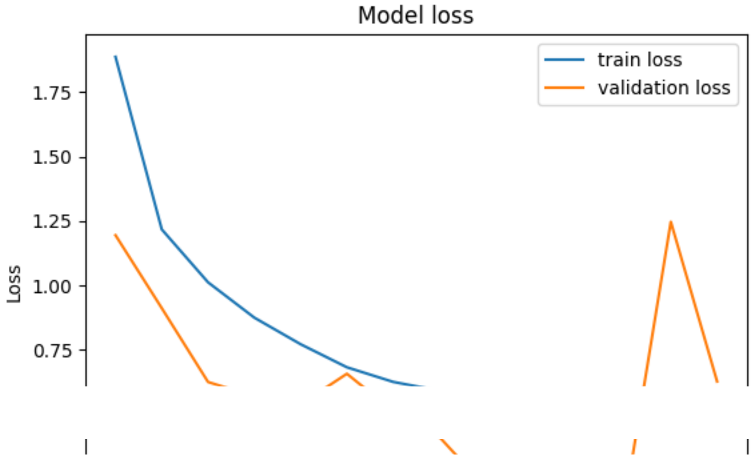
```
from keras.engine import sequential
def get_model():
    model =  tf.keras.Sequential([
        Dense(50, activation='relu', kernel_regularizer=regularizers.l2(0.005)),
        Dense(50, activation='relu'),
        Dense(60, activation='relu'),
        Dense(70, activation='relu', kernel_regularizer=regularizers.l2(0.005)),
        Dense(80, activation='relu'),
        Dense(90, activation='relu'),
        Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.005)),
        Dense(banyak_kategori, activation='softmax')
    ])

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

```
#x_train=np.asarray(x_train).astype(np.int)

#y_train=np.asarray(y_train).astype(np.int)


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 = 20,
                      validation_data = (x_test, y_test),
                      callbacks=my_callbacks)
```

```
Epoch 1/20
36221/36221 [==============================] - 253s 7ms/step - loss: 1.8872 - accuracy: 0.3240 - val_loss: 1.1942 - val_a
Epoch 2/20
36221/36221 [==============================] - 253s 7ms/step - loss: 1.2168 - accuracy: 0.5375 - val_loss: 0.9108 - val_a
Epoch 3/20
36221/36221 [==============================] - 256s 7ms/step - loss: 1.0112 - accuracy: 0.6439 - val_loss: 0.6240 - val_a
Epoch 4/20
36221/36221 [==============================] - 241s 7ms/step - loss: 0.8739 - accuracy: 0.7262 - val_loss: 0.5679 - val_a
Epoch 5/20
36221/36221 [==============================] - 257s 7ms/step - loss: 0.7707 - accuracy: 0.7808 - val_loss: 0.5371 - val_a
Epoch 6/20
36221/36221 [==============================] - 256s 7ms/step - loss: 0.6812 - accuracy: 0.8325 - val_loss: 0.6560 - val_a
Epoch 7/20
36221/36221 [==============================] - 240s 7ms/step - loss: 0.6244 - accuracy: 0.8623 - val_loss: 0.5109 - val_a
Epoch 8/20
36221/36221 [==============================] - 256s 7ms/step - loss: 0.5901 - accuracy: 0.8794 - val_loss: 0.4103 - val_a
Epoch 9/20
36221/36221 [==============================] - 241s 7ms/step - loss: 0.5433 - accuracy: 0.8990 - val_loss: 0.2148 - val_a
Epoch 10/20
36221/36221 [==============================] - 258s 7ms/step - loss: 0.5203 - accuracy: 0.9104 - val_loss: 0.2109 - val_a
Epoch 11/20
36221/36221 [==============================] - 257s 7ms/step - loss: 0.4841 - accuracy: 0.9179 - val_loss: 0.1799 - val_a
Epoch 12/20
36221/36221 [==============================] - 240s 7ms/step - loss: 0.4349 - accuracy: 0.9330 - val_loss: 0.1440 - val_a
Epoch 13/20
36221/36221 [==============================] - 241s 7ms/step - loss: 0.4380 - accuracy: 0.9382 - val_loss: 1.2464 - val_a
Epoch 14/20
36221/36221 [==============================] - 237s 7ms/step - loss: 0.4199 - accuracy: 0.9357 - val_loss: 0.6261 - val_a
```

```
def plot_accuracy(history):

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

```
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(y_test_class, classes_x)
```

```
    9056/9056 [==============================] - 18s 2ms/step
    9056/9056 [==============================] - 18s 2ms/step
    array([[ 418,  728,    0, ...,    0,    0,    0],
           [   0,    0, 1188, ...,    0,    0,    0],
           [   0,    0,   30, ...,    0,    0,    0],
           ...,
           [   0,    0,    0, ...,   79,    0,    0],
           [   0,    0,    0, ...,  704,  447,    0],
           [   0,    0,    0, ...,    0,    0, 1091]])
```

```
print(classification_report(y_test_class, classes_x))
```

```
         206       0.00      0.00      0.00      1219
         207       0.50      1.00      0.66      1201
         208       1.00      1.00      1.00      1176
         209       1.00      1.00      1.00      1167
         210       1.00      1.00      1.00      1149
         211       1.00      1.00      1.00      1135
         212       1.00      1.00      1.00      1178
         213       1.00      1.00      1.00      1179
         214       1.00      1.00      1.00      1202
         215       0.98      1.00      0.99      1147
         216       0.49      0.98      0.65      1135
         217       0.00      0.00      0.00      1175
         218       1.00      0.15      0.26      1175
         219       0.60      1.00      0.75      1153
         220       1.00      0.35      0.52      1159
         221       0.69      1.00      0.82      1163
         222       0.78      0.54      0.64      1115
         223       0.94      0.85      0.89      1147
         224       0.50      0.95      0.65      1180
         225       0.05      0.06      0.05      1189
         226       0.00      0.00      0.00      1191
         227       0.00      0.00      0.00      1189
         228       0.47      1.00      0.64      1145
         229       0.00      0.00      0.00      1177
         230       1.00      0.92      0.96      1168
         231       1.00      1.00      1.00      1209
         232       1.00      0.48      0.65      1117
         233       0.65      0.91      0.76      1202
         234       0.91      0.95      0.93      1111
         235       0.95      1.00      0.98      1129
         236       1.00      1.00      1.00      1208
         237       1.00      1.00      1.00      1115
         238       0.99      0.10      0.19      1193
         239       0.07      0.07      0.07      1199
         240       0.46      0.81      0.59      1174
         241       0.48      1.00      0.65      1174
         242       0.10      0.10      0.10      1170
         243       0.00      0.00      0.00      1149
         244       0.51      1.00      0.67      1167
         245       0.10      0.07      0.08      1206
         246       1.00      0.39      0.56      1151
         247       1.00      1.00      1.00      1091

    accuracy                           0.79    289764
   macro avg       0.79      0.79      0.77    289764
weighted avg       0.79      0.79      0.77    289764

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
  _warn_prf(average, modifier, msg_start, len(result))
```

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

```
# 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)
```

```
accuracy: 0.7946915420825224
Precision: 0.7947833781341143
Recall: 0.7947102555418392
F1-score: 0.7692691298497252
support 289764
```

✓ 0s completed at 12:38 PM ● ✕