```
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())
```

```
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

```
import tensorflow as tf
tf.convert_to_tensor(numeric_features)
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(numeric_features)


"""
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(o
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)
```

```
[[ 1.64361927  0.53985829  1.4347476  -1.36190586]
 [ 0.64552452  1.22952903  1.0848822   1.05460402]
 [ 0.9390818   0.18628699  0.52509755 -0.57968301]
 ...
 [-1.17453061 -0.7300818   0.17523214  1.40381066]
 [-0.11772441 -0.85572637 -1.71404105  0.42603209]
 [-1.35066498 -0.51733811  0.07027252 -0.56571474]]
[[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.]]
[[-1.46808789 -0.62374758  1.50472069 -1.65523943]
 [-1.64422226 -0.79452828  1.32978798 -0.81714352]
 [-0.46999314 -0.79199386 -1.57409489  0.17460332]
 ...
 [-0.88097333 -0.57334515 -1.22422948 -0.88698484]
 [ 0.76294743 -0.49186449 -1.43414872  0.38412729]
 [ 0.17583287  1.39604637  1.57469377 -1.23619147]]
[[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.]]
```

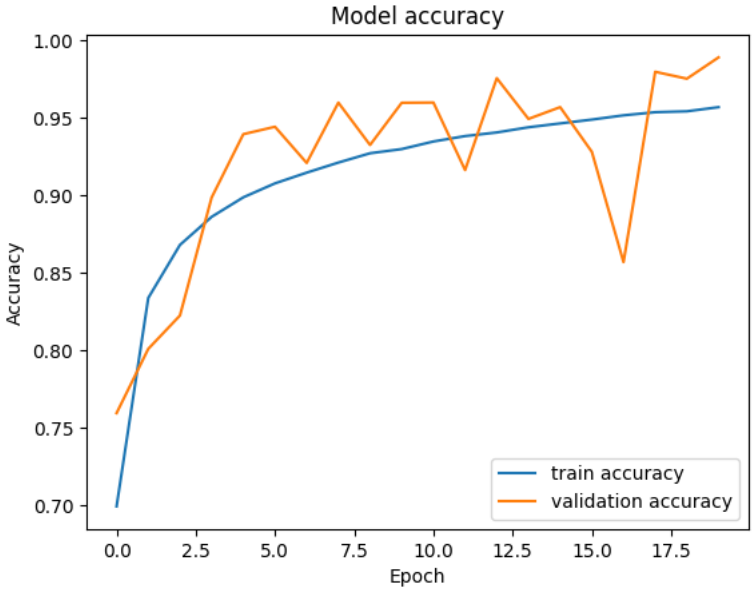
```
from keras.engine import sequential
def get_model():
  """
  model = tf.keras.Sequential([
    #normalizer,

    Dense(50, activation='relu'),
    Dense(50, activation='relu'),

    Dense(60, activation='relu'),
    Dense(70, activation='relu'),
    Dense(80, activation='relu'),

    Dense(80, activation='relu'),
```

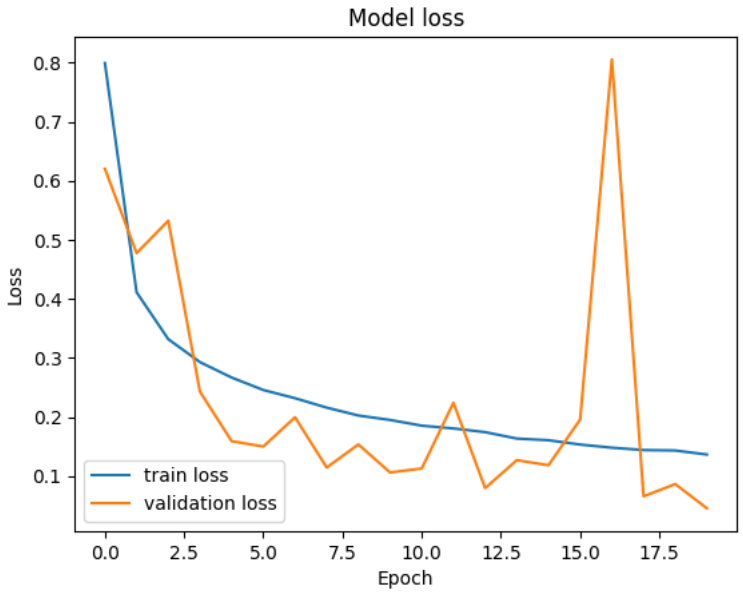
```
    Dense(90, activation='relu'),
    Dense(100, activation='relu'),



    Dense(banyak_kategori, activation = 'softmax')
  ])
  """
  model = sequential()
  model.add(Dense(50, activation='relu'))
  model.add(Dense(50, activation='relu'))
  model.add(Dense(60, activation='relu'))
  model.add(regularizers.L1L2(l1=1e-5, l2=1e-4))
  model.add(Dense(70, activation='relu'))
  model.add(Dense(80, activation='relu'))
  model.add(Dense(90, activation='relu'))
  model.add(regularizers.L2(1e-4))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(banyak_kategori, activation = 'softmax'))

  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-3),
                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)


model = get_model()



model_fit = model.fit(x_train,
                      y_train,
                      epochs = 20,
                      validation_data = (x_test, y_test))
```

```
Epoch 1/20
36221/36221 [==============================] - 118s 3ms/step - loss: 0.7991 - accuracy: 0.6994 - val_loss: 0.6201 - val_a
Epoch 2/20
36221/36221 [==============================] - 123s 3ms/step - loss: 0.4112 - accuracy: 0.8338 - val_loss: 0.4774 - val_a
Epoch 3/20
36221/36221 [==============================] - 117s 3ms/step - loss: 0.3317 - accuracy: 0.8680 - val_loss: 0.5322 - val_a
Epoch 4/20
36221/36221 [==============================] - 114s 3ms/step - loss: 0.2929 - accuracy: 0.8861 - val_loss: 0.2428 - val_a
Epoch 5/20
36221/36221 [==============================] - 123s 3ms/step - loss: 0.2668 - accuracy: 0.8987 - val_loss: 0.1590 - val_a
Epoch 6/20
36221/36221 [==============================] - 135s 4ms/step - loss: 0.2457 - accuracy: 0.9077 - val_loss: 0.1499 - val_a
Epoch 7/20
36221/36221 [==============================] - 117s 3ms/step - loss: 0.2318 - accuracy: 0.9146 - val_loss: 0.1996 - val_a
Epoch 8/20
36221/36221 [==============================] - 138s 4ms/step - loss: 0.2158 - accuracy: 0.9211 - val_loss: 0.1145 - val_a
Epoch 9/20
36221/36221 [==============================] - 140s 4ms/step - loss: 0.2025 - accuracy: 0.9271 - val_loss: 0.1535 - val_a
Epoch 10/20
36221/36221 [==============================] - 128s 4ms/step - loss: 0.1950 - accuracy: 0.9298 - val_loss: 0.1060 - val_a
Epoch 11/20
36221/36221 [==============================] - 116s 3ms/step - loss: 0.1853 - accuracy: 0.9347 - val_loss: 0.1128 - val_a
Epoch 12/20
36221/36221 [==============================] - 121s 3ms/step - loss: 0.1806 - accuracy: 0.9382 - val_loss: 0.2241 - val_a
Epoch 13/20
36221/36221 [==============================] - 114s 3ms/step - loss: 0.1743 - accuracy: 0.9405 - val_loss: 0.0795 - val_a
Epoch 14/20
36221/36221 [==============================] - 113s 3ms/step - loss: 0.1635 - accuracy: 0.9439 - val_loss: 0.1269 - val_a
Epoch 15/20
36221/36221 [==============================] - 113s 3ms/step - loss: 0.1606 - accuracy: 0.9464 - val_loss: 0.1183 - val_a
Epoch 16/20
36221/36221 [==============================] - 127s 4ms/step - loss: 0.1533 - accuracy: 0.9488 - val_loss: 0.1958 - val_a
Epoch 17/20
36221/36221 [==============================] - 115s 3ms/step - loss: 0.1480 - accuracy: 0.9516 - val_loss: 0.8054 - val_a
Epoch 18/20
36221/36221 [==============================] - 123s 3ms/step - loss: 0.1440 - accuracy: 0.9536 - val_loss: 0.0656 - val_a
Epoch 19/20
36221/36221 [==============================] - 115s 3ms/step - loss: 0.1433 - accuracy: 0.9542 - val_loss: 0.0864 - val_a
Epoch 20/20
36221/36221 [==============================] - 113s 3ms/step - loss: 0.1364 - accuracy: 0.9569 - val_loss: 0.0453 - 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 [==============================] - 22s 2ms/step
9056/9056 [==============================] - 17s 2ms/step
array([[1165,    0,    0, ...,    0,    0,    0],
       [  26, 1105,    1, ...,    0,    0,    0],
       [   0,   34, 1119, ...,    0,    0,    0],
       ...,
       [   0,    0,    0, ..., 1082,   10,    0],
       [   0,    0,    0, ...,    0, 1125,    0],
       [   0,    0,    0, ...,    0,   12, 1153]])
```

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

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1165
           1       0.95      0.98      0.96      1132
           2       0.99      0.97      0.98      1153
           3       0.97      0.98      0.97      1144
           4       1.00      0.98      0.99      1199
           5       0.98      0.98      0.98      1094
           6       0.74      0.98      0.84      1153
           7       1.00      0.65      0.79      1182
           8       0.93      1.00      0.96      1202
           9       0.99      0.93      0.96      1207
          10       1.00      0.99      0.99      1168
          11       1.00      1.00      1.00      1148
          12       0.98      1.00      0.99      1209
          13       1.00      0.98      0.99      1164
          14       0.96      1.00      0.98      1149
          15       1.00      0.96      0.98      1196
          16       0.95      1.00      0.97      1145
          17       0.98      0.94      0.96      1144
          18       0.97      0.98      0.98      1182
          19       1.00      0.97      0.99      1183
          20       1.00      1.00      1.00      1192
          21       1.00      1.00      1.00      1158
          22       1.00      1.00      1.00      1131
          23       1.00      1.00      1.00      1179
          24       1.00      1.00      1.00      1196
          25       1.00      1.00      1.00      1191
          26       1.00      1.00      1.00      1185
          27       0.98      1.00      0.99      1173
          28       1.00      0.98      0.99      1164
          29       1.00      0.98      0.99      1102
          30       0.98      1.00      0.99      1135
          31       1.00      1.00      1.00      1163
          32       0.99      1.00      1.00      1165
          33       1.00      0.99      1.00      1121
          34       1.00      1.00      1.00      1185
          35       1.00      1.00      1.00      1146
          36       1.00      1.00      1.00      1163
          37       1.00      1.00      1.00      1152
          38       1.00      1.00      1.00      1174
          39       1.00      1.00      1.00      1115
          40       1.00      1.00      1.00      1173
          41       1.00      0.99      1.00      1206
          42       0.99      1.00      1.00      1145
          43       1.00      0.98      0.99      1186
          44       0.98      1.00      0.99      1195
          45       1.00      1.00      1.00      1125
          46       1.00      1.00      1.00      1161
          47       1.00      1.00      1.00      1149
          48       1.00      1.00      1.00      1173
          49       1.00      1.00      1.00      1198
          50       1.00      0.97      0.98      1143
          51       0.97      1.00      0.98      1156
          52       1.00      0.99      0.99      1140
          53       0.98      1.00      0.99      1153
          54       1.00      0.99      1.00      1169
          55       1.00      1.00      1.00      1164
```

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

```
# 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: 0.9889703344790933
Precision: 0.9895256736249316
Recall: 0.9889645833902586
F1-score: 0.9889139504211104
support 289764
```

✓ 4s completed at 6:04 PM