```
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(c
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.tenso
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)
```

```
[[ 0.82165889 -0.34516799 -0.52449867 -0.06285719]
 [ 0.76294743  0.09807718  0.49011101  1.51555678]
 [ 1.40877344  0.04354082 -0.06967364  0.97079443]
 ...
 [-0.58741606 -0.37492604  1.39976106  1.38984239]
 [ 1.17392762  0.54896895 -0.17463326  0.71936566]
 [-1.40937644 -0.74266119  1.11986874  0.81714352]]
[[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.]]
[[ 1.11521616  0.28610276 -0.73441791  0.35619076]
 [-1.17453061 -0.62040046  1.53970723 -0.69142913]
 [-0.64612751  0.04930557  0.17523214  0.49587342]
 ...
 [ 0.52810161  0.63505792  1.60968031  1.65523943]
 [-1.29195353 -0.02749033  1.64466685  0.67746086]
 [ 0.05840996 -0.65239735 -1.36417564  0.48190515]]
[[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', kernel_regularizer=regularizers.l2(0.001)),
        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 [==============================] - 243s 7ms/step - loss: 0.8134 - accuracy: 0.7132 - val_loss: 0.7621 - val_a
Epoch 2/20
36221/36221 [==============================] - 238s 7ms/step - loss: 0.4284 - accuracy: 0.8344 - val_loss: 0.3442 - val_a
Epoch 3/20
36221/36221 [==============================] - 222s 6ms/step - loss: 0.3470 - accuracy: 0.8658 - val_loss: 0.2561 - val_a
Epoch 4/20
36221/36221 [==============================] - 221s 6ms/step - loss: 0.2994 - accuracy: 0.8854 - val_loss: 0.4478 - val_a
Epoch 5/20
36221/36221 [==============================] - 227s 6ms/step - loss: 0.2628 - accuracy: 0.9014 - val_loss: 0.3941 - val_a
```

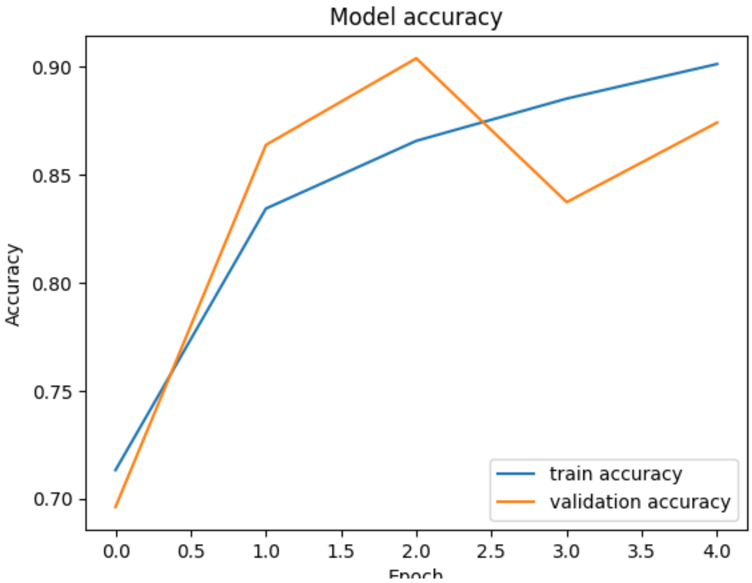
```
def plot_accuracy(history):

    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([[1119,   23,    0, ...,    0,    0,    0],
           [  32, 1074,    0, ...,    0,    0,    0],
           [   0,    0, 1125, ...,    0,    0,    0],
           ...,
           [   0,    0,    0, ..., 1080,    2,   16],
           [   0,    0,    0, ...,   16, 1159,   29],
           [   0,    0,    0, ...,    0,    0, 1213]])
```

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

```
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
      _warn_prf(average, modifier, msg_start, len(result))
                  precision    recall  f1-score   support

               0       0.97      0.98      0.98      1144
               1       0.98      0.92      0.95      1169
               2       1.00      0.97      0.98      1165
               3       0.92      1.00      0.96      1153
               4       0.99      0.98      0.99      1168
               5       1.00      0.99      0.99      1217
               6       0.95      0.99      0.97      1203
               7       0.95      0.97      0.96      1188
               8       0.98      0.92      0.95      1208
               9       0.98      0.98      0.98      1160
              10       0.98      1.00      0.99      1142
              11       1.00      0.97      0.99      1186
              12       0.97      0.89      0.93      1159
              13       1.00      0.99      0.99      1128
              14       0.90      1.00      0.95      1170
              15       1.00      0.91      0.96      1164
              16       0.93      0.83      0.88      1170
              17       0.82      0.94      0.88      1158
              18       0.84      0.97      0.90      1134
              19       0.98      0.87      0.92      1131
              20       1.00      0.93      0.96      1155
```

```
          21       1.00      0.99      0.99      1192
          22       0.93      1.00      0.96      1146
          23       0.94      1.00      0.97      1193
          24       1.00      1.00      1.00      1173
          25       1.00      0.87      0.93      1112
          26       1.00      0.94      0.97      1138
          27       0.88      1.00      0.93      1182
          28       1.00      0.98      0.99      1167
          29       1.00      0.99      0.99      1165
          30       0.98      1.00      0.99      1145
          31       0.87      0.99      0.92      1190
          32       0.99      0.81      0.89      1150
          33       1.00      0.98      0.99      1129
          34       0.95      1.00      0.97      1157
          35       1.00      0.99      0.99      1156
          36       1.00      0.97      0.99      1178
          37       0.96      0.96      0.96      1123
          38       1.00      0.98      0.99      1184
          39       0.92      1.00      0.96      1121
          40       1.00      0.96      0.98      1184
          41       0.99      0.80      0.89      1153
          42       0.79      1.00      0.88      1160
          43       1.00      0.94      0.97      1166
          44       0.94      0.99      0.97      1167
          45       1.00      0.92      0.96      1198
          46       1.00      0.99      1.00      1192
          47       0.98      0.98      0.98      1122
          48       1.00      0.99      0.99      1136
          49       0.96      1.00      0.98      1150
          50       0.70      0.99      0.82      1171
          51       1.00      0.57      0.73      1192
```

```
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)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `
  and should_run_async(code)
accuracy: 0.8742804489170497
Precision: 0.8896886902307414
Recall: 0.8743332509215013
F1-score: 0.8649489543507053
support 289764
```

Colab paid products - Cancel contracts here

✓ 3s completed at 11:14 AM