```
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 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())
```

```
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

```
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)
```

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



    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)
```

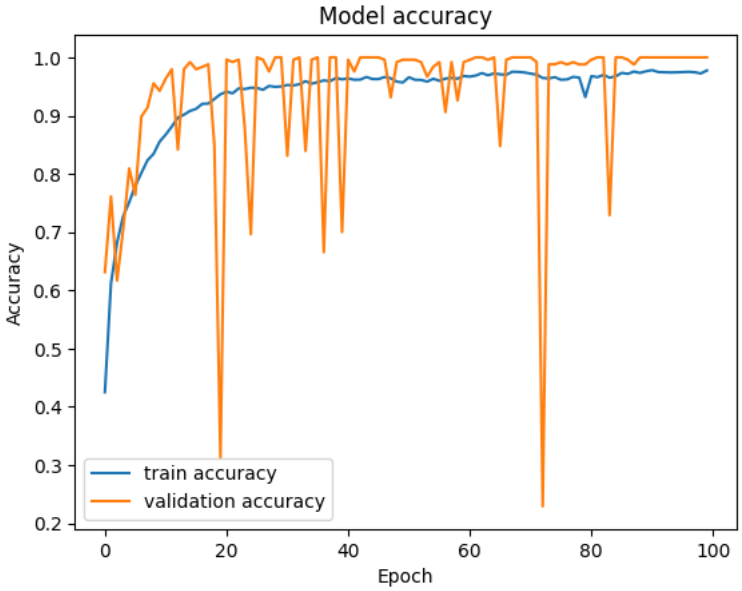
```
    <ipython-input-12-802a2bd9ef09>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence t
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
      x_train=np.asarray(x_train).astype(np.int)
    <ipython-input-12-802a2bd9ef09>:3: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence t
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
      y_train=np.asarray(y_train).astype(np.int)
```

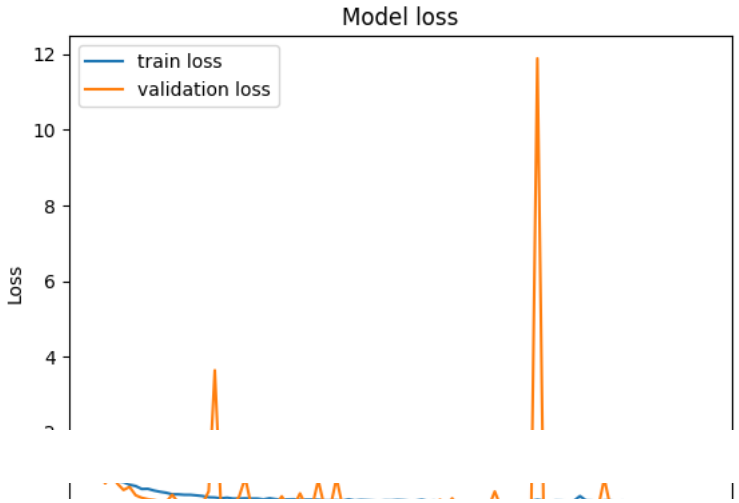
```
model = get_model()
```

```
model_fit = model.fit(x_train,
                      y_train,
                      epochs = 100,
                      validation_data = (x_test, y_test))
```

```
Epoch 70/100
36221/36221 [==============================] - 163s 4ms/step - loss: 0.1848 - accuracy: 0.9741 - val_loss: 0.0213 - val_a
Epoch 71/100
36221/36221 [==============================] - 159s 4ms/step - loss: 0.1770 - accuracy: 0.9721 - val_loss: 0.0227 - val_a
Epoch 72/100
36221/36221 [==============================] - 166s 5ms/step - loss: 0.1828 - accuracy: 0.9704 - val_loss: 0.0353 - val_a
Epoch 73/100
36221/36221 [==============================] - 159s 4ms/step - loss: 0.2169 - accuracy: 0.9646 - val_loss: 11.8844 - val_
Epoch 74/100
36221/36221 [==============================] - 161s 4ms/step - loss: 0.1890 - accuracy: 0.9639 - val_loss: 0.0414 - val_a
Epoch 75/100
36221/36221 [==============================] - 165s 5ms/step - loss: 0.1776 - accuracy: 0.9660 - val_loss: 0.0870 - val_a
Epoch 76/100
36221/36221 [==============================] - 160s 4ms/step - loss: 0.1958 - accuracy: 0.9616 - val_loss: 0.0393 - val_a
Epoch 77/100
36221/36221 [==============================] - 160s 4ms/step - loss: 0.1931 - accuracy: 0.9622 - val_loss: 0.0478 - val_a
Epoch 78/100
36221/36221 [==============================] - 167s 5ms/step - loss: 0.1842 - accuracy: 0.9665 - val_loss: 0.0347 - val_a
Epoch 79/100
36221/36221 [==============================] - 158s 4ms/step - loss: 0.1870 - accuracy: 0.9654 - val_loss: 0.0361 - val_a
Epoch 80/100
36221/36221 [==============================] - 165s 5ms/step - loss: 0.3155 - accuracy: 0.9319 - val_loss: 0.1069 - val_a
Epoch 81/100
36221/36221 [==============================] - 162s 4ms/step - loss: 0.1966 - accuracy: 0.9678 - val_loss: 0.0357 - val_a
Epoch 82/100
36221/36221 [==============================] - 158s 4ms/step - loss: 0.1919 - accuracy: 0.9660 - val_loss: 0.0235 - val_a
Epoch 83/100
36221/36221 [==============================] - 162s 4ms/step - loss: 0.1765 - accuracy: 0.9698 - val_loss: 0.0211 - val_a
Epoch 84/100
36221/36221 [==============================] - 160s 4ms/step - loss: 0.1960 - accuracy: 0.9651 - val_loss: 0.7783 - val_a
Epoch 85/100
36221/36221 [==============================] - 159s 4ms/step - loss: 0.1974 - accuracy: 0.9675 - val_loss: 0.0216 - val_a
Epoch 86/100
36221/36221 [==============================] - 162s 4ms/step - loss: 0.1768 - accuracy: 0.9733 - val_loss: 0.0203 - val_a
Epoch 87/100
36221/36221 [==============================] - 160s 4ms/step - loss: 0.1926 - accuracy: 0.9717 - val_loss: 0.0369 - val_a
Epoch 88/100
36221/36221 [==============================] - 160s 4ms/step - loss: 0.1520 - accuracy: 0.9756 - val_loss: 0.0503 - val_a
Epoch 89/100
36221/36221 [==============================] - 165s 5ms/step - loss: 0.1745 - accuracy: 0.9733 - val_loss: 0.0228 - val_a
Epoch 90/100
36221/36221 [==============================] - 159s 4ms/step - loss: 0.1674 - accuracy: 0.9762 - val_loss: 0.0195 - val_a
Epoch 91/100
36221/36221 [==============================] - 166s 5ms/step - loss: 0.1491 - accuracy: 0.9782 - val_loss: 0.0195 - val_a
Epoch 92/100
36221/36221 [==============================] - 163s 5ms/step - loss: 0.1738 - accuracy: 0.9748 - val_loss: 0.0197 - val_a
Epoch 93/100
36221/36221 [==============================] - 156s 4ms/step - loss: 0.1719 - accuracy: 0.9744 - val_loss: 0.0201 - val_a
Epoch 94/100
36221/36221 [==============================] - 159s 4ms/step - loss: 0.1653 - accuracy: 0.9741 - val_loss: 0.0173 - val_a
Epoch 95/100
36221/36221 [==============================] - 155s 4ms/step - loss: 0.1613 - accuracy: 0.9743 - val_loss: 0.0180 - val_a
Epoch 96/100
36221/36221 [==============================] - 157s 4ms/step - loss: 0.1728 - accuracy: 0.9747 - val_loss: 0.0241 - 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
```

```
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, y_pred_class)
```

```
print(classification_report(y_test_class, y_pred_class))
```

✓ 0s completed at 1:07 PM