Mount Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Prepare the Dataset

Import library

```
import tensorflow as tf
import subprocess
```

Copy dataset from drive

```
cp /content/drive/MyDrive/indo_food_datasets/jadi/food-dataset-500.zip /content/
```

```
cp -R /content/drive/MyDrive/indo_food_datasets/jadi/food-dataset-500 /content/
```

Unzip file

```
import zipfile

# Extract the archive
local_zip = './food-dataset-500.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/food-dataset')
zip_ref.close()

# local_zip = './rps-test-set.zip'
# zip_ref = zipfile.ZipFile(local_zip, 'r')
# zip_ref.extractall('tmp/rps-test')
# zip_ref.close()
```

Delete unused dataset

```
food_classes = ['bakso','soto','pempek','pepes','rendang','onde-onde']
```

```
for food_class in food_classes:
  subprocess.run(["rm", "-rf", "/content/food-dataset-500/test/"+food_class])
  subprocess.run(["rm", "-rf", "/content/food-dataset-500/train/"+food_class])
```

```
rm -rf /content/tmp/food-dataset/food-dataset-500/test/soto
```

```
rm -rf /content/tmp/food-dataset/food-dataset-500/train/soto
```

Build Model

```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    # The fourth convolution
    # tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    # tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    #tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])

# Print the model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 148, 148, 32)      896

max_pooling2d_3 (MaxPooling  (None, 74, 74, 32)       0
2D)

conv2d_4 (Conv2D)            (None, 72, 72, 32)        9248

max_pooling2d_4 (MaxPooling  (None, 36, 36, 32)       0
2D)
```

```
 conv2d_5 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 17, 17, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 18496)             0

 dense_2 (Dense)             (None, 128)               2367616

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 2,396,901
Trainable params: 2,396,901
Non-trainable params: 0
_________________________________________________________________
```

```
# Set the training parameters
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Prepare the ImageDataGenerator

```
from keras_preprocessing.image import ImageDataGenerator

TRAINING_DIR = "/content/food-dataset-500/train"
training_datagen = ImageDataGenerator(
      rescale = 1./255,
        rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

VALIDATION_DIR = "/content/food-dataset-500/test"
validation_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_directory(
    TRAINING_DIR,
    target_size=(150,150),
    class_mode='categorical',
  batch_size=126
)

validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(150,150),
```

```
    class_mode='categorical',
  batch_size=126
  #batch_size=126
)
```

```
Found 2000 images belonging to 5 classes.
Found 500 images belonging to 5 classes.
```

Train the model and evaluate the results

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    '''
    Halts the training after reaching 60 percent accuracy

    Args:
      epoch (integer) - index of epoch (required but unused in the function definition below)
      logs (dict) - metric results from the training epoch
    '''

    # Check accuracy
    # if(logs.get('loss') < 0.4):

    #   # Stop if threshold is met
    #   print("\nLoss is lower than 0.4 so cancelling training!")
    #   self.model.stop_training = True
    if(logs.get('val_accuracy') > 0.75):
      # Stop if threshold is met
      print("\nVal_accuracy is higher than 0.8 so cancelling training!")
      self.model.stop_training = True

# Instantiate class
callbacks = myCallback()
```

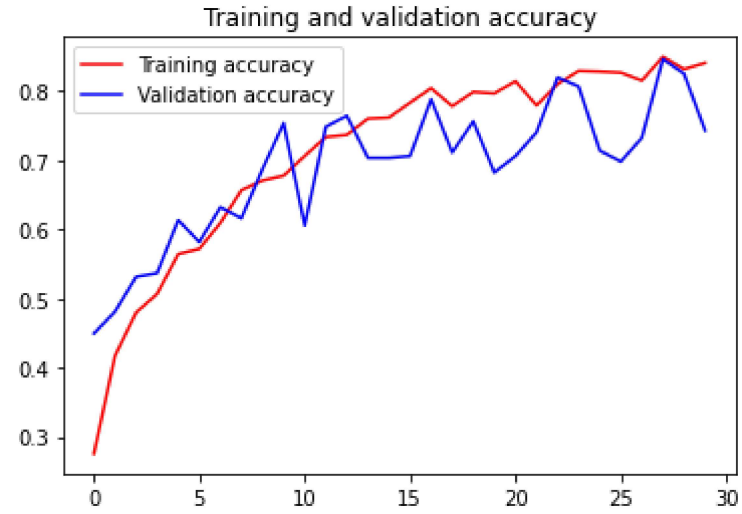
```
history = model.fit(train_generator, epochs=30, validation_data = validation_generator, verbo
```

```
Epoch 2/30
16/16 [==============================] - 15s 909ms/step - loss: 1.3774 - accuracy: 0.4
Epoch 3/30
16/16 [==============================] - 15s 907ms/step - loss: 1.2426 - accuracy: 0.4
Epoch 4/30
16/16 [==============================] - 14s 902ms/step - loss: 1.1483 - accuracy: 0.
Epoch 5/30
16/16 [==============================] - 14s 899ms/step - loss: 1.0404 - accuracy: 0.
Epoch 6/30
16/16 [==============================] - 14s 901ms/step - loss: 1.0076 - accuracy: 0.
Epoch 7/30
16/16 [==============================] - 14s 899ms/step - loss: 0.9327 - accuracy: 0.
Epoch 8/30
16/16 [==============================] - 15s 915ms/step - loss: 0.8710 - accuracy: 0.
Epoch 9/30
```

```
16/16 [==============================] - 14s 901ms/step - loss: 0.8337 - accuracy: 0.
Epoch 10/30
16/16 [==============================] - 14s 900ms/step - loss: 0.7937 - accuracy: 0.
Epoch 11/30
16/16 [==============================] - 14s 899ms/step - loss: 0.7465 - accuracy: 0.
Epoch 12/30
16/16 [==============================] - 15s 912ms/step - loss: 0.7049 - accuracy: 0.
Epoch 13/30
16/16 [==============================] - 15s 903ms/step - loss: 0.6779 - accuracy: 0.
Epoch 14/30
16/16 [==============================] - 14s 901ms/step - loss: 0.6324 - accuracy: 0.
Epoch 15/30
16/16 [==============================] - 15s 909ms/step - loss: 0.6353 - accuracy: 0.
Epoch 16/30
16/16 [==============================] - 15s 910ms/step - loss: 0.5893 - accuracy: 0.
Epoch 17/30
16/16 [==============================] - 15s 908ms/step - loss: 0.5309 - accuracy: 0.
Epoch 18/30
16/16 [==============================] - 14s 901ms/step - loss: 0.5852 - accuracy: 0.
Epoch 19/30
16/16 [==============================] - 14s 895ms/step - loss: 0.5492 - accuracy: 0.
Epoch 20/30
16/16 [==============================] - 14s 898ms/step - loss: 0.5355 - accuracy: 0.
Epoch 21/30
16/16 [==============================] - 14s 895ms/step - loss: 0.5091 - accuracy: 0.
Epoch 22/30
16/16 [==============================] - 14s 894ms/step - loss: 0.5786 - accuracy: 0.
Epoch 23/30
16/16 [==============================] - 14s 895ms/step - loss: 0.5130 - accuracy: 0.
Epoch 24/30
16/16 [==============================] - 14s 884ms/step - loss: 0.4716 - accuracy: 0.
Epoch 25/30
16/16 [==============================] - 14s 895ms/step - loss: 0.4603 - accuracy: 0.
Epoch 26/30
16/16 [==============================] - 14s 888ms/step - loss: 0.4535 - accuracy: 0.
Epoch 27/30
16/16 [==============================] - 14s 893ms/step - loss: 0.4721 - accuracy: 0.
Epoch 28/30
16/16 [==============================] - 14s 899ms/step - loss: 0.4045 - accuracy: 0.
Epoch 29/30
16/16 [==============================] - 14s 897ms/step - loss: 0.4573 - accuracy: 0.
Epoch 30/30
16/16 [==============================] - 14s 888ms/step - loss: 0.4167 - accuracy: 0.
```

```
# Train the model
history = model.fit(train_generator, epochs=20, validation_data = validation_generator, valid


import matplotlib.pyplot as plt

# Plot the results
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()

plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```

Model Prediction

```
## CODE BLOCK FOR NON-SAFARI BROWSERS
## SAFARI USERS: PLEASE SKIP THIS BLOCK AND RUN THE NEXT ONE INSTEAD

import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
```

```
print(fn)
print(classes)
```

Choose Files 12 files

- **710.png**(image/png) - 6544 bytes, last modified: 5/13/2022 - 100% done
- **718.png**(image/png) - 8671 bytes, last modified: 5/13/2022 - 100% done
- **719.png**(image/png) - 6778 bytes, last modified: 5/13/2022 - 100% done
- **721.png**(image/png) - 8701 bytes, last modified: 5/13/2022 - 100% done
- **724.png**(image/png) - 9474 bytes, last modified: 5/13/2022 - 100% done
- **726.png**(image/png) - 5476 bytes, last modified: 5/13/2022 - 100% done
- **732.png**(image/png) - 11569 bytes, last modified: 5/13/2022 - 100% done
- **734.png**(image/png) - 7473 bytes, last modified: 5/13/2022 - 100% done
- **736.png**(image/png) - 7320 bytes, last modified: 5/13/2022 - 100% done
- **737.png**(image/png) - 8108 bytes, last modified: 5/13/2022 - 100% done
- **749.png**(image/png) - 6049 bytes, last modified: 5/13/2022 - 100% done
- **823.png**(image/png) - 6649 bytes, last modified: 5/13/2022 - 100% done

```
Saving 710.png to 710.png
Saving 718.png to 718.png
Saving 719.png to 719.png
Saving 721.png to 721.png
Saving 724.png to 724.png
Saving 726.png to 726.png
Saving 732.png to 732.png
Saving 734.png to 734.png
Saving 736.png to 736.png
Saving 737.png to 737.png
Saving 749.png to 749.png
Saving 823.png to 823.png
710.png
[[0.0000000e+00 0.0000000e+00 0.0000000e+00 7.0753937e-15 1.0000000e+00]]
718.png
[[0. 0. 0. 0. 1.]]
719.png
[[0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 4.607697e-24]]
721.png
[[0. 0. 0. 0. 1.]]
724.png
[[0. 0. 0. 0. 1.]]
726.png
[[0. 0. 0. 0. 1.]]
732.png
[[0. 0. 0. 0. 1.]]
734.png
[[0. 0. 0. 0. 1.]]
736.png
[[0. 0. 0. 0. 1.]]
737.png
[[0. 0. 0. 0. 1.]]
749.png
[[0. 0. 0. 0. 1.]]
823.png
[[0. 0. 0. 0. 1.]]
```

Finish