TUGAS BESAR KULIAH SISTEM PENGUKURAN BERBASIS CITRA

Nama : Ronggur Mahendra Widya Putra

NIM : 13519008

In [1]:

```
# Import Library
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
import scipy
from skimage import color, data, restoration
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from random import uniform
from tensorflow.keras.layers import BatchNormalization
```

Loading Data dari folder ./Data/ClassifiedData

In [2]:

```
ClassifiedDataDir = "./Data/ClassifiedData/Train"
BATCH_SIZE = 32
IMG_SIZE = (128, 128)
train_dataset = tf.keras.utils.image_dataset_from_directory(
    directory = ClassifiedDataDir,
        labels='inferred',
        label_mode='int',
        class_names=None,
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        image_size=IMG_SIZE,
        shuffle=True,
        seed=1337,
        subset="training",
        validation_split=0.1,
        interpolation='bilinear',
        follow_links=False,
        crop_to_aspect_ratio=False,
        # rescale = 1./255,
    )
validation_dataset = tf.keras.utils.image_dataset_from_directory(
    directory = ClassifiedDataDir,
        labels='inferred',
        label_mode='int',
        class_names=None,
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        image_size=IMG_SIZE,
        shuffle=True,
        seed=1337,
        subset="validation",
        validation_split=0.1,
        interpolation='bilinear',
        follow_links=False,
        crop_to_aspect_ratio=False,
        # rescale = 1./255,
    )


test_dataset = tf.keras.utils.image_dataset_from_directory(
    directory = "./Data/ClassifiedData/Test",
        labels='inferred',
        label_mode='int',
        class_names=None,
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        image_size=IMG_SIZE,
        shuffle=True,
        seed=None,
        interpolation='bilinear',
        follow_links=False,
        crop_to_aspect_ratio=False,
        # rescale = 1./255,
    )
```

```
Found 3525 files belonging to 24 classes.
Using 3173 files for training.
```

```
Found 3525 files belonging to 24 classes.
Using 352 files for validation.
Found 312 files belonging to 24 classes.
```

List Class

In [3]:

```
# for element in dataset.as_numpy_iterator():
#     print(element)
class_names = train_dataset.class_names
print(class_names)
print(len(class_names))
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']
24
```

In [4]:

```
# print('Number of training batches: %d' % tf.data.experimental.cardinality(train_dataset).
# print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_da
```

Optimization buffer untuk Dataset

In [5]:

```
# DataSet
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Model 1

In [6]:

```
# # # Model 1 Initialize
# input_size = 128
# filter_size = 14
# num_filter = 8
# maxpool_size = 2
# batch_size = BATCH_SIZE
# epochs = 30

# model1 = tf.keras.models.Sequential([
#     tf.keras.layers.InputLayer(input_shape=(input_size, input_size, 3)),
#     tf.keras.layers.Reshape((256, 256 * 3)),
#     tf.keras.layers.Bidirectional(tf.keras.layers.LSTM( 256, return_sequences=True, retur
#     tf.keras.layers.Bidirectional(tf.keras.layers.LSTM( 256 )),
#     tf.keras.layers.Dense( 256 ),
#     tf.keras.layers.Dropout(.2),
#     tf.keras.layers.Dense( 256 ),
# ])

# # model = models.Sequential()
# # model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(input_size, input_s
# # model.add(layers.MaxPooling2D((2, 2)))
# # model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# # model.add(layers.MaxPooling2D((2, 2)))
# # model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# model1.add(tf.keras.layers.Flatten())
# # model.add(tf.keras.layers.Dense(24))
# model1.add(tf.keras.layers.Dense(24, activation = 'softmax'))

# # model.add(layers.Flatten())
# # model.add(layers.Dense(64, activation='relu'))
# # model.add(layers.Dense(24))

# model1.summary()
```

In [7]:

```
# # Optimizer
# optimizer = tf.keras.optimizers.Nadam(
#     learning_rate=0.00001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
#     name='Nadam'
# ) # 0.00001

# # Loss Fn
# lossfn = tf.keras.losses.SparseCategoricalCrossentropy( from_logits=False, reduction=tf.k

# # Model Summary
# model.compile(
#     optimizer="rmsprop",
#     loss="sparse_categorical_crossentropy",
#     metrics=["sparse_categorical_accuracy"],
# )
```

Creating CNN Model

In [8]:

```
# Model 2

input_size = 128
filter_size = 3
num_filter = 8
maxpool_size = 2
batch_size = BATCH_SIZE
epochs = 30

steps_per_epoch = 24720/batch_size

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(16, (filter_size,filter_size),
                 input_shape= (input_size,input_size,3),
                 activation ='relu',
                 padding='same'))
model.add(tf.keras.layers.Conv2D(16, (filter_size,filter_size),
                 input_shape= (input_size,input_size,3),
                 activation ='relu',
                 padding='same'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(maxpool_size, maxpool_size),strides=1))
model.add(tf.keras.layers.Dropout(uniform(0, 1)))

model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(maxpool_size, maxpool_size),strides=2))
model.add(tf.keras.layers.Dropout(uniform(0, 1)))

model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(BatchNormalization())

model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(tf.keras.layers.Conv2D(32, (filter_size,filter_size),
                 activation='relu',
                 padding='valid'))
model.add(BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(maxpool_size, maxpool_size),strides=2))
model.add(tf.keras.layers.Dropout(uniform(0, 1)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(24,activation='softmax'))
```

In [9]:

```
METRICS = [ 'accuracy']#, 'precision','recall']
model.compile( optimizer= tf.keras.optimizers.Adam(lr=0.001),loss='sparse_categorical_cross

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 16)      448

 conv2d_1 (Conv2D)           (None, 128, 128, 16)      2320

 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
 )

 dropout (Dropout)           (None, 127, 127, 16)      0

 conv2d_2 (Conv2D)           (None, 125, 125, 32)      4640

 conv2d_3 (Conv2D)           (None, 123, 123, 32)      9248

 batch_normalization (BatchN  (None, 123, 123, 32)     128
 ormalization)

 max_pooling2d_1 (MaxPooling  (None, 61, 61, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 61, 61, 32)        0

 conv2d_4 (Conv2D)           (None, 59, 59, 32)        9248

 conv2d_5 (Conv2D)           (None, 57, 57, 32)        9248

 batch_normalization_1 (Batc  (None, 57, 57, 32)       128
 hNormalization)

 conv2d_6 (Conv2D)           (None, 55, 55, 32)        9248

 conv2d_7 (Conv2D)           (None, 53, 53, 32)        9248

 batch_normalization_2 (Batc  (None, 53, 53, 32)       128
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 32)       0
 2D)

 dropout_2 (Dropout)         (None, 26, 26, 32)        0

 flatten (Flatten)           (None, 21632)             0

 dense (Dense)               (None, 120)               2595960

 dense_1 (Dense)             (None, 24)                2904

=================================================================
Total params: 2,652,896
Trainable params: 2,652,704
```

```
Non-trainable params: 192
_________________________________________________________________

c:\Users\ZEPHYRUS GU502GU\AppData\Local\Programs\Python\Python39\lib\site-pa
ckages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argu
ment is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Makingsure Training using GPU

In [10]:

```
print(tf.test.is_built_with_cuda())
print(tf.config.list_physical_devices('GPU'))
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
True
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
Num GPUs Available:  1
```

Training

In [11]:

```
# Training
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data = (validation_dataset)
    )
```

```
Epoch 1/100
100/100 [==============================] - 18s 104ms/step - loss: 3.2411 -
accuracy: 0.3577 - val_loss: 3.0462 - val_accuracy: 0.4205
Epoch 2/100
100/100 [==============================] - 11s 108ms/step - loss: 0.8062 -
accuracy: 0.7476 - val_loss: 1.9172 - val_accuracy: 0.5483
Epoch 3/100
100/100 [==============================] - 11s 105ms/step - loss: 0.2797 -
accuracy: 0.9136 - val_loss: 2.2510 - val_accuracy: 0.4858
Epoch 4/100
100/100 [==============================] - 11s 107ms/step - loss: 0.1822 -
accuracy: 0.9423 - val_loss: 3.6804 - val_accuracy: 0.4545
Epoch 5/100
100/100 [==============================] - 10s 94ms/step - loss: 0.1255 -
accuracy: 0.9625 - val_loss: 2.4303 - val_accuracy: 0.4602
Epoch 6/100
100/100 [==============================] - 10s 94ms/step - loss: 0.0902 -
accuracy: 0.9701 - val_loss: 16.0349 - val_accuracy: 0.2159
Epoch 7/100
```

In [12]:

```
loss, acc = model.evaluate(test_dataset)
print(loss, acc)
```

```
10/10 [==============================] - 1s 68ms/step - loss: 0.0069 - accur
acy: 0.9968
0.006871971767395735 0.9967948794364929
```

Saving model to load later

In [13]:

```
model.save('saved_model/FinalModel')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_co
nvolution_op, _jit_compiled_convolution_op while saving (showing 5 of 8). Th
ese functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: saved_model/FinalModel\assets

INFO:tensorflow:Assets written to: saved_model/FinalModel\assets
```

Model's Performance

In [16]:

```
TrainingAccuracy = history.history['accuracy'][len(history.history['accuracy'])-1]
print("Train accuracy: {:5.2f}%".format(100 * TrainingAccuracy))
print("Test accuracy: {:5.2f}%".format(100 * acc))
```

```
Train accuracy: 99.81%
Test accuracy: 99.68%
```

Training History

In [15]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

# test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

Out[15]:

```
<matplotlib.legend.Legend at 0x1f74b241670>
```