TECHNICAL
REPORT

Blockchain

TEAM MEMBER

HABIB IRFAN MAHAASIN

NIM : 1103194186

BRILLIANT FRIEZKA AINA

NIM : 1103194186

GILANG RAMADHAN UTAMA

NIM : 1103194051

PENGGUNAAN RESNET

DALAM SEBUAH KASUS

MACHINE LEARNING

RESNET

ResNet atau Residual Network adalah jenis arsitektur Convolutional Neural Network (CNN) dengan menggunakan model yang sudah dilatih sebelumnya

IMPORT LIBRARY

In [38]:

```
import os
import h5py
import math
import numpy as np
import tensorflow as tf
from keras.utils.data_utils import get_file
from tensorflow.keras.applications.imagenet_utils import preprocess_in
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow import keras
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Add, Dense, Activation, Zer
from tensorflow.keras.preprocessing import image
from keras.utils import layer_utils
import scipy.misc
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
%matplotlib inline
import tensorflow.keras.backend as K
K.set_image_data_format('channels_last')
```

Library yang digunakan yaitu

- os
- h5py
- Math
- Numpy
- Tensorflow
- Spicy
- Matplotlib

MEMUAT DATA

In [ ]:

```
def load_dataset():
    train_dataset = h5py.File('train_signs.h5', "r")

    # Train set features
    train_set_x_orig = np.array(train_dataset["train_set_x"][:])
    # Train set labels
    train_set_y_orig = np.array(train_dataset["train_set_y"][:])

    test_dataset = h5py.File('test_signs.h5', "r")

    # Test set features
    test_set_x_orig = np.array(test_dataset["test_set_x"][:])
    # Test set labels
    test_set_y_orig = np.array(test_dataset["test_set_y"][:])

    # Test of classes
    classes = np.array(test_dataset["list_classes"][:])

    # reshape
    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes
```

Dalam memuat dataset terdapat dua jenis yaitu ***dataset train*** (data uji) dan ***dataset test*** (data latih)

MENGIDENTIFIKASI BLOCK

In [ ]:

```
def identity_block(X, f, filters, stage, block):

  # Defining base name for block
  # Stage is the block position in the network
  conv_base_name = 'res' + str(stage) + block + '_'
  bn_base_name = 'bn' + str(stage) + block + '_'

  # Defining the number of filters in each layer of the main path
  # f3 must be equal to n_C. That way dimensions of the third component will match the dimension of origi
  f1, f2, f3 = filters

  # Batch normalization must be performed on the 'channels' axis for input.
  # We use 3 for this case.
  bn_axis = 3

  # Save input for "addition" to last layer output
  X_skip_connection = X

# Building layers/component of identity block using Keras functional API

  # First component/layer of main path
  X = Conv2D(filters= f1, kernel_size = (1,1), strides = (1,1), padding='valid', name=conv_base_name+'fir
  X = BatchNormalization(axis=bn_axis, name=bn_base_name+'first_component')(X)
  X = Activation('relu')(X)

  # Second component/layer of main path
  X = Conv2D(filters= f2, kernel_size = (f,f), strides = (1,1), padding='same', name=conv_base_name+'seco
  X = BatchNormalization(axis=bn_axis, name=bn_base_name+'second_component')(X)
  X = Activation('relu')(X)

  # Third component/layer of main path
  X = Conv2D(filters= f3, kernel_size = (1,1), strides = (1,1), padding='valid', name=conv_base_name+'thi
  X = BatchNormalization(axis=bn_axis, name=bn_base_name+'third_component')(X)

  # "Addition step" - skip-connection value merges with main path
  # NOTE: both values have same dimensions at this point, so no operation is required to match dimensions
  X = Add()([X, X_skip_connection])
  X = Activation('relu')(X)
```

CONVOLUTIONAL BLOCK

In [ ]:

```
def convolutional_block(X, f, filters, stage, block, s = 2):
    conv_base_name = 'res' + str(stage) + block + '_'
    bn_base_name = 'bn' + str(stage) + block + '_'

    f1, f2, f3 = filters

    bn_axis = 3

    X_skip_connection = X

    X = Conv2D(f1, (1, 1), strides = (s,s), padding = 'valid', name = conv_base_name + 'first_component', kernel_initializer = glorot_uniform(seed=
    X = BatchNormalization(axis = bn_axis, name = bn_base_name + 'first_component')(X)
    X = Activation('relu')(X)

    X = Conv2D(f2,  kernel_size = (f, f), strides = (1,1), padding = 'same', name = conv_base_name + 'second_component', kernel_initializer = gloro
    X = BatchNormalization(axis = bn_axis, name = bn_base_name + 'second_component')(X)
    X = Activation('relu')(X)

    X = Conv2D(f3, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_base_name + 'third_component', kernel_initializer = glorot
    X = BatchNormalization(axis = bn_axis, name = bn_base_name + 'third_component')(X)

    X_skip_connection = Conv2D(f3, (1, 1), strides = (s,s), padding = 'valid', name = conv_base_name + 'merge', kernel_initializer = glorot_uniform
    X_skip_connection = BatchNormalization(axis = 3, name = bn_base_name + 'merge')(X_skip_connection)

    X = Add()([X, X_skip_connection])
    X = Activation('relu')(X)

    return X
```

MACHINE LEARNING

RESNET

Melakukan implementasi ResNet

In [ ]:

```
def ResNet50(input_shape = (64, 64, 3), classes = 6):
    X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)
    # Tahap 1
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv_1', kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Tahap 2
    X = convolutional_block(X, f = 3, filters = [64, 64, 256], stage = 2, block='a', s = 1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

    # Tahap 3
    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Tahap 4
    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Tahap 5
    X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = AveragePooling2D((2, 2), name='avg_pool')(X)

    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform(seed=0))(X)

    model = Model(inputs = X_input, outputs = X, name='ResNet50')
```

TRAIN THE RESNET

In [ ]:
```
model.fit(X_train, Y_train, epochs = 20, batch_size = 32)
```

```
Epoch 1/20
34/34 [==============================] - 194s 5s/step - loss: 1.8266 - accuracy: 0.4713
Epoch 2/20
34/34 [==============================] - 192s 6s/step - loss: 0.6448 - accuracy: 0.7759
Epoch 3/20
34/34 [==============================] - 186s 5s/step - loss: 0.5320 - accuracy: 0.8417
Epoch 4/20
34/34 [==============================] - 186s 5s/step - loss: 0.2855 - accuracy: 0.8972
Epoch 5/20
34/34 [==============================] - 188s 6s/step - loss: 0.3545 - accuracy: 0.8991
Epoch 6/20
34/34 [==============================] - 191s 6s/step - loss: 0.2191 - accuracy: 0.9361
Epoch 7/20
34/34 [==============================] - 189s 6s/step - loss: 0.1309 - accuracy: 0.9574
Epoch 8/20
34/34 [==============================] - 189s 6s/step - loss: 0.0624 - accuracy: 0.9787
Epoch 9/20
34/34 [==============================] - 192s 6s/step - loss: 0.0450 - accuracy: 0.9833
Epoch 10/20
34/34 [==============================] - 188s 6s/step - loss: 0.1013 - accuracy: 0.9741
Epoch 11/20
34/34 [==============================] - 184s 5s/step - loss: 0.1754 - accuracy: 0.9546
Epoch 12/20
34/34 [==============================] - 182s 5s/step - loss: 0.1384 - accuracy: 0.9546
Epoch 13/20
34/34 [==============================] - 182s 5s/step - loss: 0.1195 - accuracy: 0.9639
Epoch 14/20
34/34 [==============================] - 181s 5s/step - loss: 0.0583 - accuracy: 0.9815
Epoch 15/20
34/34 [==============================] - 179s 5s/step - loss: 0.0683 - accuracy: 0.9759
Epoch 16/20
34/34 [==============================] - 184s 5s/step - loss: 0.0445 - accuracy: 0.9806
Epoch 17/20
34/34 [==============================] - 185s 5s/step - loss: 0.0305 - accuracy: 0.9870
Epoch 18/20
34/34 [==============================] - 183s 5s/step - loss: 0.0607 - accuracy: 0.9824
Epoch 19/20
34/34 [==============================] - 186s 5s/step - loss: 0.1729 - accuracy: 0.9556
Epoch 20/20
34/34 [==============================] - 178s 5s/step - loss: 0.1182 - accuracy: 0.9667
```

TESTING
THE PERFORMANCE

In [ ]:

```
predictions = model.evaluate(X_test, Y_test)
print("Loss = " + str(predictions[0]))
print("Test Accuracy = " + str(predictions[1]))
```

```
4/4 [==============================] - 3s 445ms/step - loss: 1.2145 - accuracy: 0.6750
Loss = 1.2144526243209839
Test Accuracy = 0.675000011920929
```

In [ ]:

```
new_model = load_model('RESNET50.h5')
predictions = new_model.evaluate(X_test, Y_test)
print(predictions)
```

```
4/4 [==============================] - 3s 389ms/step - loss: 1.2145 - accuracy: 0.6750
[1.2144526243209839, 0.675000011920929]
```

TERIMA KASIH

BLOCKCHAIN