Tensorflow

Library yang digunakan (Tensorflow version 2.x)

- __future__
- H5py
- Math
- Numpy
- Matplotlib
- Random
- cv2

Dataset? (KMNIST - Kuzushiji)

https://www.kaggle.com/anokas/kuzushiji

Historical Document, Japanese handwritten Kuzushiji. Awal mula dari dataset ini ada yaitu karena banyak penduduk asli jepang yang tidak dapat membaca huruf tersebut.

10 kelas, 60.000 training dataset, 10.000 test dataset

Class Map :

0 = お
1 = き
2 = す
3 = つ
4 = な
5 = は
6 = ま
7 = や
8 = れ
9 = を

How we load dataset?

- Download from kaggle the “.npz”
- Using the load function from numpy library

```
data = np.load('/content/drive/My Drive/Colab Notebooks/kmnist-test-imgs.npz')
test_images = data['arr_0']
data = np.load('/content/drive/My Drive/Colab Notebooks/kmnist-test-labels.npz')
test_labels = data['arr_0']
data = np.load('/content/drive/My Drive/Colab Notebooks/kmnist-train-imgs.npz')
train_images = data['arr_0']
data = np.load('/content/drive/My Drive/Colab Notebooks/kmnist-train-labels.npz')
train_labels = data['arr_0']
```

- Testing shape :

```
print(test_images.shape)
print(test_labels.shape)
print(train_images.shape)
print(train_labels.shape)
```

```
(10000, 28, 28)
(10000,)
(60000, 28, 28)
(60000,)
```

Appearance of the images

28x28

Deep Learning Model

1. Layer 1 : CNN, 6 Kernel, 3x3, act_f = relu, no_bias
2. Maxpooling : 2x2
3. Layer 2 : CNN, 6 Kernel, 3x3, act_f = relu, no_bias
4. Maxpooling : 2x2
5. Flatten
6. Layer 3 : Dense Layer, 64 kernel, act_f = relu
7. Layer output : Dense Layer, 10 kernel, act_f = relu

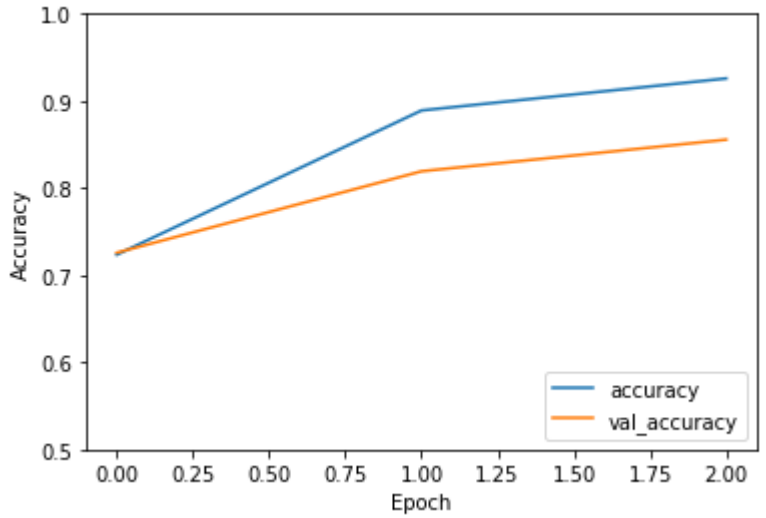
Summary

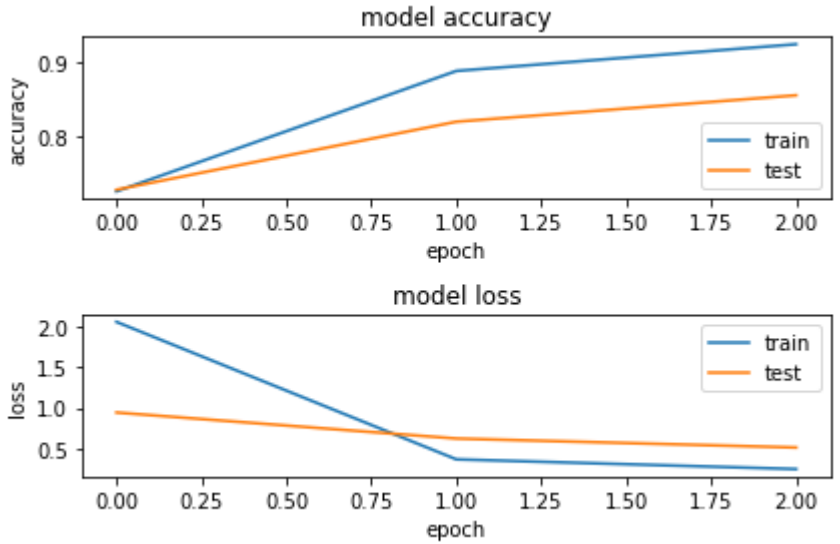
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_1 (Conv2D)             (None, 26, 26, 6)         54
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 6)         0
_________________________________________________________________
layer_2 (Conv2D)             (None, 11, 11, 6)         324
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 6)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 150)               0
_________________________________________________________________
layer_3 (Dense)              (None, 64)                9600
_________________________________________________________________
layer_Output (Dense)         (None, 10)                640
=================================================================
Total params: 10,618
Trainable params: 10,618
Non-trainable params: 0
```

Compiler & Training

- Optimizer = adam
- Loss = Sparse Categorical Crossentropy
- Ephocs = 3
- Test Accuracy = 85,56%

```
10000/1 - 2s - loss: 0.8121 - accuracy: 0.8556
```





Weight Shape ?

```
print(layer_1_data.shape)
print(layer_2_data.shape)
print(layer_3_data.shape)
print(layer_out_data.shape)
```

```
(3, 3, 1, 6)
(3, 3, 6, 6)
(150, 64)
(64, 10)
```

Test Accuracy of Quantized Weight and Images

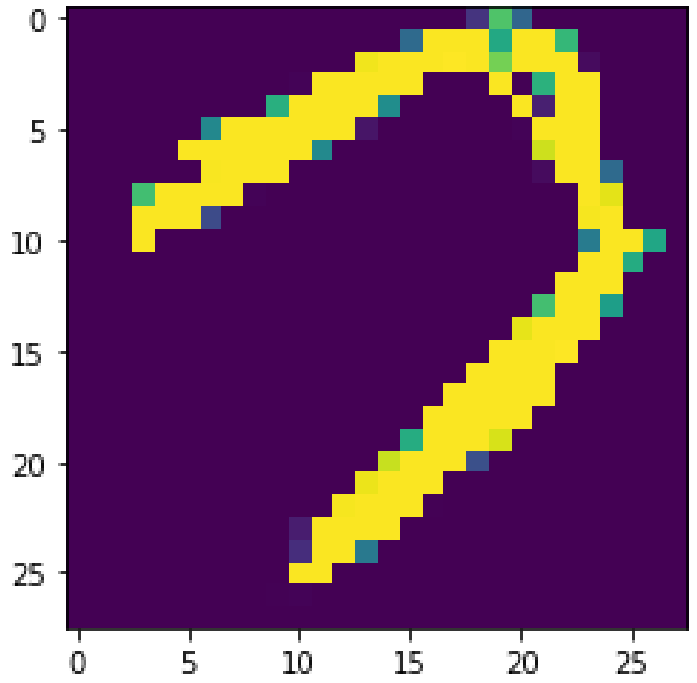
```
val_num_keras_model : 8556
val_num_quantized_model : 8558
val val_num_keras_model : 85.56 %
val quantized  : 85.58 %
```

Testing with image from user

```
#Train from file Label 3

indata = timg3/255
res = Model2(indata.reshape(1,28,28))
plt.imshow(timg3)
print("Gambar ini diklasifikasikan sebagai :")
print(np.argmax(res))
```

```
Gambar ini diklasifikasikan sebagai :
3
```



Weight Value

```
Weights Layer 1 :

[[[[ -1.  13. -36.]
   [  7.  12. -54.]
   [  4.  51.   1.]]]
 [[[ 18.  26.  44.]
   [ 27.  -2. -31.]
   [  4.  17.  25.]]]
 [[[ -1. -24. -70.]
   [-69.  24. -20.]
   [ 52. -69.  -3.]]]
 [[[  4.  37. -33.]
   [ 29.  10. -44.]
   [-29.  37. -12.]]]
 [[[-21. -75. -17.]
   [ 14. -35.  26.]
   [-12.   0.  17.]]]
 [[[ 25. -20. -19.]
   [ -9.  31. -20.]
   [ 25.  -1.  10.]]]]
```

```
Weights Layer 2 ke-0 :

[[[[ 15. -15. -14.]
   [  6. -54.  19.]
   [ 25. -19. -12.]]]
 [[[ 26.  28.  -6.]
   [-20.  -4.  -1.]
   [-16.  10. -53.]]]
 [[[-26. -26.  -5.]
   [ 25.  36. -31.]
   [ -8.  58.  16.]]]
 [[[-12.  31.  37.]
   [-41.  25. -19.]
   [-41.  -3. -14.]]]
 [[[ 24.  -7.   6.]
   [-40. -13.  15.]
   [-23.  10.   0.]]]
 [[[ 14.   6. -28.]
   [-12.  17.  10.]
   [  5. -35.   7.]]]]
```

```
Weights Layer 3 :

[[ 10.   8.  23.  11.   0.  17. -15. -19.  14.  -1. -13.  -6.   5.   7.
  -20. -10.  -3.  18.  -5.  25.  16. -14. -17.  -9.  19.  14. -17. -13.
  -15. -19.   1. -29.   1.  -8.   7.   8.  14. -21. -19.  11.  23. -17.
   26. -20.  11. -15.  -8.   0. -11. -13.  -6.  -9.   6.  18.  23.  19.
  -18.  -1. -16.   1.  12.  20.   9.  -7.]
 [  9.   8.  -3. -14.   3. -20.  19. -21.  -6.   1. -16.   3.  -4.  21.
   -3.  18. -21.   5.   2.  11.  14.  18. -19. -15.  -3.  19. -23. -21.
  -17. -29. -13.   7. -15.  12.  28.   4.  -5.  16.  -8.  13.   7.   8.
    3.   5.  10.   9.  -5. -26. -16.   8.  19. -20.  -5. -31.  12.  -4.
   -5. -16.  -1. -19.   7.   6. -20.  16.]
 [-14. -16.  24. -29.   8. -22.  -8. -29.  -8.  -7. -10.  13. -12. -12.
  -21. -16.  33.   1. -21.  -1.  12. -15.   7. -24.  18.  -8.  -9.  -1.
    9. -18.  -4. -10.  20. -22. -12.   5.  -7.  15.  -2. -27.   3. -23.
   -6.   7.   0.   3. -11.  -9.  10. -30.  -5.  21.  17.   7.   3. -19.
  -13.  -6.   5. -14.   3.  -2.  11.   6.]
```

```
Weights Layer 4 :

[[ -7.  23. -35.   9. -28. -13. -36.   4.  -7. -42.]
 [  4. -18.  12.  23. -27.  35. -11.   2.  43. -29.]
 [-20. -24.  23.   0.  -8.  25.  31.  23. -15.  23.]
 [-49.  41. -19.   4. -32.  40.  -4. -46.  54.  -8.]
 [-12.   3. -18.  11. -32. -39.  -2. -32. -23.  34.]
 [ 19.  13.   4.  28.  19. -14.  41.  45.  13.  14.]
 [ 37. -29. -10. -35. -29.  23. -11.  11.  -2.  11.]
 [ 35.  54.  11. -30.  31. -15. -39.   3.  27.   2.]
 [-41. -22. -27.  -6. -14.   0. -41. -15.  11. -27.]
 [ 24. -42. -20.  14. -43. -53.  32. -42. -36.  28.]
 [ 14. -12.  50.  24. -32.   3.  47.  19. -29.   2.]
 [ 33. -21.  33.   1.  34.  19.   0.  19. -17. -15.]
 [ 19. -10. -37.   5. -12.  36.  25.  11.  10.  -9.]
 [ 38.  15.  17.  29. -11. -40. -49.  42.  -3. -16.]
 [ 28.  -4.  -2.  -9. -35. -31. -47. -50.  21. -33.]
 [-24.   7. -33.  20.  51. -36.  -1.   1.   4. -24.]
```

(array([0.09881388, 0.09752722, 0.09821551, 0.10697497, 0.10182809, 0.10083442, 0.10087133, 0.09974821, 0.09546809, 0.09971828]),)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "weights.h"
#include "cnnfunction.h"

int main(void)
{
   //Output Layer
   float layer_1_out[6][13][13];
   float layer_2_out[6][5][5];
   float layer_3_out[64];
   float layer_last_out[10];
   float result;
   //Konstanta pengali untuk output layer sebelum masuk kuantisasi  untuk input testing gambar 3
   float Scale_w1 =0.0032400540479524866;
   float Scale_d1 =0.007874015748031496;
   float Scale_w2 =0.004680760732785923;
   float Scale_d2 =0.0034186397041388442;
   float Scale_w3 =0.006520035698657899;
   float Scale_d3 =0.0009874517863943284;
   float Scale_w4 =0.005369161526987872;
   float Scale_d4 =0.0013629764335707172;
   //Variabel input layer terkuantisasi
   //sebelum masuk layer 2
   int quantized_d2_0[6][150];
   int quantized_d3[150];     //sebelum masuk layer 3
   int quantized_d4[64];      //sebelum masuk layer 4
   float flat[150];
   int i,j,k;
   //inisialisasi nilai 0 pada array output
   zero(&layer_last_out);
   zero(&layer_3_out);
   //Keluaran layer 1 dari HW (abis max pooling) masuk ke pengali skala
   scale2D(&layer_1_out,Scale_d1,Scale_w1);
   //Kuantisasi Output Layer 2
   kuantisasi_float3d(layer_1_out[6][13][13],7,&quantized_d2_0[6][150]);
   //Passing ke HW hasil kuantisasi
   //Keluaran layer 2 dari HW (abis max pooling) masuk ke pengali skala
   scale2D(&layer_2_out,Scale_d2,Scale_w2);
   int count=0;
   //Flatten
   for(i=0;i<5;i++)
   {
      for(j=0;j<5;j++)
      {
         for(k=0;k<5;k++)
         {
            flat[count] = layer_2_out[i][j][k];
            count++;
         }
      }
   }
   //Quantize Flatten
   kuantisasi_float1d(flat[150],7,&quantized_d3[150]);
   //Perhitungan Layer 3 (Dense 1)

matmul(quantized_d3,quantized_w3,150,64,Scale_d3,Scale_w3,&layer_3_out);
   //Kuantisasi Output Layer 3
   kuantisasi_float1d(layer_3_out[64],7,&quantized_d4[64]);
   //Perhitungan Layer Output (Dense 2)

matmul(quantized_d4,quantized_w4,64,10,Scale_d4,Scale_w4,&layer_last_out);
   result = softmax(layer_last_out);
   printf("\nHasil dari klasifikasi menunjukkan bahwa citra ini masukk ke
kelas:\n");
   printf("%.2f\n", result);
}
```

```
void kuantisasi_float3d(float data_lama[][][], int bits,int *output[][][]){
                        float max, min, temp;
                        float range_real;
                        float range_real;
                        int data_baru;
                        int l,m,n;
                        temp = 0;
                        for ( l =0; l<length1; l++)
  {
    for ( m =0; m<length2; m++)
    {
      for(n=0;n<lenght3;n++)
      {
        if (data_lama[l][m][n] > temp)
        {
          max = data_lama[l][m][n];
          temp = data_lama[l][m][n];
        }
      }
    }
                        }
                        temp = 0;
                        for ( l =0; l<length1; l++)
  {
    for ( m =0; m<length2; m++)
    {
      for(n=0;n<lenght3;n++)
      {
        if (data_lama[l][m][n] < temp)
        {
          min = data_lama[l][m][n];
          temp = data_lama[l][m][n];
        }
      }
    }
                        }
                        range_real = max - min;
                        if (range_real == 0)
    range_real = 1
                        scale = ((range_real/pow(2,bits))-1);
                        for ( l =0; l<length1; l++)
  {
    for ( m =0; m<length2; m++)
    {
      for(n=0;n<lenght3;n++)
      {
*(output+l+m+n) = round(data_lama[l][m][n]/scale);
                        }
    }
                        }
}
```

```
void kuantisasi_float1d(float data_lama[], int bits,int *output[]){
                        float max, min, temp;
                        float range_real;
                        float range_real;
                        int data_baru;
                        int l,m,n;
                        temp = 0;
                        for ( l =0; l<length1; l++)
  {
    if (data_lama[l] > temp)
    {
      max = data_lama[l];
      temp = data_lama[l];
    }
                        }
                        temp = 0;
                        for ( l =0; l<length1; l++)
  {
    if (data_lama[l][m] < temp)
    {
      min = data_lama[l];
      temp = data_lama[l];
    }
                        }
                        range_real = max - min;
                        if (range_real == 0)
    range_real = 1
                        scale = ((range_real/pow(2,bits))-1);
                        for ( l =0; l<length1; l++)
  {
    *(output+l) = round(data_lama[l]/scale);
                        }
}
float softmax(float layer_last_out[10])
{
  float expo[10];
  float expo_sum=0;
  for(int i = 0;i<10;i++)
  {
    expo[i]=exp(layer_last_out[i]);
    expo_sum += expo[i];
  }
  return expo_sum;
}
void relu(float *input)
{
  if(*(input)<0)
  {
    *(input)=0;
  }
}
void zero(float *array)
{
  for(int i=0;i<sizeof(array);i++)
  {
    *(array+i) = 0;
  }
}
```

```
void matmul(int quantized_d[], int quantized_w[][], int sizein, int sizeout, float Scale_d, float Scale_w, float *out)
{
  int i,j;
  for(i=0;i<sizeout;i++)
  {
    for(j=0;j<sizein;j++)
    {
      *(out+i) += quantized_d[j] * quantized_w[j][i];
    }
    if(sizein == 150)
    {
      relu(out)
      *(out+i) = *(out+i) * Scale_d * Scale_w;
    }
    else
    {
      *(out+i) =  *(out+i) * Scale_d * Scale_w;
    }

  }
}
void scale2D(int *input[][][], float scale_d, float scale_w)
{
  int l,m,n;
  for(l=0;l<5;l++)
  {
    for(m=0;m<5;m++)
    {
      for(n=0;n<5;n++)
      {
        *(input+l+m+n) =*(input+l+m+n) * scale_d * scale_w;
      }
    }
  }
}
```