LAPORAN PROYEK MACHINE LEARNING - KELOMPOK 10

Anggota:

- Jennifer Angelina (C14190099)
- Geraldy Cornelius (C14190122)

Topik: *MNIST Number Recognition 0-9 with CNN*

I. Pembagian Kerja

a. Jennifer Angelina (C14190099)
   - Membuat 250 dataset
   - Memfilter data agar memiliki jumlah class yang sama
   - Melakukan uji coba dengan variasi jumlah dataset 10,000
   - Melakukan uji coba dengan variasi jumlah dataset 60,000

b. Geraldy Cornelius (C14190122)
   - Membuat 250 dataset
   - Melakukan uji coba dengan variasi jumlah dataset 30,000
   - Melakukan uji coba dengan variasi jumlah dataset 60,000
   - Melakukan uji coba untuk tipe binary

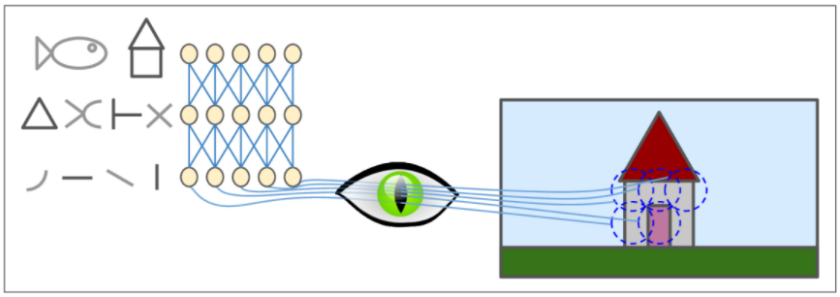
II. Teori dan Model yang Digunakan

Pada proyek kali ini, kami mengimplementasikan salah satu metode Machine Learning yaitu Convolutional Neural Network (CNN). Convolutional Neural Network merupakan salah satu metode yang memiliki hasil signifikan dalam image processing. Convolutional Neural Network muncul dari studi otak tentang visual cortex dan metode ini telah digunakan sejak tahun 1980an. Dalam beberapa tahun terakhir, dengan jumlah training data yang semakin banyak, CNN semakin mampu menyaingi kinerja manusia dalam mengenali visual yang kompleks. CNN banyak digunakan dalam image search service, self-driving cars, automatic video classification system, dan masih banyak lagi. Bahkan, CNN tidak hanya bisa mengenali visual, tetapi juga bisa untuk hal lain seperti voice recognition, natural language processing (NLP).

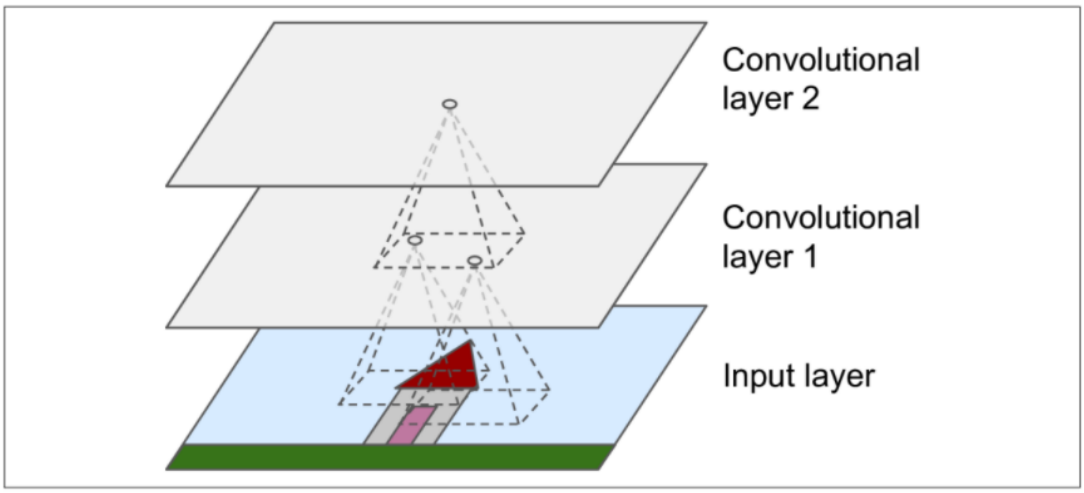
CNN sendiri merupakan pengembangan dari Multilayer Perceptron (MLP) yang didesain untuk mengolah data dua dimensi. CNN tergolong dalam jenis Deep Neural Network karena kedalaman jaringan yang tinggi dan banyak diaplikasikan pada data citra. Cara kerja CNN mirip dengan MLP, namun dalam CNN, tiap neuron direpresentasikan dalam bentuk dua dimensi, tidak seperti MLP yang tiap neuron hanya berukuran satu dimensi. Pre-processing yang dibutuhkan dalam CNN lebih rendah daripada algoritma lain untuk klasifikasi.

Sebuah studi oleh David H. Hubel dan Torsten Wiesel menunjukkan bahwa banyak neuron di visual cortex mempunyai bidang reseptif lokal kecil, yang berarti

neuron tersebut hanya bereaksi terhadap rangsangan visual yang terletak di area bidang visual yang terbatas. Seperti pada gambar di bawah, bidang reseptif lokal dari lima neuron digambarkan dengan lingkaran putus-putus.



Hal yang paling penting dalam implementasi CNN adalah convolutional layer itu sendiri. Neuron yang berada pada convolutional layer pertama tidak akan terhubung ke setiap pixel pada input image, tetapi hanya terhubung ke bidang reseptif tertentu. Begitu juga neuron pada convolutional layer kedua hanya terhubung ke neuron yang berada pada kotak di layer pertama. Arsitektur ini akan membuat network hanya fokus pada fitur low level pada hidden layer pertama, kemudian menyatukan mereka menjadi fitur tingkat tinggi yang lebih besra, dan begitu seterusnya. Struktur hierarki ini umum dalam gambar di dunia nyata, yang merupakan salah satu alasan mengapa CNN bekerja sangat baik dalam image recognition.



Dalam implementasi CNN untuk image recognition, kami menggunakan Tensorflow

Untuk dataset, kami menggunakan dataset MNIST dari Keras yang berisi 60.000 dataset tulisan tangan angka 0 sampai 9 untuk training dan dataset tulisan tangan sendiri (1500 dataset) untuk training. Model CNN akan dikembangkan dengan menggunakan library Tensorflow (sequential, Conv2D, Activation, MaxPooling2D).

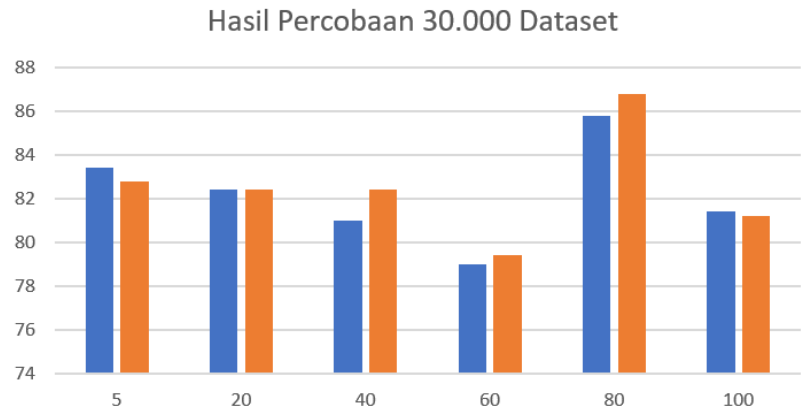
III. Hasil Uji Coba

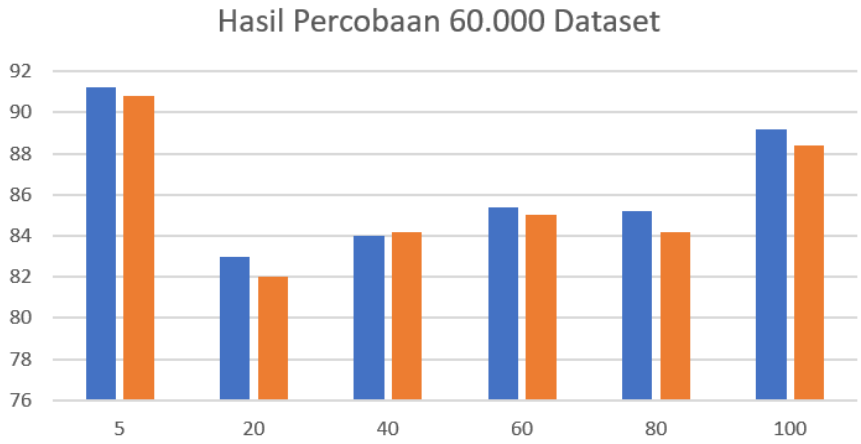
1. Uji coba dengan *Dataset Training* hanya dari Keras (60000)

Jumlah Dataset Training	Jumlah Epoch	Grayscale	Binary
10000	5	76%	76%
	20	85.2%	84.4%
	40	77.4%	75.8%
	60	79.6%	78.8%
	80	77.8%	77.2%
	100	81.8%	80.4%
30000	5	83.4%	82.8%
	20	82.4%	82.4%
	40	81%	82.4%
	60	79%	79.4%
	80	85.8%	86.8%
	100	81.4%	81.2%
60000	5	91.2%	90.8%
	20	83%	82%
	40	84%	84.2%
	60	85.4%	85%
	80	85.2%	84.2%

	100	89.2%	88.4%

Grafik:

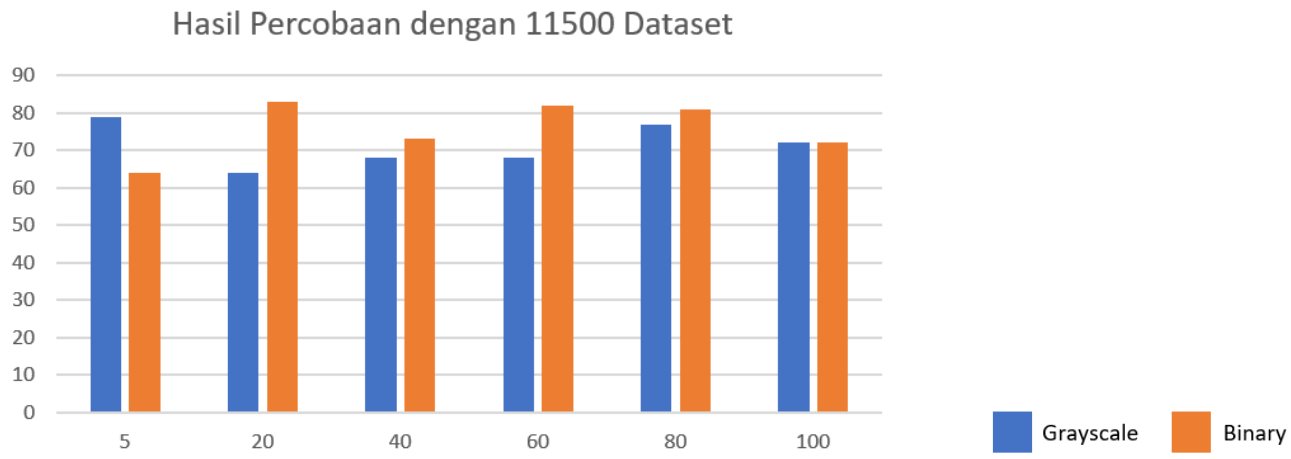


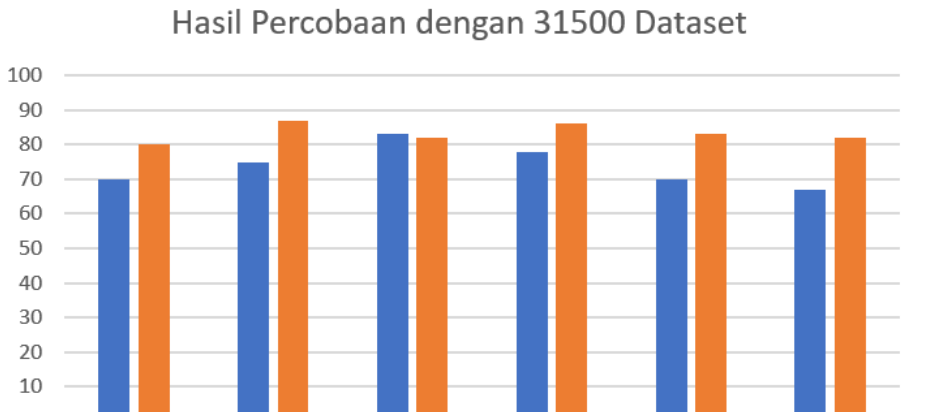


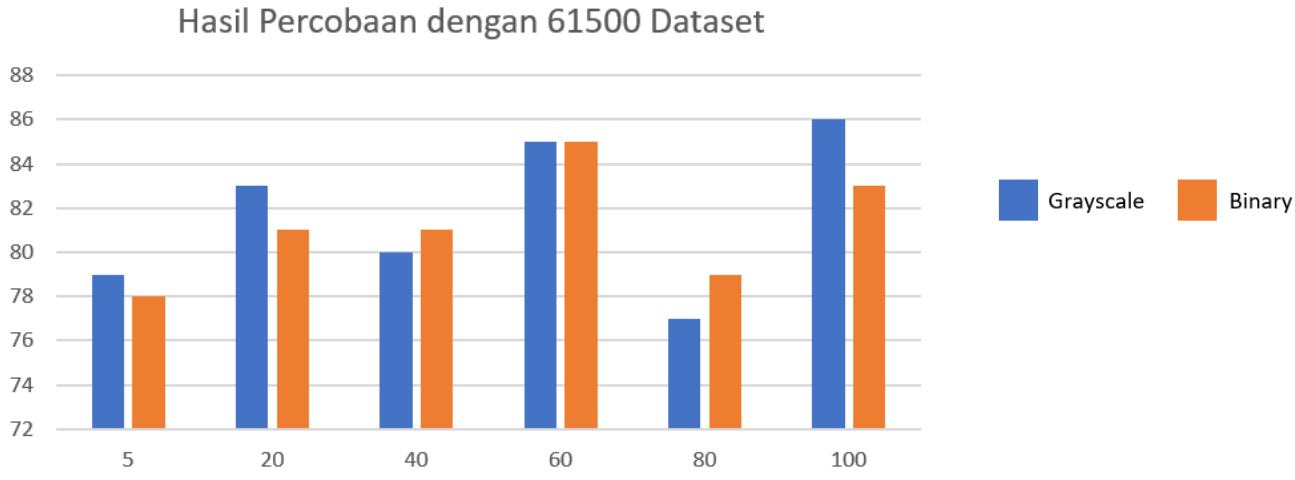
2. Uji coba dengan *Dataset Training* dari Keras + dari dataset buatan sendiri (60000 + 1500)

Jumlah Dataset Training	Jumlah Epoch	Grayscale	Binary
11500	5	79%	64%
	20	64%	83%
	40	68%	73%
	60	68%	82%
	80	77%	81%
	100	72%	72%
31500	5	70%	80%
	20	75%	87%
	40	83%	82%
	60	78%	86%
	80	70%	82%
	100	67%	82%
61500	5	79%	78%
	20	83%	81%
	40	80%	81%
	60	85%	85%
	80	77%	79%
	100	86%	83%

Grafik:







IV. Kesimpulan

Dari percobaan yang telah dilakukan:

1) Perbandingan variasi jumlah dataset training: kami melihat bahwa dengan adanya variasi perbandingan jumlah dataset pada saat training, akurasi meningkat seiring bertambahnya jumlah dataset training.
2) Perbandingan jumlah epoch: jumlah epoch tidak selalu mempengaruhi akurasi, terkadang akurasi naik dan terkadang akurasi turun.
3) Perbandingan tipe grayscale dan binary: dari hasil uji coba yang telah dilakukan, testing dengan menggunakan data grayscale sedikit lebih akurat daripada data binary. Hal ini diduga karena grayscale memiliki struktur yang lebih kompleks yang akan lebih mengenali objek lebih baik daripada binary, sehingga menghasilkan akurasi yang lebih tinggi daripada binary.
4) Perbandingan dataset training yang hanya dari Keras dengan dataset training yang ditambahkan dengan buatan sendiri: kebanyakan hasil akurasi dengan dataset yang hanya dari Keras lebih tinggi (namun terkadang ada juga yang hasilnya lebih rendah). Selisih perbedaan akurasi ini tidak terlalu jauh. Dapat disimpulkan bahwa memang dataset yang hanya berasal dari satu sumber saja cenderung lebih mudah untuk ditraining daripada dataset yang berasal dari banyak sumber.

LAMPIRAN (DOKUMENTASI)

1. Import library & dataset MNIST dari Keras

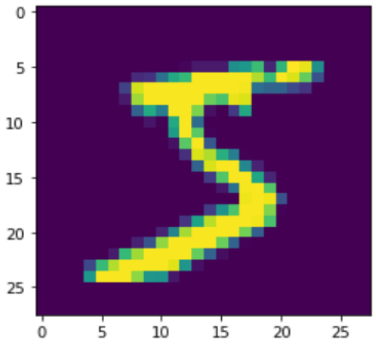
```
In [36]: import tensorflow as tf
         import numpy as np
         import matplotlib.pyplot as plt

         mnist = tf.keras.datasets.mnist
         (x_train, y_train),(x_test, y_test) = mnist.load_data()
```

Mencoba print data training

```
In [37]: x_train.shape
Out[37]: (60000, 28, 28)

In [38]: plt.imshow(x_train[0])
         plt.show()
```



2. Load dataset buatan sendiri dan kelompok lain untuk dimasukkan ke dalam array

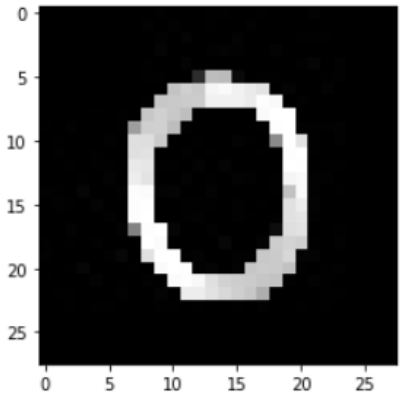
In [5]:
```
import os
import cv2
from PIL import Image
from PIL import ImageChops
```

In [6]:
```
IMG_SIZE = 28
```

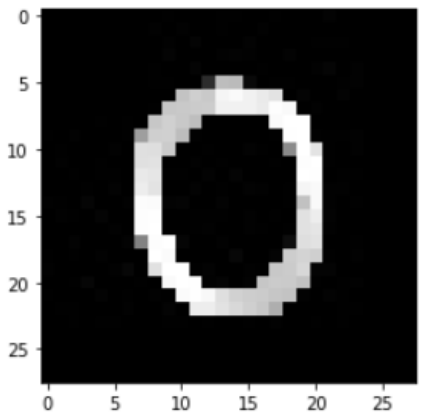
In [7]:
```
numbers = os.listdir("dataset_test")
```

In [8]:
```
DATADIR = "C:/Users/ACER/Documents/MNIST/dataset_test_dilate"
CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```



In [9]:
```
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap= 'gray')
plt.show()
```



3. Memasukkan datatest grayscale dan binary kedalam array data_test

In [10]:
```
# Grayscale

testing_data_test = [] # membuat data test untuk melakukan feature extraction

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                testing_data_test.append([new_array, class_num])
            except Exception as e:
                pass

create_testing_data()
```

In [13]:
```
for features, label in testing_data_test: #melakukan feature extraction untuk data test
    X_test_fix.append(features)
    Y_test_fix.append(label)
```

In [14]:
```
X_test_fix = np.array(X_test_fix)
Y_test_fix = np.array(Y_test_fix)
```

In [15]:
```
# Binary

testing_binary_data_test = [] # membuat data test untuk melakukan feature extraction

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
                (thresh, im_bw) = cv2.threshold(img_array, 127, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                testing_binary_data_test.append([new_array, class_num])
            except Exception as e:
                pass

create_testing_data()
```

In [18]:
```
for features, label in testing_data_test: #melakukan feature extraction untuk data test
    X_test_binary_fix.append(features)
    Y_test_binary_fix.append(label)
```

In [19]:
```
X_test_binary_fix = np.array(X_test_binary_fix)
Y_test_binary_fix = np.array(Y_test_binary_fix)
```

In [20]:
```
DATADIR = "C:/Users/ACER/Documents/MNIST/dataset_dilate_3"
CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```

In [22]:
```
testing_data_3 = [] # membuat data test untuk melakukan feature extraction

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                testing_data_3.append([new_array, class_num])
            except Exception as e:
                pass

create_testing_data()
```

In [24]:
```
DATADIR = "C:/Users/ACER/Documents/MNIST/dataset_dilate_4"
CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```

In [26]:
```
testing_data_4 = [] # membuat data test untuk melakukan feature extraction

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                testing_data_4.append([new_array, class_num])
            except Exception as e:
                pass

create_testing_data()
```

In [42]:
```
X_train_combined_10 = np.concatenate((x_train, X_test))
Y_train_combined_10 = np.concatenate((y_train, Y_test))

X_train_combined_3 = np.concatenate((X_train_combined_10, X_test_3))
Y_train_combined_3 = np.concatenate((Y_train_combined_10, Y_test_3))

X_train_combined = np.concatenate((X_train_combined_3, X_test_4))
Y_train_combined = np.concatenate((Y_train_combined_3, Y_test_4))
```

4. Normalisasi: mengubah intensitas pixel (yang tadinya range 0-255 menjadi range 0-1).

In [47]:
```
# normalisasi -> mengubah intensitas pixel
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

```
In [42]: print(x_train[0])
```

```
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
 0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
 0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
 0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
 0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.04500225 0.4219755  0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.28826244 0.26543758 0.34149427 0.31128482
 0.         0.         0.         0.         0.         0.
```

5. Mengambil sebagian data dari seluruh data untuk variasi jumlah training data

```
In [48]: # data full

         from sklearn.model_selection import train_test_split
```

```
In [56]: # FILTER DATA -

         # GRAYSCALE filter
         X_train_filter, _, Y_train_filter, _ = train_test_split(X_train_normalized, Y_train_combined, train_size=31500, random_state=42,

         # BINARY filter
         X_train_binary_filter, _, Y_train_filter, _ = train_test_split(X_train_combined_binary, Y_train_combined, train_size=11500, randc
```

6. Mengubah *shape image* agar bisa digunakan dalam operasi *Convolution* (*kernel operation* membutuhkan 1 dimensi lagi)

```
In [58]: # me-reshape // mengubah image size agar bisa digunakan dalam operasi Convolution
         # untuk bisa kernel operation diperlukan 1 dimensi lagi
         # Reshape Grayscale

         IMG_SIZE=28 #ukuran gambar sebesar 28 pixel
         X_trainr_full = np.array(X_train_combined).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # menambah 1 dimensi untuk kernel operation pada da
         X_testr_full = np.array(X_test_fix).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # menambah 1 dimensi untuk kernel operation pada data uji
         print("Training Samples Dimention: ", X_trainr_full.shape) # menampilkan bentuk dari data X_train
         print("Testing Samples Dimention: ", X_testr_full.shape) # menampilkan bentuk dari data X_test
```

```
Training Samples Dimention:  (61500, 28, 28, 1)
Testing Samples Dimention:  (100, 28, 28, 1)
```

```
In [59]: # Reshape Binary

         IMG_SIZE=28 #ukuran gambar sebesar 28 pixel
         X_trainr_binary_full = np.array(X_train_combined_binary).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # menambah 1 dimensi untuk kernel ope
         X_testr_binary_full = np.array(X_test_binary_fix).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # menambah 1 dimensi untuk kernel operation
         print("Training Samples Dimention: ", X_trainr_full.shape) # menampilkan bentuk dari data X_train
         print("Testing Samples Dimention: ", X_testr_binary_full.shape) # menampilkan bentuk dari data X_test
```

```
Training Samples Dimention:  (61500, 28, 28, 1)
Testing Samples Dimention:  (100, 28, 28, 1)
```

7. *Import* komponen-komponen dari Tensorflow

In [44]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
```

8. Membuat *neural network*

In [9]:
```
model = Sequential() # membuat model

#First layer
model.add(Conv2D(16, (3,3), input_shape = x_trainr.shape[1:])) # menggunakan konvolusi 2 dimensi sebagai layer
model.add(Activation("relu")) # menggunakan fungsi aktivasi relu, untuk aktivin neuron
model.add(MaxPooling2D(pool_size=(2,2)))  # melakukan pooling dalam bentuk 2 dimensi

#Second layer
model.add(Conv2D(32, (3,3))) # menggunakan konvolusi sebagai layer
model.add(Activation("relu")) # menggunakan fungsi aktivasi relu
model.add(MaxPooling2D(pool_size=(2,2))) # melakukan pooling dalam bentuk 2 dimensi

#Third layer
model.add(Conv2D(64, (3,3))) # menggunakan konvolusi sebagai layer
model.add(Activation("relu")) # menggunakan fungsi aktivasi relu
model.add(MaxPooling2D(pool_size=(2,2)))  # melakukan pooling dalam bentuk 2 dimensi

#Fully connected layer # 1
model.add(Flatten()) # di-flatten dari 2D ke 1D
model.add(Dense(128)) # Hanya lapisan NN yang terhubung secara padat.
model.add(Activation("relu"))

#Fully connected layer # 2
model.add(Dense(64)) # Hanya lapisan NN yang terhubung secara padat.
model.add(Activation("relu"))
model.add(Dropout(0.4))

#Fully connected layer # 3
model.add(Dense(10)) # menggunakan 10 objek kategorikal
model.add(Activation('softmax')) # Menggunakan softmax karena memiliki lebih dari 2 objek yang di latih
```

9. Melakukan training atau fitting model

```
In [66]: model.compile(loss="sparse_categorical_crossentropy", optimizer="SGD", metrics=['accuracy'])

In [67]: # use checkpoint to save best loss
         cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath='checkpoint.h5',save_best_only=True, save_weights_only=True, verbose=1)

In [68]: # Train data
         # Grayscale
         # full -> X_trainr_full, Y_train_combined
         # filter -> X_trainr_filter, Y_train_filter

         # Binary
         # full -> X_trainr_binary_full, Y_train_combined
         # filter -> X_trainr_binary_filter, Y_train_filter

         history = model.fit(X_trainr_binary_filter, Y_train_filter, epochs= 100, validation_split = 0.2, callbacks = [cp_callback]) # mel
```

```
288/288 [==============================] - 2s 5ms/step - loss: 0.0039 - accuracy: 0.9986 - val_loss: 0.3622 - val_accuracy:
0.9639

Epoch 00097: val_loss did not improve from 0.18300
Epoch 98/100
288/288 [==============================] - 2s 5ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.3978 - val_accuracy:
0.9622

Epoch 00098: val_loss did not improve from 0.18300
Epoch 99/100
288/288 [==============================] - 2s 5ms/step - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.3713 - val_accuracy:
0.9652

Epoch 00099: val_loss did not improve from 0.18300
Epoch 100/100
288/288 [==============================] - 2s 6ms/step - loss: 0.0077 - accuracy: 0.9982 - val_loss: 0.4139 - val_accuracy:
0.9600

Epoch 00100: val_loss did not improve from 0.18300
```

*Pada tahap ini, kami melakukan variasi jumlah *epoch* (5, 20, 40, 60, 80, 100). Gambar di atas adalah pada saat epoch = 100.

10. Mengecek akurasi data untuk grayscale dan biner

In [39]:
```
# check grayscale data accuracy

test_loss, test_acc = model.evaluate(X_test, Y_test, batch_size = 1)
print("Test loss on 60,000 test samples", test_loss) # menampilkan akurasi loss dari data dalam bentuk grayscale
print("Validation Accuracy on 60,000 test samples", test_acc) # menampilkan akurasi validasi dari data dalam bentuk grayscale
```

```
500/500 [==============================] - 2s 3ms/step - loss: 349.2086 - accuracy: 0.8520
Test loss on 60,000 test samples 349.2086486816406
Validation Accuracy on 60,000 test samples 0.8519999980926514
```

In [40]:
```
# check binary data accuracy

test_loss, test_acc = model.evaluate(X_binary_test, Y_binary_test, batch_size = 1)
print("Test loss on 60,000 test samples", test_loss) # menampilkan akurasi loss dari data dalam bentuk biner
print("Validation Accuracy on 60,000 test samples", test_acc) # menampilkan validasi akurasi dari data dalam bentuk biner
```

```
500/500 [==============================] - 1s 3ms/step - loss: 370.4521 - accuracy: 0.8420
Test loss on 60,000 test samples 370.45208740234375
Validation Accuracy on 60,000 test samples 0.8420000076293945
```

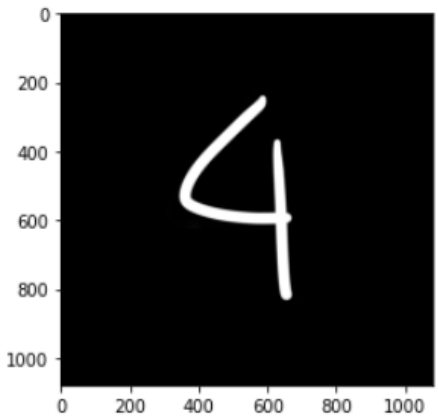
11. Melakukan percobaan pada data grayscale

In [45]:
```
img1 = cv2.imread("dataset/4/img (1).jpg")
plt.imshow(img1)

gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_AREA)
newing = tf.keras.utils.normalize (resized, axis=1)
newing = np.array(newing).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
predictions = model.predict(newing)
print(np.argmax(predictions))
```

Out[45]:
```
<matplotlib.image.AxesImage at 0x28709043be0>

9
```

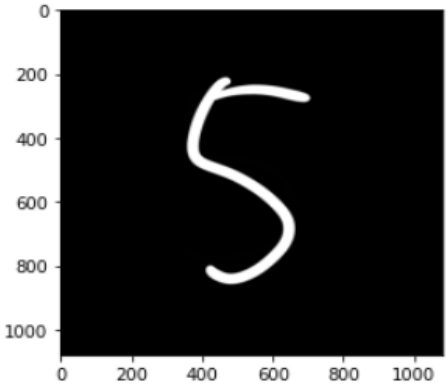


In [46]:
```
img1 = cv2.imread("dataset/5/img (1).jpg")
plt.imshow(img1)

gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_AREA)
newing = tf.keras.utils.normalize (resized, axis=1)
newing = np.array(newing).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
predictions = model.predict(newing)
print(np.argmax(predictions))
```

Out[46]: <matplotlib.image.AxesImage at 0x2870b745be0>

```
5
```

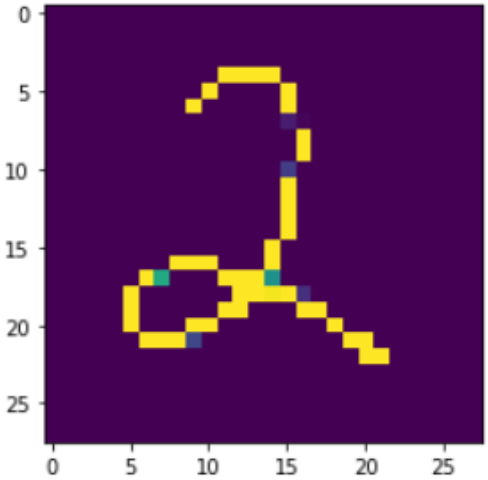


12. Melakukan percobaan pada data binary

In [543]:
```
predictions = model.predict([X_binary_test])
print(np.argmax(predictions[102]))
plt.imshow(X_binary_test[102])
```

```
2
```

Out[543]: <matplotlib.image.AxesImage at 0x28800a1c5e0>



In [544]:
```
predictions = model.predict([X_binary_test])
print(np.argmax(predictions[153]))
plt.imshow(X_binary_test[153])
```

```
3
```

Out[544]:
```
<matplotlib.image.AxesImage at 0x28800a7c8b0>
```