In [31]:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

In [5]:

```
print('GPU',tf.test.is_gpu_available())
```

```
WARNING:tensorflow:From C:\Users\10207\AppData\Local\Temp\ipykernel_17284\698793533.py:
1: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will
be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
GPU False
```

In [6]:

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnis
t.npz
11490434/11490434 [==============================] - 2s 0us/step
```

In [9]:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

In [10]:

```
predictions = model(x_train[:1]).numpy()
predictions
```

Out[10]:

```
array([[-0.50658125, -0.16483425, -0.61695194,  1.0483593 , -0.6917839 ,
        -0.62253964, -0.01198217,  0.417538  , -0.56368655,  0.022507  ]],
      dtype=float32)
```

In [11]:

```
tf.nn.softmax(predictions).numpy()
```

Out[11]:

```
array([[0.06038456, 0.08498547, 0.0540745 , 0.2859091 , 0.05017569,
        0.05377319, 0.09902105, 0.15214783, 0.05703289, 0.10249577]],
      dtype=float32)
```

In [13]:

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

In [14]:

```
loss_fn(y_train[:1], predictions).numpy()
```

Out[14]:

```
2.9229803
```

In [15]:

```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

In [17]:

```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 2s 865us/step - loss: 0.0659 - accuracy:
0.9788
Epoch 2/5
1875/1875 [==============================] - 2s 840us/step - loss: 0.0580 - accuracy:
0.9813
Epoch 3/5
1875/1875 [==============================] - 2s 852us/step - loss: 0.0515 - accuracy:
0.9829
Epoch 4/5
1875/1875 [==============================] - 2s 892us/step - loss: 0.0476 - accuracy:
0.9845
Epoch 5/5
1875/1875 [==============================] - 2s 884us/step - loss: 0.0446 - accuracy:
0.9854
```

Out[17]:

```
<keras.callbacks.History at 0x2bf9b33b880>
```

In [18]:

```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.0772 - accuracy: 0.9778 - 344ms/epoch - 1ms/step
```

Out[18]:

```
[0.07715286314487457, 0.9778000116348267]
```

In [19]:

```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
```

In [20]:

```
probability_model(x_test[:5])
```

Out[20]:

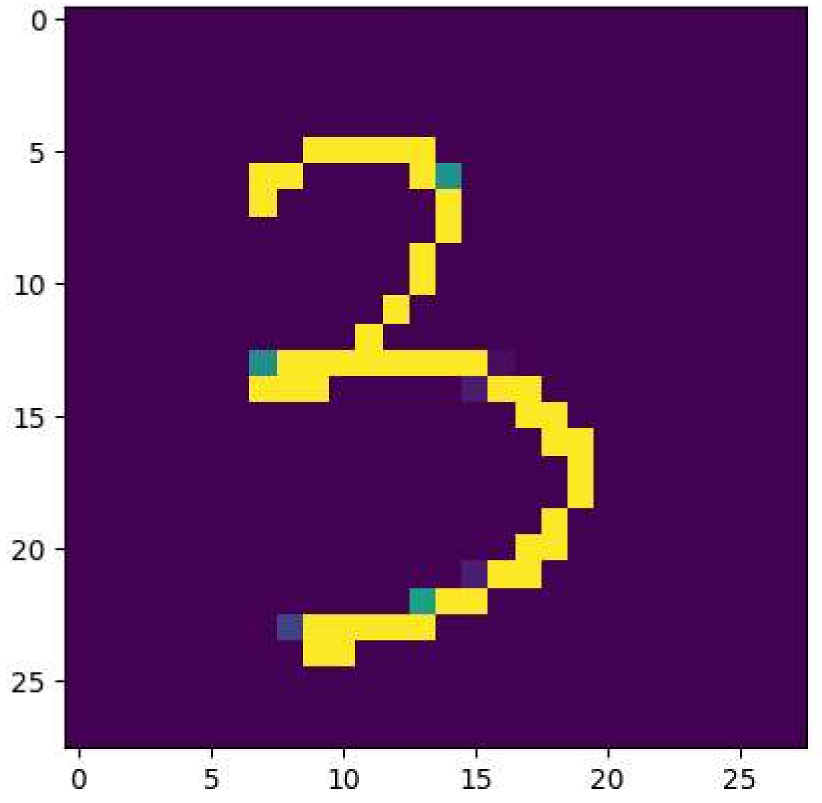
```
<tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[5.93325444e-10, 2.71807044e-09, 2.04512205e-07, 7.79293805e-06,
        3.15525262e-14, 1.04687725e-08, 2.97147959e-17, 9.99991655e-01,
        1.58447602e-08, 3.13529597e-07],
       [4.94734340e-05, 4.51975466e-05, 9.99904752e-01, 3.53856365e-07,
        6.29674962e-17, 3.61147627e-08, 2.18845926e-07, 6.83842521e-16,
        3.73840408e-08, 3.93181643e-17],
       [1.01124149e-08, 9.99959946e-01, 4.12794691e-07, 2.82256227e-08,
        3.14567774e-06, 2.80615780e-08, 1.25230201e-07, 2.64463652e-05,
        9.83014434e-06, 1.29734072e-08],
       [9.99993920e-01, 4.84432598e-11, 4.05346185e-07, 1.04751869e-08,
        1.88618565e-09, 5.31647572e-07, 3.45458773e-07, 3.33543994e-06,
        4.63883509e-08, 1.40403495e-06],
       [1.59372092e-07, 4.87476240e-11, 1.06778600e-07, 9.67293346e-12,
        9.99779761e-01, 4.94739638e-10, 1.15609225e-07, 1.93879441e-06,
        2.92315931e-08, 2.18007801e-04]], dtype=float32)>
```

In [36]:

```
    def decode_predictions(a):
        return np.argmax(a)
    raw = tf.io.read_file("3.PNG")
    tensor = tf.io.decode_image(raw, channels=1, dtype=tf.dtypes.float32)
    tensor = tf.image.resize(tensor, [28, 28])

    d = [[[] for i in range(28)] for j in range (28)]
    for i in range(28):
        for j in range(28):
            d[i][j].append(1-tensor[i][j][0])
```

```
d = np.array(d)
#print(tensor.shape)
inputs_tensor = tf.expand_dims(d, axis=0)
#print(inputs_tensor.shape)
plt.imshow(d)
plt.show()
preds = model.predict(inputs_tensor)
print(decode_predictions(preds))
```



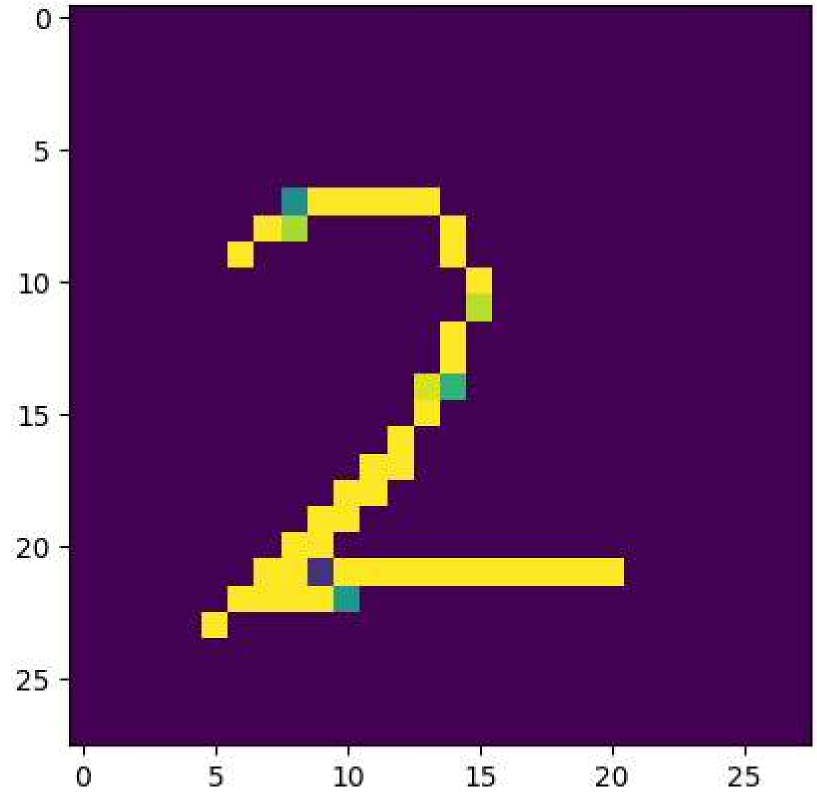
```
1/1 [==============================] - 0s 12ms/step
3
```

In [37]:

```
def decode_predictions(a):
    return np.argmax(a)
raw = tf.io.read_file("2.PNG")
tensor = tf.io.decode_image(raw, channels=1, dtype=tf.dtypes.float32)
tensor = tf.image.resize(tensor, [28, 28])

d = [[[] for i in range(28)] for j in range (28)]
for i in range(28):
    for j in range(28):
        d[i][j].append(1-tensor[i][j][0])


d = np.array(d)
#print(tensor.shape)
inputs_tensor = tf.expand_dims(d, axis=0)
#print(inputs_tensor.shape)
plt.imshow(d)
plt.show()
preds = model.predict(inputs_tensor)
print(decode_predictions(preds))
```



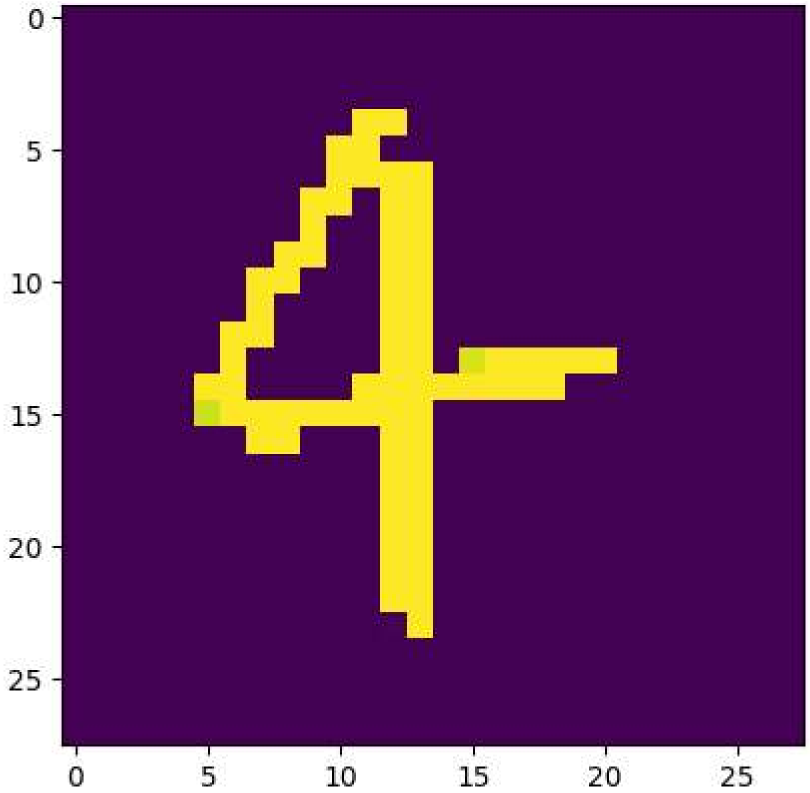
```
1/1 [==============================] - 0s 11ms/step
2
```

In [41]:

```
def decode_predictions(a):
    return np.argmax(a)
raw = tf.io.read_file("4.PNG")
tensor = tf.io.decode_image(raw, channels=1, dtype=tf.dtypes.float32)
tensor = tf.image.resize(tensor, [28, 28])

d = [[[] for i in range(28)] for j in range (28)]
for i in range(28):
    for j in range(28):
        d[i][j].append(1-tensor[i][j][0])


d = np.array(d)
#print(tensor.shape)
inputs_tensor = tf.expand_dims(d, axis=0)
#print(inputs_tensor.shape)
plt.imshow(d)
plt.show()
preds = model.predict(inputs_tensor)
print(decode_predictions(preds))
```



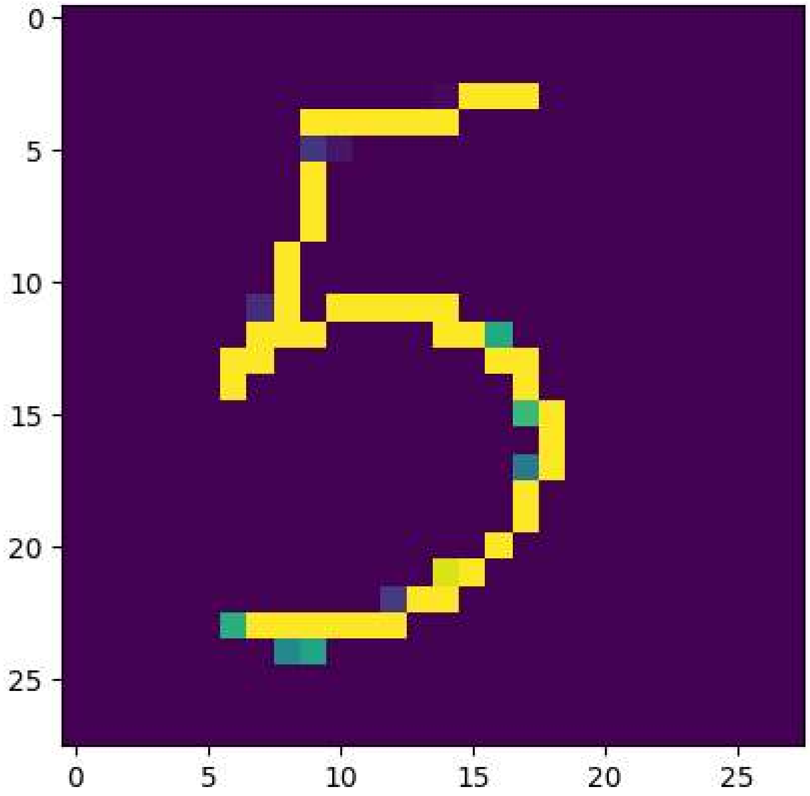
```
1/1 [==============================] - 0s 12ms/step
4
```

In [43]:

```
def decode_predictions(a):
    return np.argmax(a)
raw = tf.io.read_file("5.PNG")
tensor = tf.io.decode_image(raw, channels=1, dtype=tf.dtypes.float32)
tensor = tf.image.resize(tensor, [28, 28])

d = [[[] for i in range(28)] for j in range (28)]
for i in range(28):
    for j in range(28):
        d[i][j].append(1-tensor[i][j][0])


d = np.array(d)
#print(tensor.shape)
inputs_tensor = tf.expand_dims(d, axis=0)
#print(inputs_tensor.shape)
plt.imshow(d)
plt.show()
preds = model.predict(inputs_tensor)
print(decode_predictions(preds))
```



```
1/1 [==============================] - 0s 13ms/step
5
```

```
In [ ]:
```