In [277]:
```
import numpy as np
import matplotlib.pyplot as plt
from torch import nn
import torch
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from sklearn.metrics import accuracy_score
%matplotlib inline
```

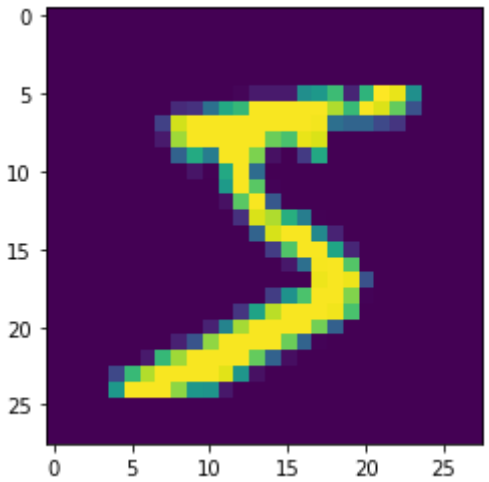
For this homework you will be using `pytorch` and `torchvision` library for neural networks and datasets. You can install them with `pip install torch torchvision`.

# Question 1 Principal Component Analysis

This problem will guide you through the principal component analysis. You will be using a classical dataset, the MNIST hand written digit dataset.

In [278]:
```
# Load the MNIST dataset
mnist = MNIST('.', download=True)
data = mnist.train_data.numpy()
labels = mnist.train_labels.numpy()
print('shapes:', data.shape, labels.shape)
plt.imshow(data[0])
print('label:', labels[0])
```

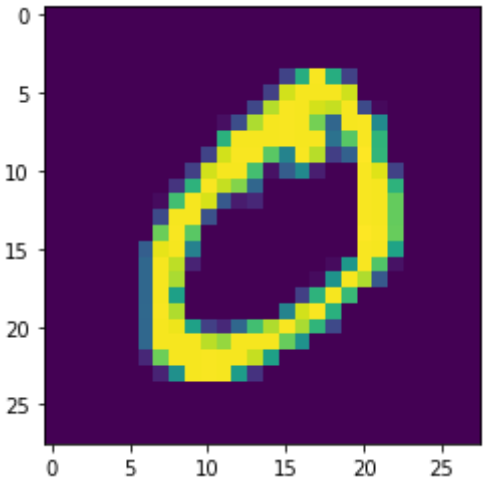
```
/Users/joshuakang/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:53: UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/Users/joshuakang/opt/anaconda3/lib/python3.7/site-packages/torchvisio
n/datasets/mnist.py:43: UserWarning: train_labels has been renamed targ
ets
  warnings.warn("train_labels has been renamed targets")

shapes: (60000, 28, 28) (60000,)
label: 5
```

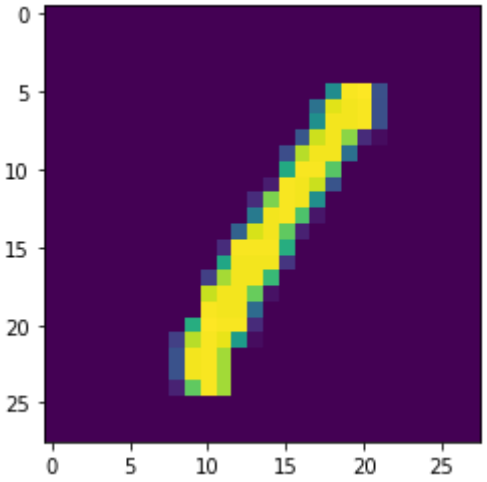


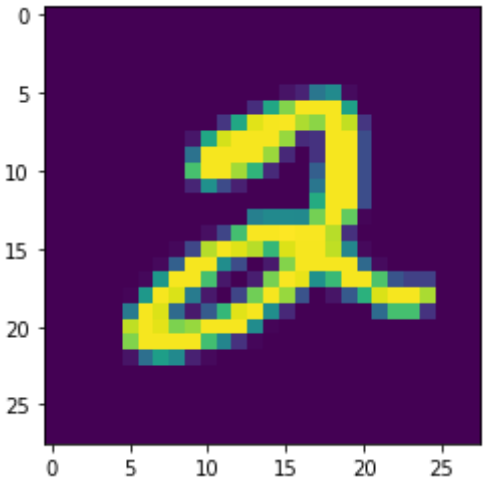

# Question 1.1 Familiarize yourself with the data [5pt]

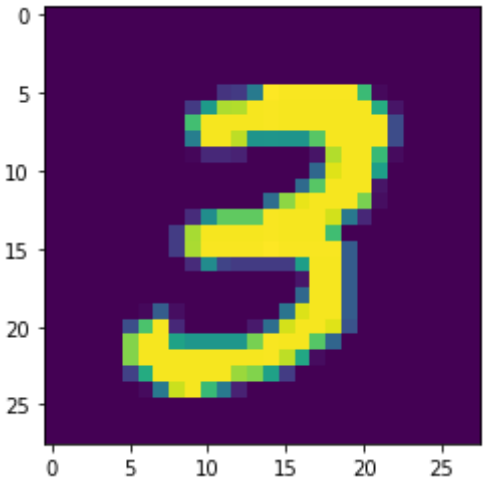
For this task, you will be using the torchvision package that provides the MNIST dataset. For each digit class(0-9), plot 1 image from the class and store those 10 images for each digit class in the array `digit_images` .

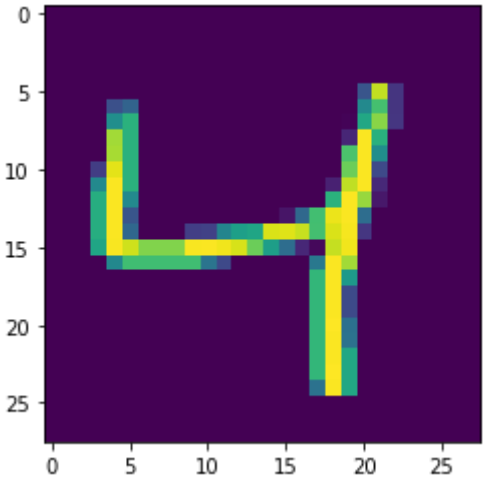
In [280]:
```
digit_images = np.zeros([10, 28, 28])
import random

test_images = []

for i in range(10):
    index = np.where(labels == i)[0][0]
    test_images.append(index)
    digit_images[i] = data[index]
    plt.figure()
    plt.imshow(data[index])



### YOUR CODE HERE
### END OF CODE
```

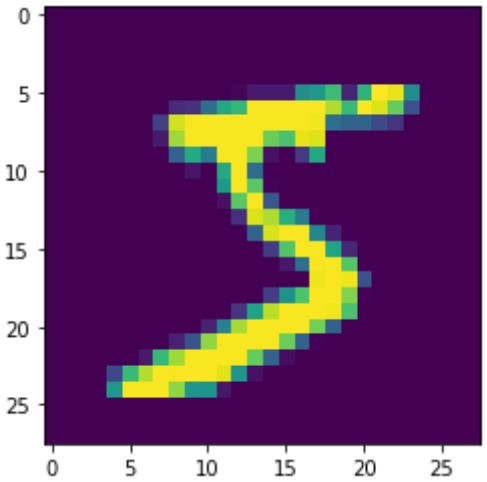

0
5
10
15
20
25
0
5
10
15
20
25

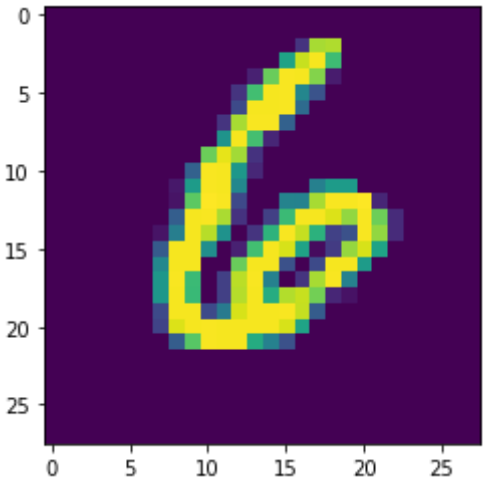

0
5
10
15
20
25
0
5
10
15
20
25

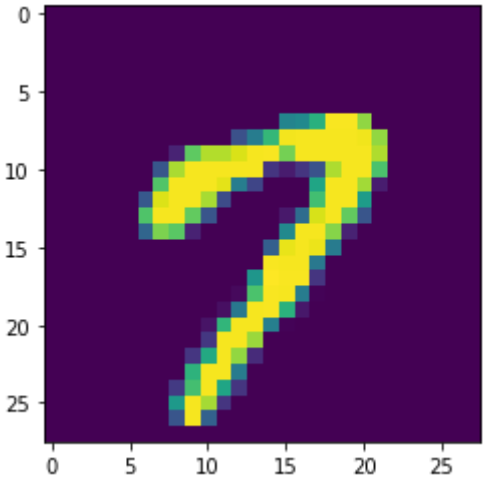

0
5
10
15
20
25
0
5
10
15
20
25

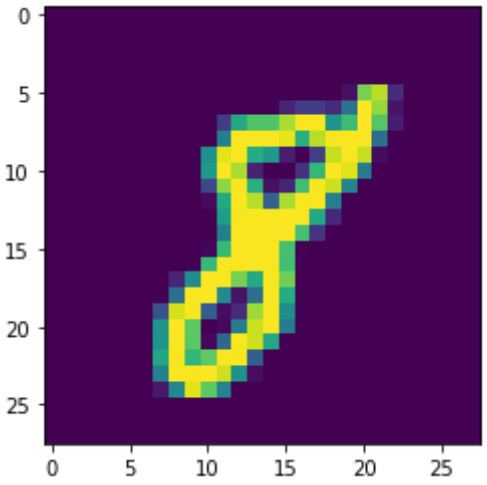

0
5
10
15
20
25
0
5
10
15
20
25

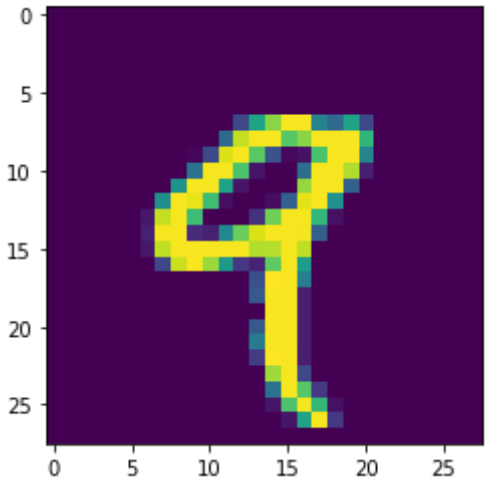

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25




# Question 1.2 PCA

The following questions will guide you through the PCA algorithm.

## Question 1.2.1 Centering the data [5pt]

For each image, flatten it to a 1-D vector. To perform PCA on the dataset, we first move the data points so they have 0 mean on each dimension. Store the centered data in variable `data_centered` and the mean of each dimension in variable `data_mean` .

In [295]:
```
data_centered = np.zeros((60000, 784))
data_mean = None
flat_data = []
for datapoint in data:
    flat_data.append(datapoint.flatten())
flat_data = np.array(flat_data)

data_mean = np.mean(flat_data,axis = 0)

for i in range(flat_data.shape[0]):
    data_centered[i] = np.array(flat_data[i] - data_mean)


### YOUR CODE HERE
### END OF CODE
```

## Question 1.2.2 Compute the covariance matrix of the data [5pt]

You need to store the covariance matrix of the data in variable `data_covmat` . You may **not** use numpy.cov

In [296]:
```
data_covmat = (data_centered.T@data_centered)/data_centered.shape[1]
# data_covmat = np.cov(data_centered)
### YOUR CODE HERE
### END OF CODE
print(data_covmat.shape)
```

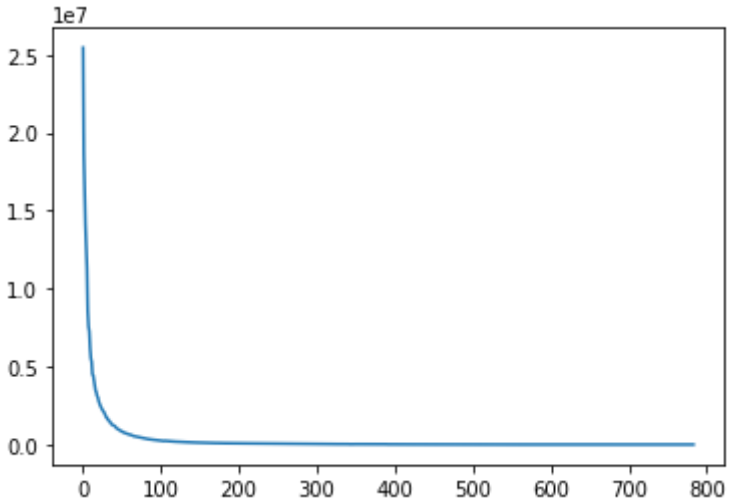
```
(784, 784)
```

## Question 1.2.3 Compute the eigenvalues of the covariance matrix [5pt]

You need to store the eigenvalues of the covariance matrix in variable `covmat_eig`, sorted in descending order. Then you need to plot the eigenvalues with `plt.plot`. You can use any numpy function.

In [307]:
```
eig_val,covmat_eig = np.linalg.eig(data_covmat)

eig_loc = np.argsort(eig_val)[::-1]

new_basis = covmat_eig[:,eig_loc[:2]]

x = []
for i in range(784):
    x.append(i)

plt.plot(x, eig_val)

print(new_basis.shape)



### YOUR CODE HERE
### END OF CODE
```

```
(784, 2)
```



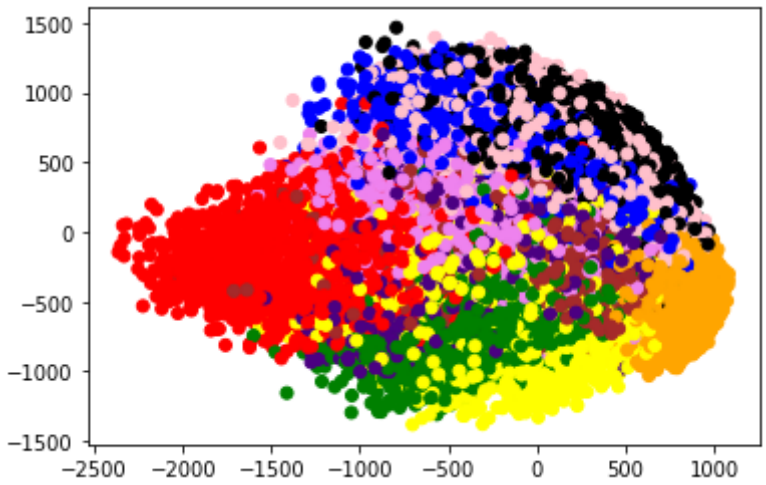
## Question 1.2.4 Project data onto the first 2 principal components [5pt]

Now you need to project the centered data on the 2D space formed by the eigenvectors corresponding to the 2 largest eigenvalues. Create a 2D scatter plot where you need to assign a unique color to each digit class.

In [308]:
```
### YOUR CODE HERE
Z = data_centered @ new_basis

print(Z.shape)

colors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"
, "black", "brown", "pink"]

fig = plt.figure()
ax = fig.add_subplot(111)

projected_x = [x for x in Z[:,0]]
projected_y = [y for y in Z[:,1]]
color_map = [colors[label] for label in labels]
plt.scatter(projected_x, projected_y, c=color_map)

### END OF CODE
```

```
(60000, 2)
```

Out[308]:
```
<matplotlib.collections.PathCollection at 0x1a2d4be2d0>
```

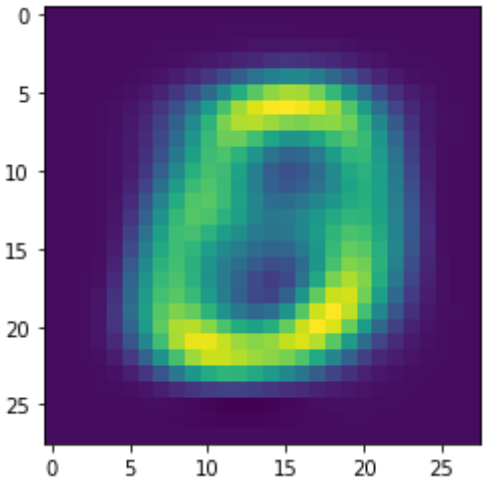


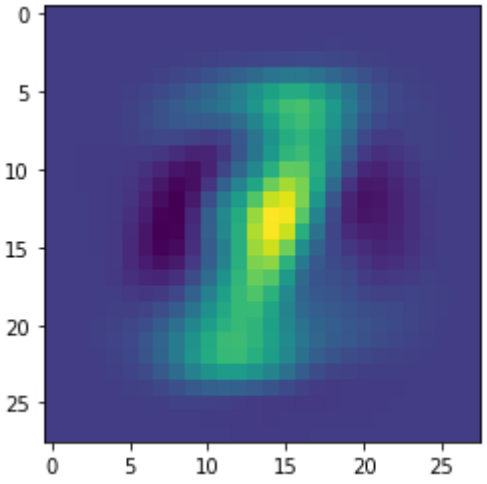

## Question 1.2.5 Unproject data back to high dimensions [10pt]

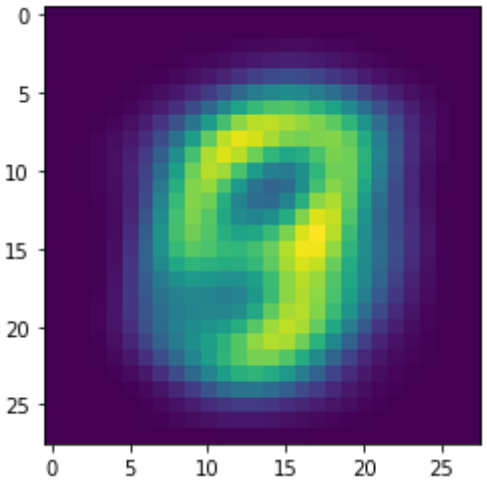
For this question, you need to project the 10 images you plotted in **1.1** on the first 2 principal components, and then unproject the "compressed" 2-D representations back to the original space. Plot the "compressed" digit (the reconstructed digit). Do they look similar to the original images?

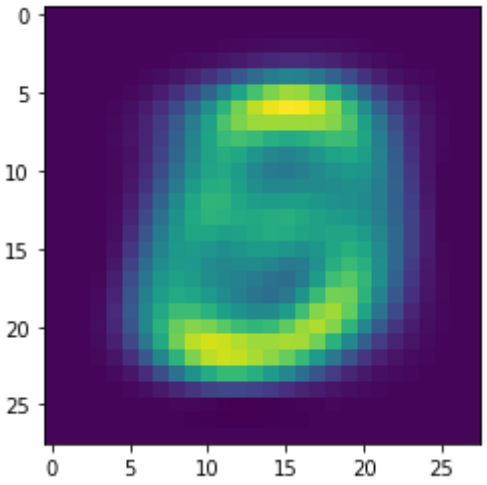
No they do no look like the image when we are using only the two largest principa components. We need to use more PCs.

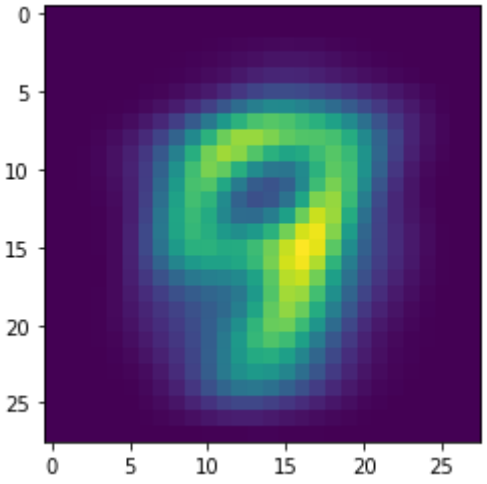
In [309]:
```
### YOUR CODE HERE
X_prime = Z @ new_basis.T
reconstructed_data = []

for image in test_images:
    plt.figure()
    current_test_image = data[image].flatten()
    #tranform the test images
    reduced_test_image = (current_test_image - data_mean) @ new_basis

    #transform them back
    test_image_prime = (reduced_test_image @ new_basis.T) + data_mean
    test_image_prime = np.array(test_image_prime)
    reconstructed_data.append(test_image_prime)

    image_shaped = np.reshape(test_image_prime, (28,28))
    plt.imshow(image_shaped)
### END OF CODE
```

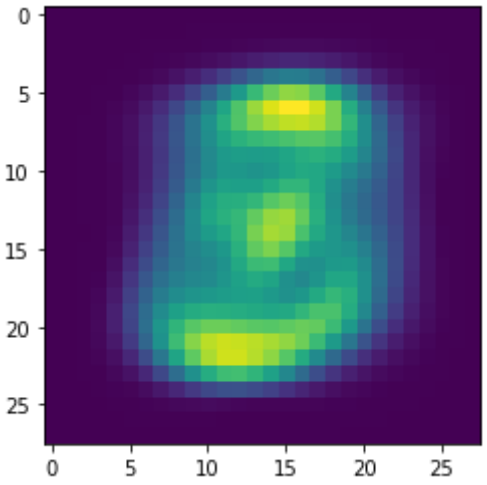

0
5
10
15
20
25
0
5
10
15
20
25

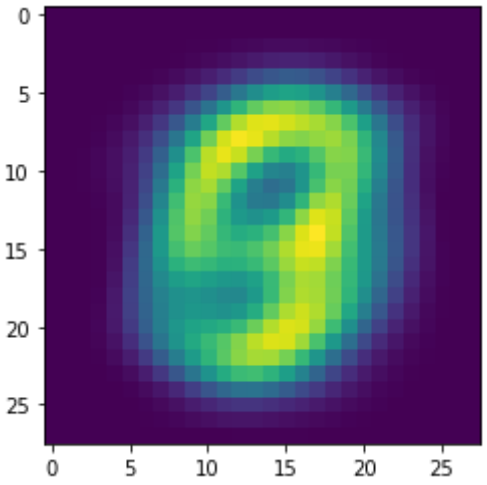

0
5
10
15
20
25
0
5
10
15
20
25

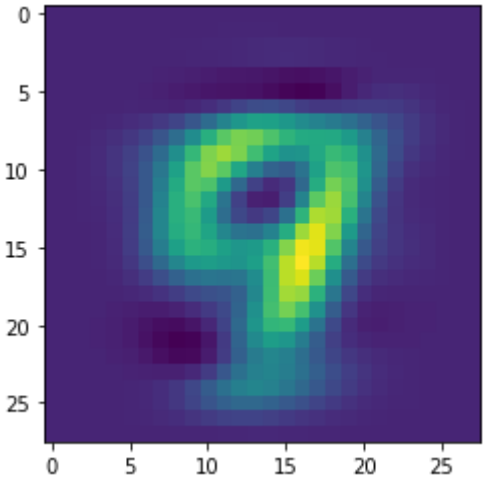

0
5
10
15
20
25
0
5
10
15
20
25

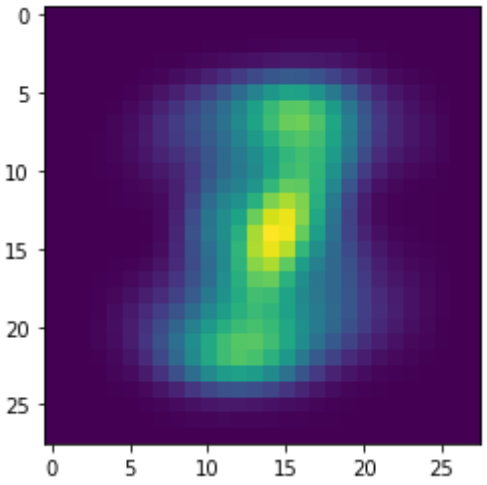

0
5
10
15
20
25
0
5
10
15
20
25

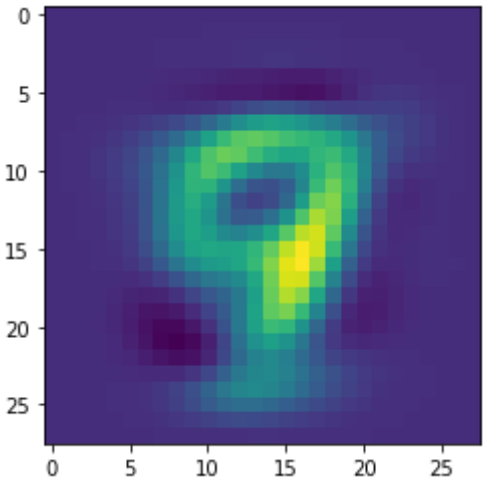

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

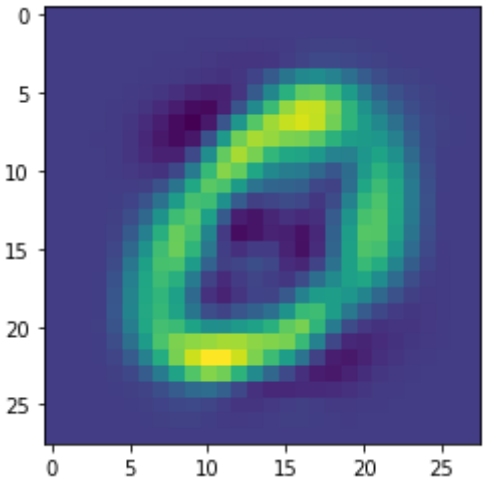


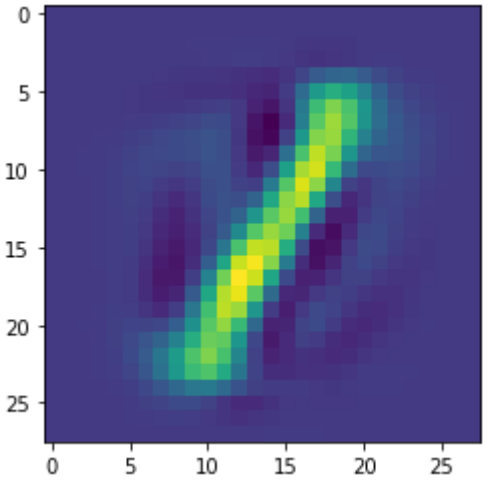

In [ ]:

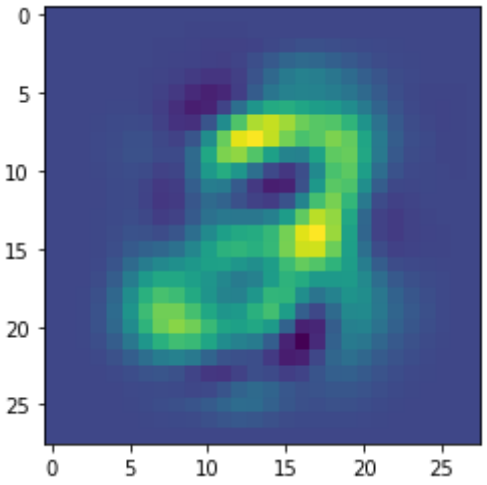
## Question 1.2.6 Choose a better low dimension space. [5pt]

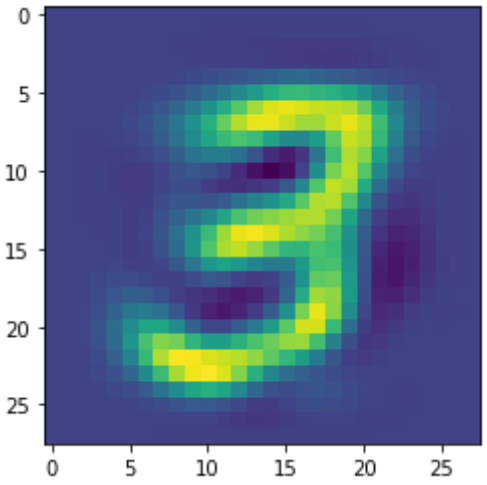
Do the previous problem with more dimensions (e.g. 3, 5, 10, 20, 50, 100). You only need to show results for one of them. Answer the following questinos. How many dimensions are required to represent the digits reasonably well? How are your results related to **question 1.2.3**?

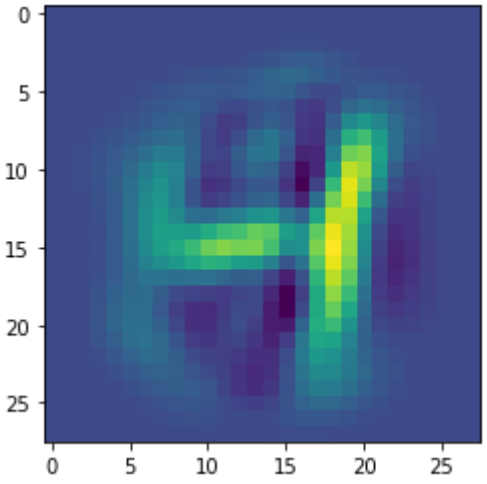
In [310]:
```
### YOUR CODE HERE
num_of_pcs = [25]
for num in num_of_pcs:
    better_basis = covmat_eig[:,eig_loc[:num]]
    for image in test_images:
        plt.figure()
        current_test_image = data[image].flatten()
        #tranform the test images
        reduced_test_image = (current_test_image - data_mean) @ better_b
asis

        #transform them back
        test_image_prime = (reduced_test_image @ better_basis.T) + data_
mean
        test_image_prime = np.array(test_image_prime)
        reconstructed_data.append(test_image_prime)

        image_shaped = np.reshape(test_image_prime, (28,28))
        plt.imshow(image_shaped)



### END OF CODE
```

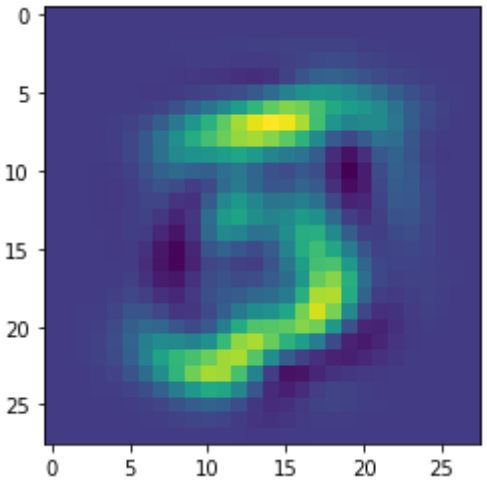

0
5
10
15
20
25
0
5
10
15
20
25

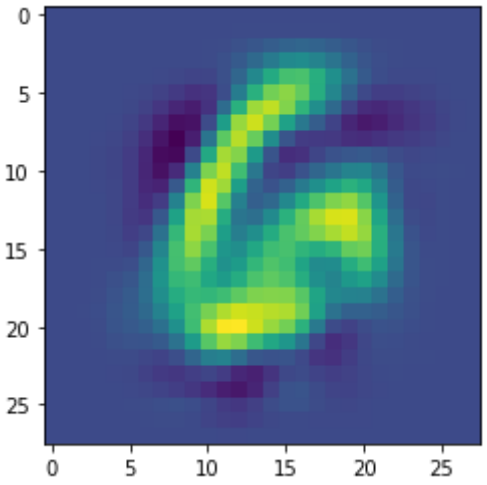

0
5
10
15
20
25
0
5
10
15
20
25

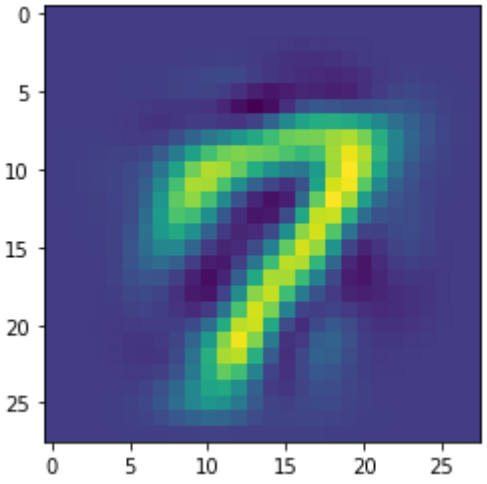

0
5
10
15
20
25
0
5
10
15
20
25

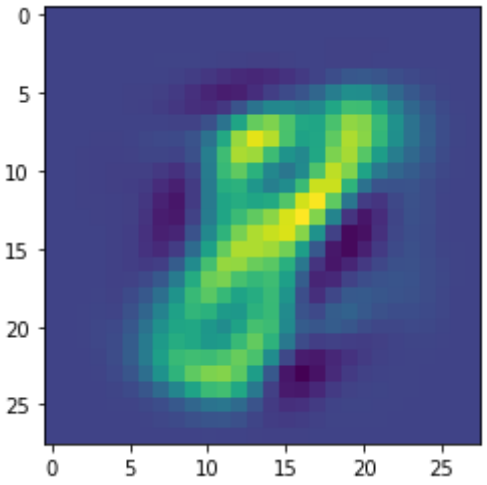

0
5
10
15
20
25
0
5
10
15
20
25

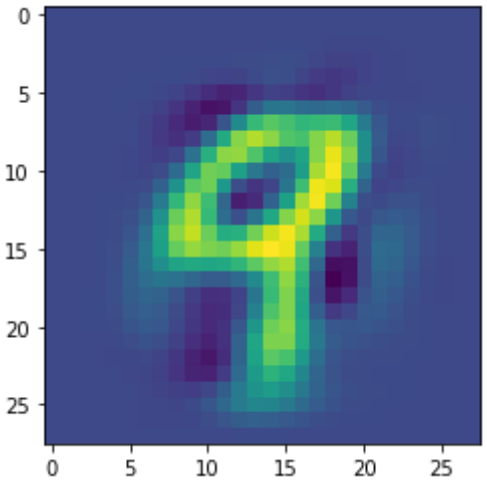

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25




(Your explanation)

I used 25 where it was to the point where I had zero issues trying to decipher what the digit was. To the point where i could identify it immediately. Using two principal components, many of digits looked like a 9 or a 0 and where therefore illegible. Using 25 dimensions you can easily identify while decreasing the dimension 30 fold.

# Question 1.3 Harris Corner and PCA [10pt]

Recall Harris corner detector algorithm:

1. Compute $x$ and $y$ derivatives ($I_x, I_y$) of an image
2. Compute products of derivatives ($I_x^2, I_y^2, I_{xy}$) at each pixel
3. Compute matrix $M$ at each pixel, where $$ M(x_0,y*0)* = *sum*{x,y} w(x-x_0,y-y_0) \begin{bmatrix}

```
I_{x}^2 & I_{x}I_{y} \\
I_{x}I_{y} & I_{y}^2
```

\end{bmatrix} $$ Here, we set weight $w(x, y)$ to be a box filter of size $3 \times 3$ (the box is placed centered at $(x_0, y_0)$).

In this problem, you need to show that Harris Corner detector is really just principal component analysis in the gradient space. Your explanation should answer the following quesions.

1. As we know, PCA is performed on data points. What are the data points in Harris corner detector when we think of it as a PCA?
2. What is the covariance matrix used in Harris corner detector and why it is a covariance matrix?
3. What are the principal components in Harris corner detector?
4. Briefly explain how principal components imply "cornerness".

(Your proof here)

1. The data points are each of the image gradients
2. The covariance matrix is the \begin{bmatrix}

```
\sum_{p \epsilon P} I_{x}^2 & \sum_{p \epsilon P} I_{x}I_{y} \\
\sum_{p \epsilon P} I_{x}I_{y} & \sum_{p \epsilon P} I_{y}^2
```

   \end{bmatrix}
3. the principle components are the lambda values (eigen values of M for each x,y). Visualizing the eigenvalues in a ellipse allows us to see the direction of the fastest and slowest change. The higher eigenvalue's eigenvector points in the direction of fast change, and same with the smaller eigenvalue.
4. The principle components tell us the points at which the eigenvectors for that M matrix that produce the largest variations when moved in any direction. Based on the C value which is = $(\lambda_1 * \lambda_2) - k * (\lambda_1 + \lambda_2)$. The higher this value, the more of a corner it is (highest variation in the directions of the eigenvectors(principle components), the lower the value it is a edge, and if it is a very small value, it is a flat region.

# Question 2 KNN, Softmax Regression

In [311]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTenso
r, download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
train_X = train_dataset.data.numpy()  # training data, uint8 type to red
uce memory and comparison cost
train_y = train_dataset.targets.numpy()  # training label
test_X = test_dataset.data.numpy() # testing data, uint8 to reduce memor
y and comparison cost
test_y = test_dataset.targets.numpy()  # testing label
```

In [312]:
```
train_X = train_X.reshape((train_X.shape[0], -1))
test_X = test_X.reshape((test_X.shape[0], -1))
```

# Question 2.1 K-Nearest Neighbor [10pt]

In this problem you will be implementing the KNN classifier. Fill in the functions in the starter code below. You are are allowed to use `scipy.spatial.KDTree` and `scipy.stats.mode` (in case of a tie, pick any one). Please avoid `sklearn.neighbors.KDTree` as it appears extremely slow. You are **not** allowed to use a library KNN function that directly solves the problem.

If you do not know what a KD-tree is, please read the documentation for `scipy.spatial.KDTree` to understand how you can use it.

Note: if you run into memory issues or neighbor queries run for more than 10 minutes, you are allowed to reduce the data size, and explain what you have done to the training data.

In [313]:
```
from scipy.spatial import KDTree
from scipy.stats import mode
```

In [314]:
```
class KNNClassifier:
    def __init__(self, num_neighbors):
        """
        construct the classifier
        Args:
            num_centers: number of neighbors
        """
        ### YOU CODE HERE
        self.num_centers = num_neighbors
        self.tree = None
        self.X = None
        self.y = None
        ### END OF CODE

    def fit(self, X, y):
        """
        train KNN classifier
        Args:
            X: training data, numpy array with shape (Nxk) where N is nu
mber of data points, k is number of features
            y: training labels, numpy array with shape (N)
        """
        ### YOU CODE HERE
        self.X = X
        self.y = y

        self.tree = KDTree(X)
        ### END OF CODE
        return self

    def predict(self, X):
        """
        predict labels
        Args:
            X: testing data, numpy array with shape (Mxk) where M is num
ber of data points, k is number of features
        Return:
            y: predicted labels, numpy array with shape (N)
        """
        pred = []
        ### YOU CODE HERE
        for x in X:
            d, i = self.tree.query([x], k=self.num_centers)
            neighbors_labels = []
            for index in i[0]:
                neighbors_labels.append(self.y[index])
            most_common, _ = mode(neighbors_labels)
            pred.append(most_common[0])
        ### END OF CODE
        return pred
```

In [315]:
```
from sklearn.metrics import accuracy_score
knn = KNNClassifier(6).fit(train_X, train_y)
pred_y = knn.predict(test_X)
# print(pred_y)
print('KNN accuracy:', accuracy_score(test_y, pred_y))
```

```
/Users/joshuakang/opt/anaconda3/lib/python3.7/site-packages/scipy/spati
al/kdtree.py:388: RuntimeWarning: overflow encountered in ubyte_scalars
  sd[node.split_dim] = np.abs(node.split-x[node.split_dim])**p

KNN accuracy: 0.5727
```

# Question 2.2 Softmax Regression

In this problm, you will be implementing the softmax regression(multi-class logistic regression). Here is a brief recap of several important concepts. In the following explanation, I will use $x$ for data vector, $y'$ for ground truth label, and $y$ for predicted label.

Suppose we have a problem where we need to classify data points into $m$ classes.

1. Softmax function $S$ normalize a vector to have sum 1. (it turns any vector into a probability distribution)
$$S(x) = [\frac{e^{x_1}}{\sum_{j=1}^{m} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{m} e^{x_j}}, \dots, \frac{e^{x_m}}{\sum_{j=1}^{m} e^{x_j}}]$$
2. Cross entropy loss $J$ is the multiclass logistic regression loss.
$$J(y', y) = -\sum_{i=1}^{m} y'_i \log y_i$$
where $y'$ is the one-hot ground truth label and $y$ is the predicted label distribution.
3. Softmax regression is the following optimization problem.
$$\min_{W,b} \sum_{(X,y') \in \{\text{training set}\}} J(y', S(Wx + b))$$
where $W$ has shape $(m \times k)$ where $k$ is the number of features in a data point; $b$ is a $m$ dimensional vector.
4. This objective is optimized with gradient descent. Let
$$L = \sum_{(x,y') \in \{\text{training set}\}} J(y', S(Wx + b))$$
Update $W$ and $b$ with $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$.

## Question 2.2.1 Compute the gradients [10pt]

In this question, you need to do the following:

1. Compute the gradient $\frac{\partial J}{\partial y}$. i.e. compute

$$\frac{\partial J}{\partial y_i}$$

Express it in terms of $y_i'$ and $y_i$.

$$\frac{\partial J}{\partial y'} = -\sum_{i=1}^{m} \frac{y_i}{y_i'}$$

2. Let $u = Wx + b$, $y_i = S_i(u_j)$ Compute

$$\frac{\partial y_i}{\partial u_j}$$

Express it in terms of $y_i$, $y_j$ and $\delta_{ij}$, where

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

We have the case in which when we are getting the derivative of e^{a_j} (the numerator) that we need to check if the $S_i'$'s i == j. To get the derivative of the fraction, we use the quotient rule. So first when i == j :

$$= \frac{(e^{a_i} \sum_{k=1}^{N} e^{a_k}) - e^{a_j} e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2}$$

$$= \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} * \frac{(\sum_{k=1}^{N} e^{a_k}) - e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$= S_i(1 - S_j)$$

and if i = j, then this is equivalent to

$$S_j(1 - S_j) = S_j - S_j^2$$

Now in the second case that i != j, and using the same quotient rule :

$$\frac{0 - e^{a_j} e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2}$$

$$= -\frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$= -S_j S_i$$

Instead of leaving it as a system of equations, we can introduce a new variable to make it a single equation, $\delta$:

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Where

$$= \frac{\partial y_i}{\partial u_j} = S_i(\delta_{ij} - S_j)$$

1. Compute

$$\frac{\partial J}{\partial W_{jk}}$$

$$\text{and } \frac{\partial J}{\partial b_j}$$

Express them in terms of $y_j, y'_j, x_k$. Explain your results in an intuitive way. Hint: the results should have a very simple form that makes sense intuitively. In order to calculate the $\frac{\partial J}{\partial W_{jk}}$, we must use chain rule, therefore :

$$\frac{\partial J}{\partial W_{jk}} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial W_{jk}}$$

Where a is the weighted sum of the inputs. This then results in :

$$\begin{aligned}
&(\sum_{i!=k}^{m} \frac{-y'_k}{y_k}(-y_k y_i) + \sum_{i=k}^{m} y'_k + y'_k y_i) \\
&= (\sum_{i!=k}^{m} y'_k y_i + \sum_{i=k}^{m} -y'_k + y'_k y_i) \\
&= (\sum_{i!=k}^{m} y'_k y_i - \sum_{i=k}^{m} y'_k + \sum_{i=k}^{m} y'_k y_i) \\
&= (\sum_{k}^{m} y'_k y_i - \sum_{k=i}^{m} y'_k) \\
&= (1 * y_i - y'_i) \\
&= (y_i - y'_i)
\end{aligned}$$

The final term of the chain rule is the partial of the weighted sum with respect to the weight that we are updating. In this case, the result is simply the $x_i$ (the input that we multiply by the $w_{ij}$ with to add to the sum of the total weighted sum.

So putting them together, we get for a weight $w_{ij}$ the gradient is :

$$(y_j - y'_j)x_i$$

For the bias, it is simply :

$$(y_j - y'_j) * 1$$

since the input for the bias is always 1.

1. Compute

$$\frac{\partial J}{\partial W}$$

in the matrix form. It should be a matrix with the same shape as $W$, and entry $jk$ is $\frac{\partial J}{\partial W_{jk}}$. Similarly, compute

$$\frac{\partial J}{\partial b}$$

Given the input to the layer is X such that the shape is N x d (N : number of datapoints, d : number of dimensions). Also, the gradient is of shape : N x o (o : number of output units of layer). The matrix multiplication for the regular inputs is X.T @ -Gradient, and for the bias it is a [1 x n] @ -Gradient.

(Your proof here)

## Question 2.2.2 Stochastic Gradient Descent [10pt]

In gradient descent algorithm, we update $W$ and $b$ with $\partial L/\partial W$ and $\partial L/\partial b$. However, this requires the gradient w.r.t. the whole dataset. Computing such gradient is very slow. Instead, we can update the weights with per-data gradient. This is known as the SGD algorithm, which runs much faster. You need to take the following steps.

1. Implement softmax function $S$. We need to take special care in this function since $e^x$ tends to overflow easily. However, we observe that $S(x) = S(x - m)$ for any constant vector $m$. We can stabilize softmax using $S(x) = S(x - \max(x))$.
2. Implement function `J` (loss) and `dJ` (loss gradient). Note: `J` is not required to run the algorithm, but you may want to implement it for debug purposes.
3. Implement the SGD algorithm.
4. Run the algorithm for 20 epochs (each epoch iterates the whole data set once) with learning rate `1e-3` and report accuracy on test set. You may use `sklearn.metrics.accuracy_score`. You need to achieve accuracy > 90%. You are allowed to experiment with different epoch numbers and learning rates (even learning rate decay) to achieve this accuracy, but they are not required.

You may use print (or progress bar packages) to track the training progress since it might take several minutes.

In [197]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTenso
r, download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
train_X = train_dataset.data.numpy() / 255.  # normalize data to 0-1
train_y = train_dataset.targets.numpy()  # training label
test_X = test_dataset.data.numpy() / 255.  # normalize data to 0-1
test_y = test_dataset.targets.numpy()  # testing label
train_X = train_X.reshape((train_X.shape[0], -1))  # flatten the image
test_X = test_X.reshape((test_X.shape[0], -1))  # flatten the image
```

In [316]:
```
def softmax(x):
    """
    softmax function
    Args:
        x: a 1-d numpy array
    Return:
        results of softmax(x)
    """
    ### YOUR CODE HERE
    exp_x = np.exp(x)

    softmax_output = exp_x / (np.sum(exp_x))
    return softmax_output
#     return softmax_output - np.amax(softmax_output)
    ### END OF CODE

def J(W, b, y_true, x):
    """
    Softmax Loss function
    Args:
        W: weights (num_classes x num_features)
        b: bias (num_classes)
        y_true: ground truth 1-hot label (num_classes)
        x: input data
    Return:
        J(y', y)
    """
    ### YOUR CODE HERE
    weighted_sum = W @ x + b
    activation = softmax(weighted_sum)

    error = 0
    for m in x.shape[0]:
        error += y_true * np.log(activation[m])

    return -error


    ### END OF CODE

def dJ(W, b, y_true, x):
    """
    Softmax Loss gradient
    Args:
        W: weights (num_classes x num_features)
        b: bias (num_features)
        y_true: ground truth 1-hot label (num_classes)
        x: input data (num_features)
    Return:
        (dW, db): gradient w.r.t. W and b
    """
    ### YOUR CODE HERE
    weighted_sum = W @ x + b

#     print('weighted sum')
#     print(weighted_sum)
```

```
#     print('activations')
    activation = softmax(weighted_sum)
    delta = activation - y_true
#     print(activation)
    dW = delta[np.newaxis].T @ x[np.newaxis]
    db =  delta[np.newaxis].T @ np.ones((1,1))

    return dW, np.reshape(db, (10,))

    #get weighted sums
    #put it in the softmax, and get the y_pred
    #do y - t and that is the upstream gradient
    #dJ/dW is x.T @ (y-t)
    #dJ/db is (ones column) @ (y-t)

    #Then W = W - lr * dJ/dW
    #and b = b - lr * dJ/db



    ### END OF CODE
```

In [317]:
```
from tqdm import tqdm_notebook
```

In [217]:
```
def SGD(f, df, Xs, ys, n_classes=10, lr=1e-3, max_epoch=20):
    """
    Args:
        f: function to optimize
        df: the gradient of the function
        Xs: input data, numpy array with shape (num_data x num_features)
        ys: true label, numpy array with shape (num_data x num_classes)
        lr: learning rate
        max_epoch: maximum epochs to run SGD
    Return:
        optimal weights and biases
    """
    N, m = Xs.shape
    W = np.random.rand(n_classes, m) - 0.5  # you do not need to change
 random initialization
    b = np.random.rand(n_classes) - 0.5

    for epoch in range(max_epoch):
        for x,y in zip(Xs, ys):
            dW, db = dJ(W, b, y, x)
            W -= lr * dW
            b -= lr * db



    ### YOUR CODE HERE
    ### END OF CODE
    return W, b
```

In [218]:
```
train_y_onehot = np.zeros((train_y.shape[0], 10))
train_y_onehot[np.arange(len(train_y)), train_y] = 1
W, b = SGD(J, dJ, train_X, train_y_onehot, 10, max_epoch=20)
accuracy_score(test_y, np.argmax(test_X @ W.T + b, axis=1))
```

Out[218]: 0.9182

# Question 3 Convolutional Neural Networks

This question requires you to use the PyTorch framework for neural network training. You will not need GPU to train the networks for this problem.

The following is a code sample for training a simple multi-layer perceptron neural network using PyTorch. Running it should give you about 98% testing accuracy.

Since network training takes long, I recommend installing the tqdm package for progress tracking.

In [318]:
```
from tqdm import tqdm_notebook
```

In [319]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTenso
r(), download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
```

In [320]:
```
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        """init function builds the required layers"""
        super(MLP, self).__init__()  # This line is always required
        # Hidden layer
        self.layer1 = nn.Linear(input_size, hidden_size)
        # activation
        self.relu = nn.ReLU()
        # output layer
        self.layer2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        """forward function describes how input tensor is transformed to
output tensor"""
        # flatten the input from (Nx1x28x28) to (Nx784)
        torch.flatten(x, 1)
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        # Note we do not need softmax layer, since this layer is include
d in the CrossEntropyLoss provided by torch
        return x
```

In [321]:
```
model = MLP(784, 1024, 10)
model
```

Out[321]:
```
MLP(
  (layer1): Linear(in_features=784, out_features=1024, bias=True)
  (relu): ReLU()
  (layer2): Linear(in_features=1024, out_features=10, bias=True)
)
```

In [322]:
```
opts = {
    'lr': 5e-4,
    'epochs': 5,
    'batch_size': 64
}
```

In [323]:
```
optimizer = torch.optim.Adam(model.parameters(), opts['lr'])  # Adam is
 a much better optimizer compared to SGD
criterion = torch.nn.CrossEntropyLoss()  # loss function
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_
size=opts['batch_size'], shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_si
ze=opts['batch_size'], shuffle=True)
```

In [324]:
```
for epoch in range(opts['epochs']):
    train_loss = []
    for i, (data, labels) in tqdm_notebook(enumerate(train_loader), tota
l=len(train_loader)):
        # reshape data
        data = data.reshape([-1, 784])
        # pass data through network
        outputs = model(data)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()  # Important! Otherwise the optimizer will
accumulate gradients from previous runs!
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
    test_loss = []
    test_accuracy = []
    for i, (data, labels) in enumerate(test_loader):
        # reshape data
        data = data.reshape([-1, 784])
        # pass data through network
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
        loss = criterion(outputs, labels)
        test_loss.append(loss.item())
        test_accuracy.append((predicted == labels).sum().item() / predic
ted.size(0))
    print('epoch: {}, train loss: {}, test loss: {}, test accuracy: {}'.
format(epoch, np.mean(train_loss), np.mean(test_loss), np.mean(test_accu
racy)))
```

```
epoch: 0, train loss: 0.28823375489030567, test loss: 0.136084674033010
08, test accuracy: 0.960390127388535

epoch: 1, train loss: 0.11209188347289176, test loss: 0.090257487675044
57, test accuracy: 0.971437101910828

epoch: 2, train loss: 0.07314085077667144, test loss: 0.081120974455432
15, test accuracy: 0.9755175159235668

epoch: 3, train loss: 0.05211033039951701, test loss: 0.061676299605803
314, test accuracy: 0.980593152866242

epoch: 4, train loss: 0.037828655509865965, test loss: 0.06651330238006
486, test accuracy: 0.9786027070063694
```

# Question 3.1 Implementing CNN [15pt]

You need to implement a convolutional neural network for the same task as above. You may find the PyTorch documentation helpful. https://pytorch.org/docs/stable/nn.html (https://pytorch.org/docs/stable/nn.html)

We provide a working network structure below. You can adjust the network size and training options for better performance, but a correct implementation of the provided network should give you the required accuracy. For convolutional layers, (conv MxM, N) means the layer has kernel size $M$ by $M$ and $N$ output channels; for pooling layers, (maxpool MxM) means doing max pooling with kernel size $M$ by $M$.

(conv 5x5, 32) -> (relu) -> (maxpool 2x2) -> (conv 5x5, 64) -> (relu) -> (maxpool 2x2) -> (flatten) -> (linear 10) -> (output)

For full score, you need to achieve 99% testing accuracy. Also, plot the hand-written digits that your network got wrong.

In [362]:
```
class CNN(nn.Module):
    def __init__(self, input_size, num_classes):
        """
        init convolution and activation layers
        Args:
            input_size: (1,28,28)
            num_classes: 10
        """
        super(CNN, self).__init__()
        ### YOUR CODE HERE
        self.first = nn.Conv2d(1, 32 , 5, stride=1, padding=2, bias=Fals
e)

        self.main = nn.Sequential(
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32 , 64, 5,stride=1, padding=2, bias=False),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.fc = nn.Sequential(
            nn.Linear(3136, num_classes)
        )
        ### END OF CODE


    def forward(self, x):
        """
        forward function describes how input tensor is transformed to ou
tput tensor
        Args:
            x: (Nx1x28x28) tensor
        """
        ### YOUR CODE HERE
        x = self.first(x)
        x = self.main(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc(x)
        ### END OF CODE
        return x
```

In [363]:
```
model = CNN((1, 28, 28), 10)
model
```

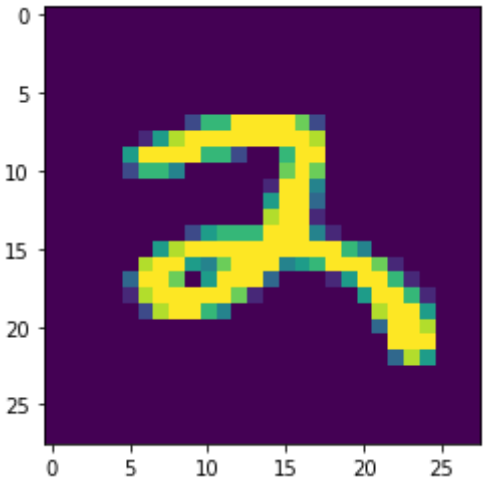
Out[363]:
```
CNN(
  (first): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), bias=False)
  (main): Sequential(
    (0): ReLU(inplace=True)
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil
_mode=False)
    (2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), bias=False)
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil
_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=3136, out_features=10, bias=True)
  )
)
```

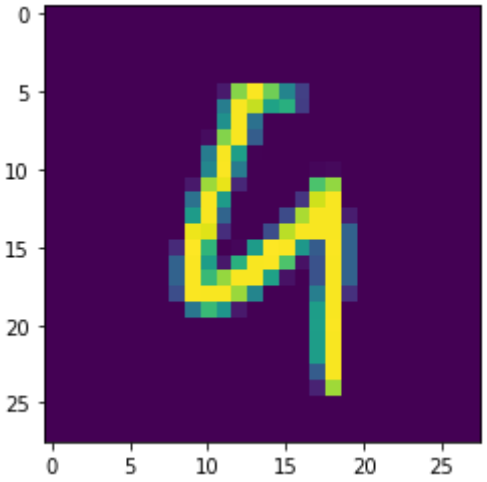
In [369]:
```
### You may (and should) change these
opts = {
    'lr': 1e-3,
    'epochs': 10,
    'batch_size': 64
}

### if you cannot get 99% with SGD, Adam optimizer can help you
optimizer = torch.optim.Adam(model.parameters(), opts['lr'])
```

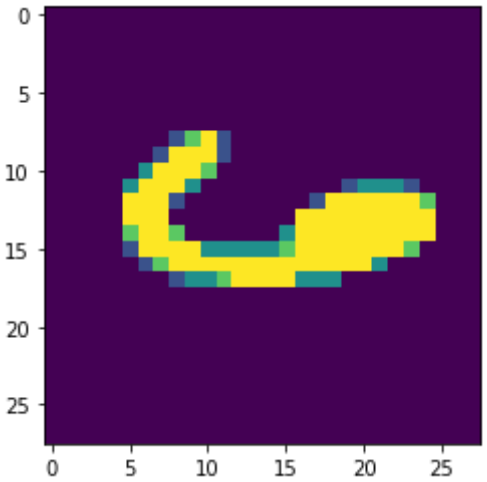
In [370]:
```
criterion = torch.nn.CrossEntropyLoss()  # loss function
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_
size=opts['batch_size'], shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_si
ze=opts['batch_size'], shuffle=True)
```

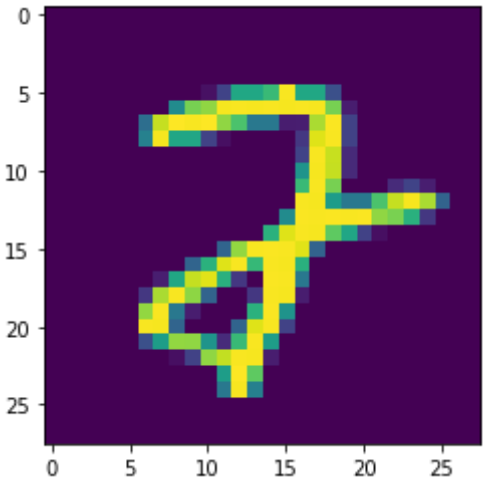
In [ ]:
```
for epoch in range(opts['epochs']):
    train_loss = []
    for i, (data, labels) in tqdm_notebook(enumerate(train_loader), tota
l=len(train_loader)):
        # pass data through network
        outputs = model(data)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
    test_loss = []
    test_accuracy = []
    for i, (data, labels) in enumerate(test_loader):
        # pass data through network
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
        loss = criterion(outputs, labels)
        test_loss.append(loss.item())

        wrong = []
        for label_i in range(len(labels)):
            if predicted[label_i] != labels[label_i]:
                wrong.append(np.squeeze(data[label_i]))
                plt.figure()
                plt.imshow(wrong[-1])
        test_accuracy.append((predicted == labels).sum().item() / predic
ted.size(0))
    print('epoch: {}, train loss: {}, test loss: {}, test accuracy: {}'.
format(epoch, np.mean(train_loss), np.mean(test_loss), np.mean(test_accu
racy)))
```

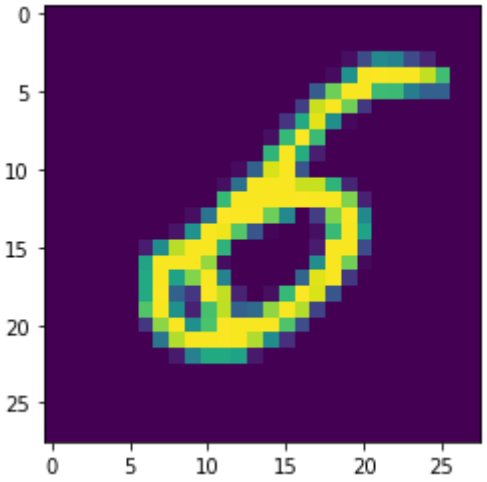
```
/Users/joshuakang/opt/anaconda3/lib/python3.7/site-packages/ipykernel_l
auncher.py:24: RuntimeWarning: More than 20 figures have been opened. F
igures created through the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may consume too much memor
y. (To control this warning, see the rcParam `figure.max_open_warning
`).

epoch: 0, train loss: 0.019554508298107145, test loss: 0.02285140537961
5997, test accuracy: 0.9931329617834395


epoch: 1, train loss: 0.013642958154125895, test loss: 0.03457956339269
142, test accuracy: 0.9898487261146497


---------------------------------------------------------------------
----
KeyboardInterrupt                         Traceback (most recent call l
ast)
<ipython-input-371-502b2b3d73b1> in <module>
      6         loss = criterion(outputs, labels)
      7         optimizer.zero_grad()
----> 8         loss.backward()
      9         optimizer.step()
     10         train_loss.append(loss.item())

~/opt/anaconda3/lib/python3.7/site-packages/torch/tensor.py in backward
(self, gradient, retain_graph, create_graph)
    148                 products. Defaults to ``False``.
    149         """
--> 150         torch.autograd.backward(self, gradient, retain_graph, c
reate_graph)
    151
    152     def register_hook(self, hook):

~/opt/anaconda3/lib/python3.7/site-packages/torch/autograd/__init__.py
 in backward(tensors, grad_tensors, retain_graph, create_graph, grad_va
riables)
     97     Variable._execution_engine.run_backward(
     98         tensors, grad_tensors, retain_graph, create_graph,
---> 99         allow_unreachable=True)  # allow_unreachable flag
    100
    101

KeyboardInterrupt:
```

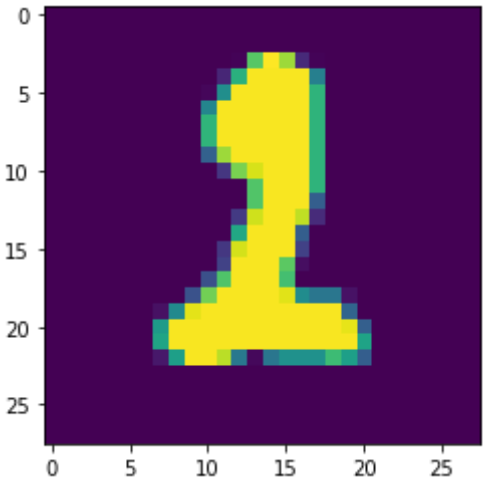

0
5
10
15
20
25
0
5
10
15
20
25

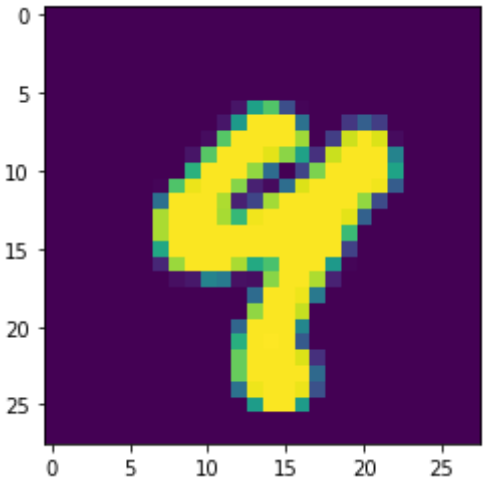

0
5
10
15
20
25
0
5
10
15
20
25

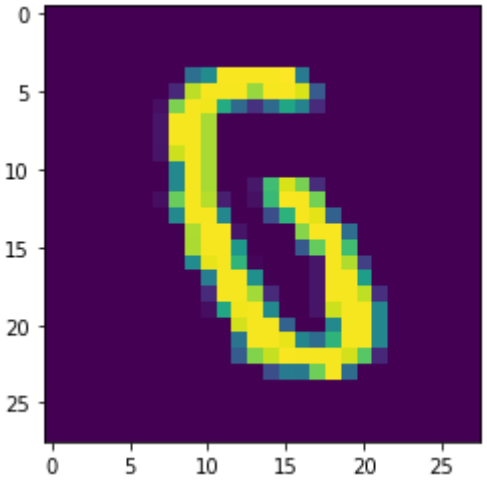

0
5
10
15
20
25
0
5
10
15
20
25

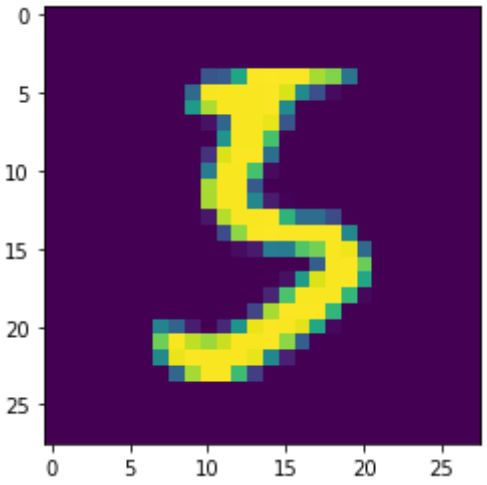

0
5
10
15
20
25
0
5
10
15
20
25

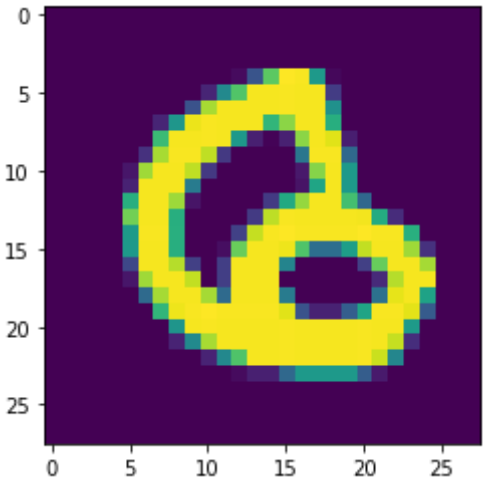

0
5
10
15
20
25
0
5
10
15
20
25

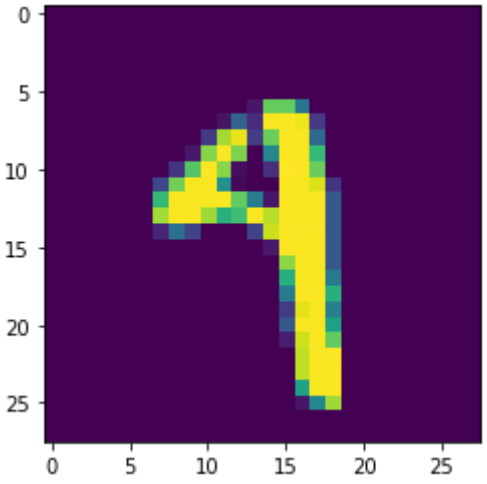

0
5
10
15
20
25
0
5
10
15
20
25

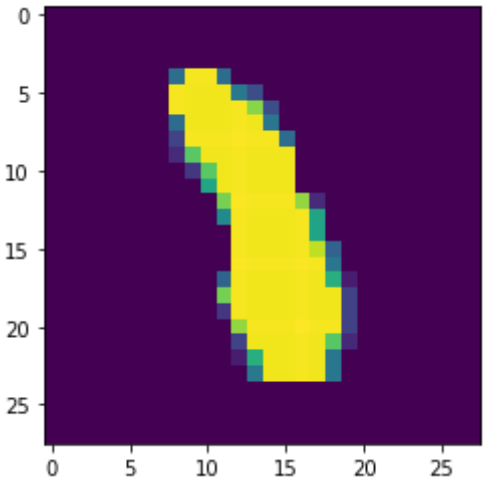

0
5
10
15
20
25
0
5
10
15
20
25

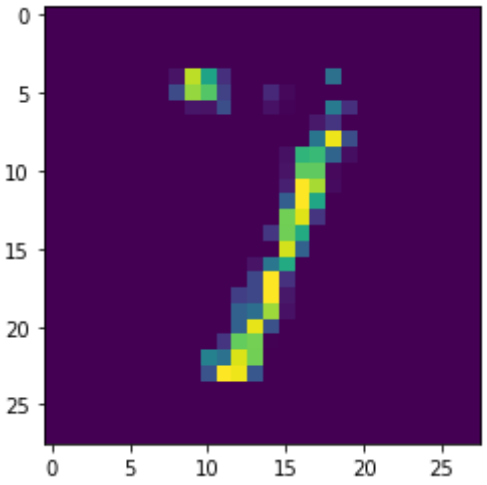

0
5
10
15
20
25
0
5
10
15
20
25

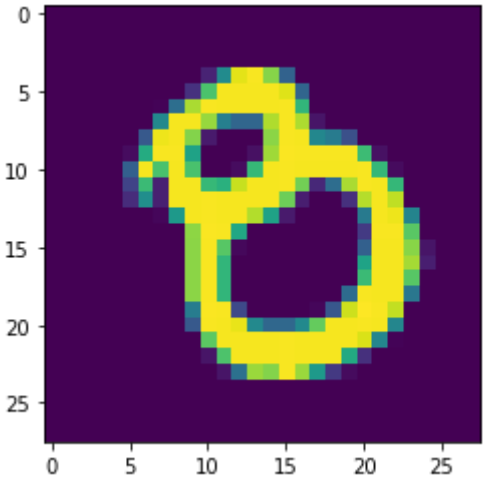

0
5
10
15
20
25
0
5
10
15
20
25

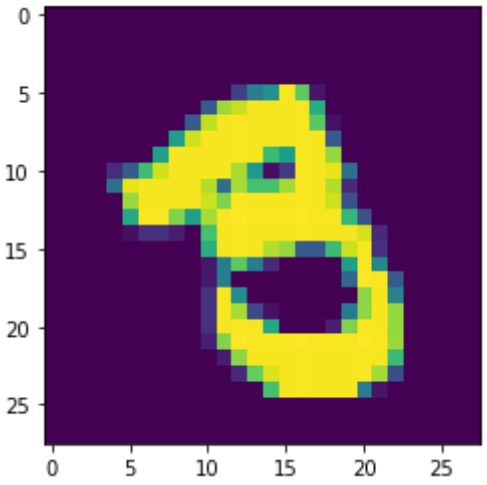

0
5
10
15
20
25
0
5
10
15
20
25

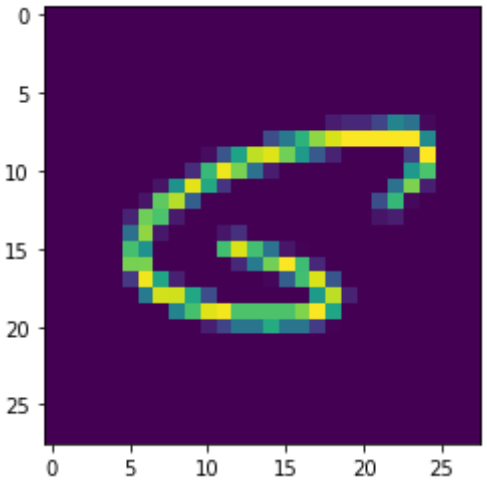

0
5
10
15
20
25
0
5
10
15
20
25

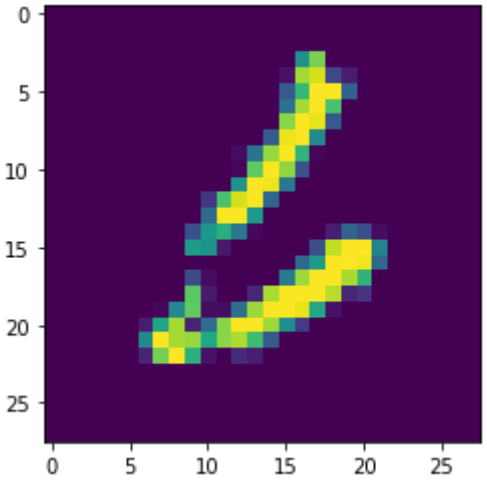

0
5
10
15
20
25
0
5
10
15
20
25

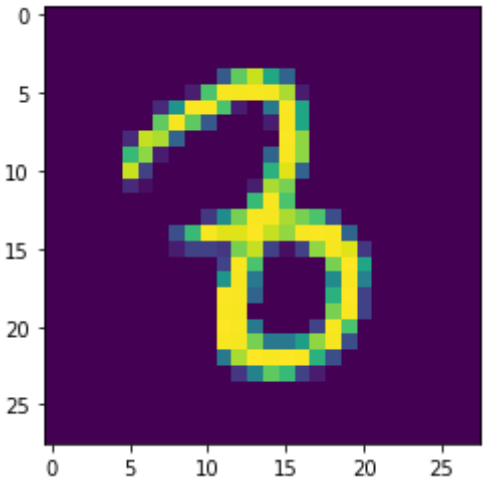

0
5
10
15
20
25
0 5 10 15 20 25

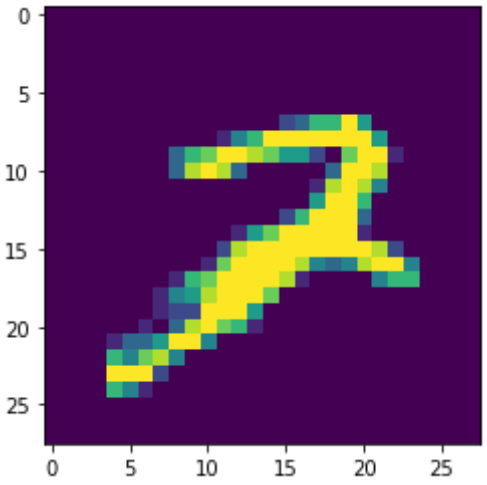

0
5
10
15
20
25
0 5 10 15 20 25

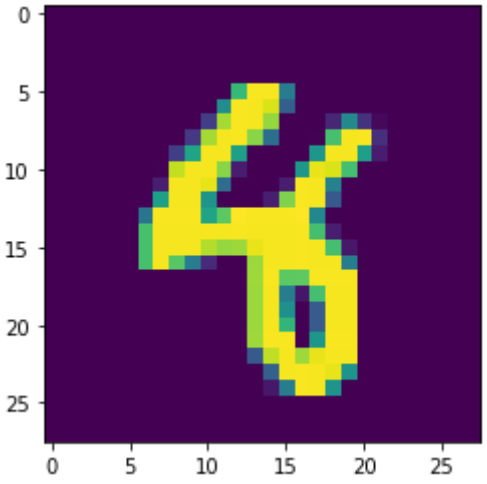

0
5
10
15
20
25
0 5 10 15 20 25

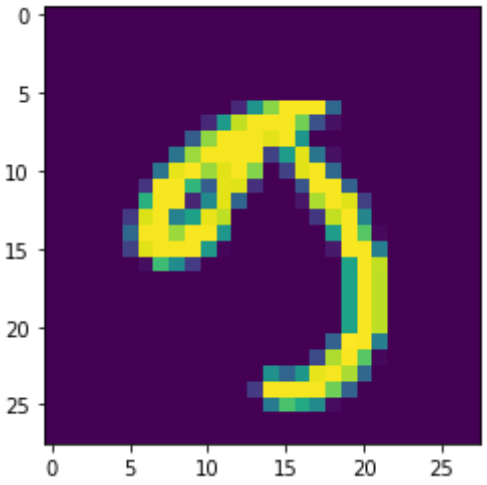

0
5
10
15
20
25
0
5
10
15
20
25

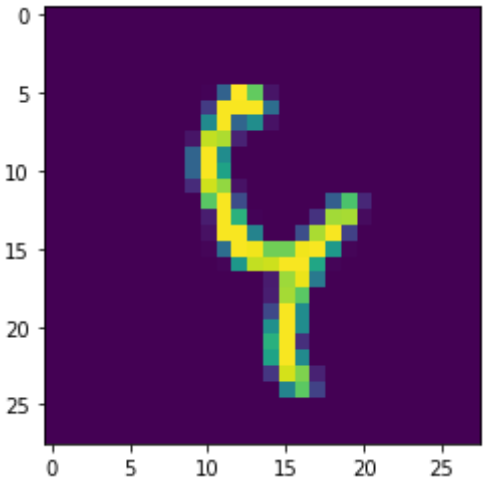

0
5
10
15
20
25
0
5
10
15
20
25

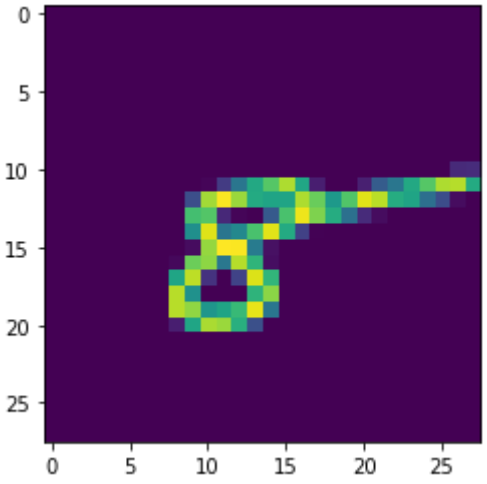

0
5
10
15
20
25
0
5
10
15
20
25

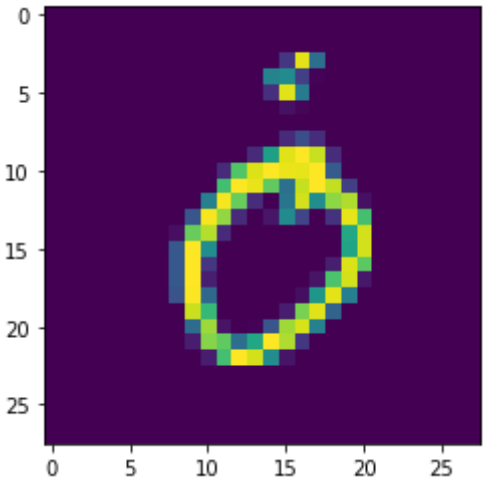

0
5
10
15
20
25
0
5
10
15
20
25

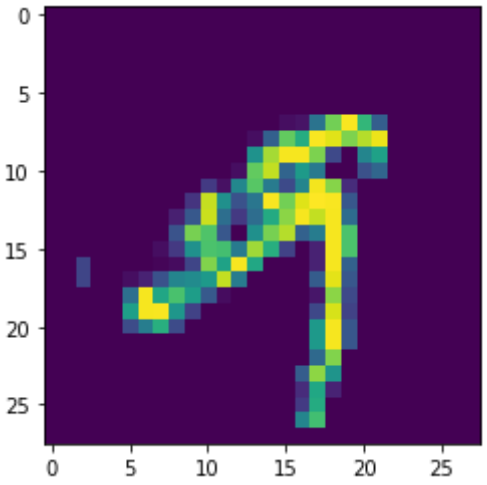

0
5
10
15
20
25
0
5
10
15
20
25

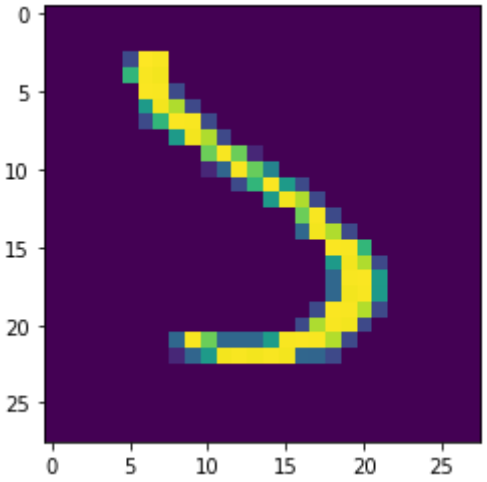

0
5
10
15
20
25
0
5
10
15
20
25

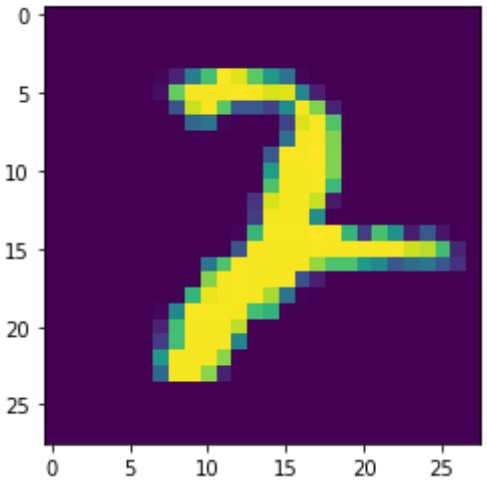

0
5
10
15
20
25
0
5
10
15
20
25

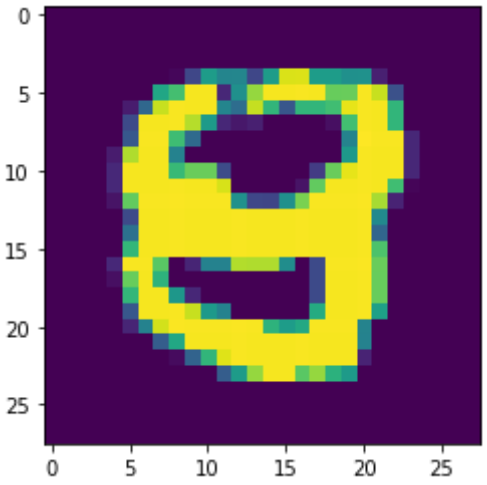

0
5
10
15
20
25
0
5
10
15
20
25

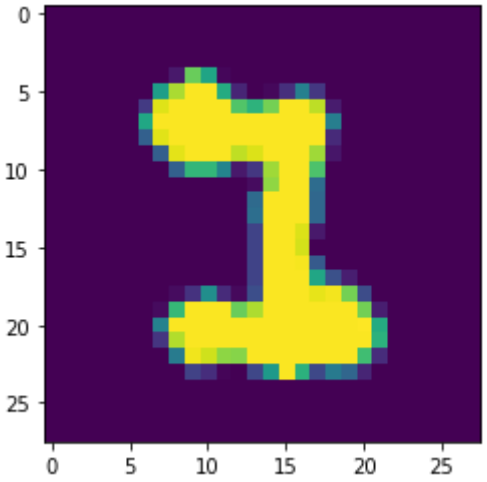

0
5
10
15
20
25
0
5
10
15
20
25

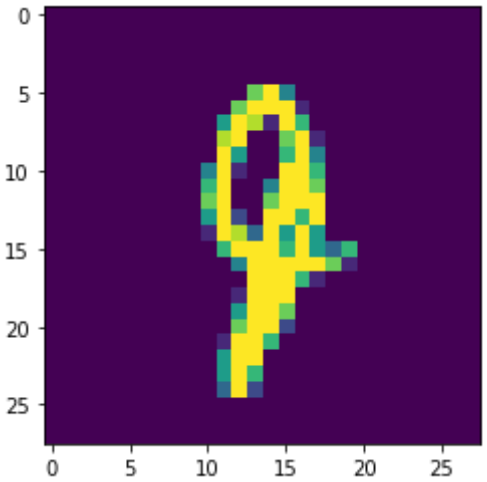

0
5
10
15
20
25
0
5
10
15
20
25

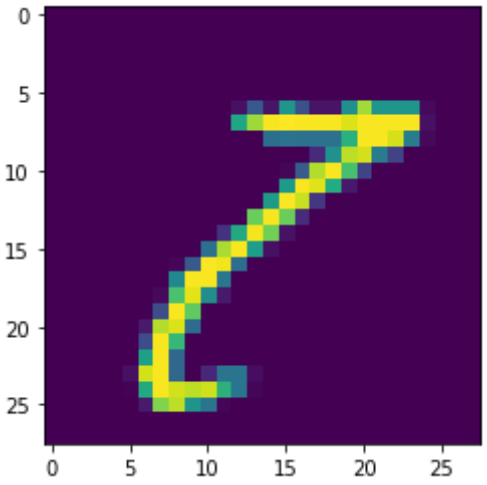

0
5
10
15
20
25
0
5
10
15
20
25

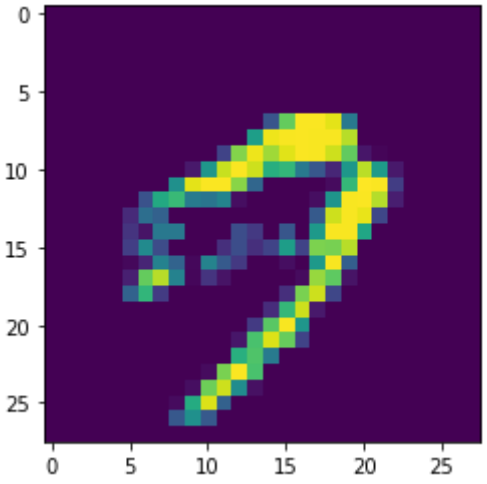

0
5
10
15
20
25
0
5
10
15
20
25

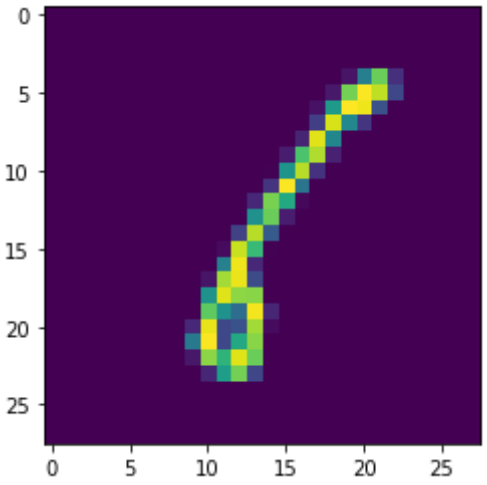

0
5
10
15
20
25
0
5
10
15
20
25

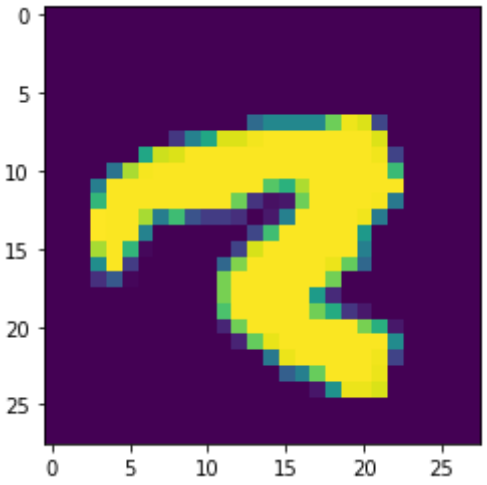

0
5
10
15
20
25
0
5
10
15
20
25

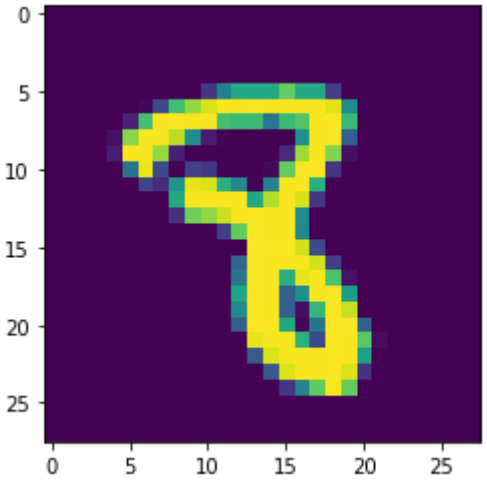

0
5
10
15
20
25
0
5
10
15
20
25

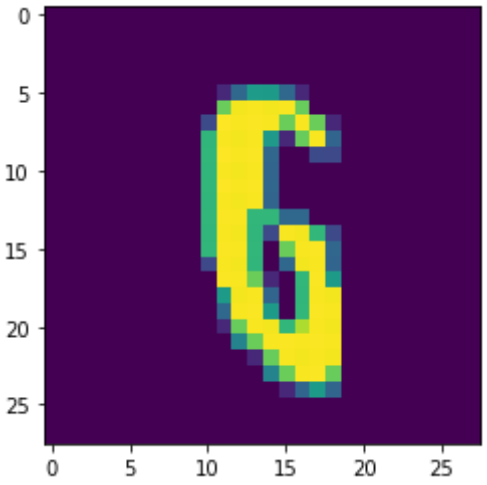

0
5
10
15
20
25
0
5
10
15
20
25

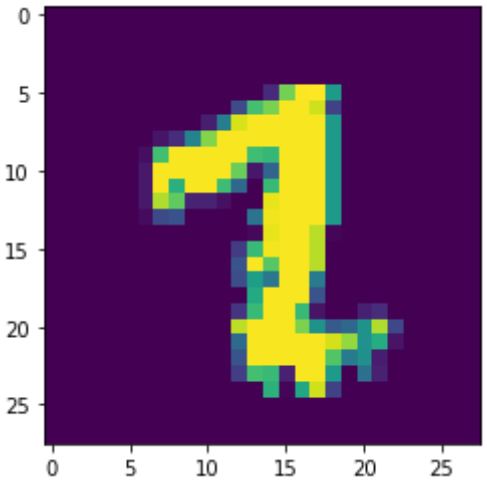

0
5
10
15
20
25
0
5
10
15
20
25

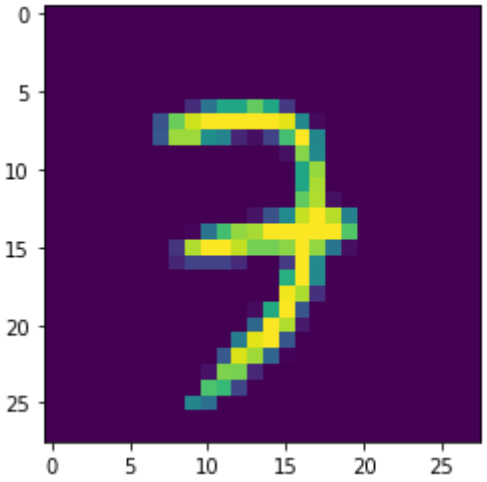

0
5
10
15
20
25
0
5
10
15
20
25

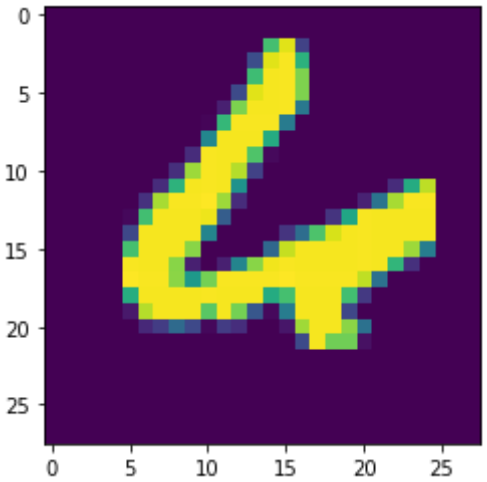

0
5
10
15
20
25
0
5
10
15
20
25

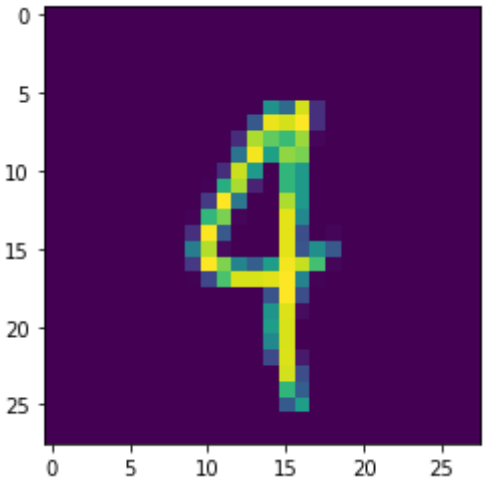

0
5
10
15
20
25
0
5
10
15
20
25

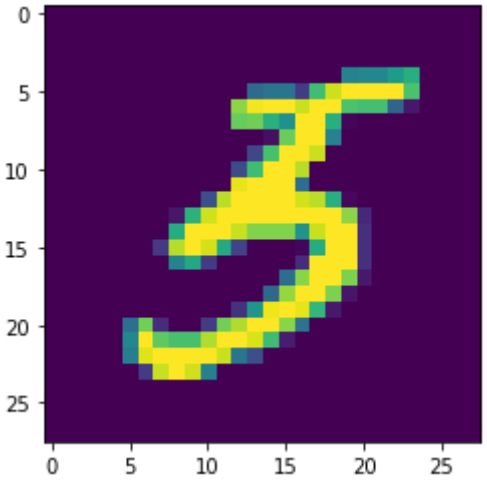

0
5
10
15
20
25
0
5
10
15
20
25

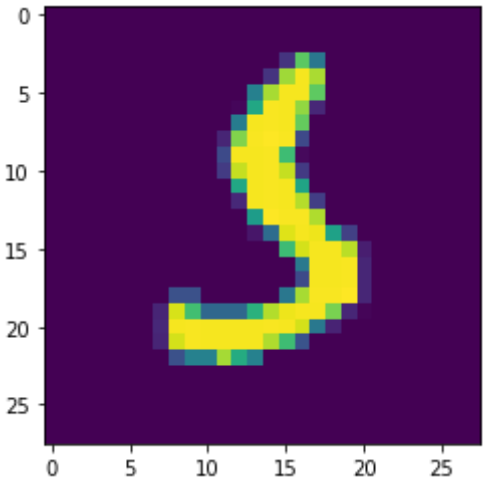

0
5
10
15
20
25
0
5
10
15
20
25

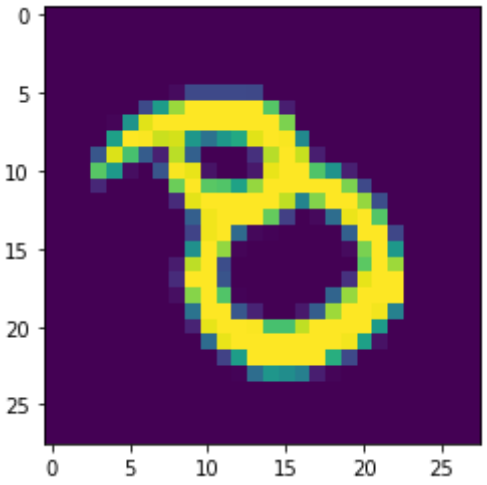

0
5
10
15
20
25
0
5
10
15
20
25

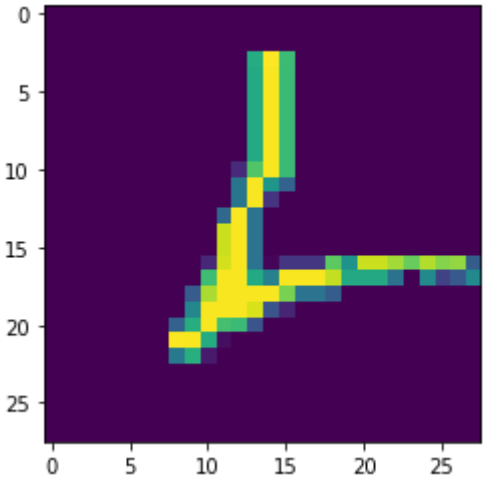

0
5
10
15
20
25
0
5
10
15
20
25

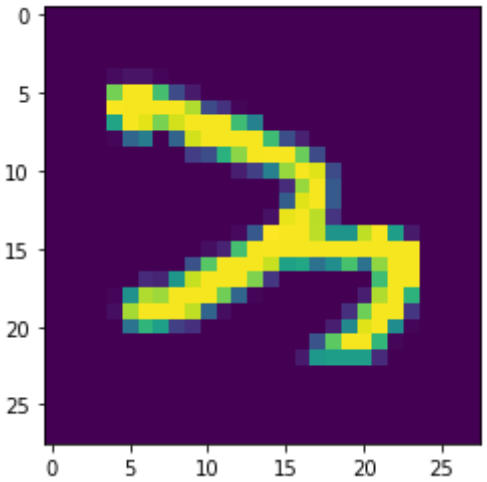

0
5
10
15
20
25
0
5
10
15
20
25

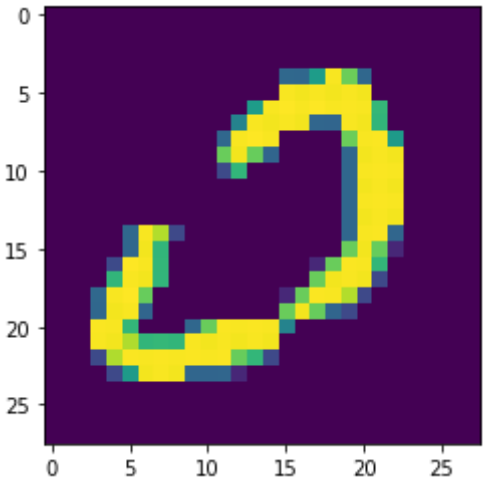

0
5
10
15
20
25
0
5
10
15
20
25

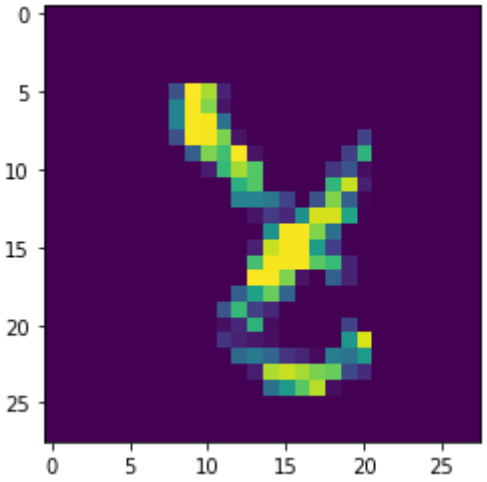

0
5
10
15
20
25
0
5
10
15
20
25

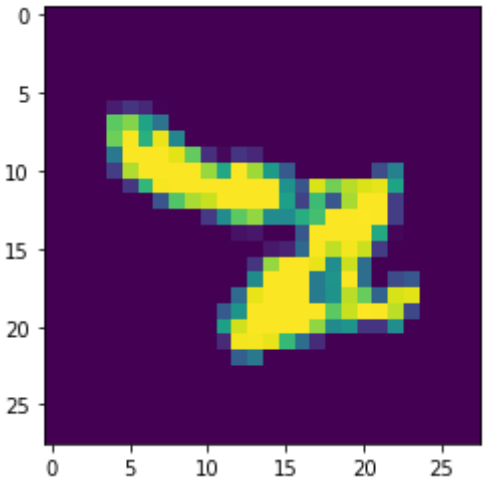

0
5
10
15
20
25
0
5
10
15
20
25

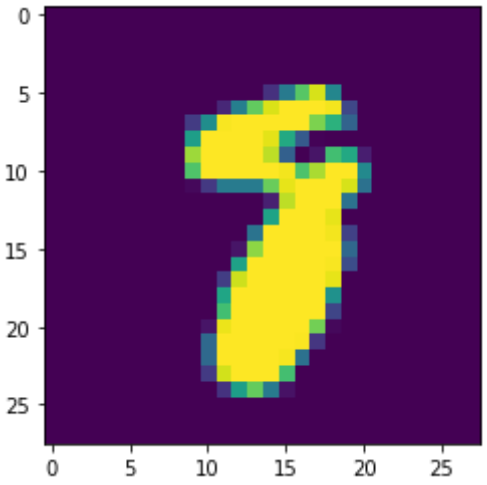

0
5
10
15
20
25
0
5
10
15
20
25

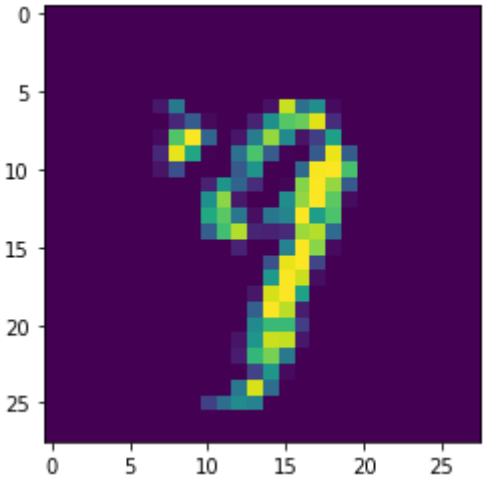

0
5
10
15
20
25
0
5
10
15
20
25

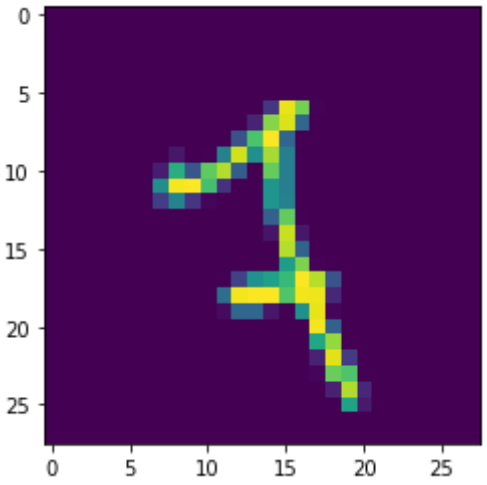

0
5
10
15
20
25
0 5 10 15 20 25

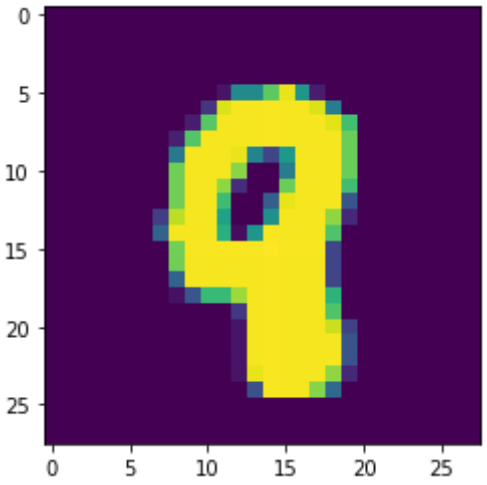

0
5
10
15
20
25
0 5 10 15 20 25

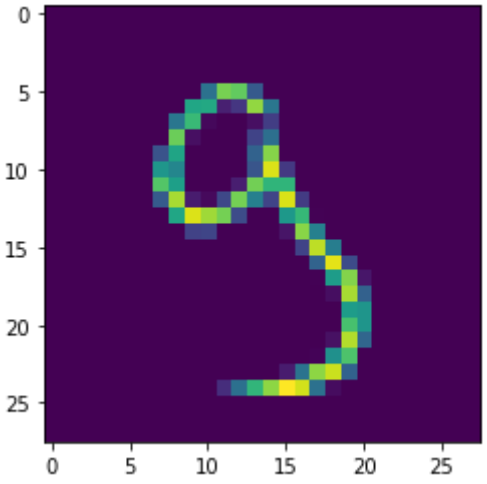

0
5
10
15
20
25
0 5 10 15 20 25

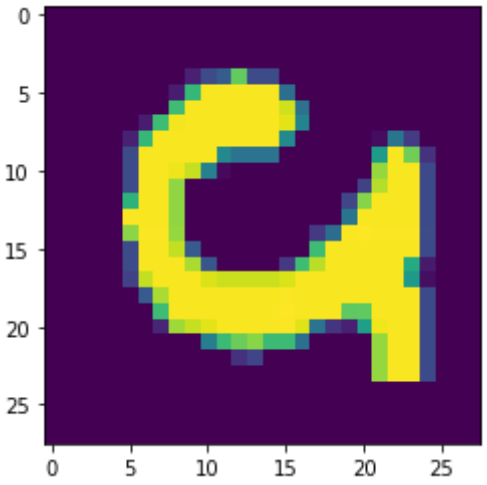

0
5
10
15
20
25
0 5 10 15 20 25

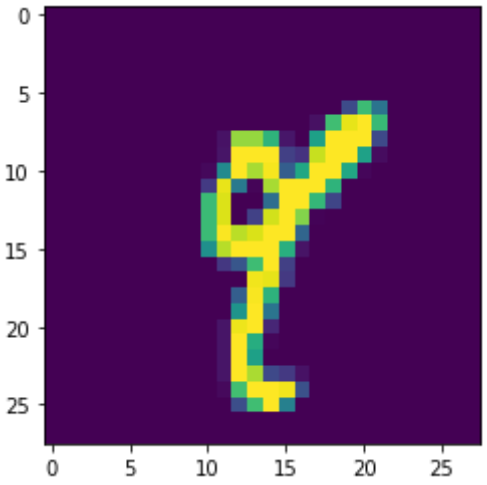

0
5
10
15
20
25
0 5 10 15 20 25

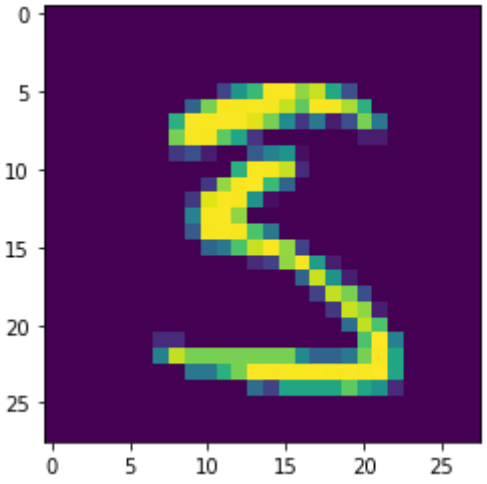

0
5
10
15
20
25
0 5 10 15 20 25

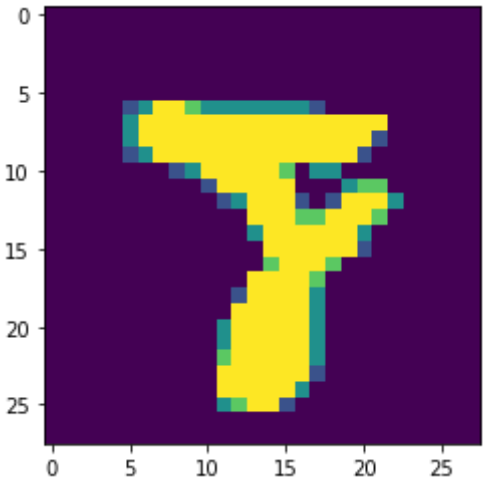

0
5
10
15
20
25
0
5
10
15
20
25

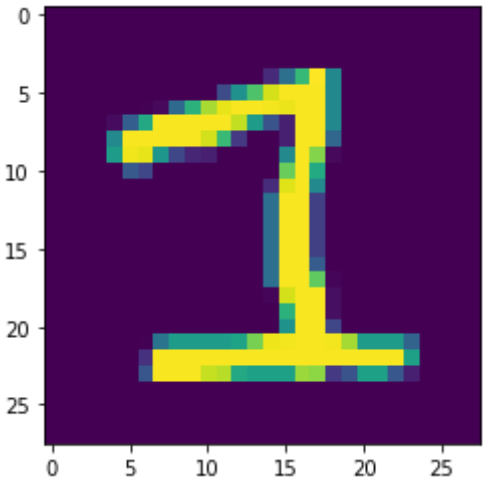

0
5
10
15
20
25
0
5
10
15
20
25

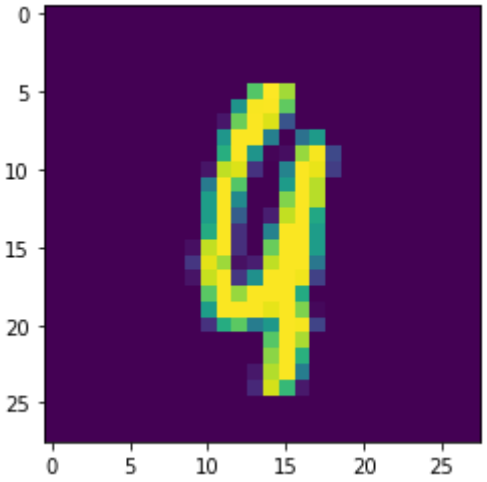

0
5
10
15
20
25
0
5
10
15
20
25

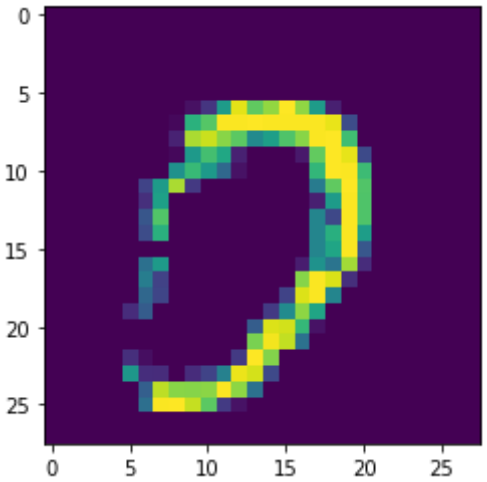

0
5
10
15
20
25
0 5 10 15 20 25

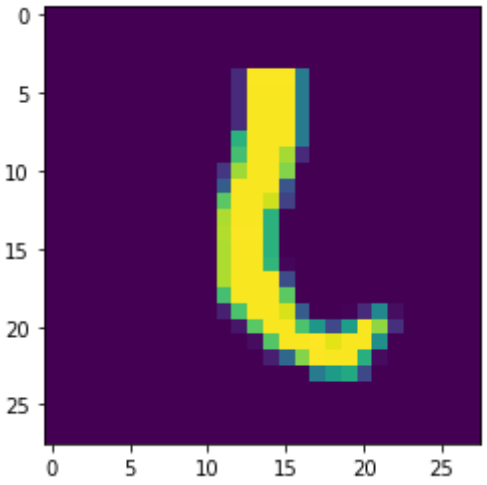

0
5
10
15
20
25
0 5 10 15 20 25

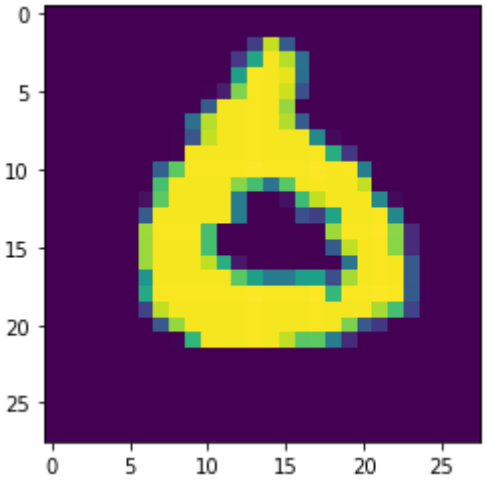

0
5
10
15
20
25
0 5 10 15 20 25

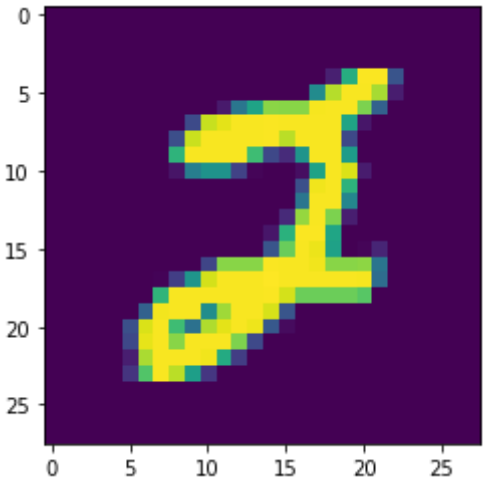

0
5
10
15
20
25
0
5
10
15
20
25

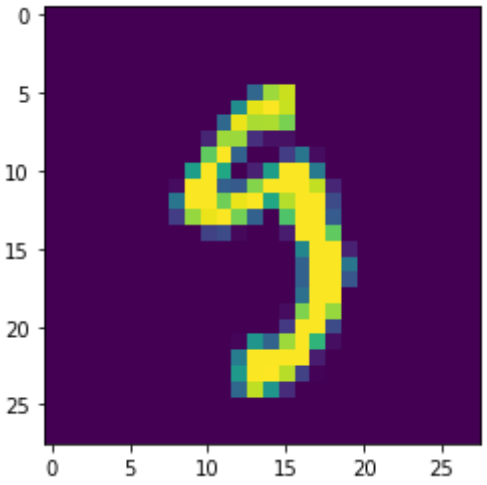

0
5
10
15
20
25
0
5
10
15
20
25

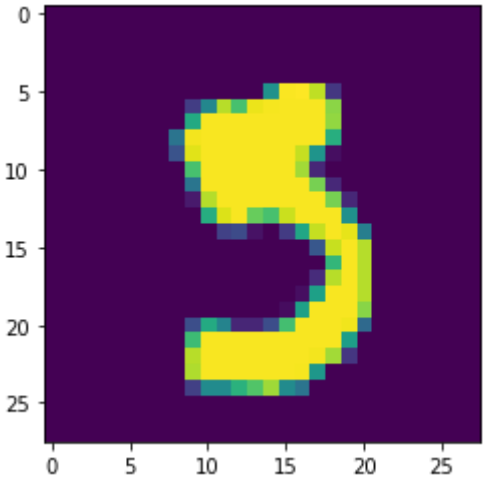

0
5
10
15
20
25
0
5
10
15
20
25

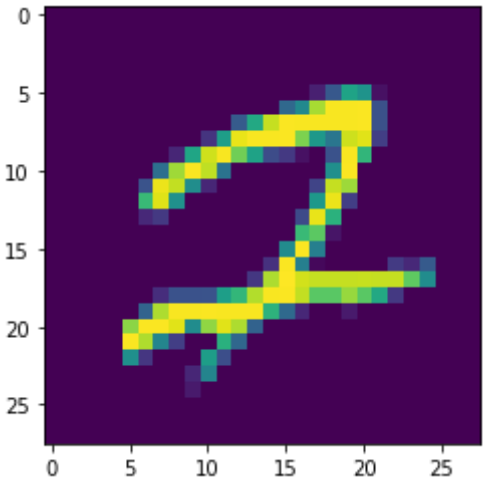

0
5
10
15
20
25
0
5
10
15
20
25

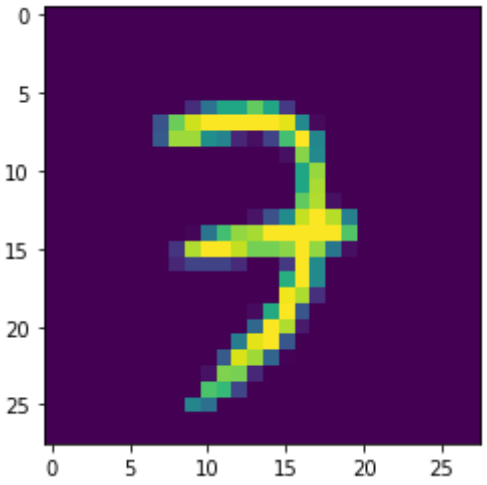

0
5
10
15
20
25
0
5
10
15
20
25

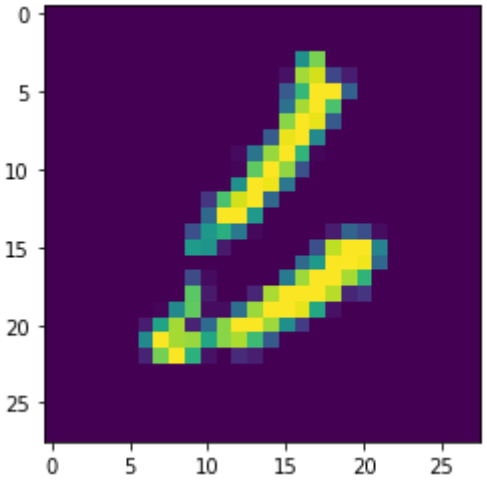

0
5
10
15
20
25
0
5
10
15
20
25

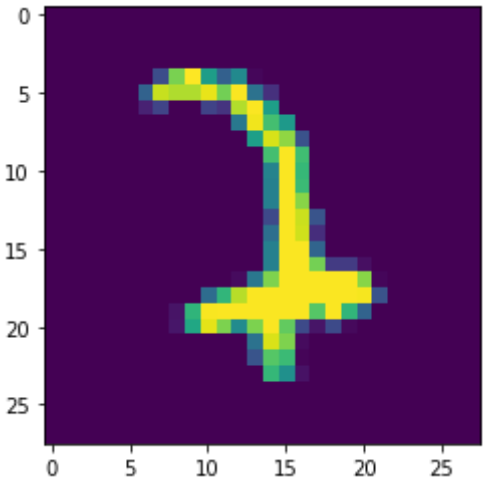

0
5
10
15
20
25
0
5
10
15
20
25

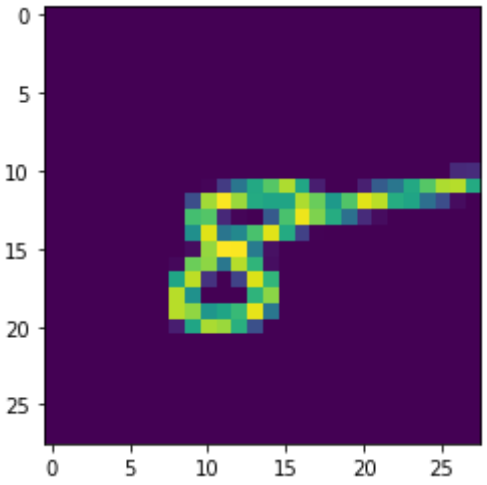

0
5
10
15
20
25
0
5
10
15
20
25

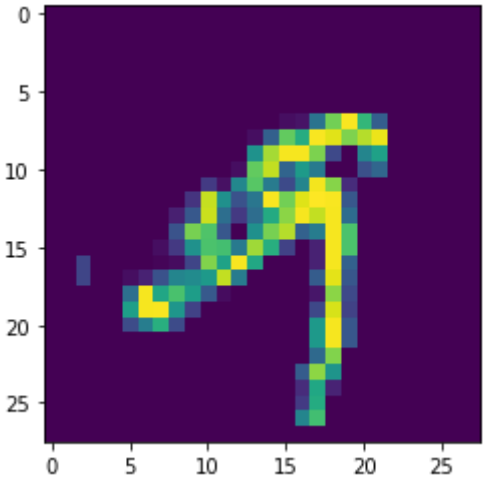

0
5
10
15
20
25
0
5
10
15
20
25

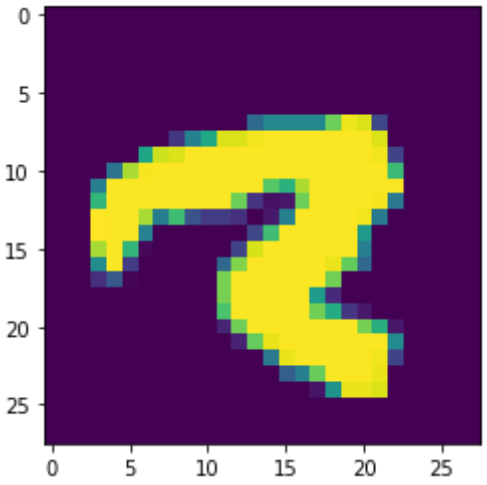

0
5
10
15
20
25
0
5
10
15
20
25

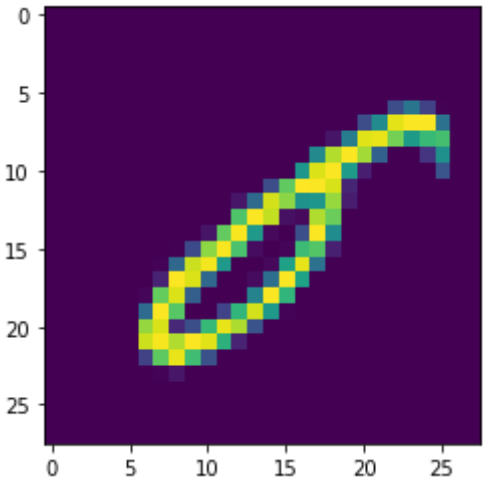

0
5
10
15
20
25
0
5
10
15
20
25

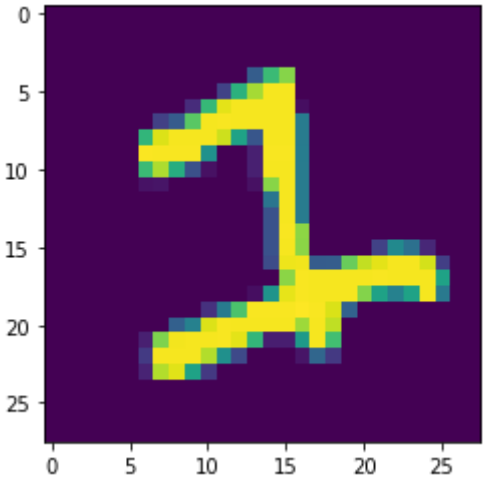

0
5
10
15
20
25
0
5
10
15
20
25

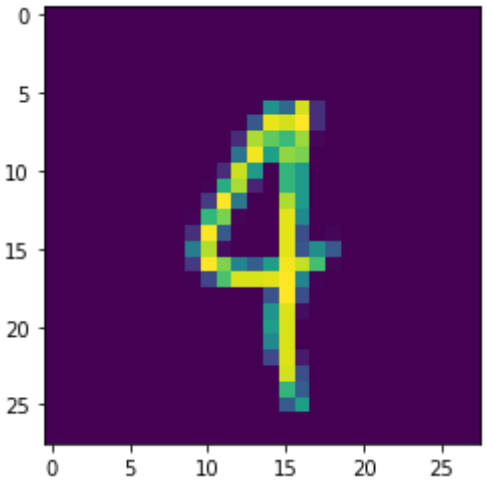

0
5
10
15
20
25
0
5
10
15
20
25

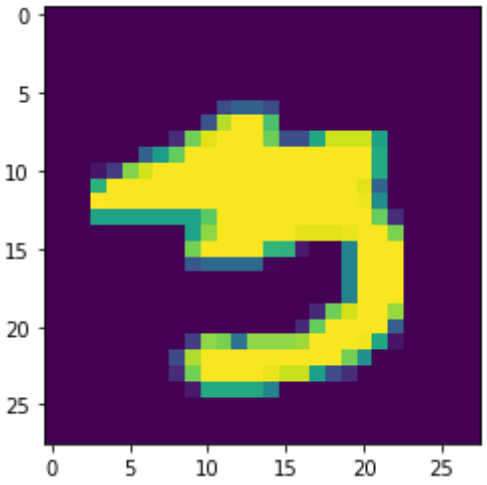

0
5
10
15
20
25
0
5
10
15
20
25

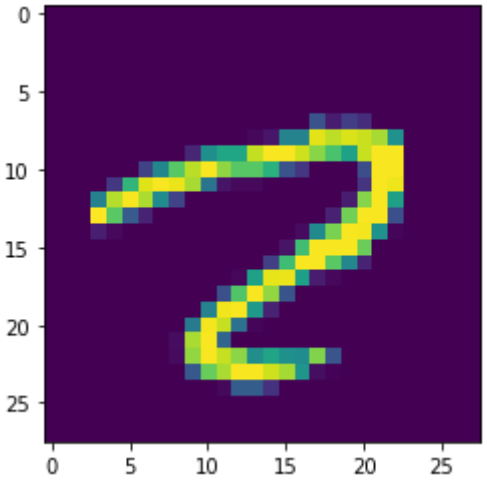

0
5
10
15
20
25
0
5
10
15
20
25

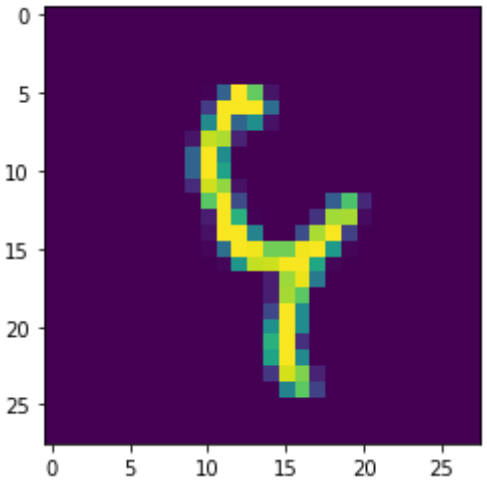

0
5
10
15
20
25
0
5
10
15
20
25

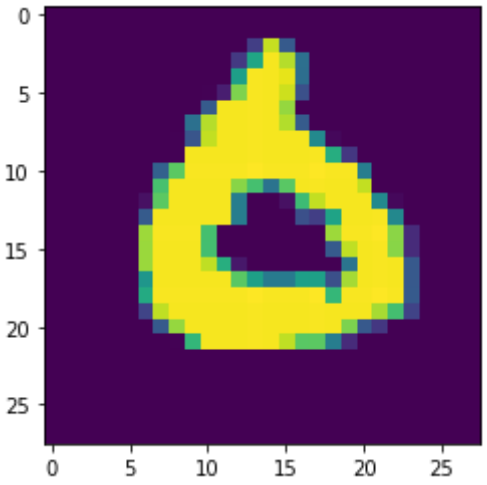

0
5
10
15
20
25
0
5
10
15
20
25

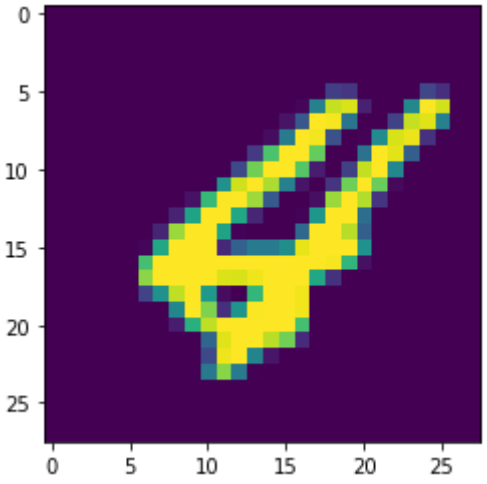

0
5
10
15
20
25
0
5
10
15
20
25

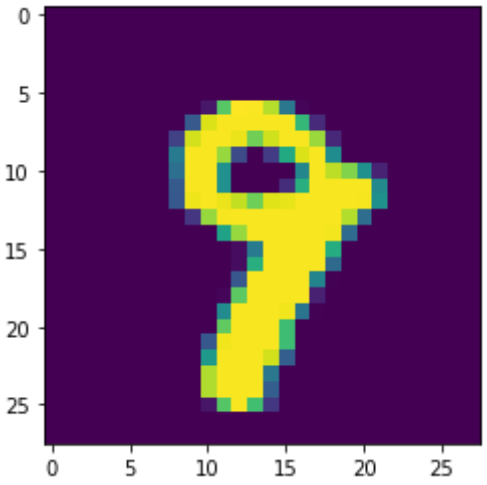

0
5
10
15
20
25
0
5
10
15
20
25

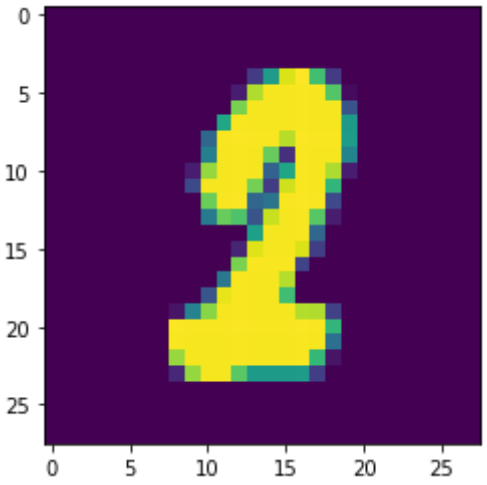

0
5
10
15
20
25
0
5
10
15
20
25

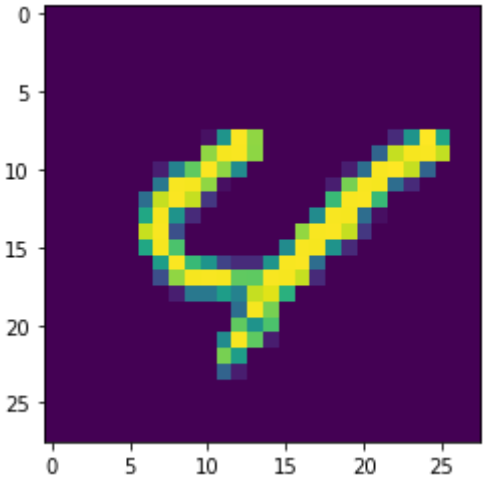

0
5
10
15
20
25
0
5
10
15
20
25

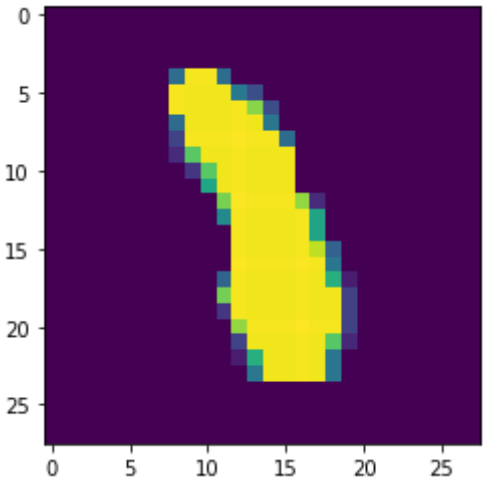

0
5
10
15
20
25
0
5
10
15
20
25

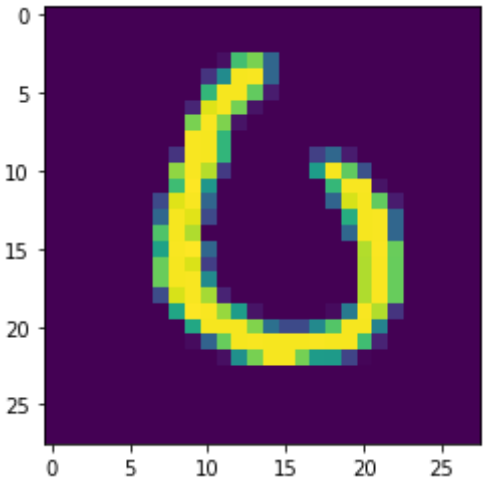

0
5
10
15
20
25
0
5
10
15
20
25

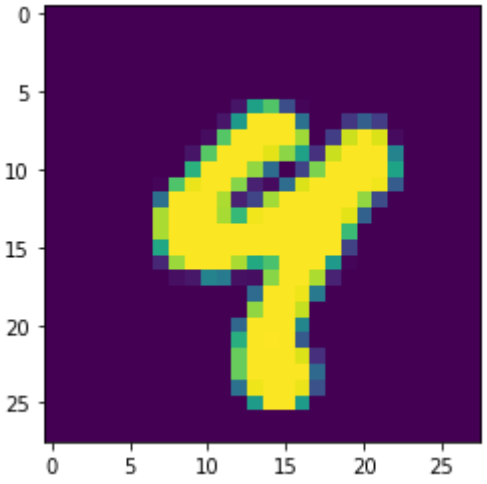

0
5
10
15
20
25
0
5
10
15
20
25

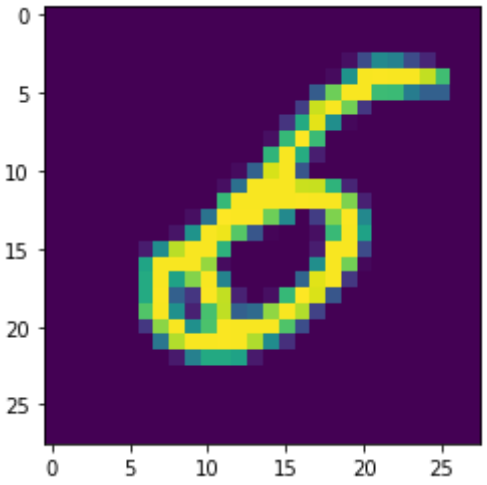

0
5
10
15
20
25
0
5
10
15
20
25

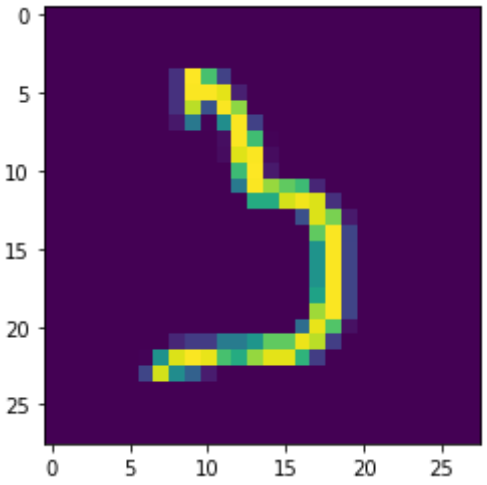

0
5
10
15
20
25
0
5
10
15
20
25

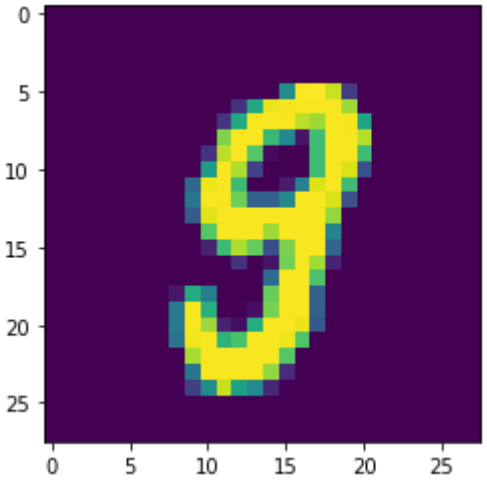

0
5
10
15
20
25
0
5
10
15
20
25

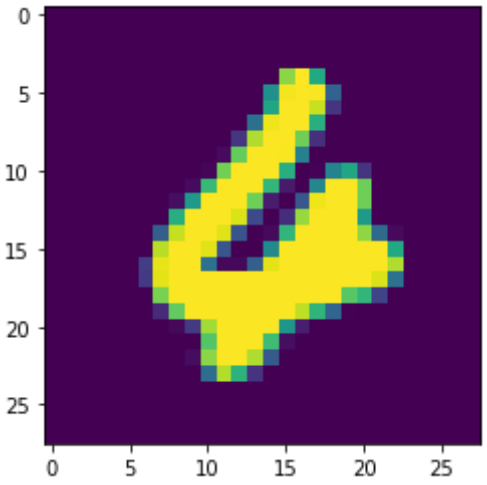

0
5
10
15
20
25
0
5
10
15
20
25

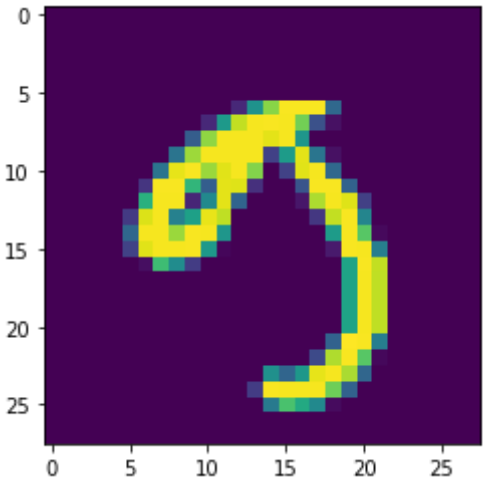

0
5
10
15
20
25
0
5
10
15
20
25

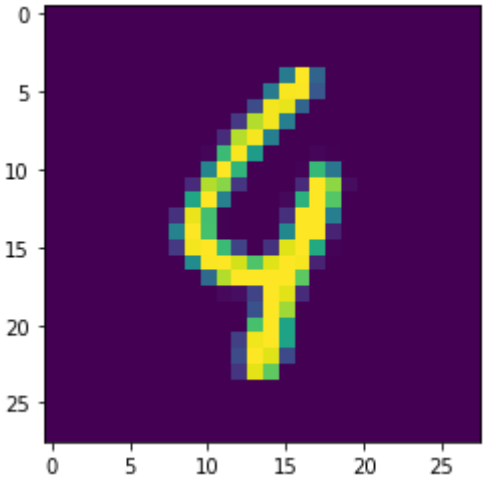

0
5
10
15
20
25
0
5
10
15
20
25

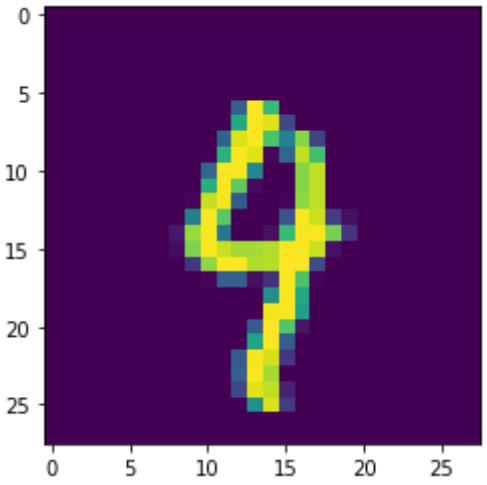

0
5
10
15
20
25
0
5
10
15
20
25

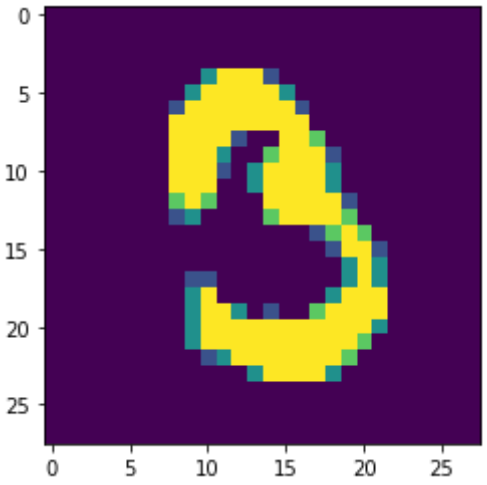

0
5
10
15
20
25
0
5
10
15
20
25

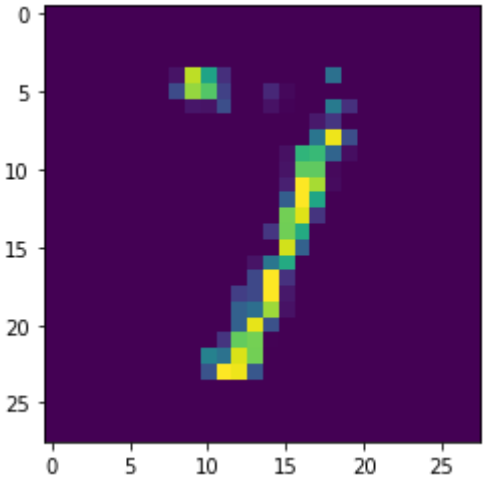

0
5
10
15
20
25
0
5
10
15
20
25

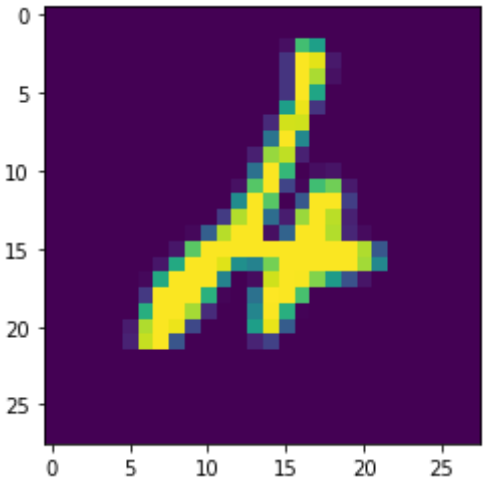

0
5
10
15
20
25
0
5
10
15
20
25

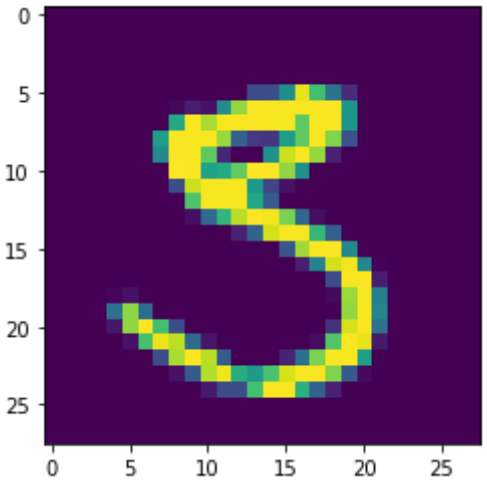

0
5
10
15
20
25
0
5
10
15
20
25

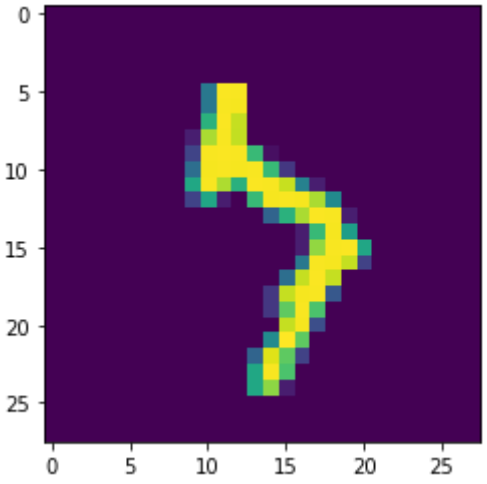

0
5
10
15
20
25
0
5
10
15
20
25

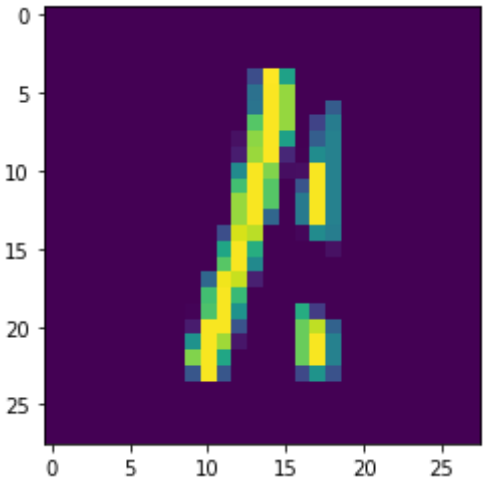

0
5
10
15
20
25
0
5
10
15
20
25

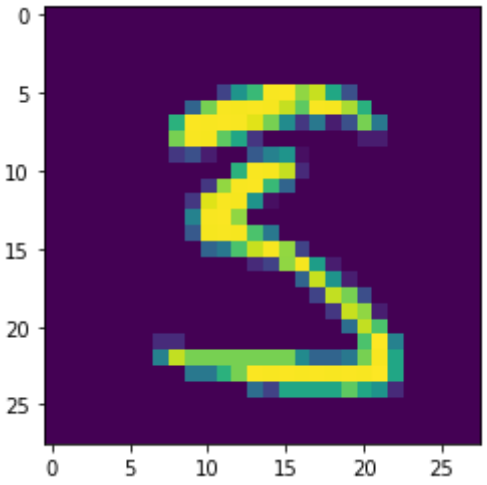

0
5
10
15
20
25
0
5
10
15
20
25

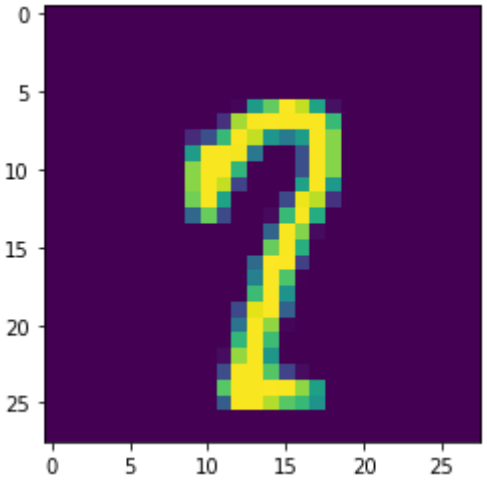

0
5
10
15
20
25
0
5
10
15
20
25

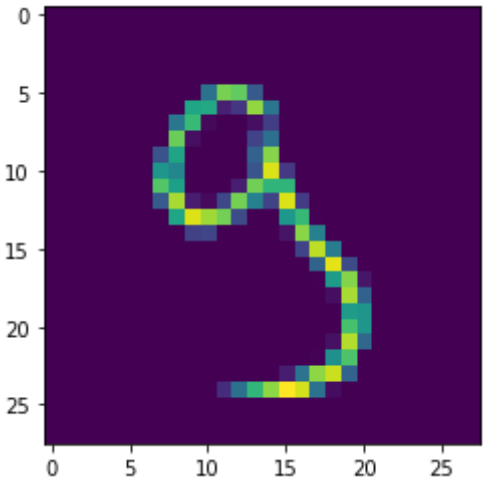

0
5
10
15
20
25
0
5
10
15
20
25

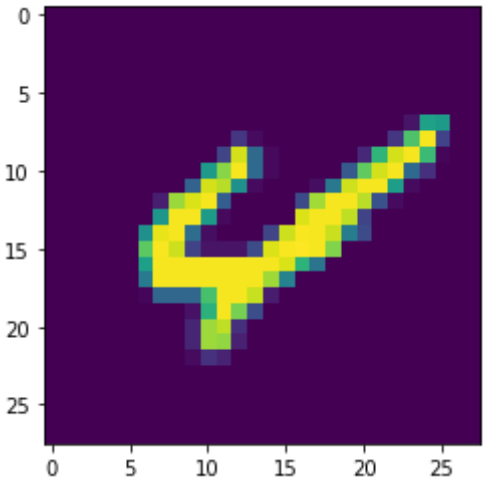

0
5
10
15
20
25
0
5
10
15
20
25

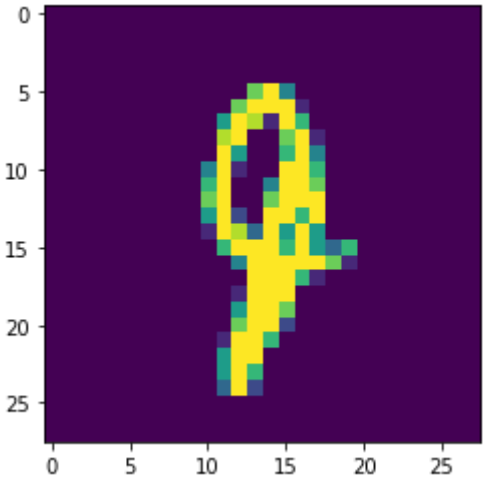

0
5
10
15
20
25
0
5
10
15
20
25

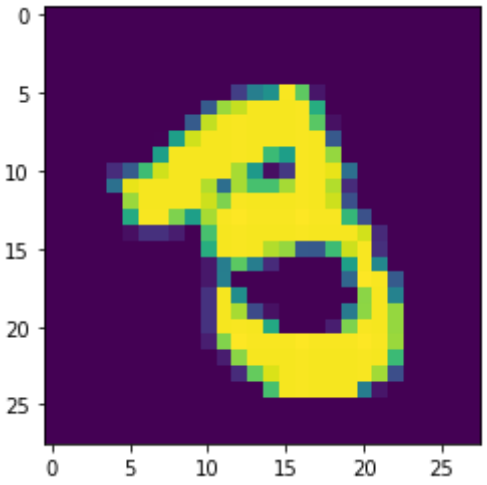

0
5
10
15
20
25
0 5 10 15 20 25

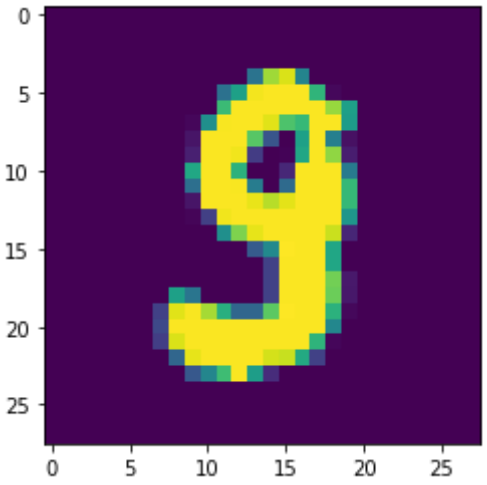

0
5
10
15
20
25
0 5 10 15 20 25

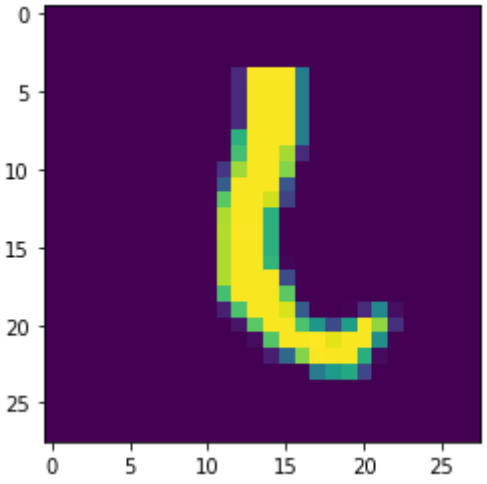

0
5
10
15
20
25
0 5 10 15 20 25

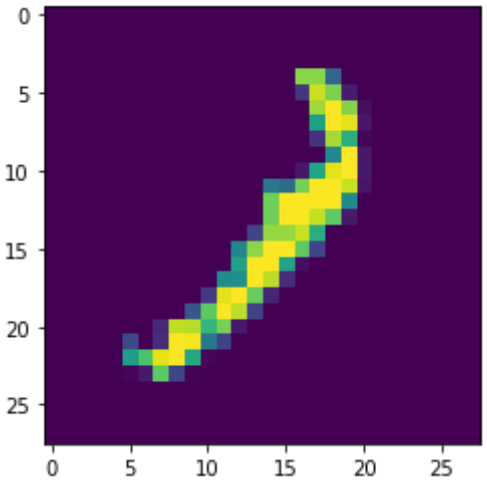

0
5
10
15
20
25
0
5
10
15
20
25

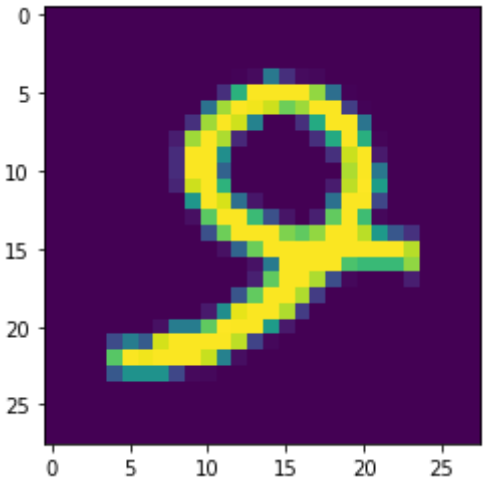

0
5
10
15
20
25
0
5
10
15
20
25

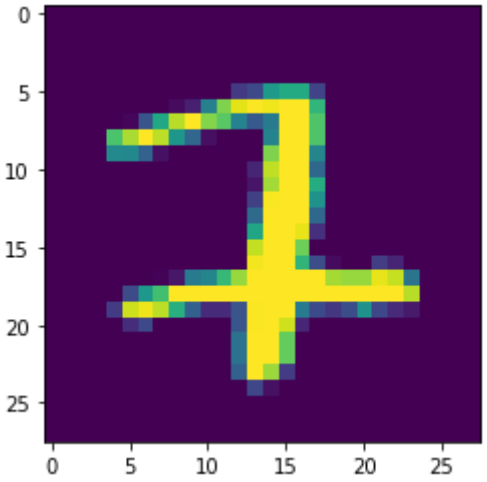

0
5
10
15
20
25
0
5
10
15
20
25

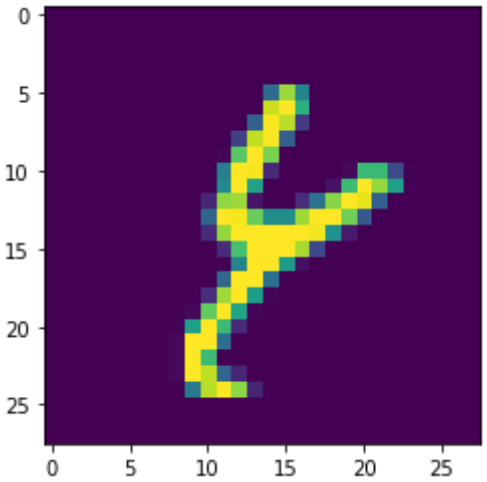

0
5
10
15
20
25
0
5
10
15
20
25

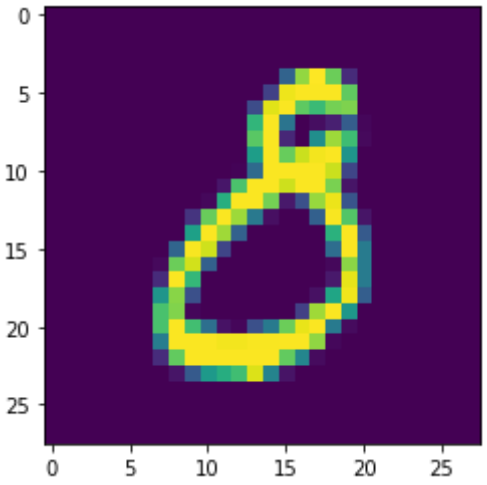

0
5
10
15
20
25
0
5
10
15
20
25

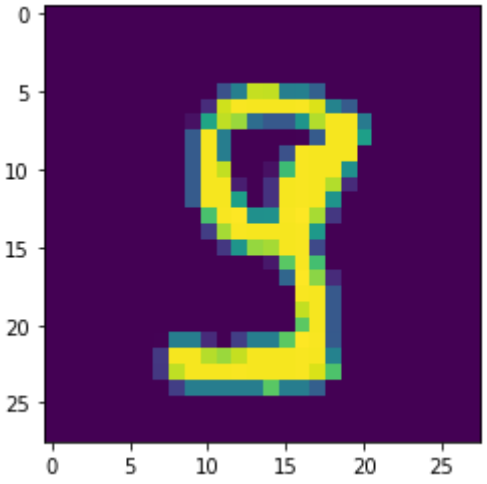

0
5
10
15
20
25
0
5
10
15
20
25

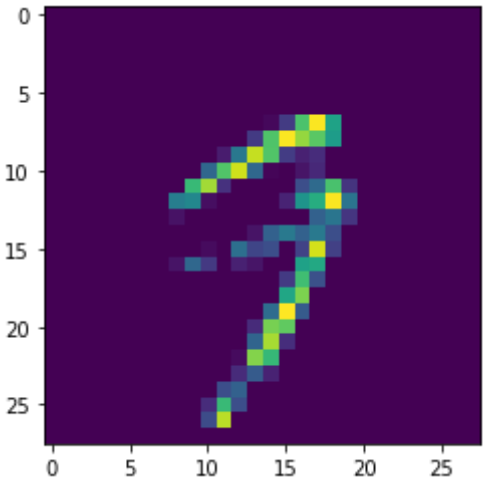

0
5
10
15
20
25
0
5
10
15
20
25

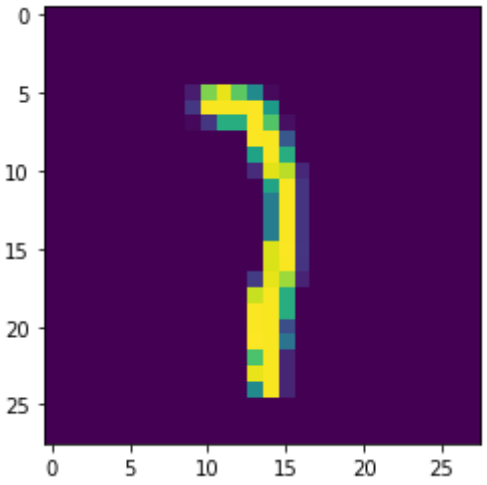

0
5
10
15
20
25
0
5
10
15
20
25

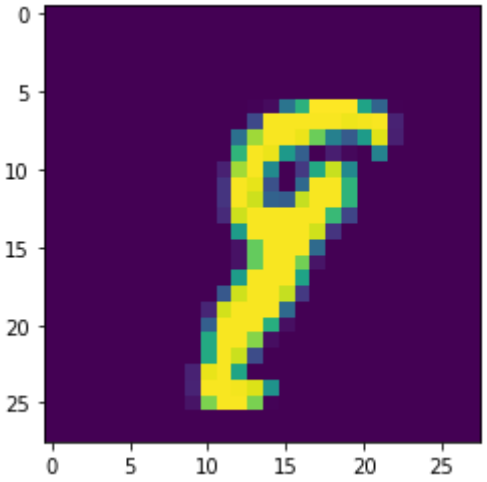

0
5
10
15
20
25
0
5
10
15
20
25

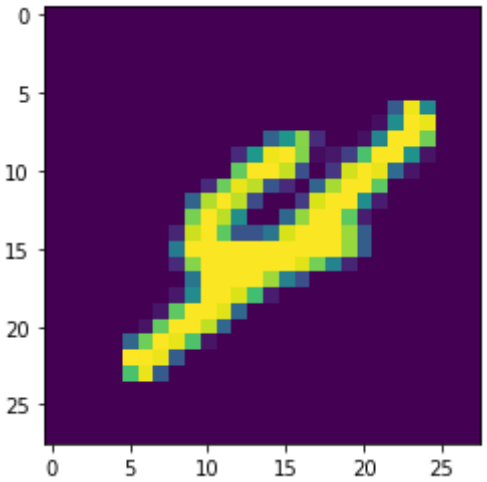

0
5
10
15
20
25
0
5
10
15
20
25

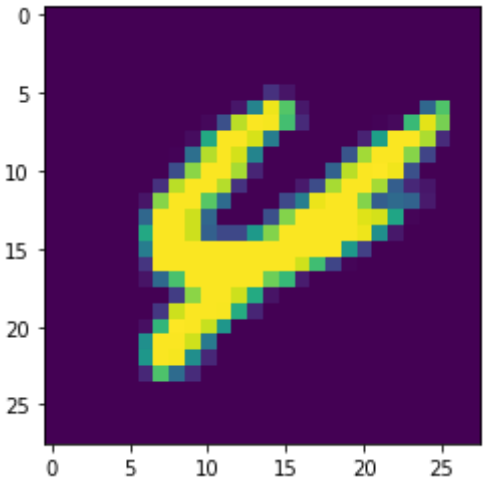

0
5
10
15
20
25
0
5
10
15
20
25

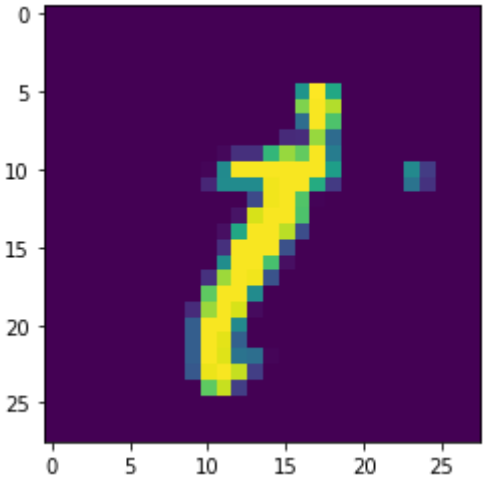

0
5
10
15
20
25
0
5
10
15
20
25

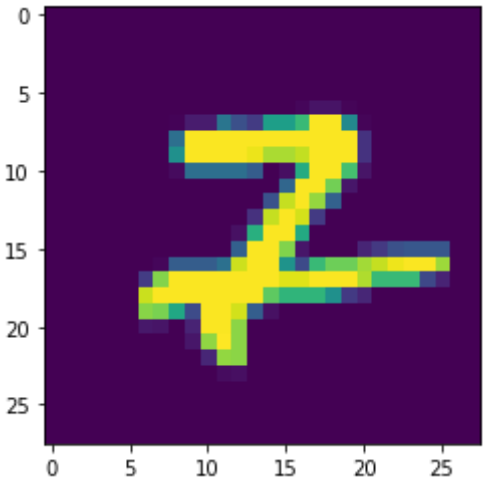

0
5
10
15
20
25
0
5
10
15
20
25

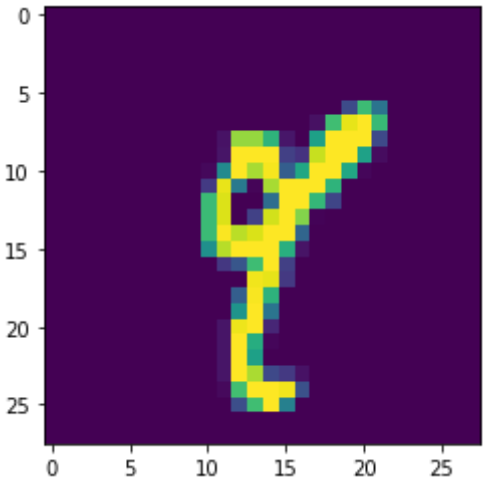

0
5
10
15
20
25
0
5
10
15
20
25

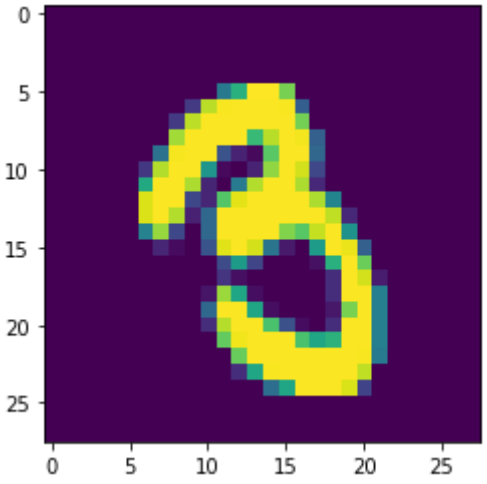

0
5
10
15
20
25
0
5
10
15
20
25

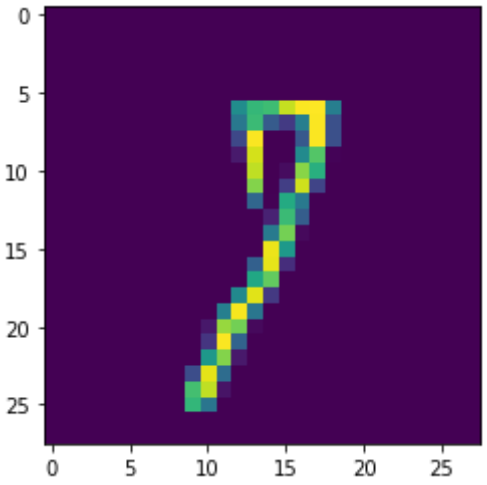

0
5
10
15
20
25
0
5
10
15
20
25

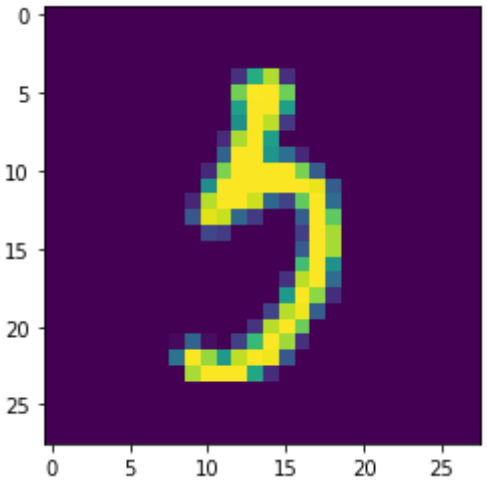

0
5
10
15
20
25
0
5
10
15
20
25

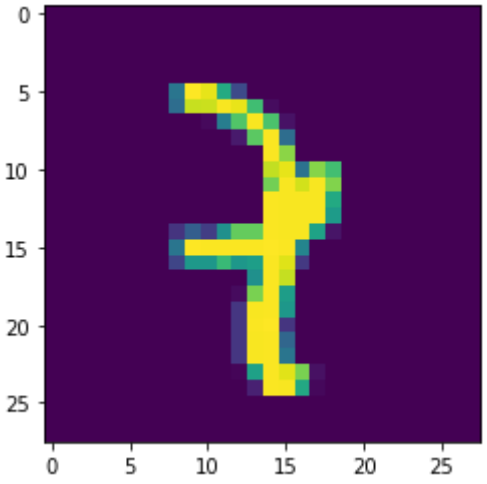

0
5
10
15
20
25
0
5
10
15
20
25

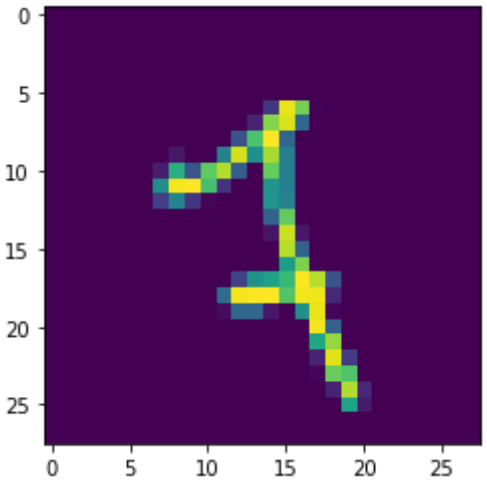

0
5
10
15
20
25
0
5
10
15
20
25

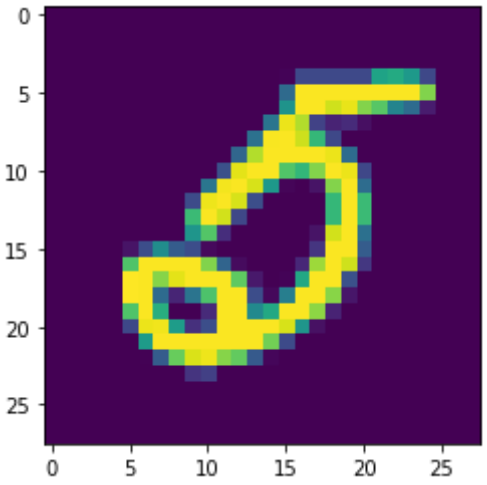

0
5
10
15
20
25
0
5
10
15
20
25

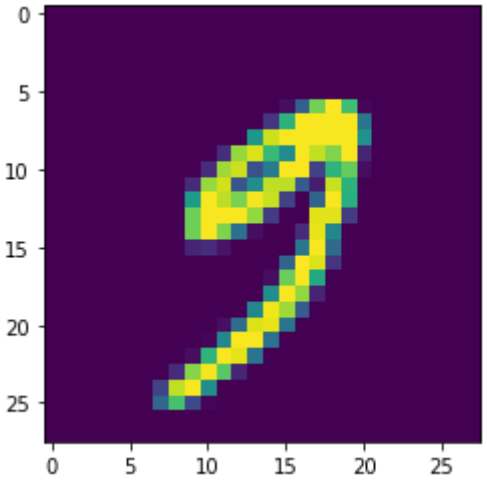

0
5
10
15
20
25
0
5
10
15
20
25

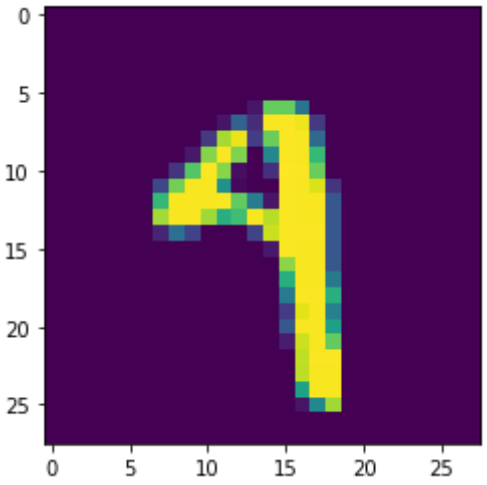

0
5
10
15
20
25
0
5
10
15
20
25

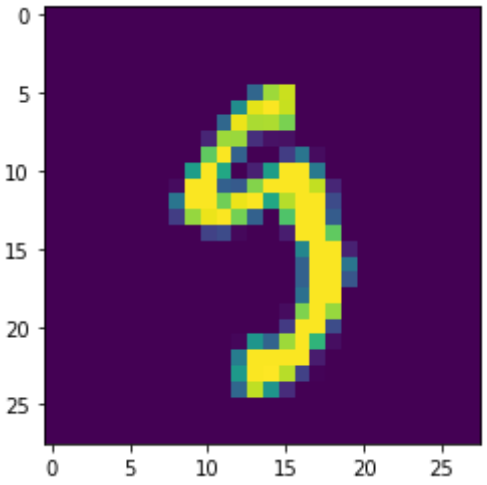

0
5
10
15
20
25
0
5
10
15
20
25

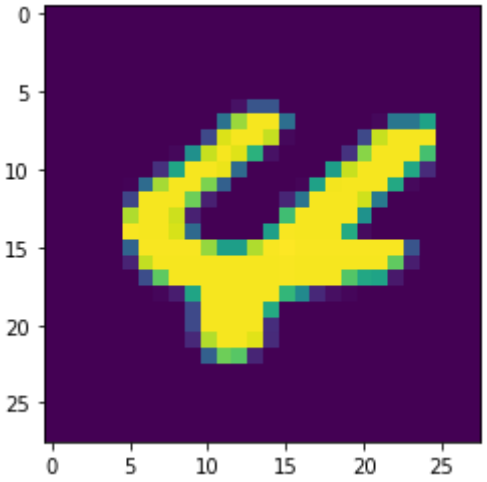

0
5
10
15
20
25
0
5
10
15
20
25

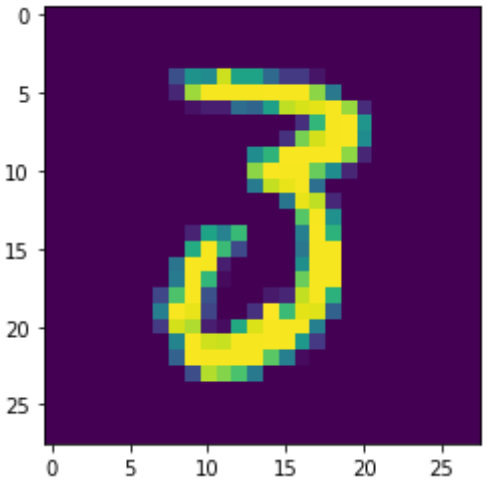

0
5
10
15
20
25
0
5
10
15
20
25

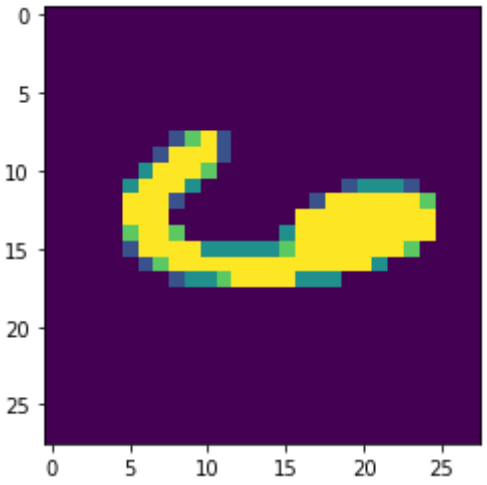

0
5
10
15
20
25
0
5
10
15
20
25

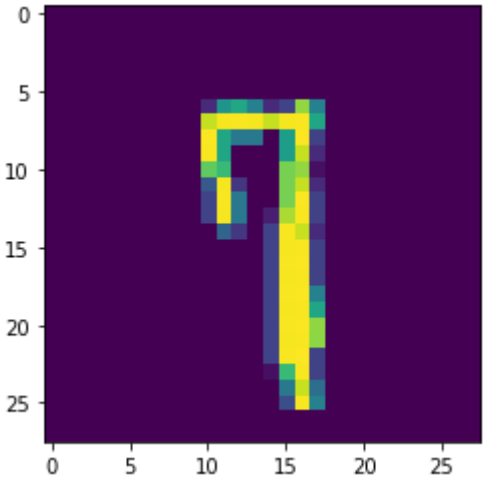

0
5
10
15
20
25
0
5
10
15
20
25

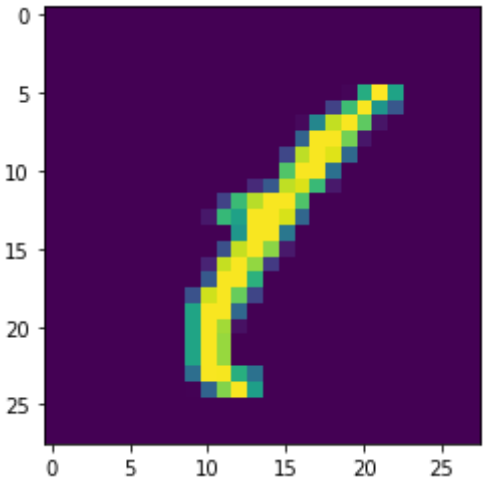

0
5
10
15
20
25
0
5
10
15
20
25

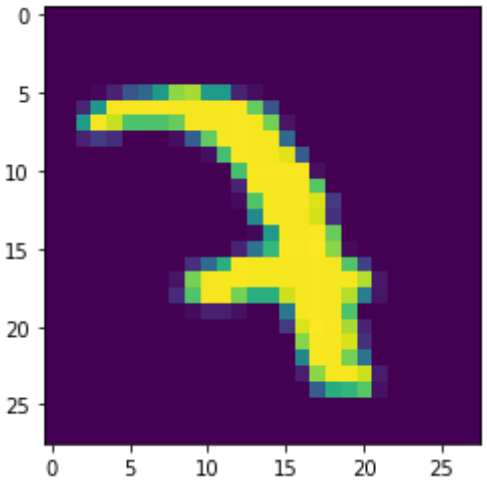

0
5
10
15
20
25
0
5
10
15
20
25

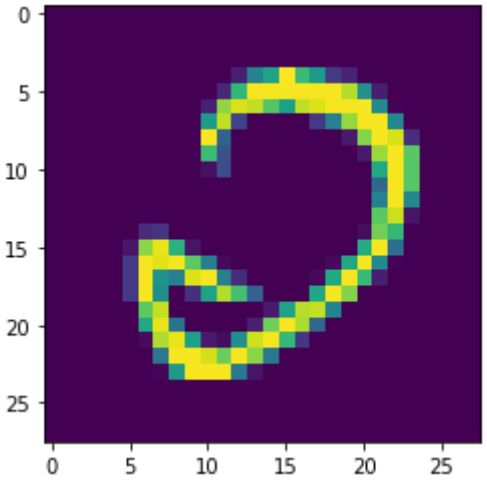

0
5
10
15
20
25
0
5
10
15
20
25

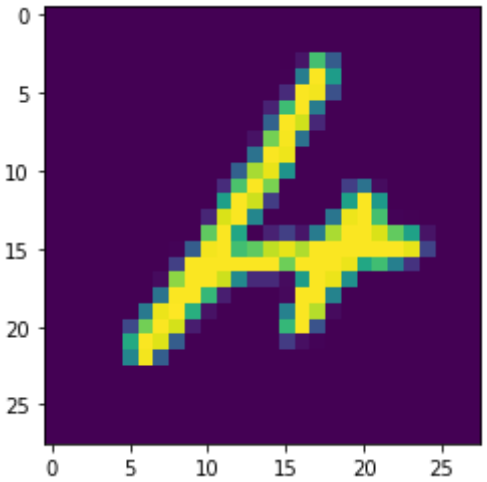

0
5
10
15
20
25
0
5
10
15
20
25

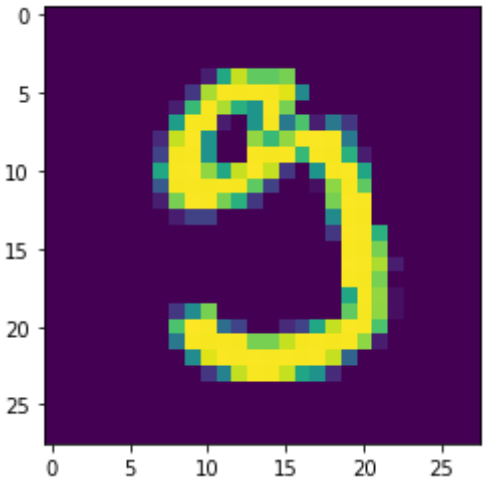

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



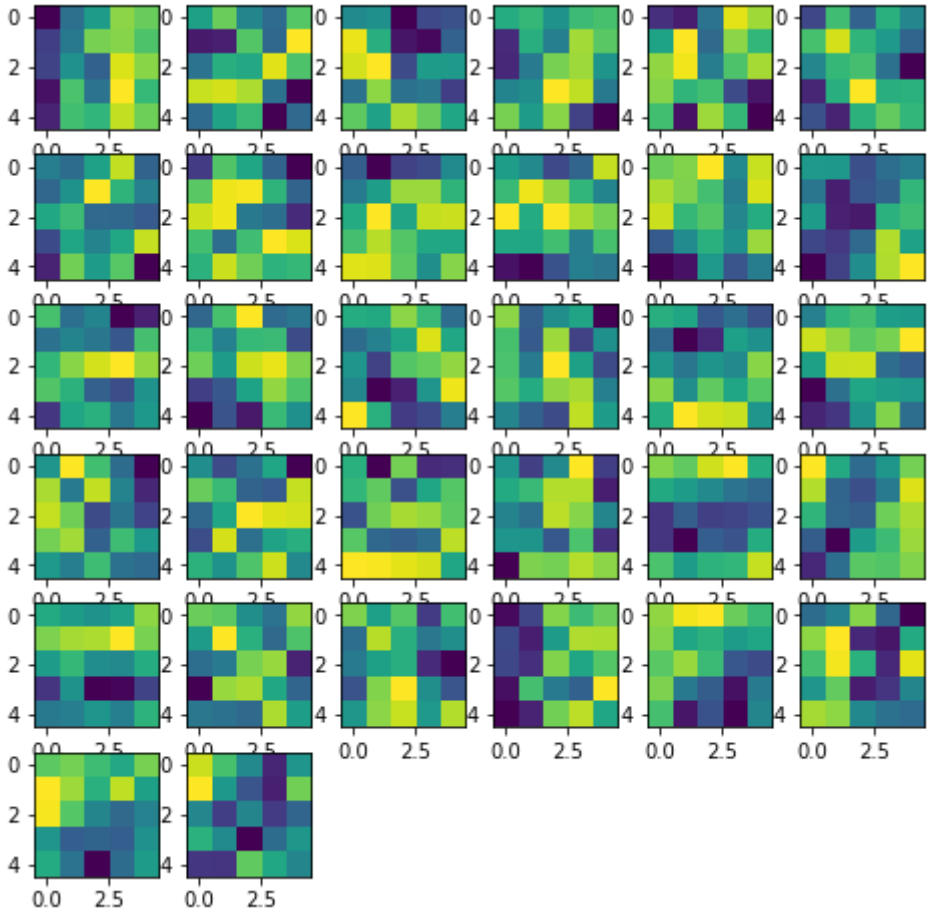

**Don't forget plotting the digits that the network got wrong.**

## Question 3.2 Kernel weights visualization [5pt]

For this question, you need to visualize the kernel weights for your first convolutional layer. Suppose you have 5x5 kernels with 32 output channels. You will plot 32 5x5 images.

hint: You might need to look at PyTorch documentation (or play with the PyTorch model) to figure out how to get the weights.

In [367]:
```
### YOUR CODE HERE
weights = model.first.weight
weights.shape

fig=plt.figure(figsize=(8, 8))

count = 1
for i in range(weights.shape[0]):
    kernel = weights[i]
    kernel = np.squeeze(kernel)
    fig.add_subplot(6, 6, count)
    plt.imshow(kernel.detach().numpy())

    count+= 1
### END OF CODE
```



In [ ]: