```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import losses

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import random
tf.keras.datasets.mnist.load_data(
    path='mnist.npz'
)
(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
xtrain=xtrain.reshape(-1,784)
print(xtrain.shape)


plt.figure()
plt.subplot(1,2,1)
plt.imshow(xtrain[100].reshape(28,28))
plt.subplot(1,2,2)
plt.imshow(xtrain[200].reshape(28,28))
plt.show()
#task1

pca=PCA(5)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.subplot(1,2,1)
plt.imshow(xpca[100][:4].reshape(2,2))
```

```
plt.subplot(1,2,2)
plt.imshow(xpca[200][:4].reshape(2,2))

#task3
xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))
pca=PCA(10)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:9].reshape(3,3))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(25)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:25].reshape(5,5))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))
```

```
pca=PCA(64)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:64].reshape(8,8))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(100)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:100].reshape(10,10))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:100].reshape(10,10))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(200)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)
```

```
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:196].reshape(14,14))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:196].reshape(14,14))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))


pca=PCA(784)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:784].reshape(28,28))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

class Autoencoder(Model):
  def __init__(self, P, Q, R, M):
    super(Autoencoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(P, activation='tanh'),
      layers.Dense(Q, activation='tanh'),
      layers.Dense(R, activation='tanh'),
      layers.Dense(M, activation='tanh'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(P, activation='swish'),
```

```
      layers.Dense(Q, activation='swish'),
      layers.Dense(R, activation='swish'),
      layers.Dense(M, activation='swish'),
      layers.Dense(784, activation='relu'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
xtrain = xtrain /255
xtest = xtest / 255

autoencoder = Autoencoder(P=128, Q=128, R=128, M=4)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:4].reshape(2,2))

plt.subplot(1,2,2)
plt.imshow(xen[200][:4].reshape(2,2))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=9)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
```

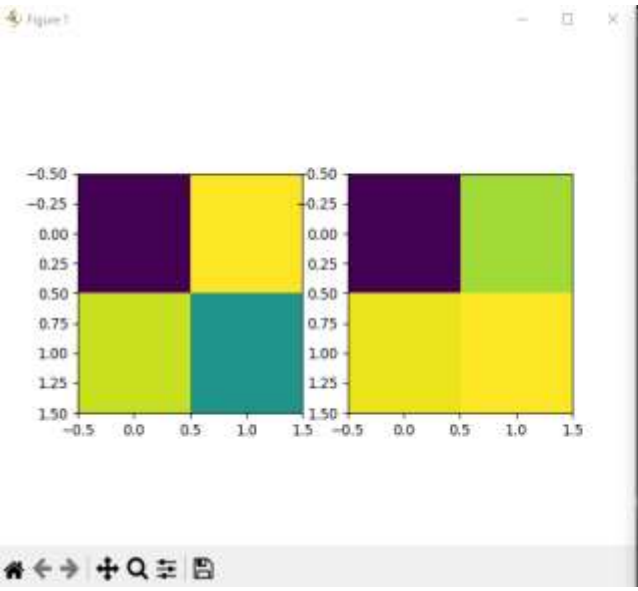
```
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xen[200][:9].reshape(3,3))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)



autoencoder = Autoencoder(P=128, Q=128, R=128, M=25)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xen[200][:25].reshape(5,5))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
```

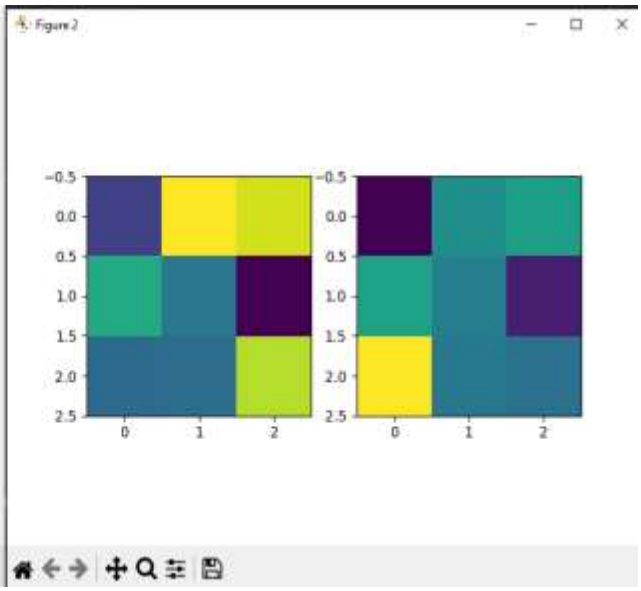
```
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=64)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xen[200][:64].reshape(8,8))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)



autoencoder = Autoencoder(P=128, Q=128, R=128, M=100)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:100].reshape(10,10))
```

```
plt.subplot(1,2,2)
plt.imshow(xen[200][:100].reshape(10,10))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)

autoencoder = Autoencoder(P=128, Q=128, R=128, M=784)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xen[200][:784].reshape(28,28))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)
```

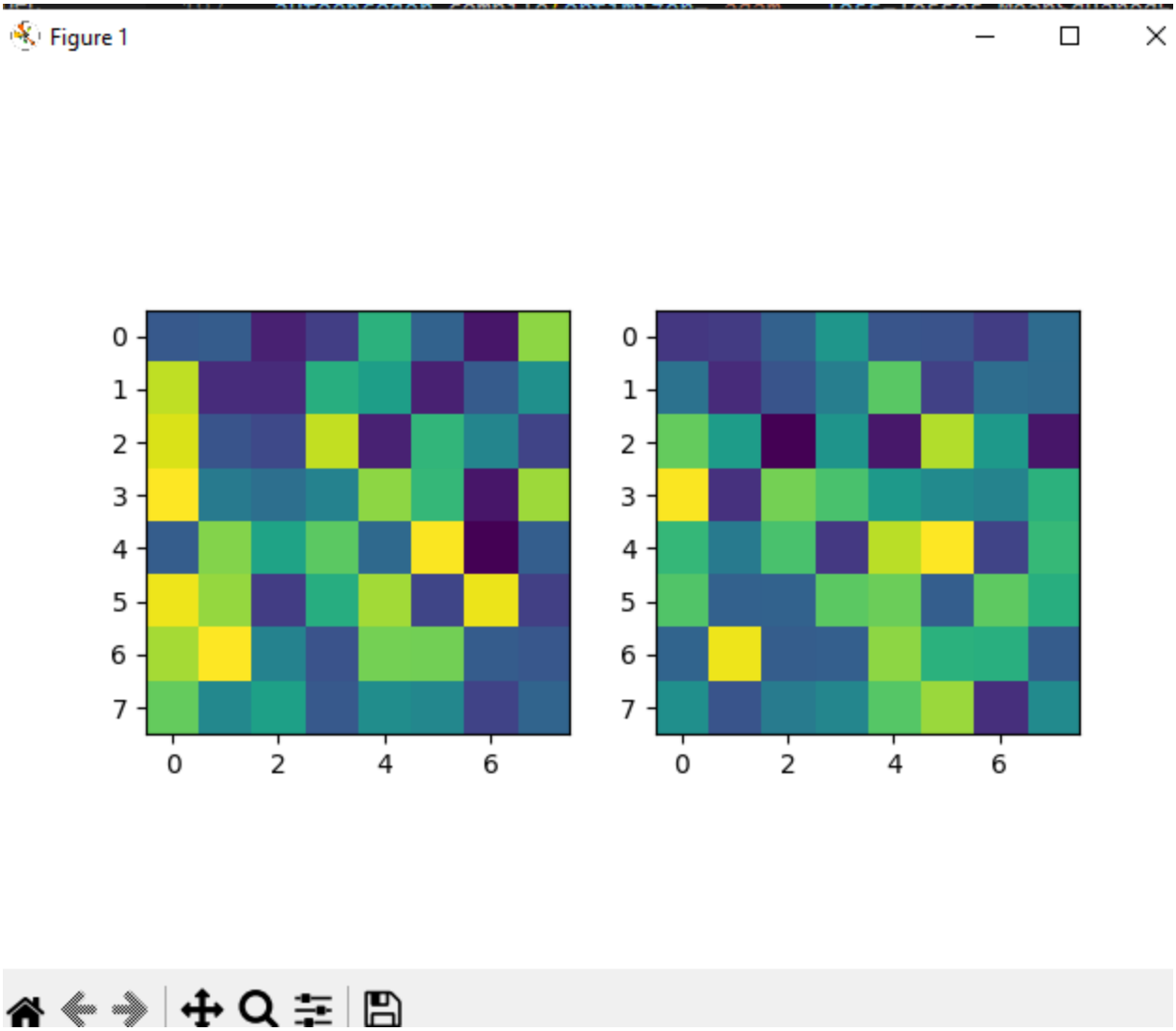
Output:-

NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017

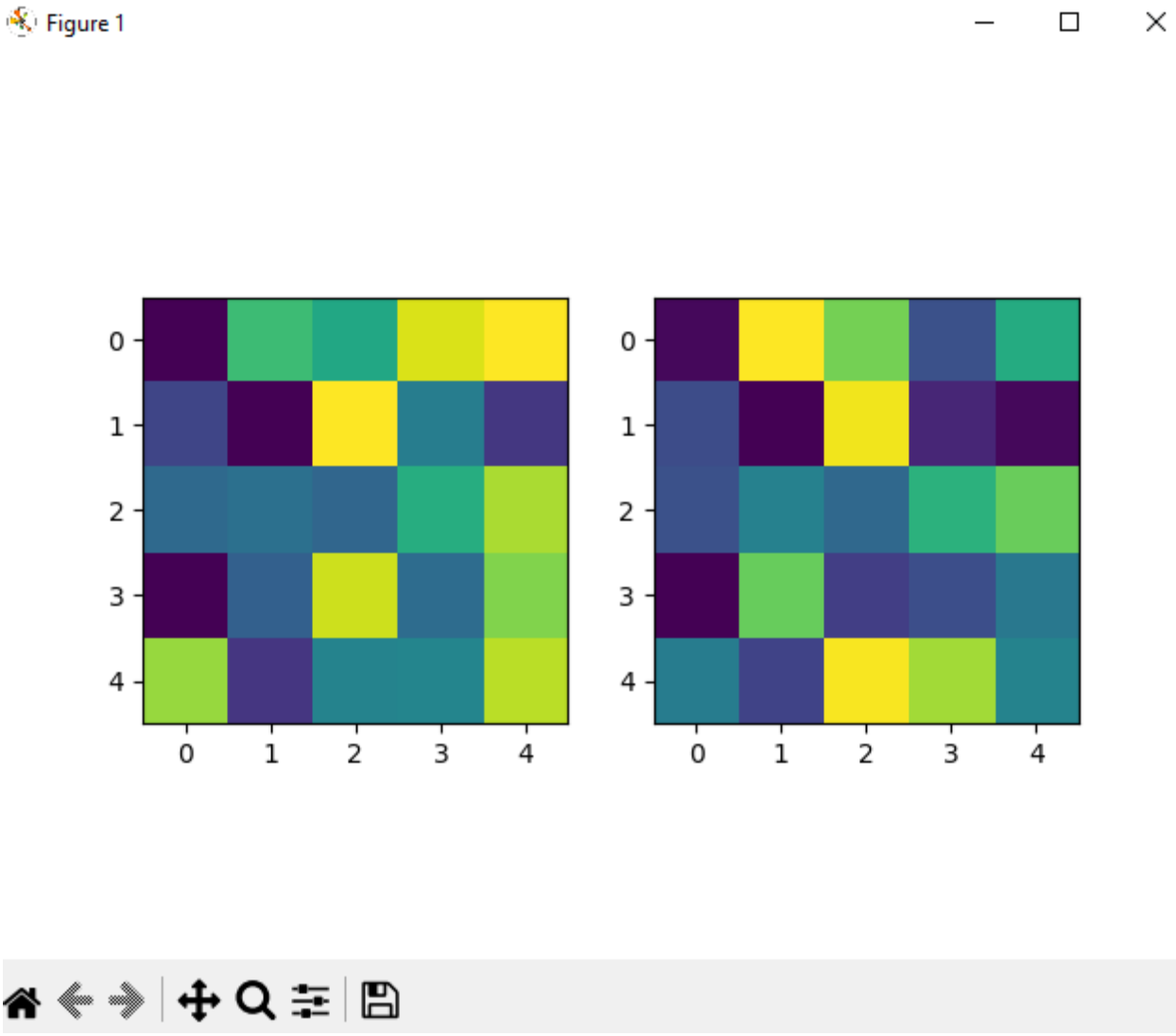




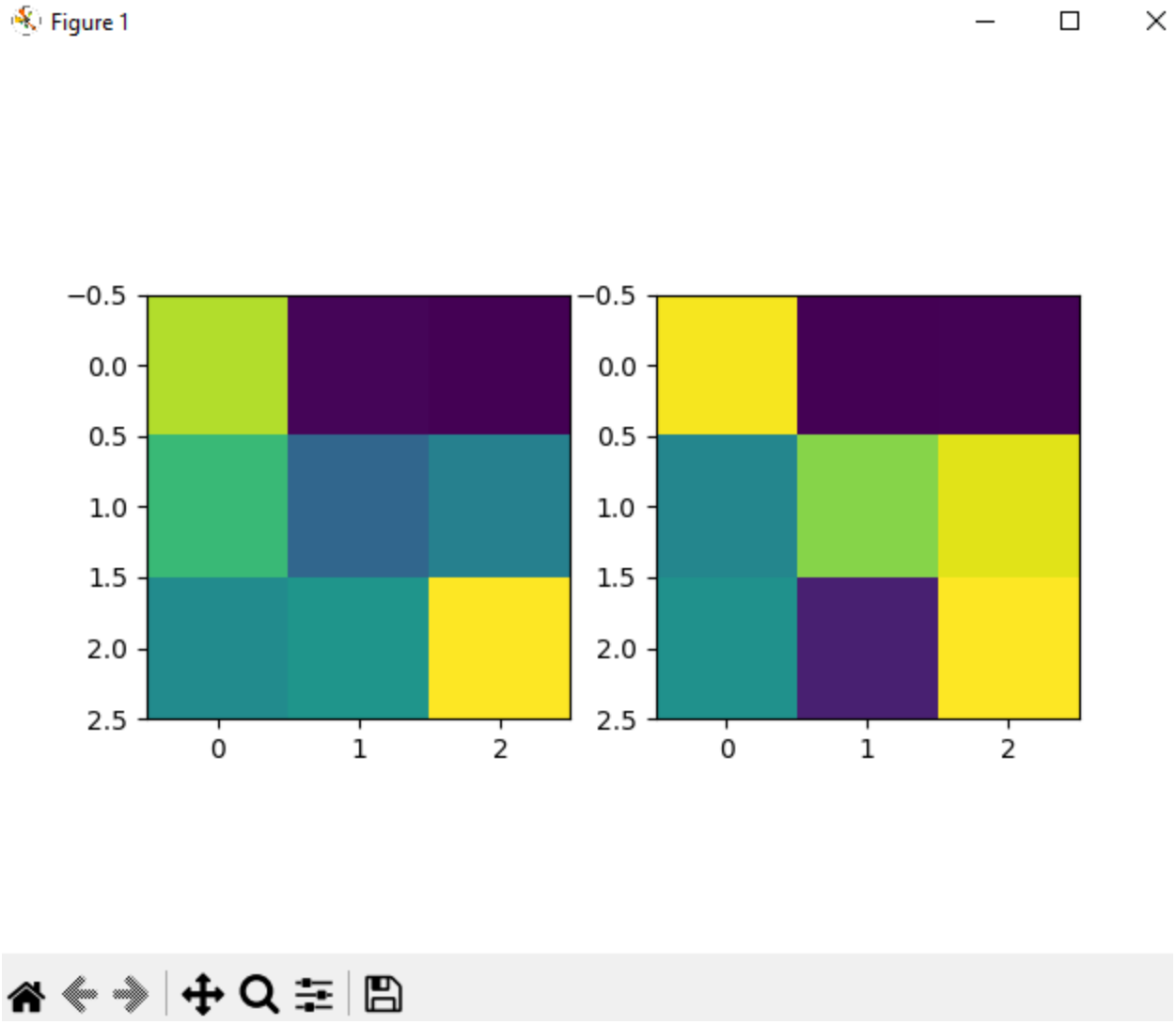
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



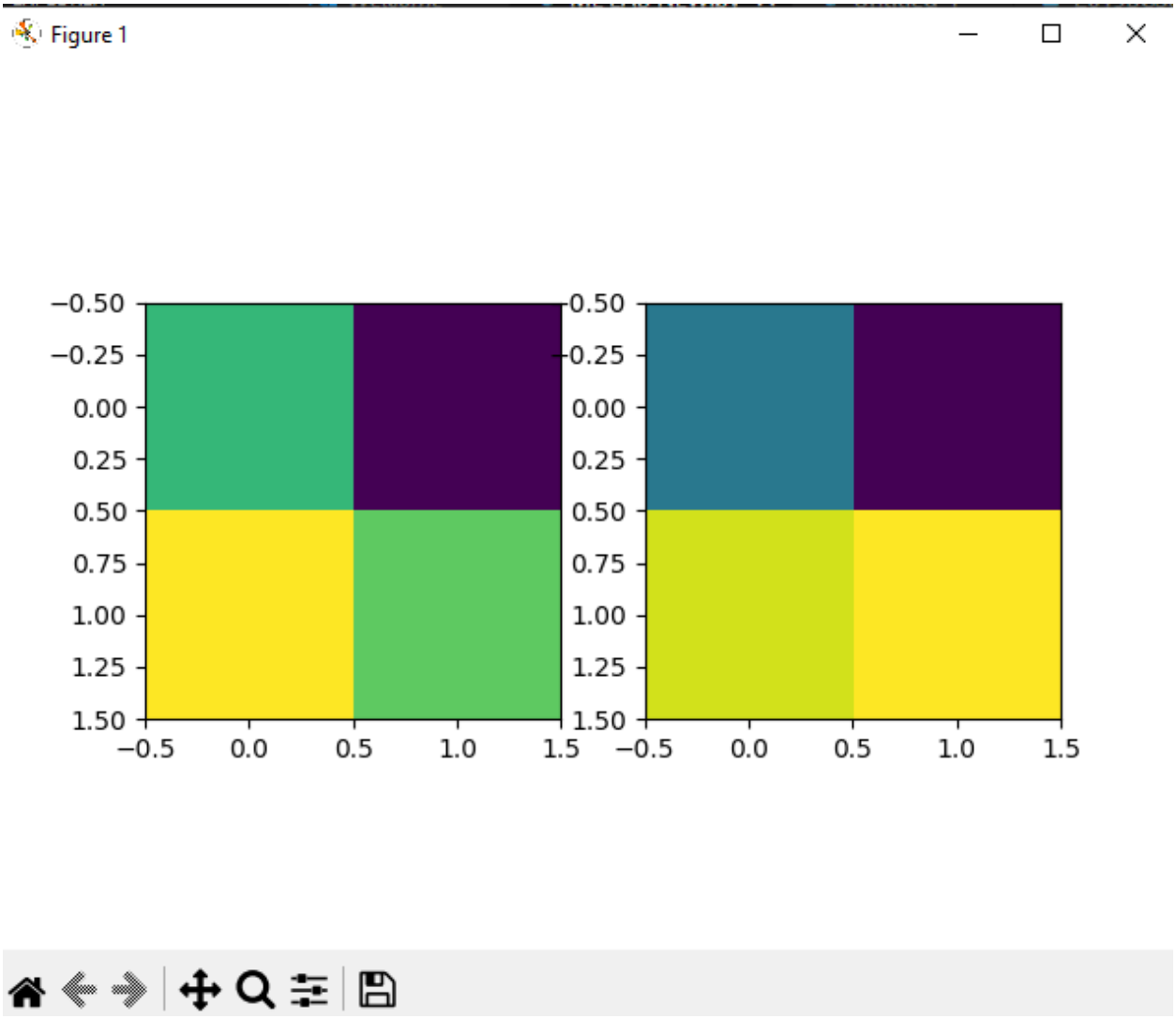
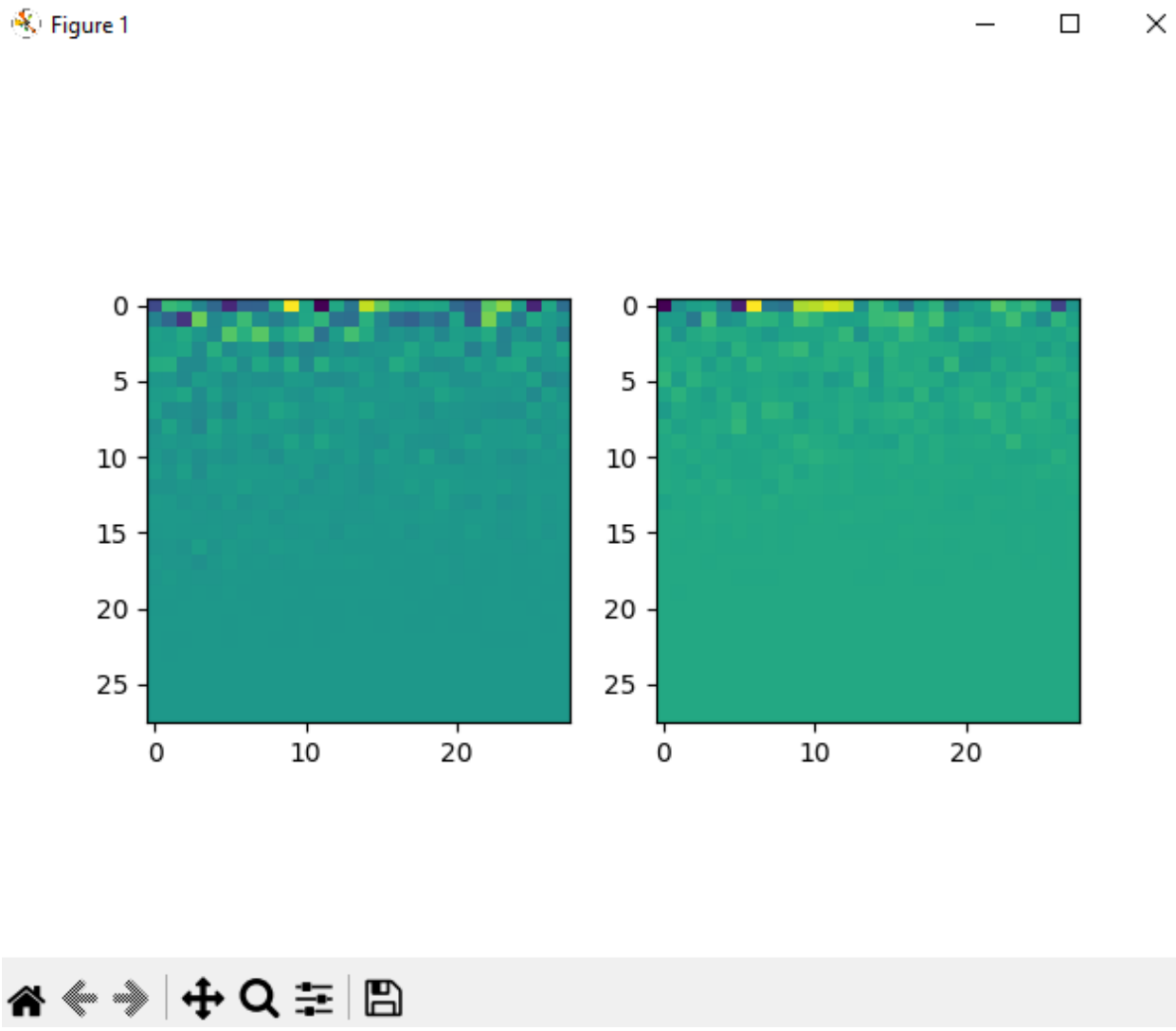
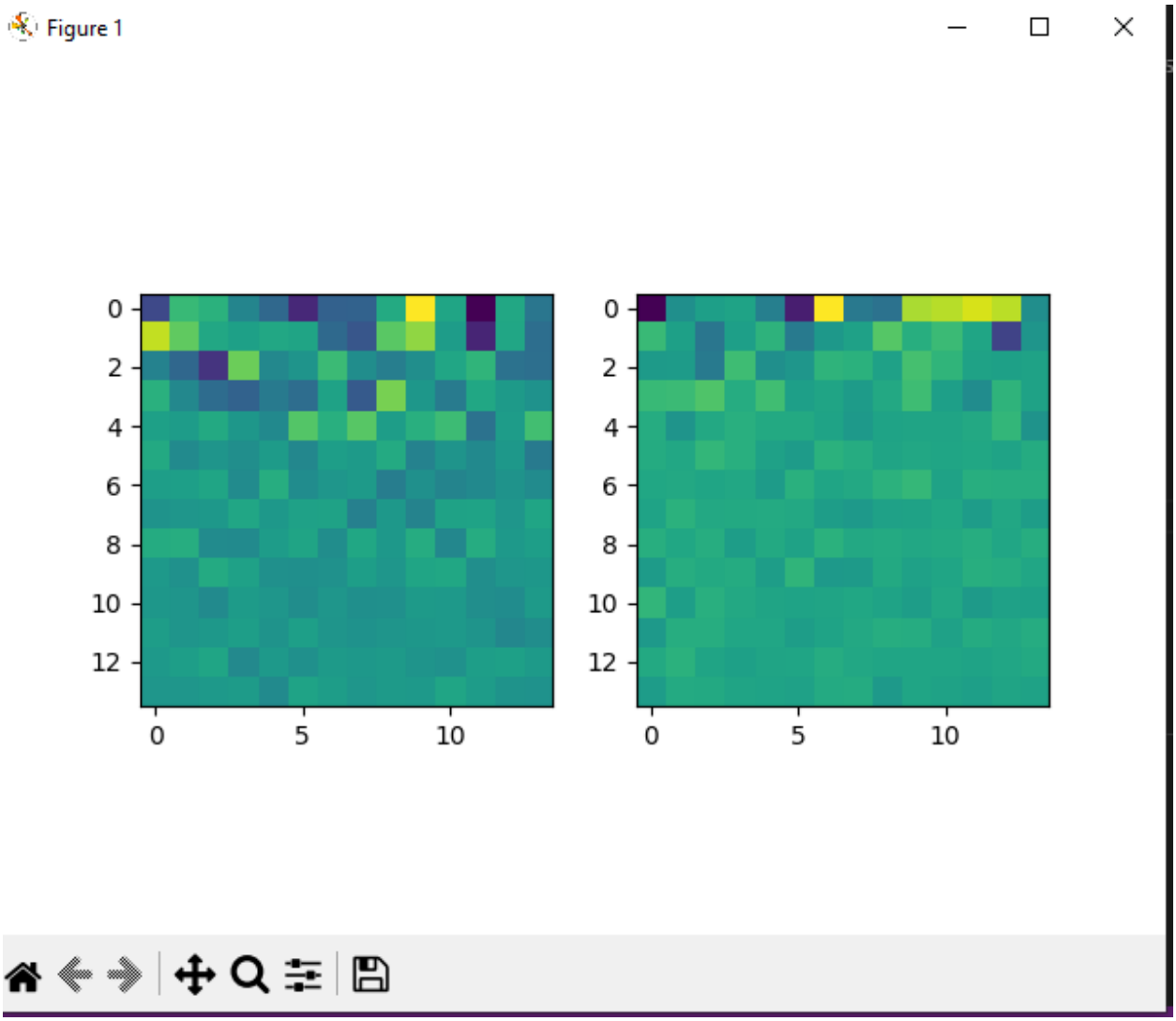


Figure 1
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
0.50
0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5

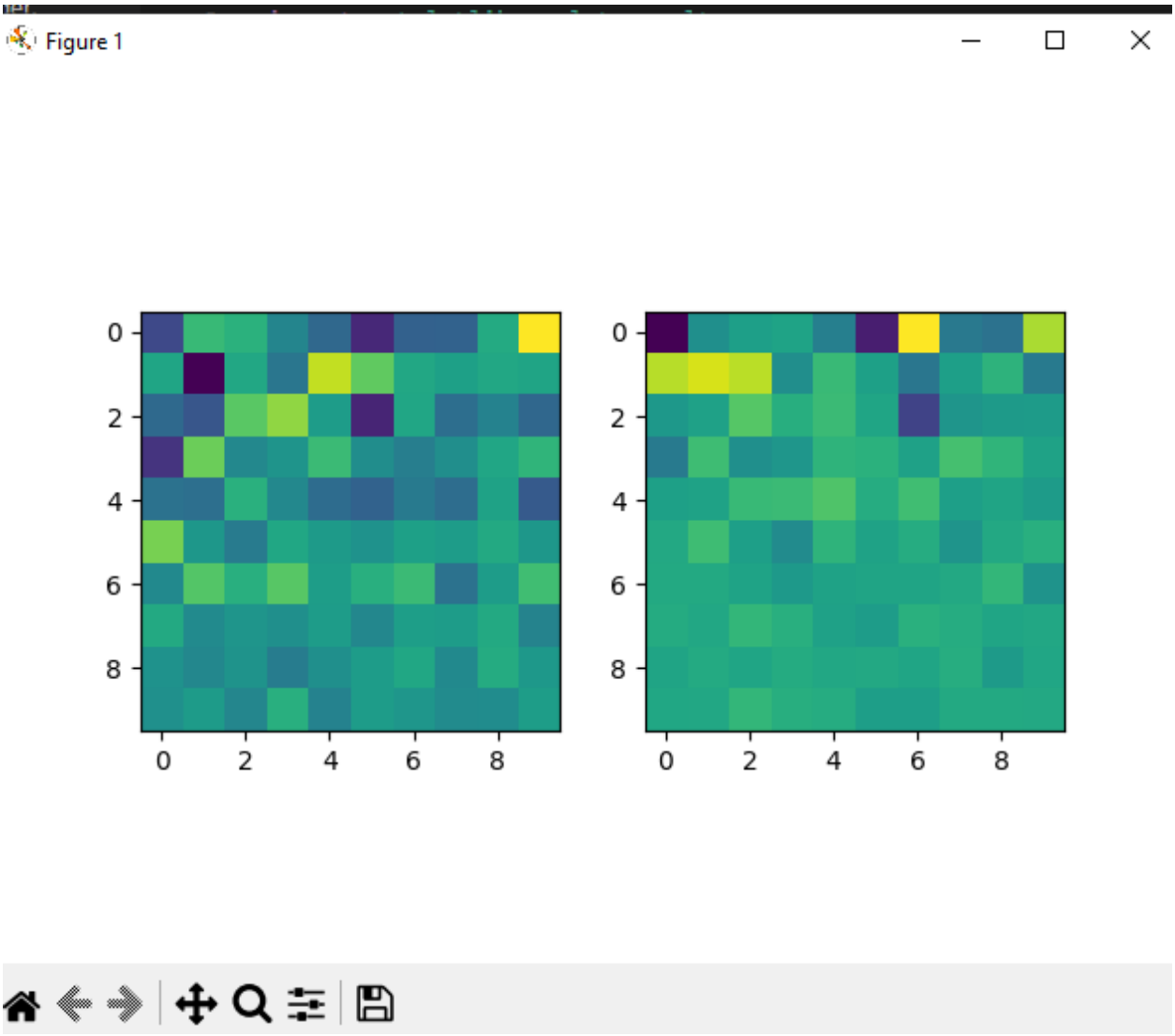
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



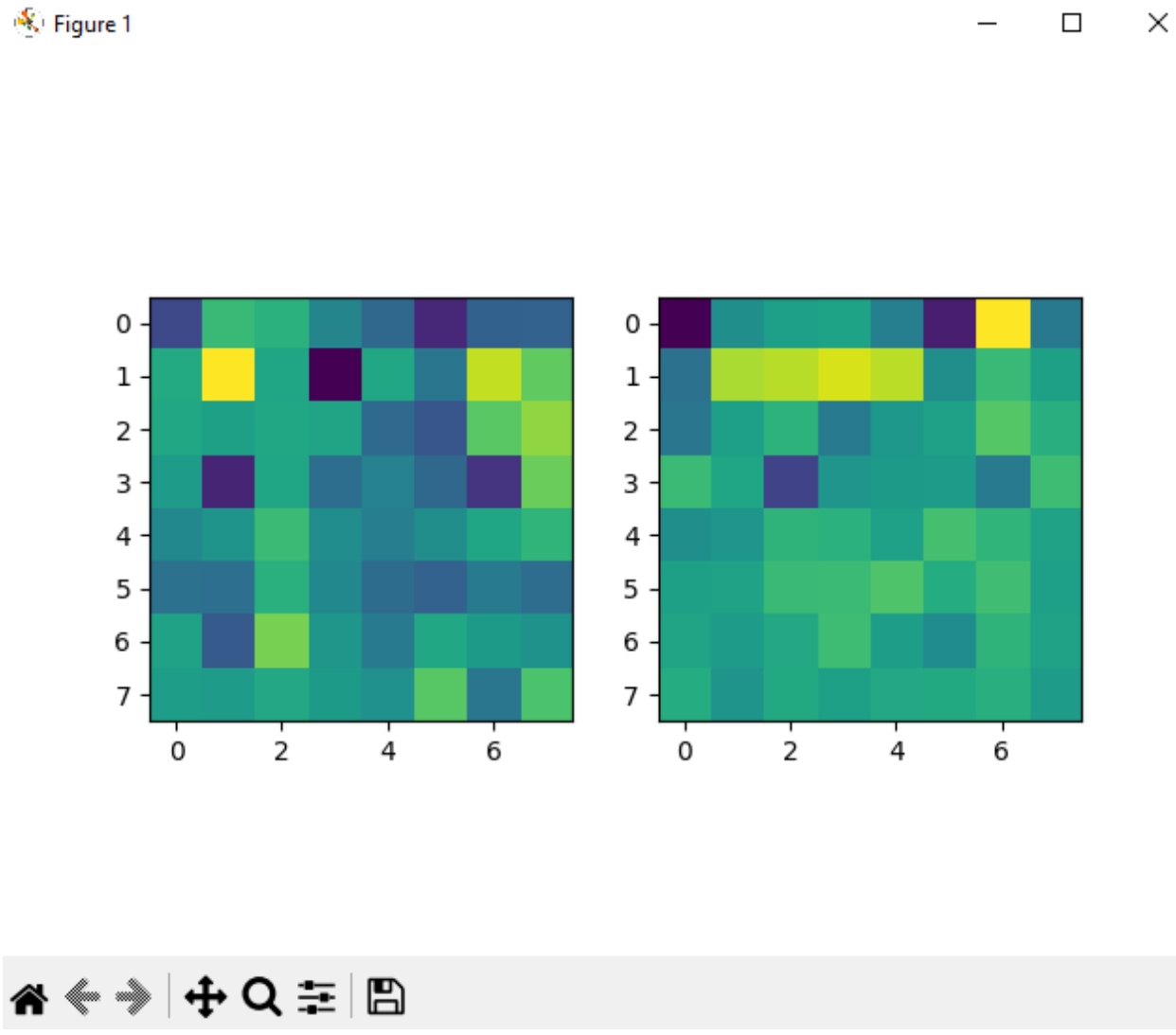
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



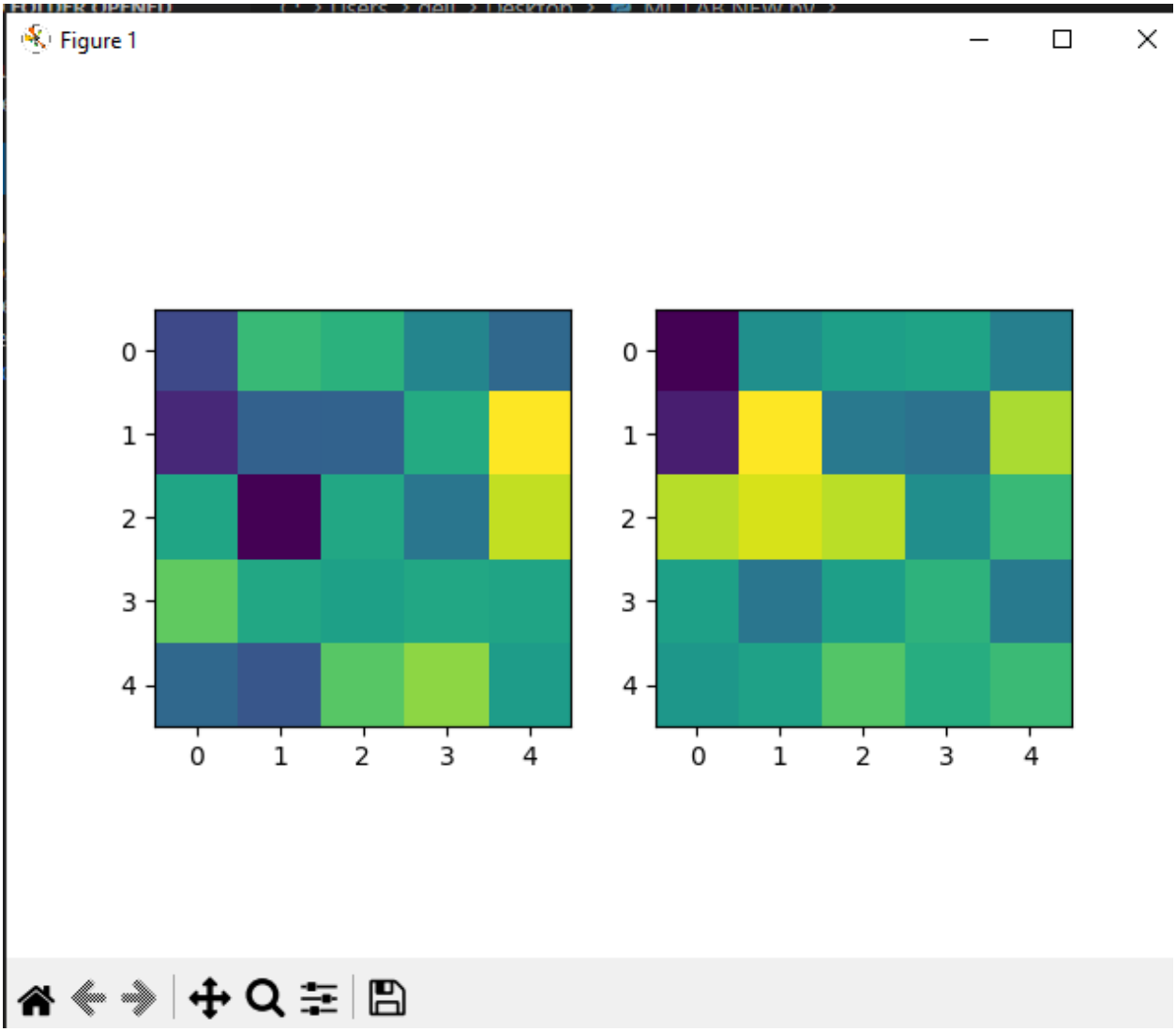
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



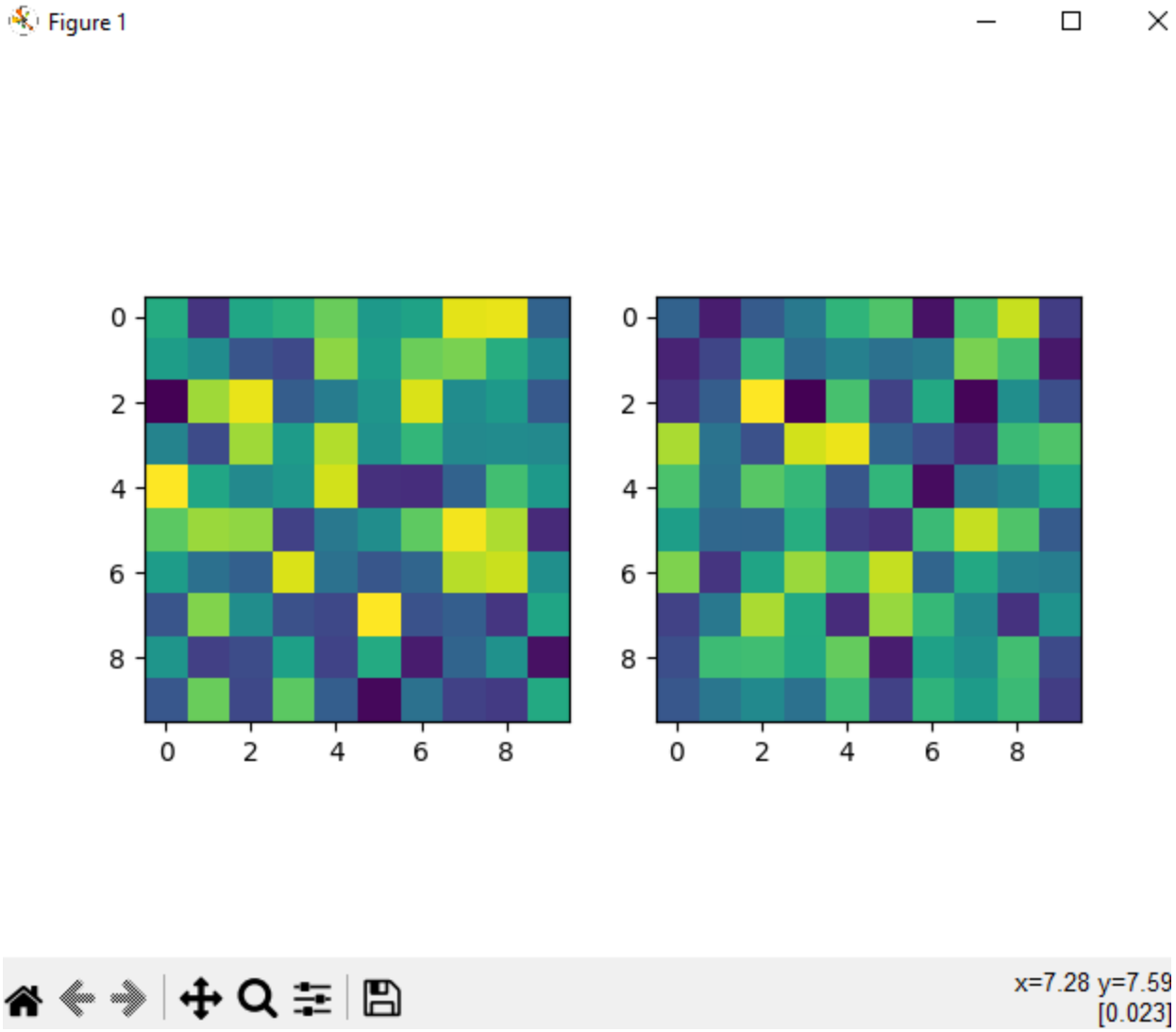
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



```
######################This code is done by gaurav
kumar################################
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import losses

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import random
tf.keras.datasets.mnist.load_data(
    path='mnist.npz'
)
(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
plt.figure()
plt.subplot(1,2,1)
print("hey")
plt.imshow(xtrain[100].reshape(28,28))
plt.subplot(1,2,2)
plt.imshow(xtrain[200].reshape(28,28))
plt.show()




import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import losses

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import random
tf.keras.datasets.mnist.load_data(
    path='mnist.npz'
)
(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
xtrain=xtrain.reshape(-1,784)
print(xtrain.shape)


plt.figure()
plt.subplot(1,2,1)
plt.imshow(xtrain[100].reshape(28,28))
plt.subplot(1,2,2)
plt.imshow(xtrain[200].reshape(28,28))
plt.show()
#task1

def pcam(X, num_components):
    # Center the data
    X = X - np.mean(X, axis=0)

    # Compute the covariance matrix
    cov_matrix = np.cov(X.T)

    # Compute the eigenvalues and eigenvectors of the covariance matrix
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

    # Sort the eigenvalues and eigenvectors in descending order
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    # Select the first num_components eigenvectors
    eigenvectors = eigenvectors[:, :num_components]

    # Transform the data into the new coordinate system
    X_pca = np.dot(X, eigenvectors)
```

```
    return X_pca


pca=PCA(5)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.subplot(1,2,1)
plt.imshow(xpca[100][:4].reshape(2,2))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:4].reshape(2,2))

#task3
xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))


pca=PCA(10)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:9].reshape(3,3))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(25)
print(pca)
```

```
xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:25].reshape(5,5))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(64)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:64].reshape(8,8))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(100)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:100].reshape(10,10))
```

```
plt.subplot(1,2,2)
plt.imshow(xpca[200][:100].reshape(10,10))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(200)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:196].reshape(14,14))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:196].reshape(14,14))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))


pca=PCA(784)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:784].reshape(28,28))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)
```

```
print(mean_squared_error(xtrain, xor))

class Autoencoder(Model):
  def __init__(self, P, Q, R, M):
    super(Autoencoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(P, activation='tanh'),
      layers.Dense(Q, activation='tanh'),
      layers.Dense(R, activation='tanh'),
      layers.Dense(M, activation='tanh'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(P, activation='swish'),
      layers.Dense(Q, activation='swish'),
      layers.Dense(R, activation='swish'),
      layers.Dense(M, activation='swish'),
      layers.Dense(784, activation='relu'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
xtrain = xtrain /255
xtest = xtest / 255

autoencoder = Autoencoder(P=128, Q=128, R=128, M=4)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:4].reshape(2,2))

plt.subplot(1,2,2)
```

```
plt.imshow(xen[200][:4].reshape(2,2))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=9)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xen[200][:9].reshape(3,3))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)



autoencoder = Autoencoder(P=128, Q=128, R=128, M=25)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))
```

```
xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xen[200][:25].reshape(5,5))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=64)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xen[200][:64].reshape(8,8))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)



autoencoder = Autoencoder(P=128, Q=128, R=128, M=100)
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:100].reshape(10,10))

plt.subplot(1,2,2)
plt.imshow(xen[200][:100].reshape(10,10))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)

autoencoder = Autoencoder(P=128, Q=128, R=128, M=784)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xen[200][:784].reshape(28,28))
plt.show()


xori = autoencoder.decoder(xen).numpy()
```

```
k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)
```

QUESTION 2

```
######################This code is done by gaurav
kumar##################################
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import losses

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import random
tf.keras.datasets.mnist.load_data(
    path='mnist.npz'
)
(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
plt.figure()
plt.subplot(1,2,1)
print("hey")
plt.imshow(xtrain[100].reshape(28,28))
plt.subplot(1,2,2)
plt.imshow(xtrain[200].reshape(28,28))
plt.show()
```

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import losses

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import random
tf.keras.datasets.mnist.load_data(
    path='mnist.npz'
)
(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
xtrain=xtrain.reshape(-1,784)
print(xtrain.shape)


plt.figure()
plt.subplot(1,2,1)
plt.imshow(xtrain[100].reshape(28,28))
plt.subplot(1,2,2)
plt.imshow(xtrain[200].reshape(28,28))
plt.show()
#task1

def pcam(X, num_components):
    # Center the data
    X = X - np.mean(X, axis=0)

    # Compute the covariance matrix
    cov_matrix = np.cov(X.T)

    # Compute the eigenvalues and eigenvectors of the covariance matrix
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

    # Sort the eigenvalues and eigenvectors in descending order
```

```
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    # Select the first num_components eigenvectors
    eigenvectors = eigenvectors[:, :num_components]

    # Transform the data into the new coordinate system
    X_pca = np.dot(X, eigenvectors)

    return X_pca


pca=PCA(5)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.subplot(1,2,1)
plt.imshow(xpca[100][:4].reshape(2,2))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:4].reshape(2,2))

#task3
xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))


pca=PCA(10)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:9].reshape(3,3))
plt.show()
```

```
xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(25)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:25].reshape(5,5))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(64)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:64].reshape(8,8))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(100)
```

```
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:100].reshape(10,10))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:100].reshape(10,10))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

pca=PCA(200)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(xpca[100][:196].reshape(14,14))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:196].reshape(14,14))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))


pca=PCA(784)
print(pca)

xpca=pca.fit_transform(xtrain)
print(xpca.shape)

plt.figure()
```

```
plt.subplot(1,2,1)
plt.imshow(xpca[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xpca[200][:784].reshape(28,28))
plt.show()

xor=pca.inverse_transform(xpca)
print(xor.shape)

print(mean_squared_error(xtrain, xor))

class Autoencoder(Model):
  def __init__(self, P, Q, R, M):
    super(Autoencoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(P, activation='tanh'),
      layers.Dense(Q, activation='tanh'),
      layers.Dense(R, activation='tanh'),
      layers.Dense(M, activation='tanh'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(P, activation='swish'),
      layers.Dense(Q, activation='swish'),
      layers.Dense(R, activation='swish'),
      layers.Dense(M, activation='swish'),
      layers.Dense(784, activation='relu'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
xtrain = xtrain /255
xtest = xtest / 255

autoencoder = Autoencoder(P=128, Q=128, R=128, M=4)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
```

```
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:4].reshape(2,2))

plt.subplot(1,2,2)
plt.imshow(xen[200][:4].reshape(2,2))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=9)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:9].reshape(3,3))

plt.subplot(1,2,2)
plt.imshow(xen[200][:9].reshape(3,3))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)
```

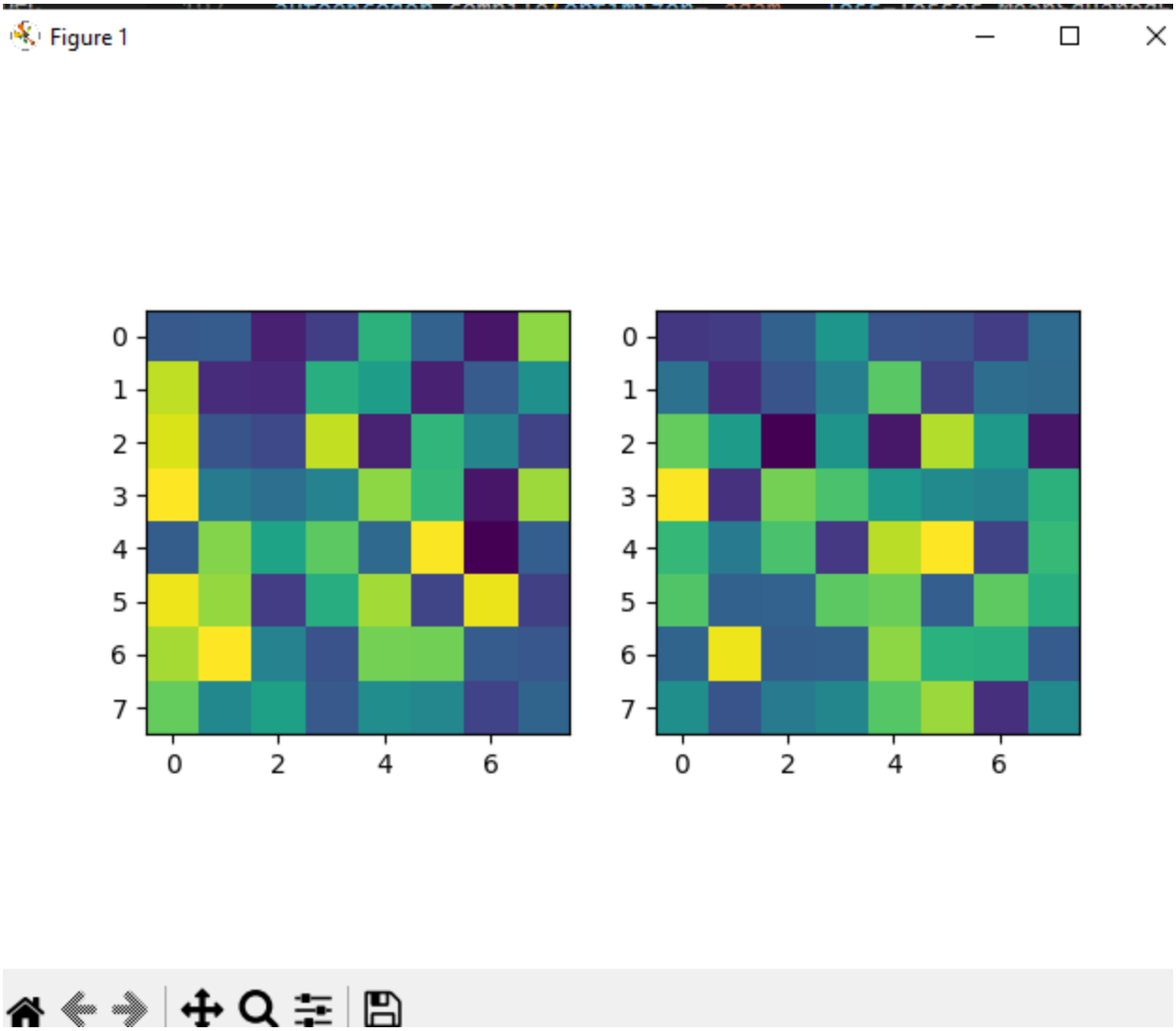
```
autoencoder = Autoencoder(P=128, Q=128, R=128, M=25)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:25].reshape(5,5))

plt.subplot(1,2,2)
plt.imshow(xen[200][:25].reshape(5,5))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)


autoencoder = Autoencoder(P=128, Q=128, R=128, M=64)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:64].reshape(8,8))

plt.subplot(1,2,2)
plt.imshow(xen[200][:64].reshape(8,8))
plt.show()
```

```
xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)



autoencoder = Autoencoder(P=128, Q=128, R=128, M=100)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
plt.imshow(xen[100][:100].reshape(10,10))

plt.subplot(1,2,2)
plt.imshow(xen[200][:100].reshape(10,10))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)

autoencoder = Autoencoder(P=128, Q=128, R=128, M=784)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(xtrain.reshape(-1, 28, 28), xtrain.reshape(-1, 28, 28),
                epochs=1,
                shuffle=True,
                validation_data=(xtest.reshape(-1, 28, 28), xtest.reshape(-1, 28,
28)))

xen = autoencoder.encoder(xtrain.reshape(-1, 28, 28)).numpy()
plt.figure()
plt.subplot(1,2,1)
```

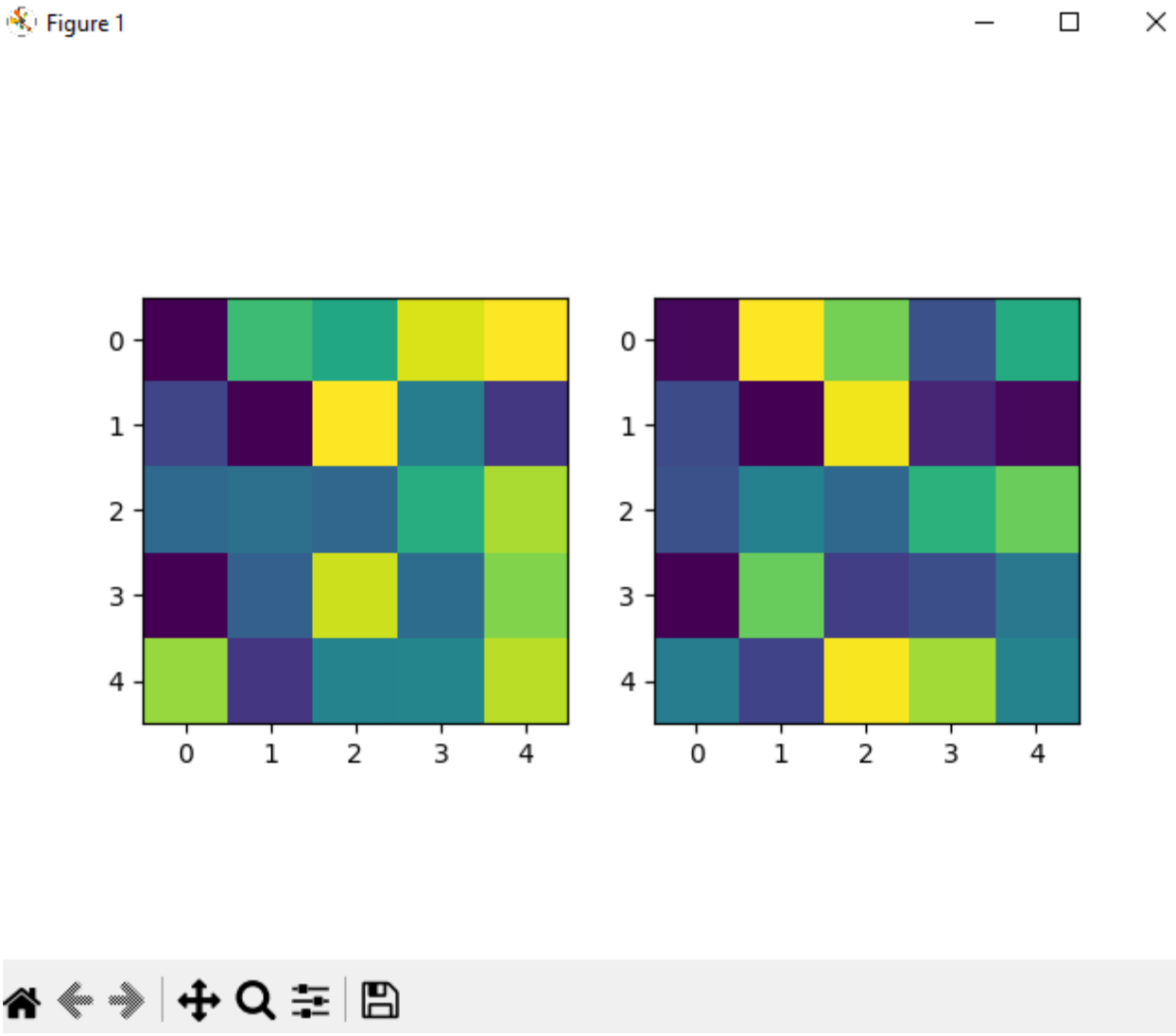
```
plt.imshow(xen[100][:784].reshape(28,28))

plt.subplot(1,2,2)
plt.imshow(xen[200][:784].reshape(28,28))
plt.show()


xori = autoencoder.decoder(xen).numpy()

k=mean_squared_error(xtrain, xori.reshape(-1, 784))
print(k)
```

OUTPUT:-

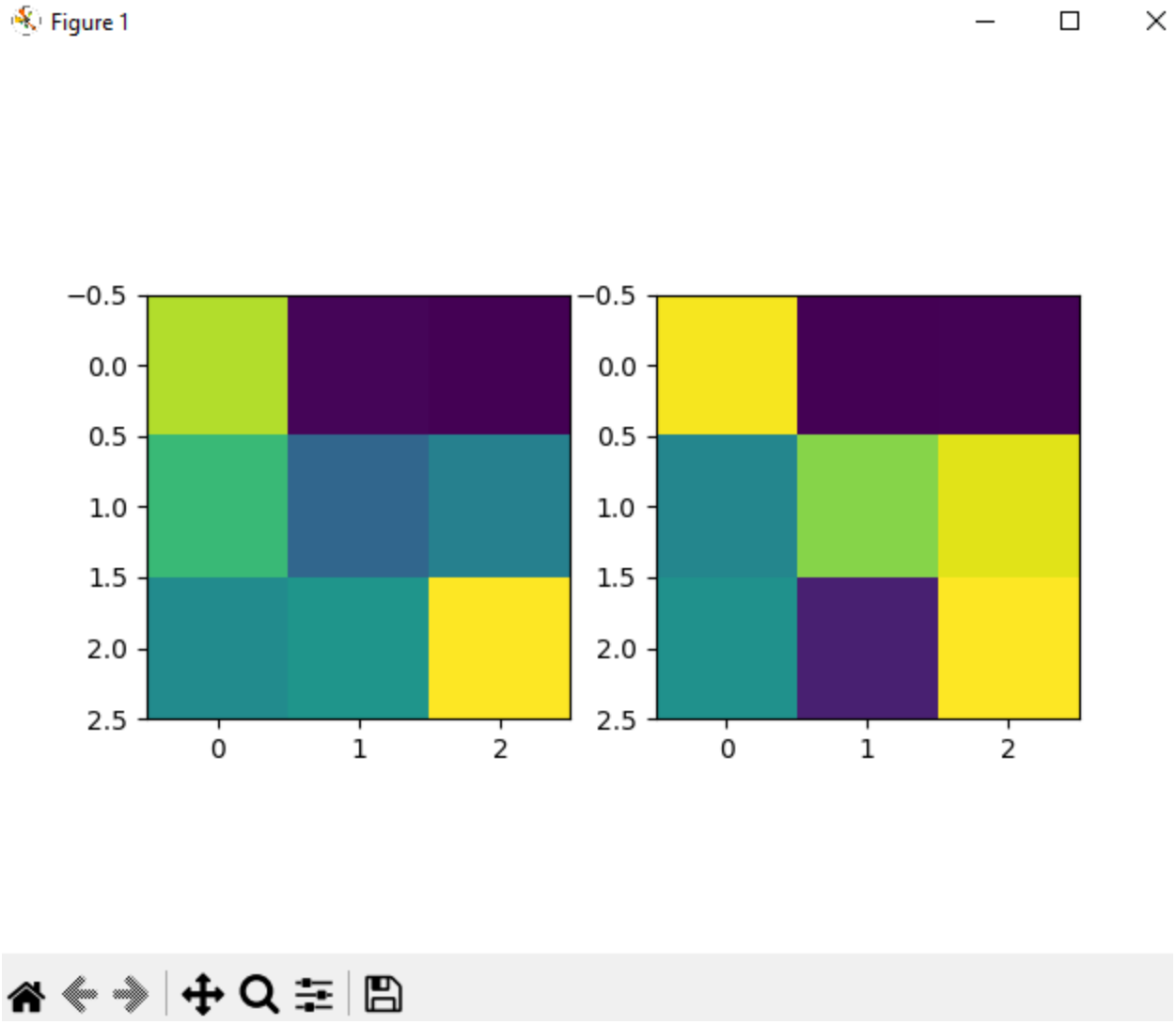
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



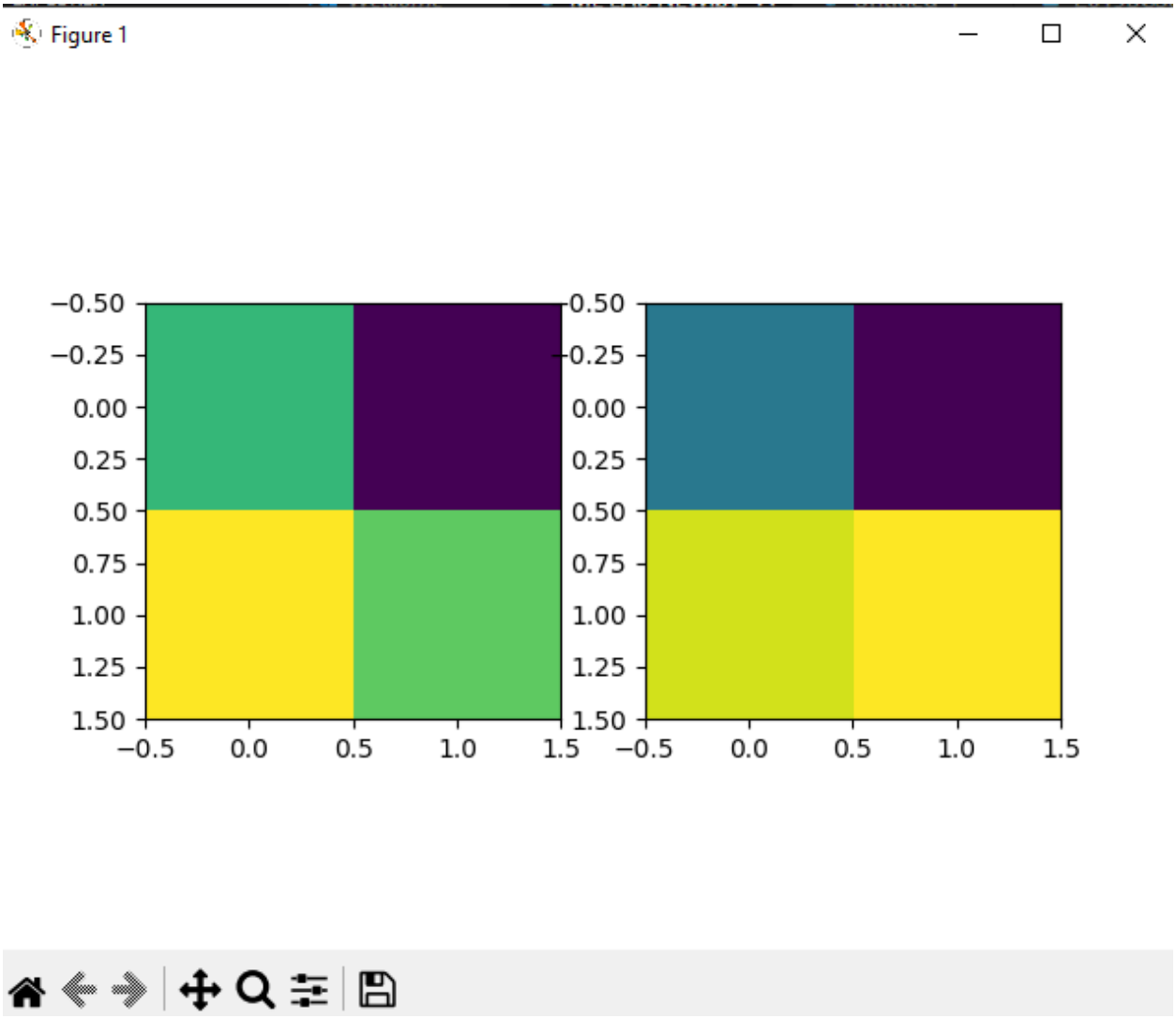
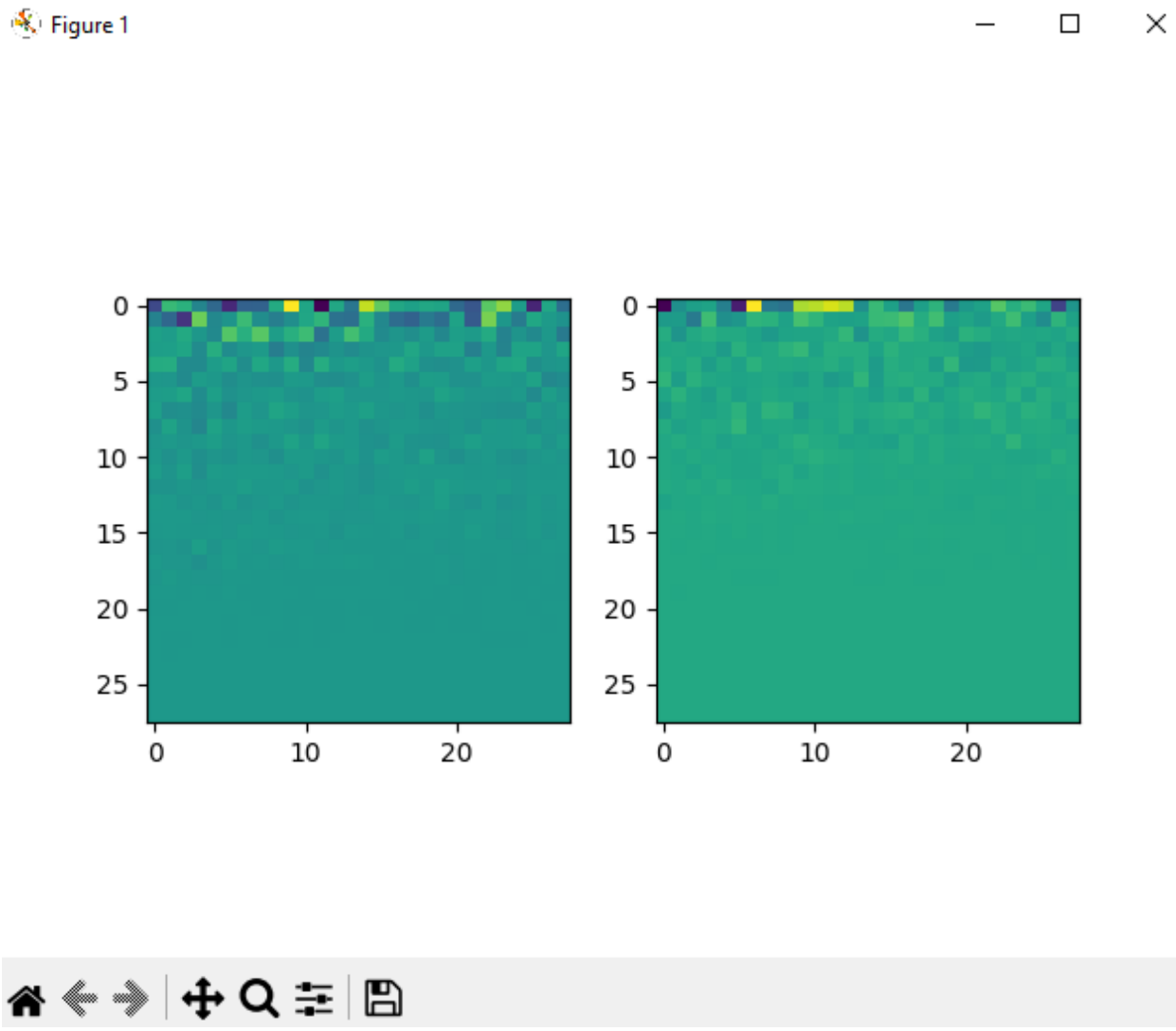
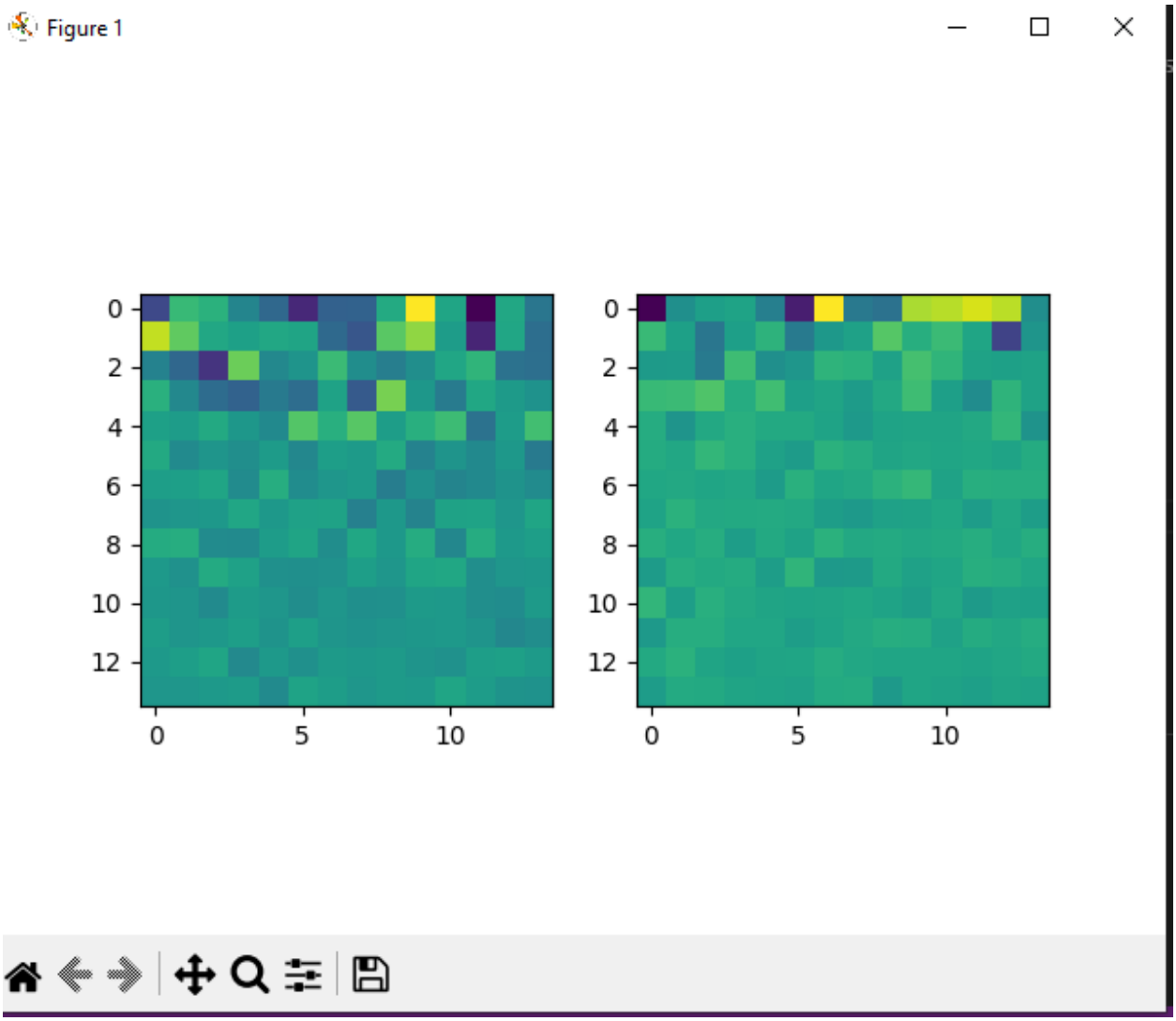


Figure 1
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
0.50
0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5

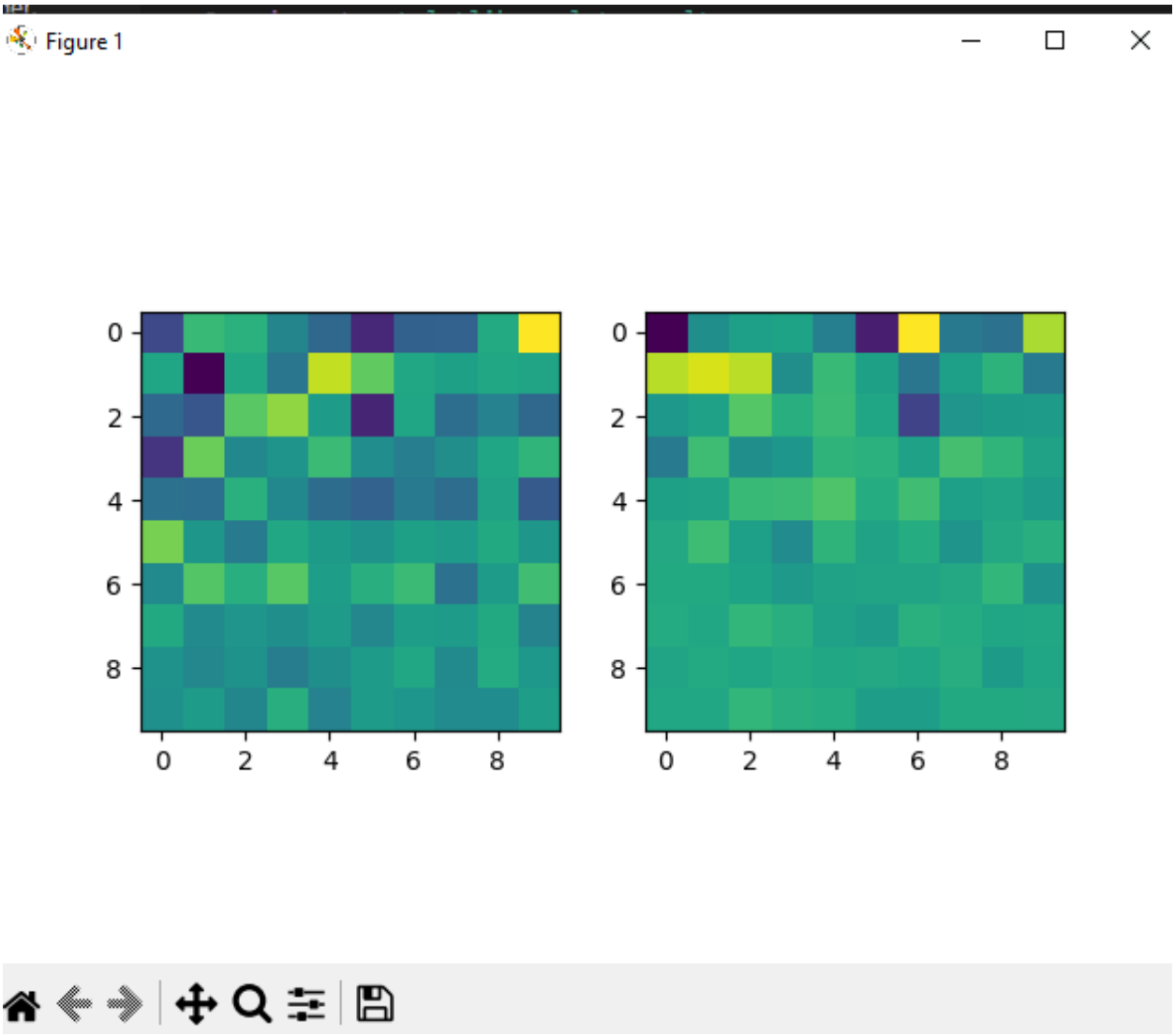
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



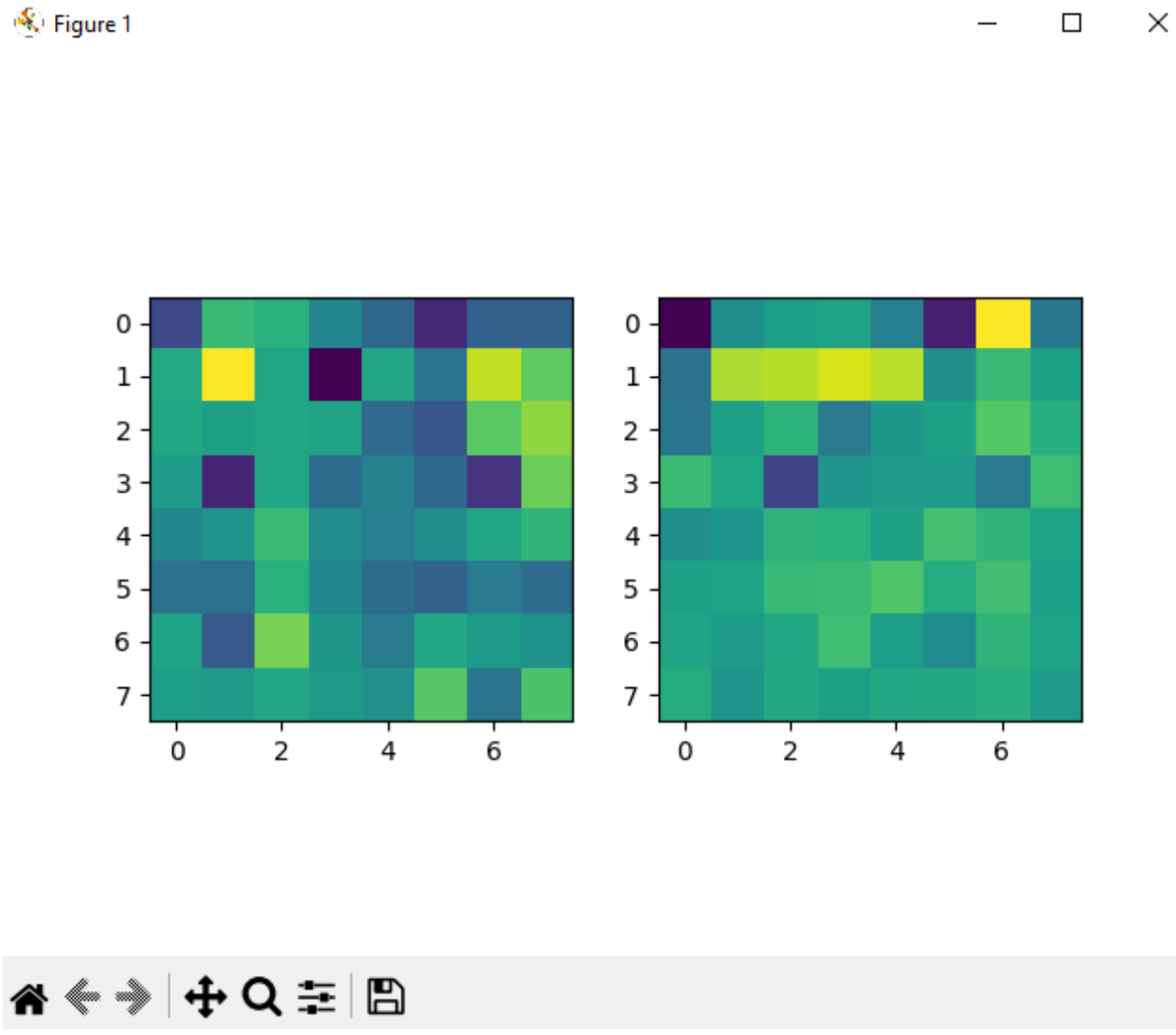
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



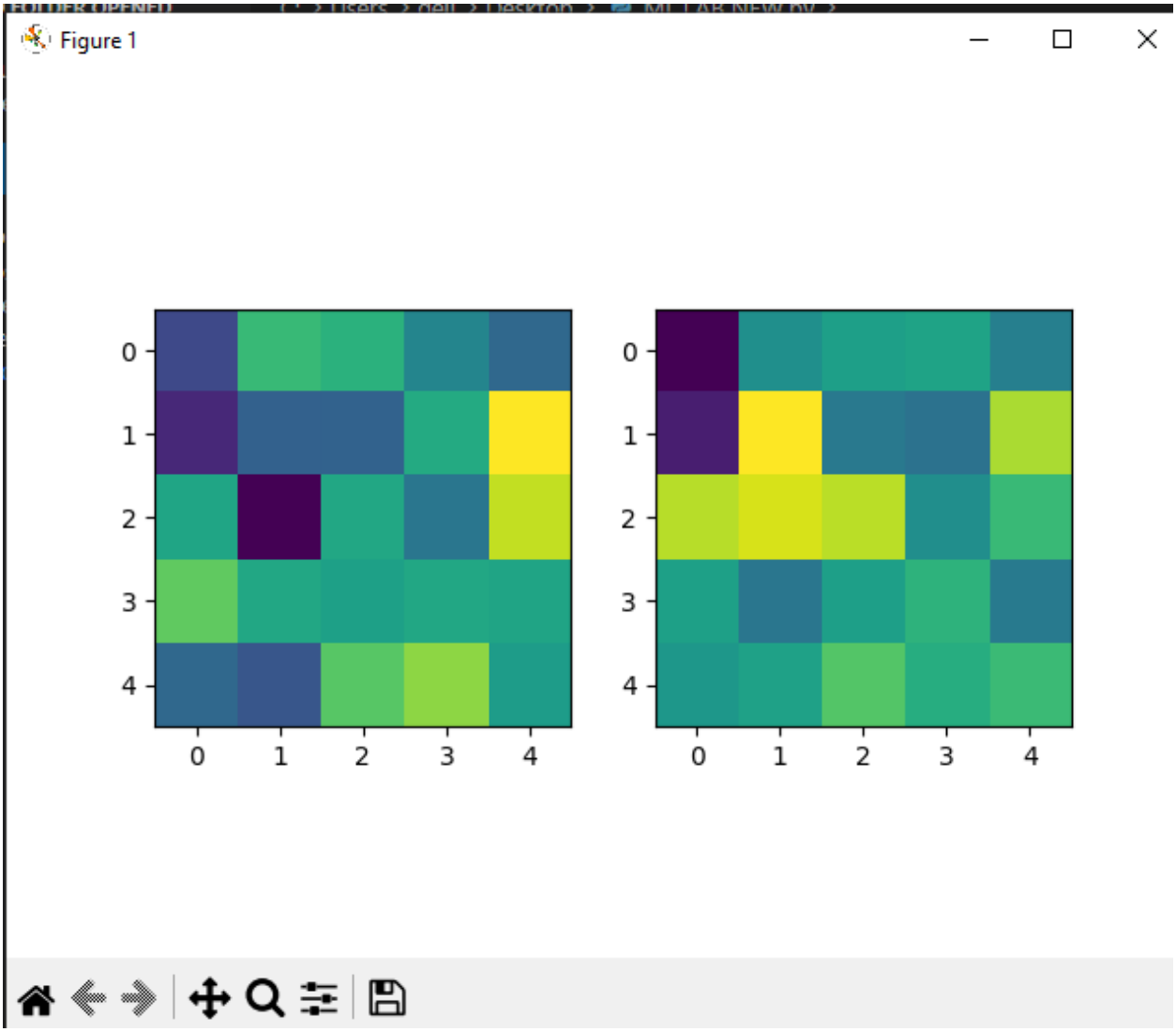
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



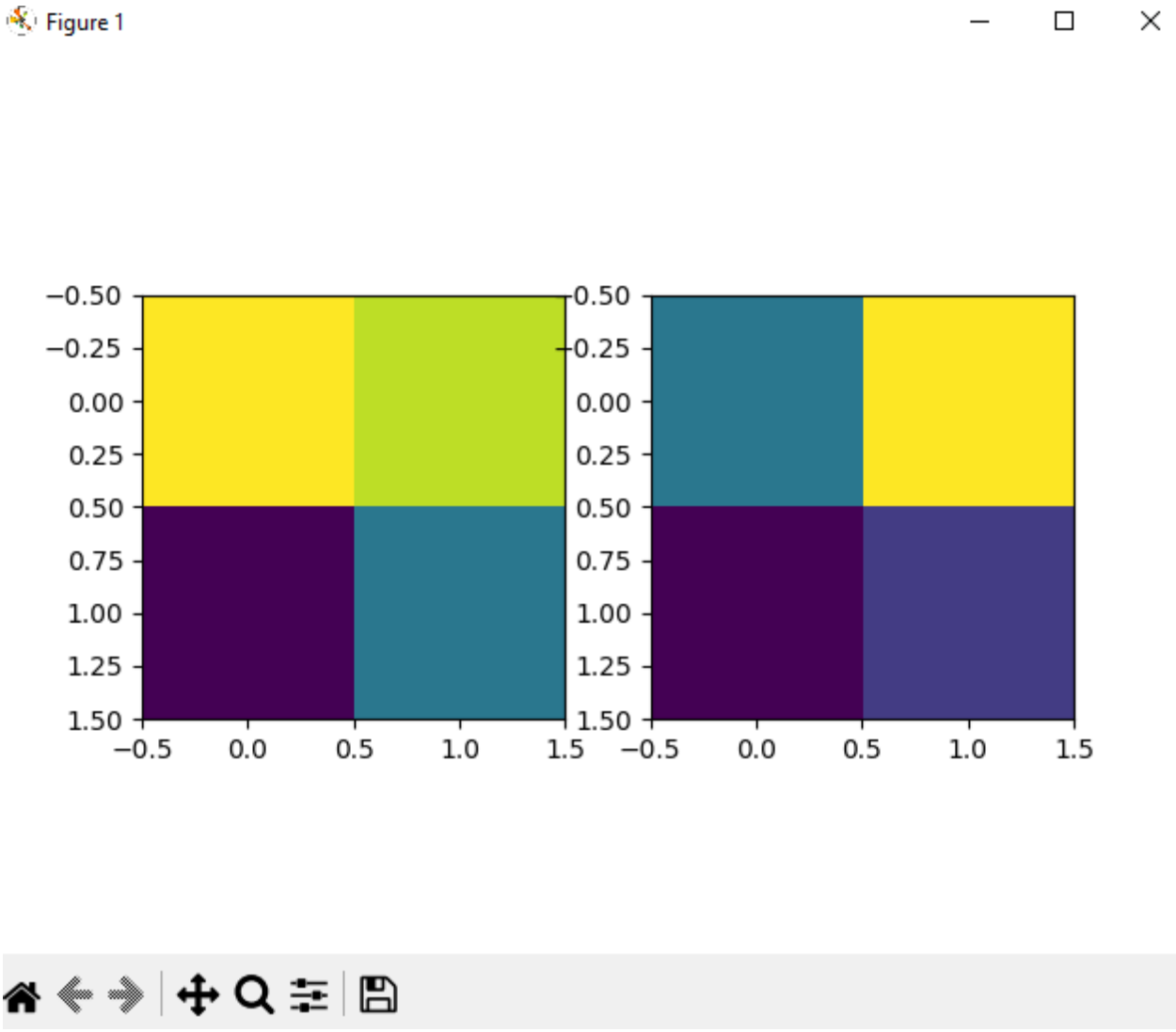
NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017



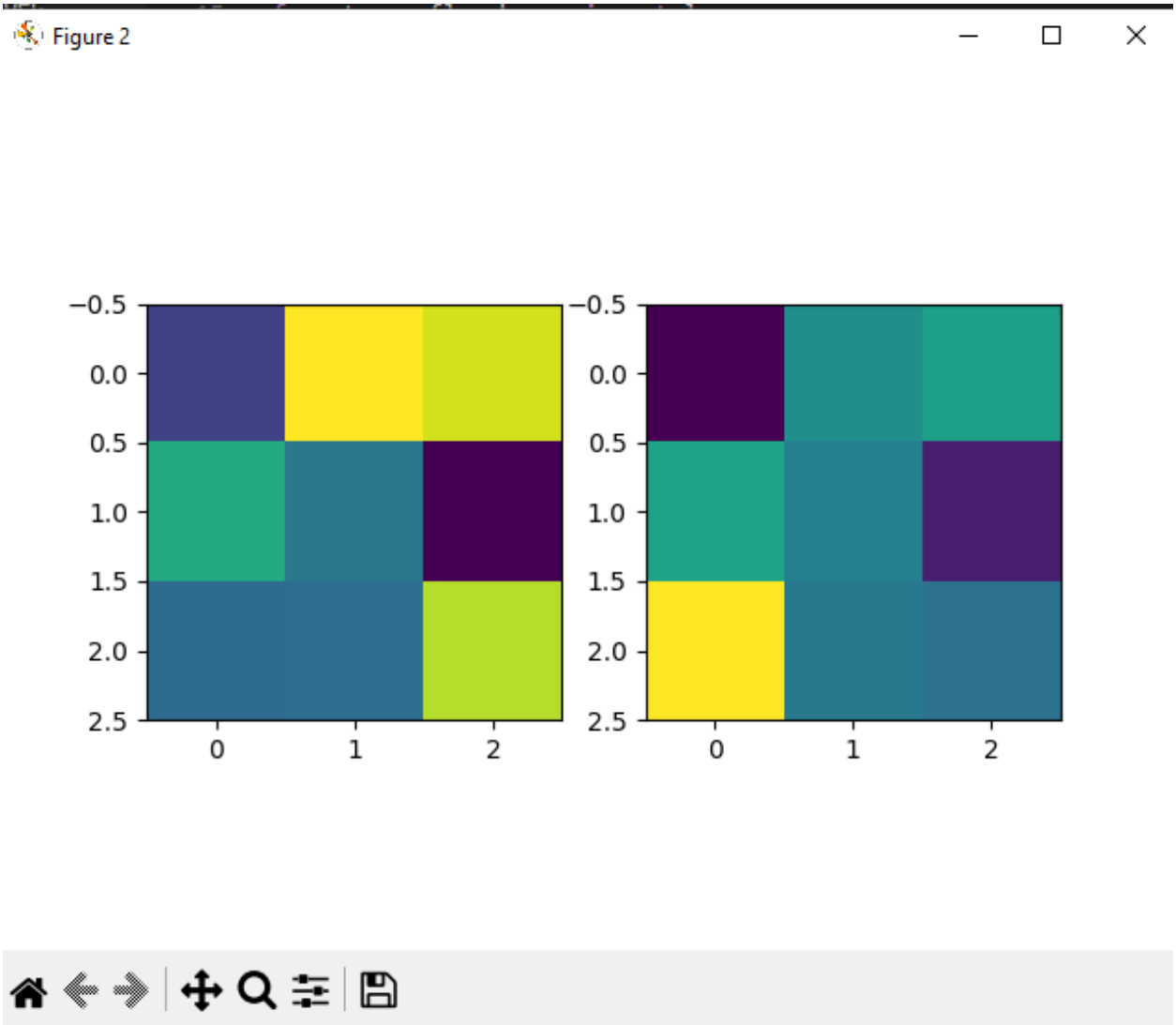


Figure 2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
0
1
2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
0
1
2

NAME- ASHWIN KUMAR SINGH ROLL-2020BCS017