Neural Networks - Image Recognition

In [1]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
```

In [2]:
```
import matplotlib.pyplot as  plt
%matplotlib inline
```

In [3]:
```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

1. Add random noise (see below on `size parameter` on `np.random.normal`) to the images in training and testing. **Make sure each image gets a different noise feature added to it. Inspect by printing out several images. Note - the `size` parameter should match the data.**

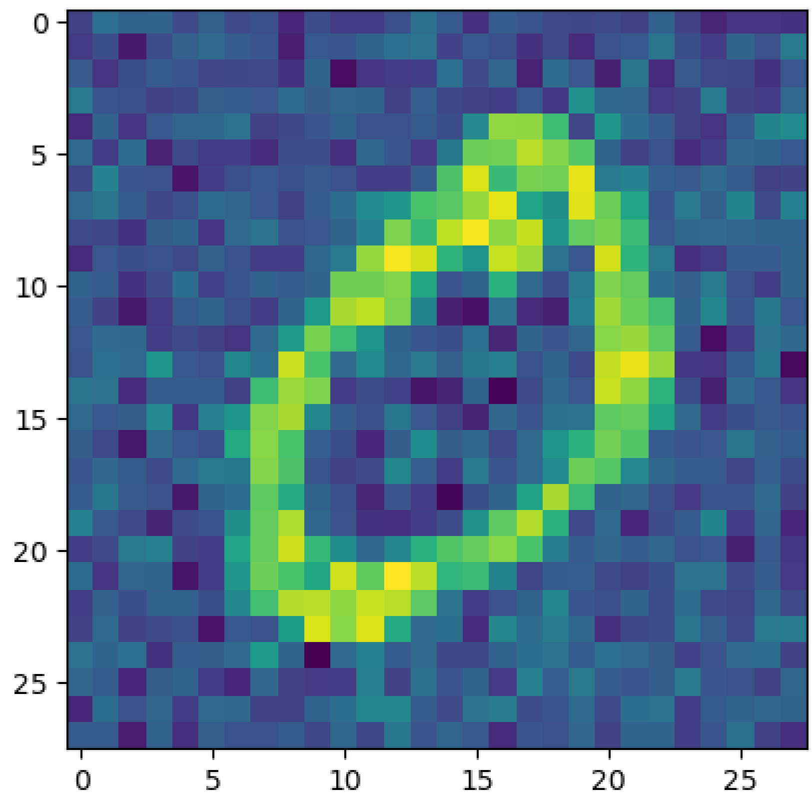
In [4]:
```
# Noise is added here

# Generate random noise with the same shape as the image
import numpy as np
mean = .5
stddev = .16
noise = np.random.normal(mean, stddev, x_train[1].shape)

# Add the noise to the image
image_noise_added = x_train[1] + noise
image_noise_added


# The max value of the noise should not grossly surpass 1.0
max(noise)
```

Out[4]:
```
0.9814309422337107
```

In [5]:
```
# Max of noise is less than .1002; very slightly over 1
```

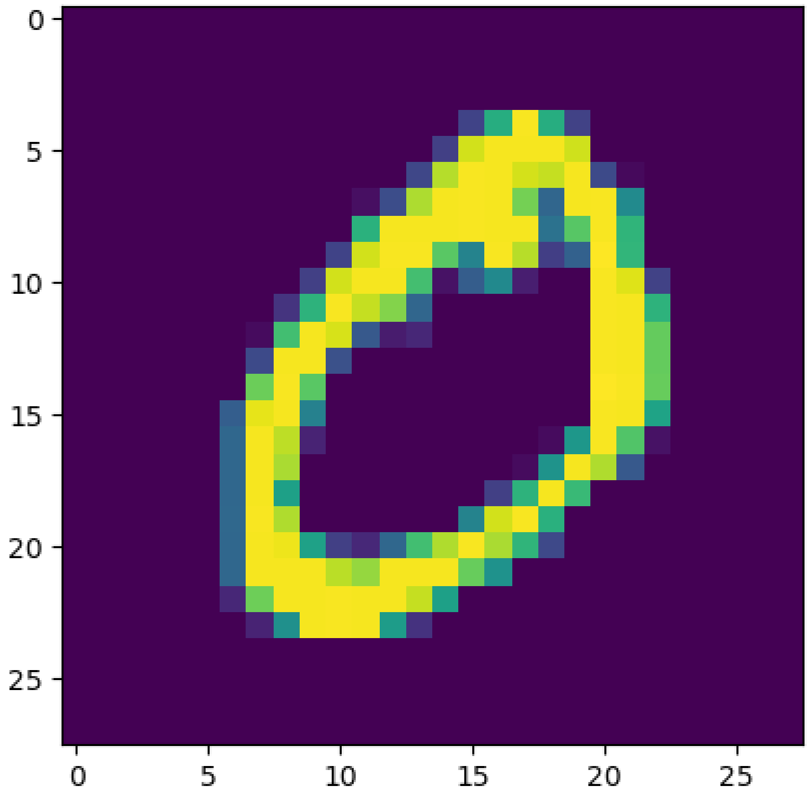
In [6]:
```
# New Image vs Old
# New Image
plt.imshow(image_noise_added.reshape(28, 28))
```

Out[6]: <matplotlib.image.AxesImage at 0x27146484400>



In [7]:
```
# Old Image
plt.imshow(x_train[1].reshape(28, 28))
```

Out[7]: <matplotlib.image.AxesImage at 0x271484f4760>



In [8]:
```
### Add noise to whole set of images
# Noise is added here

# Generate random noise with the same shape as the image
import numpy as np
mean = .5
stddev = .16
noise = np.random.normal(mean, stddev, x_train.shape)

# Generate random noise with the same shape as the image
test_noise = np.random.normal(mean, stddev, x_test.shape)
y_train_noise = np.random.normal(mean, stddev, y_train.shape)
y_test_noise = np.random.normal(mean, stddev, y_test.shape)

# Add noise
x_train_noise = x_train + noise
x_test_noise = x_test + test_noise
y_train_noise = y_train + y_train_noise
y_test_noise = y_test + y_test_noise

# Add noise
x_test_noise[1]
```

Out[8]:
```
array([ 0.57687232,  0.40826787,  0.48698683,  0.50217036,  0.645252  ,
        0.61910315,  0.41440629,  0.51303236,  0.63962366,  0.64682592,
        0.37467939,  0.52838516,  0.57147313,  0.60116104,  0.42979264,
        0.47278035,  0.17707505,  0.6037902 ,  0.38793955,  0.4623349 ,
        0.19363086,  0.37676522,  0.67193465,  0.58154555,  0.46630765,
        0.34852071,  0.42603748,  0.60274664,  0.34968219,  0.57033017,
        0.32556246,  0.34093753,  0.46699395,  0.40841534,  0.31135967,
        0.40805849,  0.55859174,  0.638774  ,  0.30170078,  0.73114616,
        0.37730497,  0.63878796,  0.44348356,  0.5813878 ,  0.60068954,
        0.28910888,  0.62700597,  0.36169634,  0.46364495,  0.46958263,
        0.50642522,  0.4631466 ,  0.48307732,  0.63050008,  0.54317324,
        0.46467005,  0.76507951,  0.41156469,  0.67368421,  0.45407504,
        0.41759021,  0.26744614,  0.52759676,  0.53368195,  0.48003363,
        0.51910648,  0.79028283,  0.72157947,  0.47613033,  0.38042947,
        0.71802766,  0.35599436,  0.50874887,  0.33294799,  0.33277805,
        0.51121625,  0.29037356,  0.54504028,  0.80172431,  0.65532005,
        0.66518034,  0.47198225,  0.36786903,  0.6233744 ,  0.42654806,
        0.626913  ,  0.39782186,  0.42302727,  0.50334411,  0.4370632 ,
        0.4251654 ,  0.30500427,  0.55564946,  0.38227468,  0.96422872,
        0.91836584,  1.24052622,  1.46307529,  1.56094511,  1.25740408,
        0.69438408,  0.6005917 ,  0.53072081,  0.45989333,  0.31914228,
        0.47099433,  0.22500937,  0.5613782 ,  0.75790809,  0.70932527,
        0.45917871,  0.39644749,  0.33798038,  0.32350235,  0.61883753,
        0.39654337,  0.70952601,  0.60944194,  0.50570285,  0.56547912,
        0.7316817 ,  1.48010999,  1.20563341,  1.5354358 ,  1.30711049,
        1.5711794 ,  1.43229737,  1.29984594,  1.38860626,  0.44369197,
        0.34746496,  0.54786675,  0.46418887,  0.72041538,  0.71813562,
        0.34223791,  0.45970136,  1.14808828,  0.47099867,  0.27251569,
        0.55721335,  0.38836619,  0.29644195,  0.41930418,  0.59695966,
        0.53640419,  0.64318424,  0.35857606,  1.12010692,  1.56477894,
        1.58059331,  1.60154089,  1.27910914,  1.04854077,  1.45419382,
        1.60501054,  1.61817443,  0.91370378,  0.5173893 ,  0.58896817,
        0.39474733,  0.58212097,  0.46727581,  0.79173531,  0.60242163,
        0.45962592,  0.54456266,  0.37801249,  0.56099927,  0.56874342,
        0.6159749 ,  0.50175787,  0.55086682,  0.6257764 ,  0.75945265,
        1.07127152,  1.36776284,  1.35008523,  1.36692701,  0.77233484,
        0.57868446,  0.30944101,  0.71401451,  1.20846498,  1.65998397,
        1.29847969,  0.66515439,  0.36115685,  0.22984019,  0.63827504,
        0.51871401,  0.41042776,  0.57781973,  0.4880221 ,  0.77902976,
        0.67478363,  0.57619222,  0.29292569,  0.3714438 ,  0.47110892,
        0.65160108,  0.67104572,  0.39016473,  0.91870886,  1.25853786,
        1.43544991,  0.73563021,  0.78235505,  0.38323818,  0.58749505,
        0.78000725,  1.69187491,  1.49744226,  0.88697527,  0.79193357,
        0.56011929,  0.63032284,  0.69932626,  0.37608275,  0.61027271,
        0.52195608,  0.48601603,  0.74008066,  0.40982626,  0.61027295,
        0.47972026,  0.3698776 ,  0.55671247,  0.61515198,  0.40591306,
        0.44684098,  0.5449375 ,  0.51541301,  0.4153461 ,  0.46303803,
        0.50765033,  0.57844954,  0.42400258,  1.22140617,  1.61883959,
        1.42202093,  0.58517293,  0.23959164,  0.52297909,  0.64580432,
        0.34583476,  0.48682419,  0.30327991,  1.04286853,  0.31668652,
        0.59552655,  0.43303125,  0.60815892,  0.21081781,  0.64153105,
        0.64774413,  0.71064805,  0.42262043,  0.40247365,  0.44596662,
        0.40861889,  0.49788938,  0.52451359,  0.62464182,  0.39184073,
        1.34829301,  1.70225717,  1.53762747,  1.51560417,  0.48299895,
        0.33936687,  0.69238985,  0.32186298,  0.2586233 ,  0.28474299,
        0.55044112,  0.52029234,  0.46978064,  0.66648489,  0.32801189,
        0.41059155,  0.58940078,  0.67963343,  0.52471417,  0.61162958,
        0.80035667,  0.66476024,  0.458238  ,  0.75578721,  0.56674768,
        0.3457525 ,  0.72454083,  0.93367028,  1.49255573,  1.35476919,
        1.32076445,  1.14075649,  0.40289053,  0.38107664,  0.444363  ,
```

```
0.57212264,  0.38351332,  0.68051138,  0.54716453,  0.62290098,
0.25550828,  0.70895076,  0.64530895,  0.27380722,  0.77455907,
0.50103175,  0.58251633,  0.41462832,  0.14251818,  0.46168851,
0.60593235,  0.66388161,  0.46078412,  0.57685837,  0.30246699,
1.2257414 ,  1.3880618 ,  1.43614418,  0.95321121,  0.45152506,
0.19778098,  0.54767181,  0.29674595,  0.41591414,  0.52391182,
0.57302419,  0.58286705,  0.676693  ,  0.45983768,  0.70680645,
0.47206472,  0.68424349,  0.13648687,  0.5869302 ,  0.55534703,
0.48605483,  0.34212007,  0.56896385,  0.60127747,  0.36992377,
0.44773107,  0.69464795,  1.10740255,  1.43281916,  1.40614555,
1.19718594,  0.56582603,  0.21850394,  0.5248323 ,  0.5604059 ,
0.49926849,  0.49443567,  0.46567549,  0.42704746,  0.60985869,
0.61681303,  0.7036364 ,  0.29623638,  0.50978999,  0.63400692,
0.41059443,  0.30164586,  0.55097491,  0.69367964,  0.44303171,
0.58091256,  0.44285741,  0.56410263,  0.61156829,  0.75049957,
1.43629381,  1.52172267,  1.34290132,  0.55388268,  0.38844558,
0.48508947,  0.3696634 ,  0.63216608,  0.48751598,  0.37656868,
0.43954913,  0.23700829,  0.4159265 ,  0.41542122,  0.47840707,
0.44554244,  0.72638413,  0.56213323,  0.76256056,  0.44031373,
0.43894338,  0.25594314,  0.33465653,  0.44415611,  0.51582468,
0.55120162,  0.66069936,  1.31538128,  1.66328685,  1.44516188,
1.02270407,  0.12003318,  0.69552721,  0.53139921,  0.33769814,
0.79705492,  0.75478203,  0.34682688,  0.70877293,  0.79783351,
0.36799432,  0.72164162,  0.46368178,  0.61005969,  0.76968896,
0.80616255,  0.46435952,  0.6975923 ,  0.59662083,  0.6061286 ,
0.33113977,  0.46146188,  0.62182404,  0.4812855 ,  0.75222937,
1.2436928 ,  1.78719514,  1.31072272,  0.3970764 ,  0.31548864,
0.56647389,  0.49042147,  0.49246477,  0.70088437,  0.32063056,
0.43199672,  0.4218937 ,  0.60392227,  0.65387342,  0.44636835,
0.61034481,  0.32493709,  0.44398562,  0.63548597,  0.47206533,
0.58278564,  0.52560018,  0.46213753,  0.53412207,  0.61547489,
0.5045515 ,  0.61784269,  1.20916063,  1.31296184,  1.53725133,
1.17186029,  0.34904539,  0.50627498,  0.50687739,  0.53640772,
0.56586986,  0.43075534,  0.60988961,  0.62925669,  0.56843169,
0.61100649,  0.43874335,  0.53837976,  0.62260578,  0.41965348,
0.6808537 ,  0.24438297,  0.49623488,  0.57181072,  0.34635712,
0.50992352,  0.5679127 ,  0.47805241,  0.50743278,  1.05503901,
1.51676278,  1.47122376,  1.43540835,  0.68739887,  0.81629148,
0.3794722 ,  0.55767566,  0.41346537,  0.49601173,  0.47879598,
0.66224115,  0.41859635,  0.63192803,  0.61052622,  0.37330411,
0.47443541,  0.28885453,  0.56104045,  0.48758825,  0.68472919,
0.7342782 , -0.0146911 ,  0.74080096,  0.44821362,  0.80058136,
0.56618772,  0.48387254,  1.40739311,  1.36079843,  1.39607238,
0.50969077,  0.5432472 ,  0.49923699,  0.541422  ,  0.3493232 ,
0.53854721,  0.52690456,  0.36931214,  0.60147836,  0.56491768,
0.6680341 ,  0.58817732,  0.49161062,  0.5382499 ,  0.66288018,
0.24496488,  0.60784583,  0.5100051 ,  0.45461803,  0.27164354,
0.40266897,  0.50877334,  0.63210122,  0.57207429,  0.51167262,
1.58497908,  1.16691554,  1.36532609,  1.02976064,  0.68604067,
0.6000592 ,  0.64854475,  0.50860073,  0.56702268,  0.39755904,
0.50356904,  0.52461119,  0.61202455,  0.75421722,  0.81392503,
1.2825361 ,  1.48882145,  1.38675446,  0.33056584,  0.21335745,
0.14683961,  0.64504509,  0.63019434,  0.56613274,  0.74820247,
0.71333899,  0.61619546,  0.07963843,  1.31058875,  1.61219212,
1.60755701,  1.45311576,  1.47463772,  1.4965448 ,  1.58811844,
1.74563024,  1.02335638,  0.93359924,  0.88620072,  1.33135528,
1.26372337,  1.6930712 ,  1.56824574,  1.5644695 ,  1.28664689,
1.39342324,  0.97731335,  0.59283548,  0.47171893,  0.45035895,
0.65857503,  0.52476677,  0.51760645,  0.6293517 ,  0.59639758,
0.57505975,  1.25416421,  1.70952544,  1.39940906,  1.32892249,
```

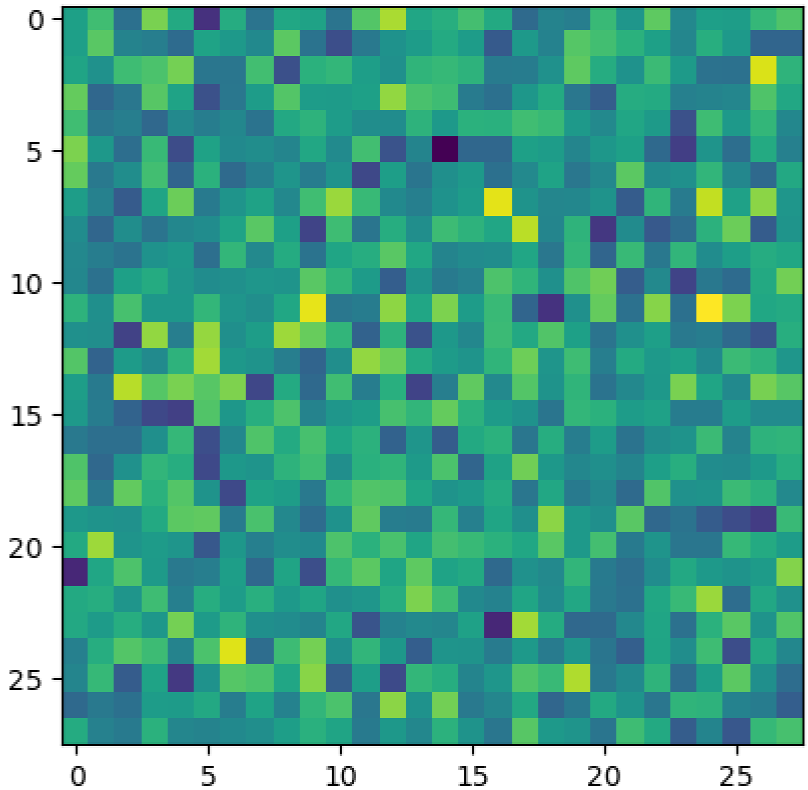
```
       1.46538103,  1.55307087,  1.59780653,  1.66271572,  1.24499921,
       1.53049975,  1.30418381,  1.54324893,  1.59013243,  1.25245885,
       1.48635307,  0.99729451,  1.08297132,  1.12765557,  0.68652973,
       0.77951035,  0.44624082,  0.40428207,  0.64056763,  0.72175819,
       0.38757577,  0.2435911 ,  0.69577388,  0.53197946,  0.50748117,
       0.92806069,  1.03204323,  0.78733256,  1.20431391,  0.96670428,
       1.76335259,  1.56037524,  1.45832627,  1.19755596,  0.90655534,
       1.00699761,  0.40481721,  0.31130171,  0.57530393,  0.38067106,
       0.2374199 ,  0.30717348,  0.59960756,  0.50586556,  0.42062948,
       0.48058779,  0.32149952,  0.47543075,  0.17546164,  0.77975522,
       0.5097491 ,  0.67831857,  0.55793433,  0.36801297,  0.19546406,
       0.24104791,  0.74321123,  0.30950039,  0.36157467,  0.6576656 ,
       0.33552428,  0.36398714,  0.44787139,  0.39748342,  0.45472615,
       0.69077358,  0.44023924,  0.49691089,  0.55107735,  0.60205863,
       0.51575186,  0.40972205,  0.5528749 ,  0.7018578 ,  0.29692798,
       0.56124433,  0.74747962,  0.77443373,  0.48150657,  0.52666454,
       0.97278558,  0.34108848,  0.36858934,  0.30622875,  0.55314357,
       0.4364178 ,  0.65828641,  0.47318976,  0.87802943,  0.50157545,
       0.58784343,  0.14233734,  0.63692678,  0.27199073,  0.33432951,
       0.55719796,  0.01497966,  0.51575917,  0.22319883,  0.18286803,
       0.59889024,  0.53215933,  0.46862034,  0.7253405 ,  0.51431413,
       0.51807683,  0.352295  ,  0.72957085,  0.24179591,  0.3677939 ,
       0.59017383,  0.4299084 ,  0.57120308,  0.54409436,  0.13898291,
       0.30659389,  0.32858998,  0.75806143,  0.25855974,  0.51663139,
       0.63212019,  0.40535639,  0.33615966,  0.74985285,  0.41696861,
       0.42385191,  0.27142034,  0.59889696,  0.53467767,  0.58737025,
       0.8085762 ,  0.43750174,  0.78269892,  0.61612576,  0.78229204,
       0.20155359,  0.49406308,  0.69594044,  0.90704146,  0.48068657,
       0.2593931 ,  0.49739493,  0.55527354,  0.2703061 ,  0.52558612,
       0.65110954,  0.84393377,  0.41588745,  0.68644318,  0.47088899,
       0.63850876,  0.61065792,  0.53675139,  0.99290694,  0.35616339,
       0.5933731 ,  0.32941645,  0.43484905,  0.63120649,  0.62207694,
       0.45278085,  0.26114508,  0.39601245,  0.54658679,  0.45476478,
       0.60711821,  0.55012664,  0.42348607,  0.76234562,  0.44878522,
       0.60790947,  0.3595148 ,  0.23428315,  0.54523711,  0.63661374,
       0.05622986,  0.49892666,  0.32963783,  0.52101281,  0.59030581,
       0.3802501 ,  0.55527802,  0.53023526,  0.73779477])
```

In [9]:

```python
# Compare Noises to Ensure they are Different
# Noise on Image 122
plt.imshow(noise[122].reshape(28, 28))
```

Out[9]:

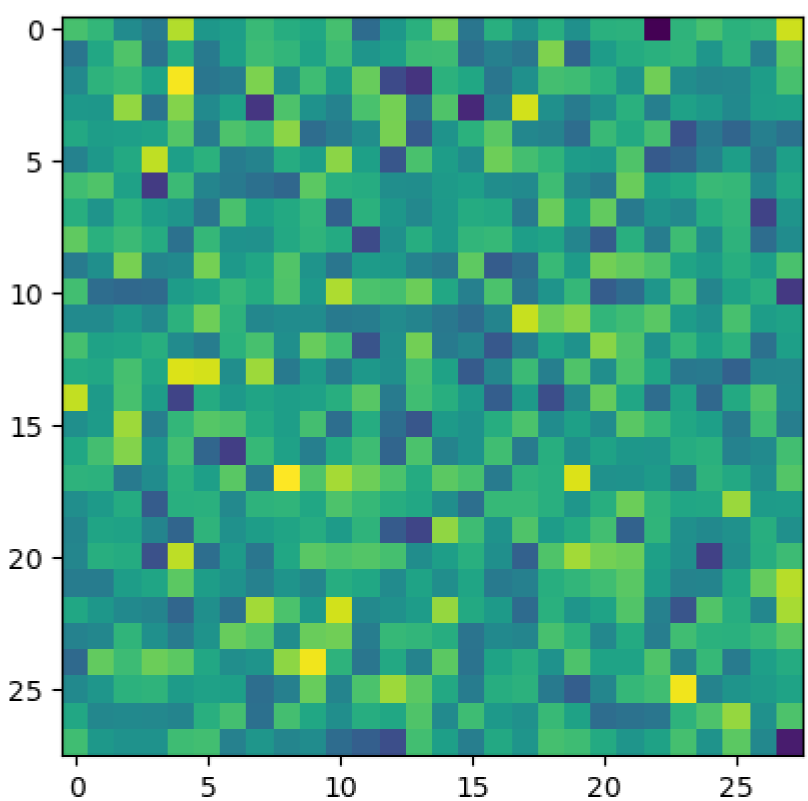
```
<matplotlib.image.AxesImage at 0x27148566200>
```



In [10]:

```
# Noise on Image 124
plt.imshow(noise[124].reshape(28, 28))
```

Out[10]: <matplotlib.image.AxesImage at 0x271485de350>



In [11]:
```
# As the two visual representations (images) of the noise show different patterns, the
# for each image
```

1. Compare the `accuracy` of train and val after N epochs for MLNN with and without noise.

In [12]:
```
%%capture
## Code From Base for Creating Neural Network Without Noise
batch_size = 128
num_classes = 10
epochs = 20

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

In [13]:
```
import pandas as pd
(
np.max(x_train_noise),
np.min(x_train_noise),
np.max(x_train),
np.min(x_train)
)
```

Out[13]:
```
(2.317632454193437, -0.4185654653815156, 1.0, 0.0)
```

In [14]:
```
# Normalization
mean = np.mean(x_train_noise)
std = np.std(x_train_noise)
x_train_noise_norm = (x_train_noise - mean) / std

mean = np.mean(x_test_noise)
std = np.std(x_test_noise)
x_test_noise_norm = (x_test_noise - mean) / std

mean = np.mean(y_train_noise)
```

```
std = np.std(y_train_noise)
y_train_noise_norm = (y_train_noise - mean) / std

mean = np.mean(y_test_noise)
std = np.std(y_test_noise)
y_test_noise_norm = (y_test_noise - mean) / std
```

In [15]:
```
%%capture
## Updated Code for Creating Neural Network With Noise
batch_size = 128
num_classes = 10
# epochs = 20 ## Commenting out; this ensures the same number of epochs is used for bo

# convert class vectors to binary class matrices - using noise added images
y_train_bcm = keras.utils.to_categorical(y_train_noise_norm, num_classes)
y_test_bcm = keras.utils.to_categorical(y_test_noise_norm, num_classes)

noisy_model = Sequential()
noisy_model.add(Dense(512, activation='relu', input_shape=(784,)))
noisy_model.add(Dropout(0.2))
noisy_model.add(Dense(512, activation='relu'))
noisy_model.add(Dropout(0.2))
noisy_model.add(Dense(10, activation='softmax'))

noisy_model.summary()

noisy_model.compile(loss='categorical_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])

history = noisy_model.fit(x_train_noise_norm, y_train_bcm,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test_noise_norm, y_test_bcm))
noisy_score = noisy_model.evaluate(x_test_noise_norm, y_test_bcm, verbose=0)
print('Test loss:', noisy_score[0])
print('Test accuracy:', noisy_score[1])
```

In [16]:
```
print('Test loss:', score[0]) # Test loss & accuracy statistics for the model trained
print('Test accuracy:', score[1])
print('Test loss:', noisy_score[0]) # The same statistics for the model trained with n
print('Test accuracy:', noisy_score[1])
```

```
Test loss: 0.0920642837882042
Test accuracy: 0.9800999760627747
Test loss: 0.1428346335887909
Test accuracy: 0.9804999828338623
```

In [17]:
```
# Comparing the without-noise and with-noise models, we find that the noise reduces ac
# a training series lasting n=20 epochs. Additionally, the test loss statistic is noic
```

1. Vary the amount of noise by changing the `scale` parameter in `np.random.normal` by a factor. Use `.1, .5, 1.0, 2.0, 4.0` for the `scale` and keep track of the `accuracy` for training and validation and plot these results.

In [18]:
```
%%capture
## Combining Noise Code with Model-Running code for a single, repeatable block:
# Listing Scale (i.e. Standard Deviation) Values over which to train models
stddev_values = [0.1, 0.5, 1.0, 2.0, 4.0]  # Values given
# Initializing Lists
loss_values = []
accuracy_values = []

# Loop:
for stddev in stddev_values:

    # Generate random noise with the same shape as the image
    mean = .5
    stddev = .16

    # Generate random noise with the same shape as the image
    x_train_noise = np.random.normal(mean, stddev, x_train.shape)
    x_test_noise = np.random.normal(mean, stddev, x_test.shape)
    y_train_noise = np.random.normal(mean, stddev, y_train.shape)
    y_test_noise = np.random.normal(mean, stddev, y_test.shape)

    # Add noise
    x_train_noise = x_train + x_train_noise
    x_test_noise = x_test + x_test_noise
    y_train_noise = y_train + y_train_noise
    y_test_noise = y_test + y_test_noise

    ## Updated Code for Creating Neural Network With Noise
    batch_size = 128
    num_classes = 10
    epochs = 20 # Always using same number of epochs

    # convert class vectors to binary class matrices
    y_train_bcm = keras.utils.to_categorical(y_train_noise_norm, num_classes)
    y_test_bcm = keras.utils.to_categorical(y_test_noise_norm, num_classes)

    noisy_model = Sequential()
    noisy_model.add(Dense(512, activation='relu', input_shape=(784,)))
    noisy_model.add(Dropout(0.2))
    noisy_model.add(Dense(512, activation='relu'))
    noisy_model.add(Dropout(0.2))
    noisy_model.add(Dense(10, activation='softmax'))

    noisy_model.summary()

    noisy_model.compile(loss='categorical_crossentropy',
                  optimizer="adam",
                  metrics=['accuracy'])

    history = noisy_model.fit(x_train_noise_norm, y_train_bcm,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test_noise_norm, y_test_bcm))
    noisy_score = noisy_model.evaluate(x_test_noise_norm, y_test_bcm, verbose=0)
    print('Test loss:', noisy_score[0])
    print('Test accuracy:', noisy_score[1])
    # Append Loss & Accuracy To List
    test_loss = noisy_score[0]
```

```
    test_accuracy = noisy_score[1]
    loss_values.append(test_loss)
    accuracy_values.append(test_accuracy)
```

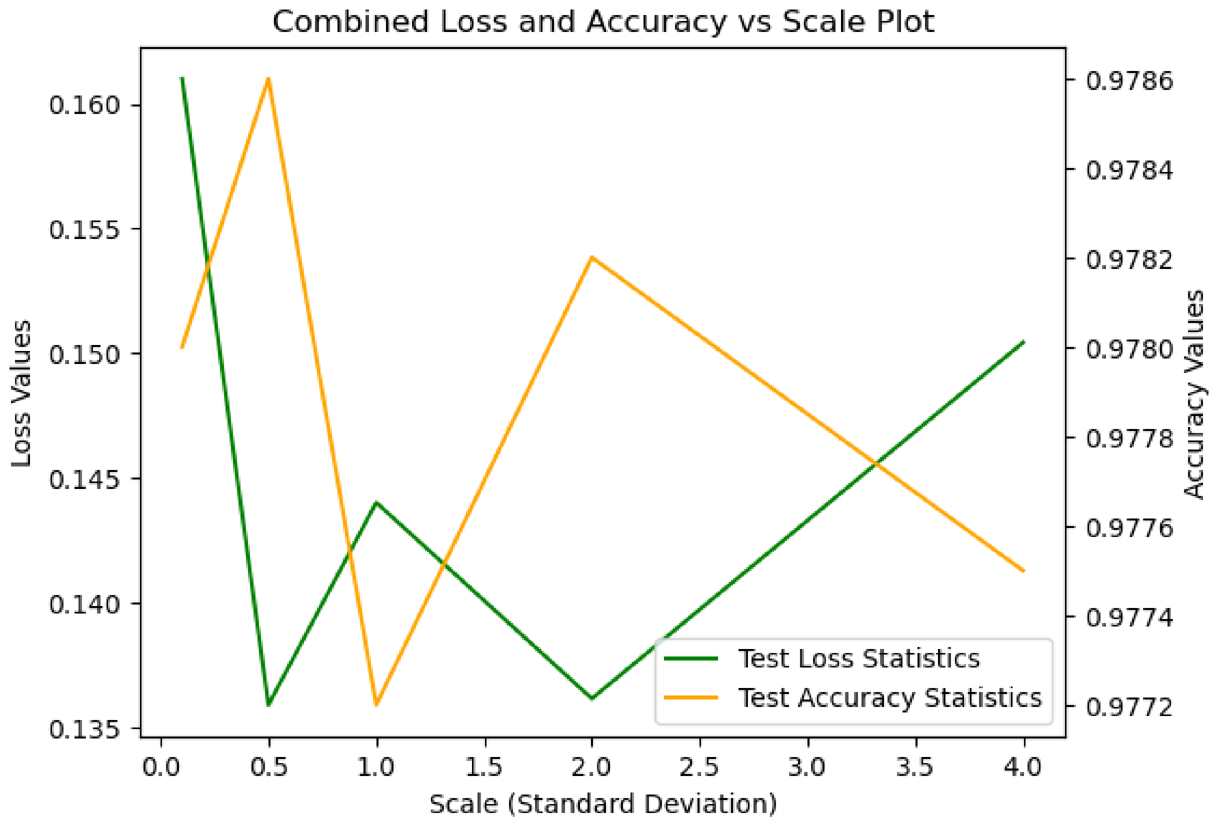
In [19]:
```
# Compile Loss and Accuracy statistics for the models into a single data frame
accuracy_loss_results_dataframe = pd.DataFrame({'stddev_values': stddev_values, 'loss_
accuracy_loss_results_dataframe
```

Out[19]:

	stddev_values	loss_values	accuracy_values
0	0.1	0.161019	0.9780
1	0.5	0.135908	0.9786
2	1.0	0.144020	0.9772
3	2.0	0.136160	0.9782
4	4.0	0.150436	0.9775

In [20]:
```
# Plotting Results
fig, ax1 = plt.subplots()
loss_line = ax1.plot(stddev_values, loss_values, color = 'green', label = 'Test Loss S

# Loss Line Plot
ax1.set_xlabel('Scale (Standard Deviation)')
ax1.set_ylabel('Loss Values')
ax1.set_title('Combined Loss and Accuracy vs Scale Plot')

# Accuracy Line Plot
ax2 = ax1.twinx()
accuracy_line = ax2.plot(stddev_values, accuracy_values, color='orange', label='Test A
ax2.set_ylabel('Accuracy Values')

# Combine Both Lines Into One Plot
lines = loss_line + accuracy_line
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='lower right')
```

Out[20]:
```
<matplotlib.legend.Legend at 0x2719c61b880>
```


Combined Loss and Accuracy vs Scale Plot
Loss Values
Accuracy Values
Scale (Standard Deviation)
Test Loss Statistics
Test Accuracy Statistics
0.160
0.155
0.150
0.145
0.140
0.135
0.9786
0.9784
0.9782
0.9780
0.9778
0.9776
0.9774
0.9772
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0