Neural Networks image recognition - ConvNet

1. Add random noise (see below on `size parameter` on `np.random.normal` (https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html)) 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. **
2. Compare the `accuracy` of train and val after N epochs for MLNN with and without noise.
3. 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.
4. Compare these results with the previous week where we used a MultiLayer Perceptron (this week we use a ConvNet).

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
```

```
2023-07-25 09:46:48.944877: I tensorflow/core/platform/cpu_feature_gu
ard.cc:182] This TensorFlow binary is optimized to use available CPU
instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

In [5]:
```
import numpy as np
import matplotlib.pyplot as  plt
```

Conv Net

Trains a simple convnet on the MNIST dataset. Gets to 99.25% test accuracy after 12 epochs (there is still a lot of margin for parameter tuning).

In [35]:
```
# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
```

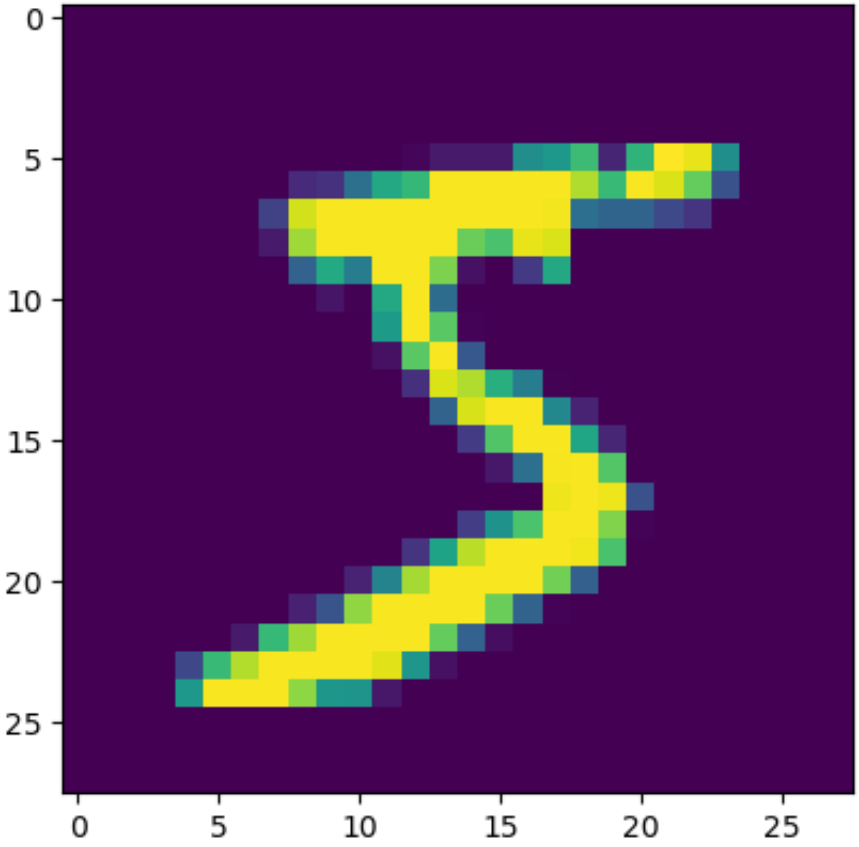
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [36]:
```
# Noise Add

x_train_noise = x_train + np.random.normal(scale = 0.1, size = (60000,

np.shape(x_train_noise)
```

Out[36]:
```
(60000, 28, 28, 1)
```

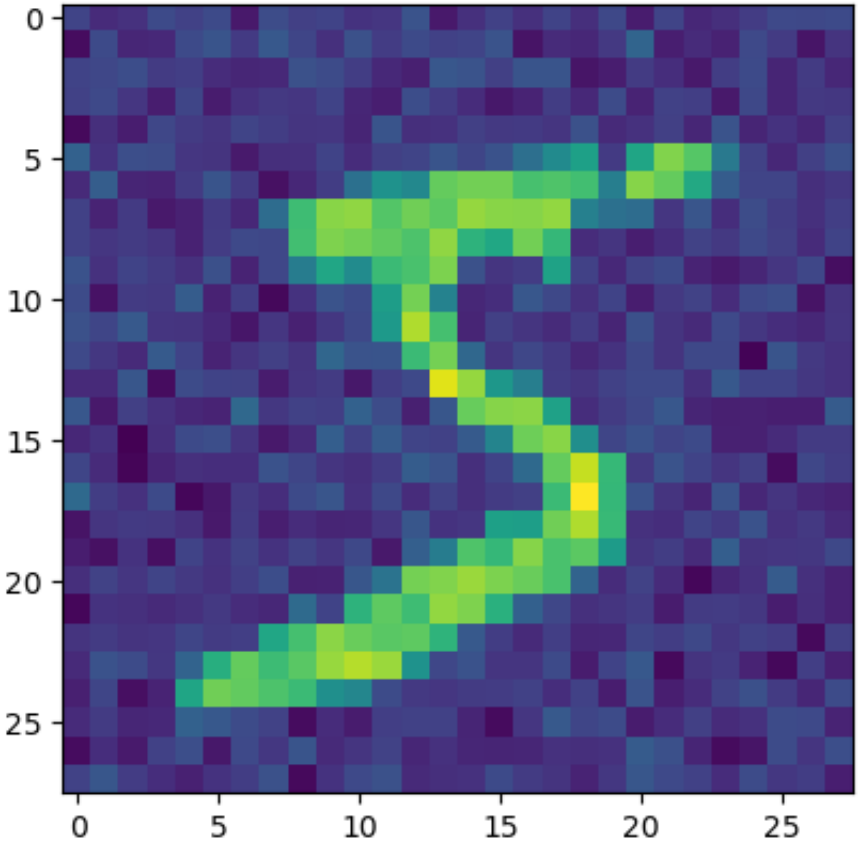
In [37]:

Out[37]: <matplotlib.image.AxesImage at 0x7fd7b427e2c0>



In [38]:

Out[38]: <matplotlib.image.AxesImage at 0x7fd6ff0ffe80>

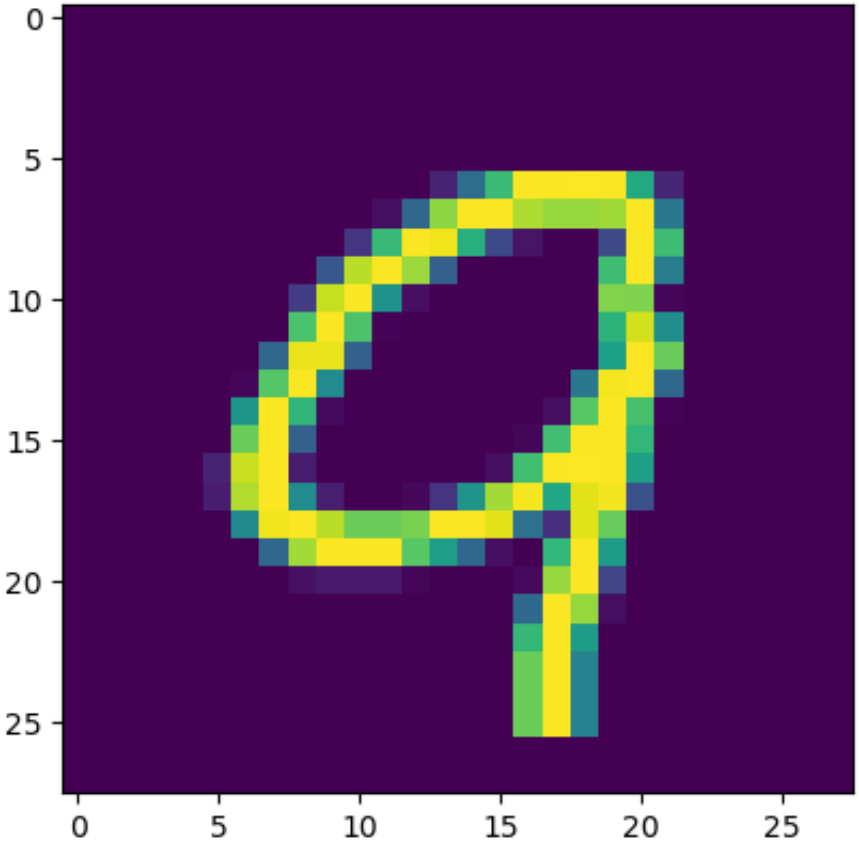


In [42]:
```
x_train_noise = x_train + np.random.normal(scale = 0.5, size = (60000,
```

Out[42]: (60000, 28, 28, 1)

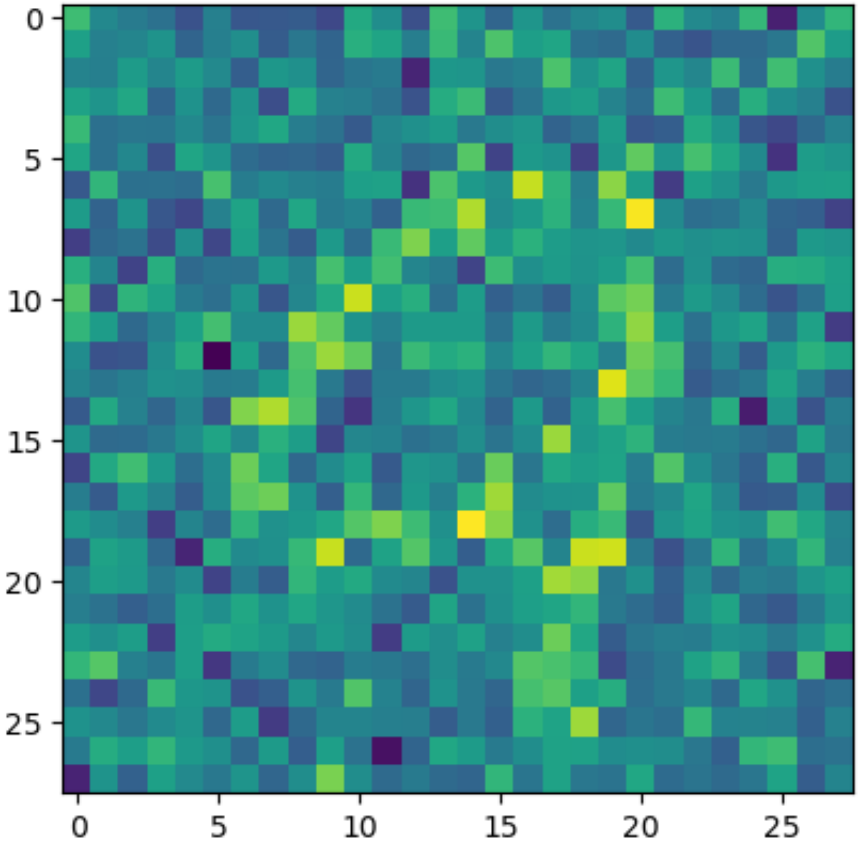
In [43]:

Out[43]: <matplotlib.image.AxesImage at 0x7fd7f3e2a3b0>



In [44]:

Out[44]: <matplotlib.image.AxesImage at 0x7fd7970a3df0>

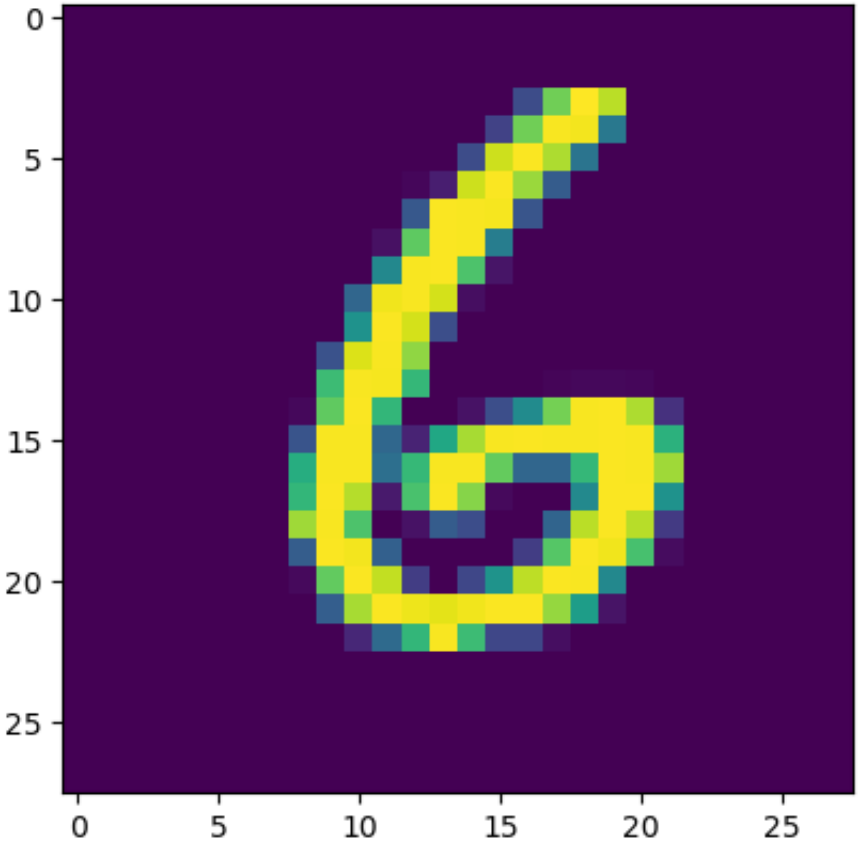


In [45]:
```
x_train_noise = x_train + np.random.normal(scale = 1.0, size = (60000,
```

Out[45]: (60000, 28, 28, 1)

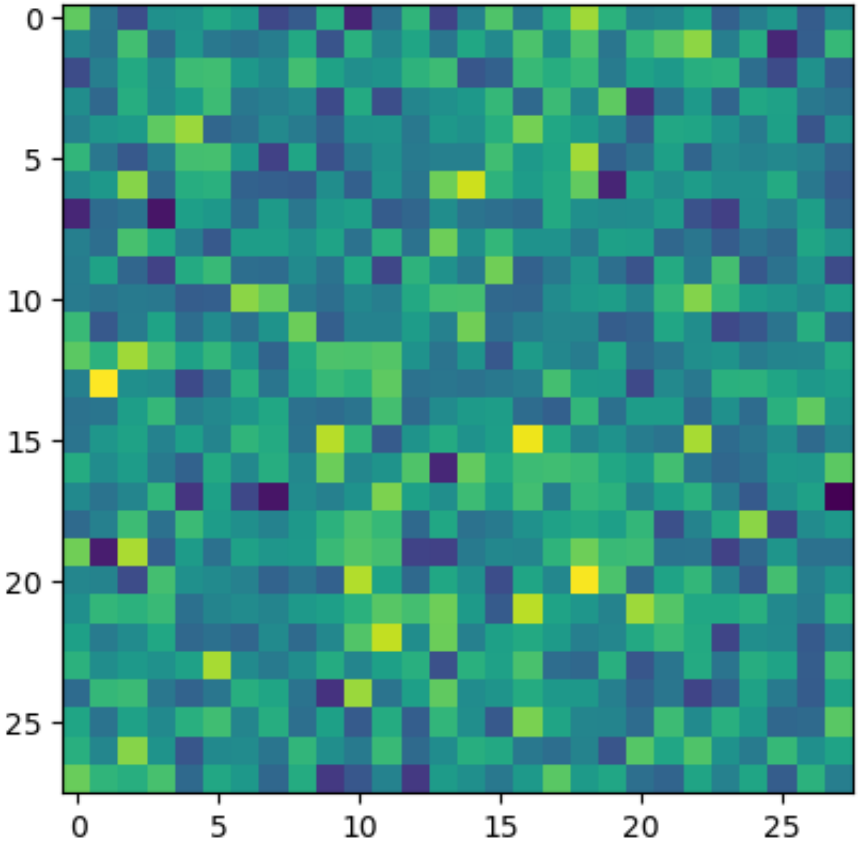
In [46]:

Out[46]: <matplotlib.image.AxesImage at 0x7fd75e470c40>



In [47]:

Out[47]: <matplotlib.image.AxesImage at 0x7fd6ae6be650>

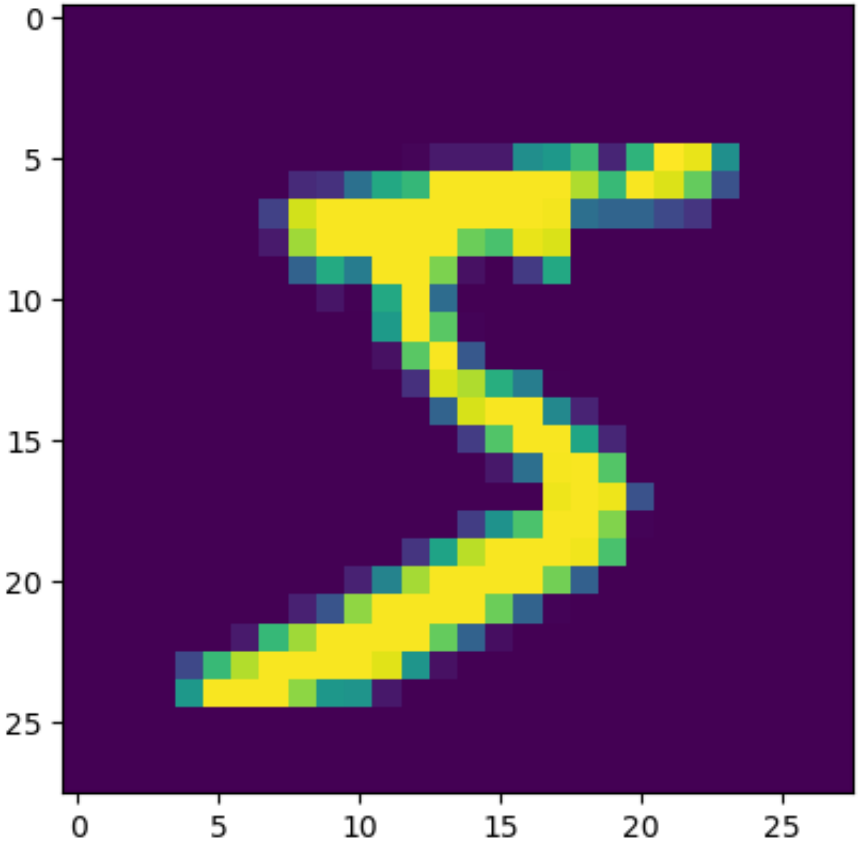


In [21]:
```
# Noise Multiply

x_train_noise = x_train * np.random.normal(scale = 0.1, size = (60000,

np.shape(x_train_noise)
```

Out[21]: (60000, 28, 28, 1)

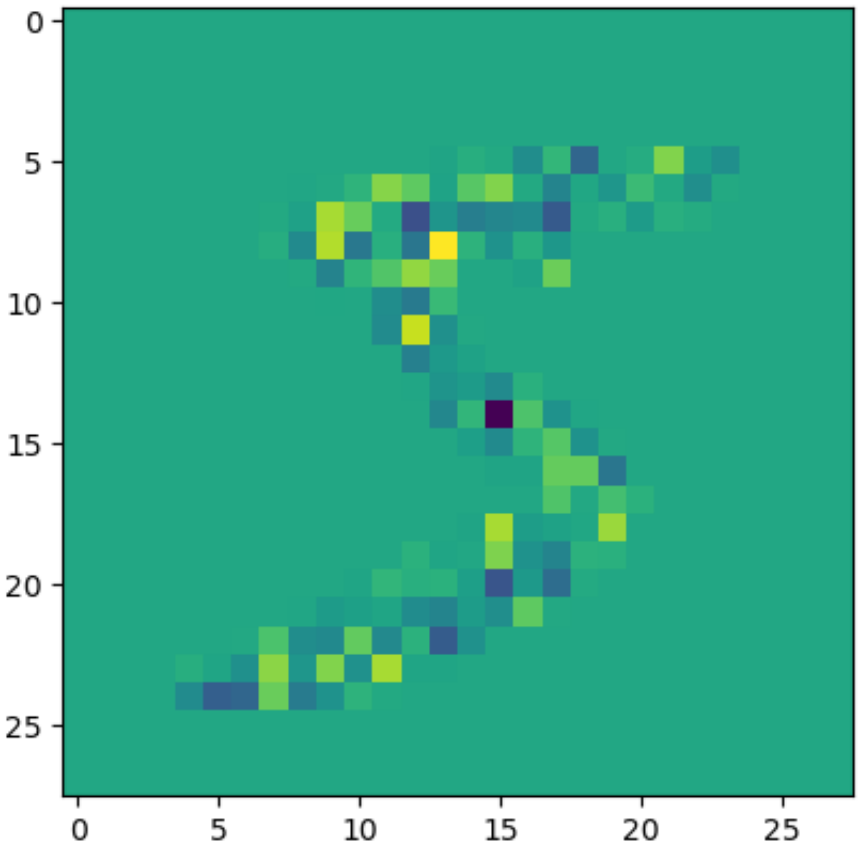
In [22]:

Out[22]: <matplotlib.image.AxesImage at 0x7fd7d5144760>



In [23]:

Out[23]: <matplotlib.image.AxesImage at 0x7fd7d51b1b40>

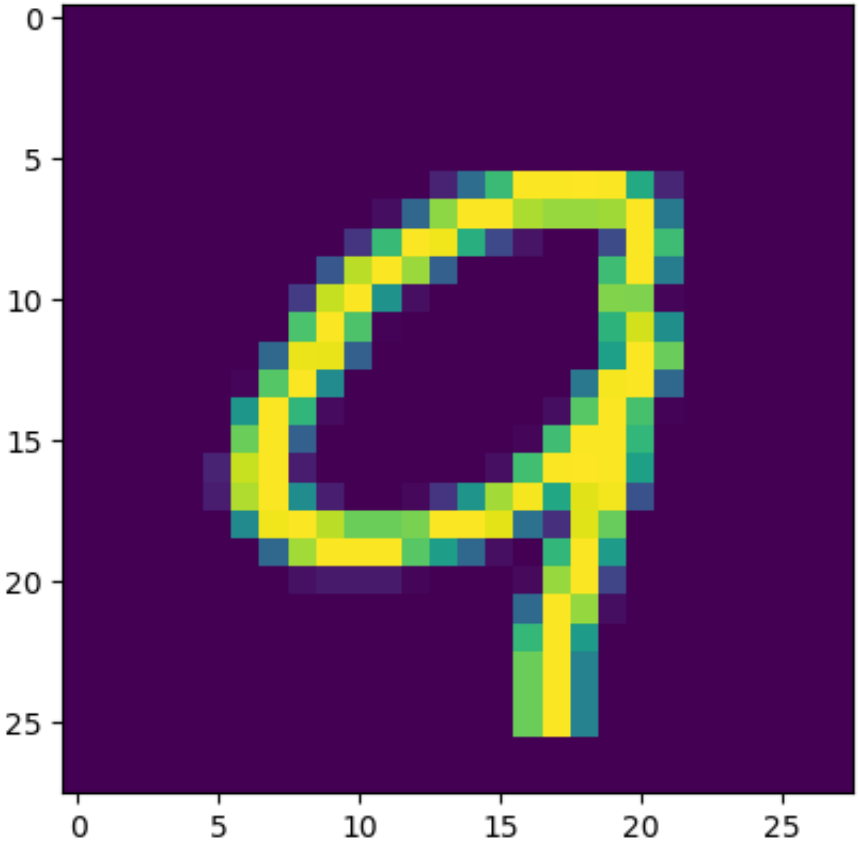


In [24]:
```
x_train_noise = x_train * np.random.normal(scale = 0.5, size = (60000,
```

Out[24]: (60000, 28, 28, 1)

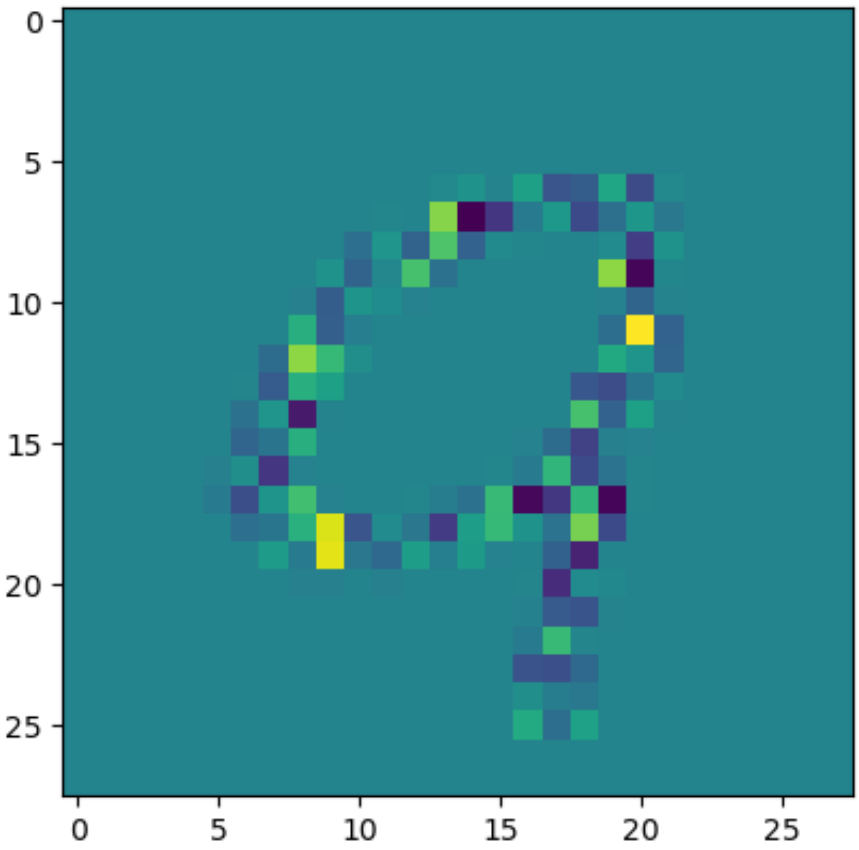
In [25]:

Out[25]: <matplotlib.image.AxesImage at 0x7fd7bdb329e0>



In [26]:

Out[26]: <matplotlib.image.AxesImage at 0x7fd7bdba3640>

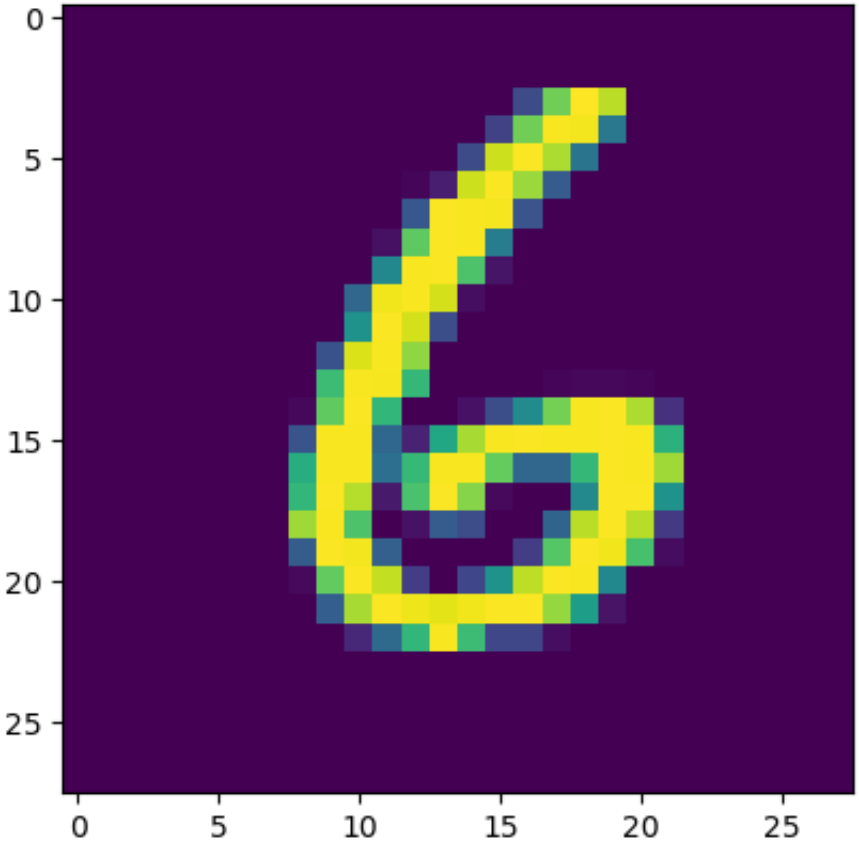


In [27]:
```
x_train_noise = x_train * np.random.normal(scale = 1.0, size = (60000,
```

Out[27]: (60000, 28, 28, 1)

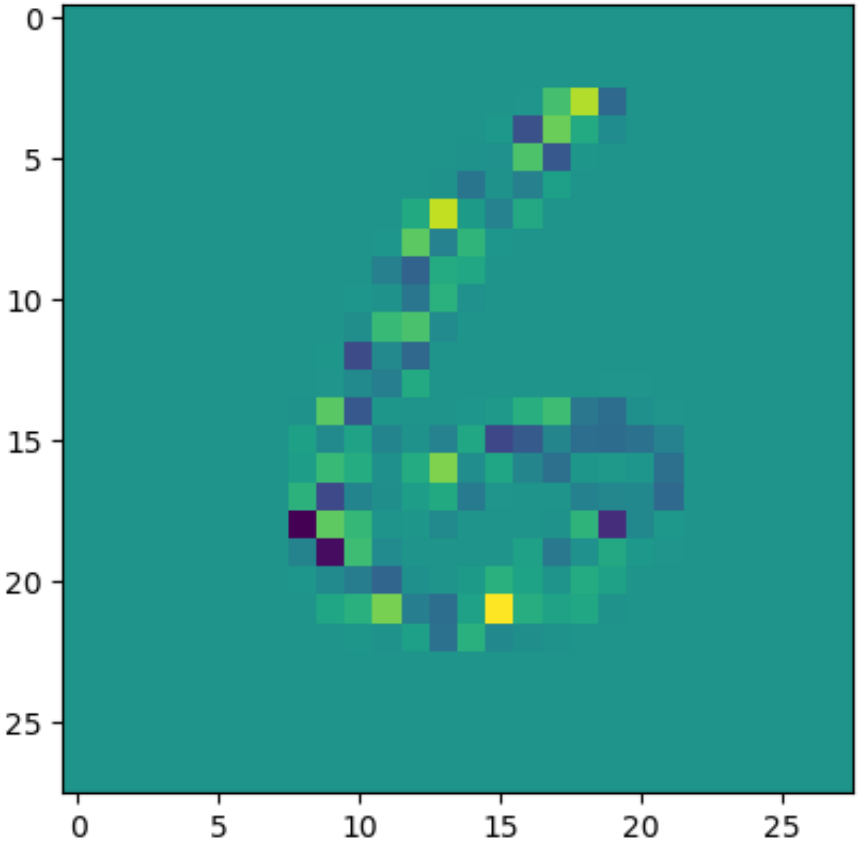
In [28]:

Out[28]: <matplotlib.image.AxesImage at 0x7fd7b41dc400>



In [29]:

Out[29]: <matplotlib.image.AxesImage at 0x7fd7b4250cd0>



In [ ]:
```
batch_size = 128
num_classes = 10
epochs = 12

# 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(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

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])
```

```
Epoch 1/12
141/469 [========>.....................] - ETA: 19s - loss: 2.3113 -
accuracy: 0.1060
```

In [31]:
```
# For Loop of Scales with addition of noise

np.random.seed(7) # Set for reproducibility

scales = [0.1,0.5,1.0,2.0,4.0]
batch_size = 128
num_classes = 10
epochs = 12
scores = []
plt.figure(figsize=(12, 8))

# 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)
```

```
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

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

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0, # processes faster when not set to verb
                    validation_data=(x_test, y_test))

plt.plot(history.history['accuracy'], label='No Noise')

model.summary()

score_baseline = model.evaluate(x_test, y_test, verbose=0)

print(score_baseline)

for scale in scales:

    x_train_noise = x_train + np.random.normal(scale = scale, size = (

    x_test_noise = x_test + np.random.normal(scale = scale, size = (10

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

    history = model.fit(x_train_noise, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0,
                    validation_data=(x_test_noise, y_test))

    plt.plot(history.history['accuracy'], label=f'{scale} noise scale'

    model.summary()
```

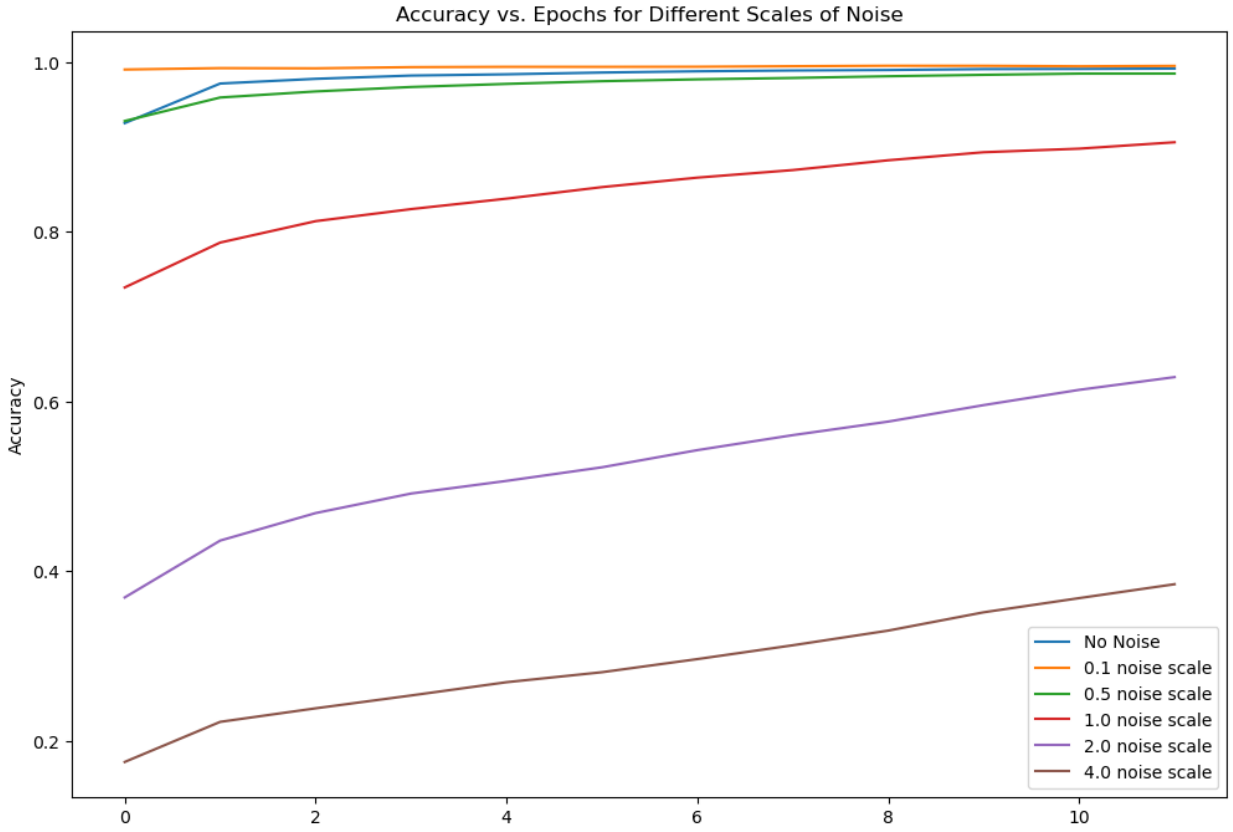
```
    score = model.evaluate(x_test_noise, y_test, verbose=0)

    print(score)

    scores.append(score)

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Epochs for Different Scales of Noise')
plt.legend()
plt.show()
printout = [item for sublist in zip(scales, scores) for item in sublis
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0


 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.028407897800207138, 0.9918000102043152]
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
```

```
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.03660549968481064, 0.9919999837875366]
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.11334235966205597, 0.9729999899864197]
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
```

```
 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.5325009226799011, 0.8461999893188477]
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[1.5234874486923218, 0.5058000087738037]
Model: "sequential"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[2.1445717811584473, 0.2538999915122986]
```



Epochs

```
Noise Scale, Loss, Accuracy: No Noise [0.028407897800207138, 0.991800
0102043152] [0.1, [0.03660549968481064, 0.9919999837875366], 0.5, [0.
11334235966205597, 0.9729999899864197], 1.0, [0.5325009226799011, 0.8
461999893188477], 2.0, [1.5234874486923218, 0.5058000087738037], 4.0,
[2.1445717811584473, 0.2538999915122986]]
```

The model without noise is able to achieve a 99.2% accuracy with a 0.028 loss. For the added noise we can see that with a scale of 0.1 the accuracy is still 99.2% with a loss of 0.037. Noise scale of 0.5 has an accuracy of 97.3% and a loss of 0.113. Noise scale 1.0 has an accuracy of 84.6% and a loss of 0.532. Noise scale 2.0 has an accuracy of 50.6% and a loss of 1.523. Finally the noise scale of 4.0 has an accuracy of 25.4% and a loss of 2.145. When compared to the Multi-Layer Perceptron models from the last assignment we can see that the ConvNet model is overall less susceptiable noise the added noise and still declines in performance as the scale of the noise increases. Generally we can see a 1-5% improvement in model accuracy performance over the Perceptron model with noise introduced. From the human eye perspective the ConvNet performance is shocking given how quickly the images degrade in to visual noise when it is added. It is additionally impressive that the Convolution Net model is able to achieve superior accuracy in only 12 epochs versus the Perceptron's 20. I suspect given equal epochs the ConvNet could achieve even higher accuracy.

In [34]:
```
# For Loop of Scales with multiplication of noise.

np.random.seed(7) # Set for reproducibility

scales = [0.1,0.5,1.0,2.0,4.0]
batch_size = 128
num_classes = 10
epochs = 12
scores = []
plt.figure(figsize=(12, 8))

# 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(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

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

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0, # processes faster when not set to verb
                    validation_data=(x_test, y_test))

plt.plot(history.history['accuracy'], label='No Noise')

model.summary()

score_baseline = model.evaluate(x_test, y_test, verbose=0)

print(score_baseline)

for scale in scales:

    x_train_noise = x_train * np.random.normal(scale = scale, size = (

    x_test_noise = x_test * np.random.normal(scale = scale, size = (10

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

    history = model.fit(x_train_noise, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0,
                    validation_data=(x_test_noise, y_test))

    plt.plot(history.history['accuracy'], label=f'{scale} noise scale'

    model.summary()

    score = model.evaluate(x_test_noise, y_test, verbose=0)

    print(score)

    scores.append(score)

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Epochs for Different Scales of Noise')
plt.legend()
```

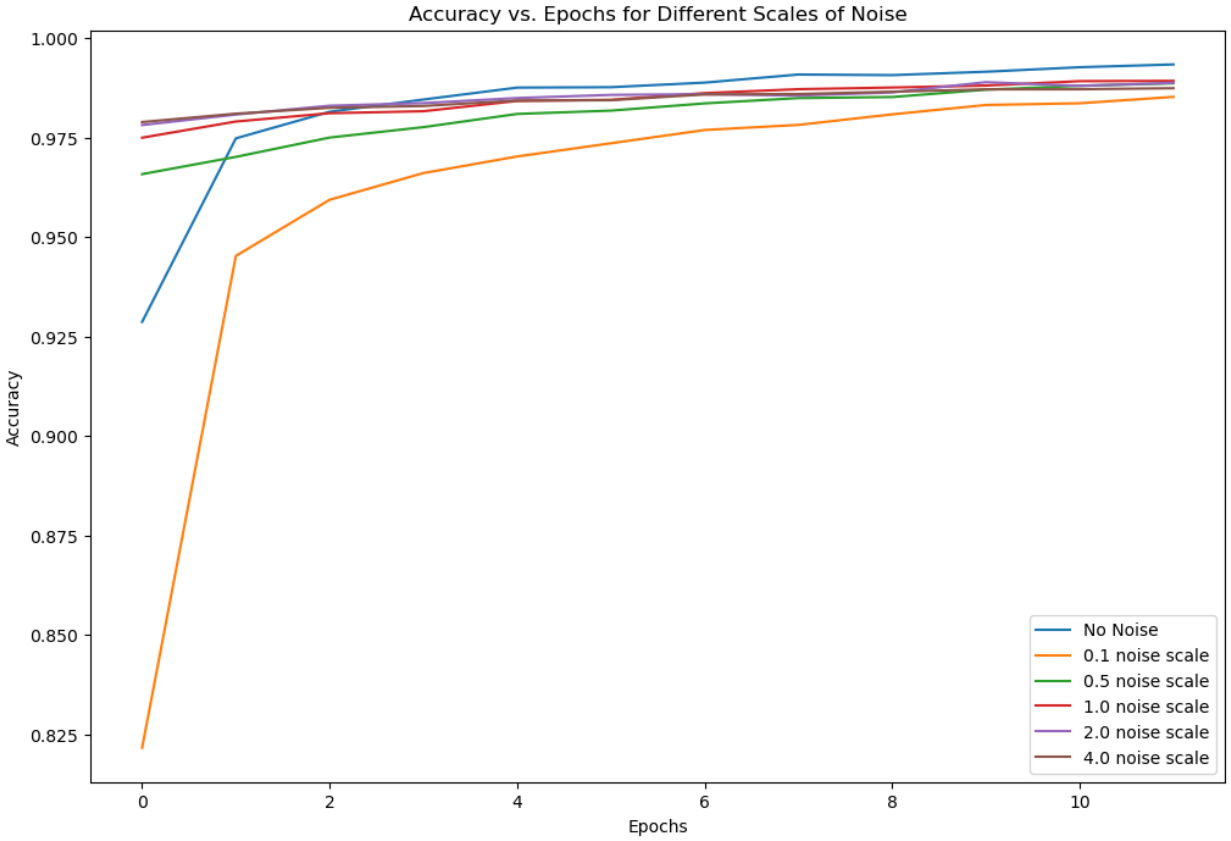
```
plt.legend()
plt.show()
printout = [item for sublist in zip(scales, scores) for item in sublis
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.02853526547551155, 0.9926000237464905]
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290
```

```
=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.06564854085445404, 0.9815000295639038]
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.06569663435220718, 0.9829999804496765]
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776
```

```
 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.07121571153402328, 0.9842000007629395]
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.07189298421144485, 0.9846000075340271]
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #

=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)
```

```
 dropout_4 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_4 (Dense)             (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
[0.06756246089935303, 0.9861000180244446]
```




```
Noise Scale, Loss, Accuracy: No Noise [0.02853526547551155, 0.9926000
237464905] [0.1, [0.06564854085445404, 0.9815000295639038], 0.5, [0.0
6569663435220718, 0.9829999804496765], 1.0, [0.07121571153402328, 0.9
842000007629395], 2.0, [0.07189298421144485, 0.9846000075340271], 4.0
, [0.06756246089935303, 0.9861000180244446]]
```

I didn't think to try multiplying the noise in to the data until the image printouts were so rough in the ConvNet shape. I went back and tried the Multi-Layer Perceptron model with the multiplied noise and got similar results to the ConvNet. Indeed the comparison between the Perceptron and the ConvNet with multiplied noise mirror the results of the added noise. The ConvNet performs ~7-10% better than the Perceptron for accuracy once noise is introduced. The additional characteristic I noted is that both models start off with worse performance at the lowest scale of noise multiplied and then improve as the noise scale gets larger. Finally both models perform nearly indentically regardless of noise scaling when the noise is multiplied.