Neural Networks image recognition - ConvNet

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

In [1]:
```
#setup
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
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# 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')
print(x_test.shape[0], 'test samples')
```

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

In [11]:
```
# model setup
batch_size = 128
num_classes = 10
epochs = 20
```

In [4]:
```
#build model

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
```

```python
                   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'])
```

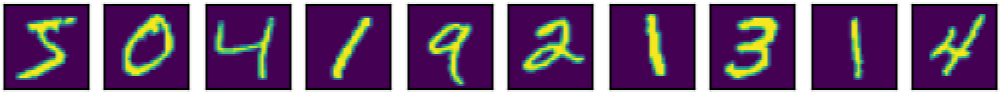
```
C:\Users\rober\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

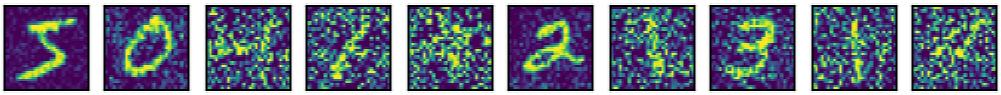
In [5]:

```python
#add noise
np.random.seed(15)
noise_feat = np.linspace(.1, 1.0, 10)
np.random.shuffle(noise_feat)
feat_img = {i: noise_feat[i] for i in range(10)}
x_train_noise = []
x_test_noise = []
#noise for train
for i in range(len(x_train)):
    class_lab = y_train[i]
    noise_scale = feat_img[class_lab]
    noise = np.random.normal(loc=0.0, scale=noise_scale, size=x_train[i].shape)
    img = np.clip(x_train[i] + noise, 0., 1.)
    x_train_noise.append(img)
#noise for test
for i in range(len(x_test)):
    class_lab = y_test[i]
    noise_scale = feat_img[class_lab]
    noise = np.random.normal(loc=0.0, scale=noise_scale, size=x_test[i].shape)
    img = np.clip(x_test[i] + noise, 0., 1.)
    x_test_noise.append(img)

x_train_noise = np.array(x_train_noise)
x_test_noise = np.array(x_test_noise)
```

In [6]:

```python
#compare plots with noise
plt.figure()
n = 10
for i in range(n):
    ax = plt.subplot(2, n, i+1)
    plt.imshow(x_train[i].reshape(28, 28))
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax = plt.subplot(2, n, i+n+1)
    plt.imshow(x_train_noise[i].reshape(28, 28))
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```





Train Data (no noise)

In [7]:

```
# 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)
```

In [12]:

```
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])
```

```
Epoch 1/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - accuracy: 0.6856 - loss: 0.9350 - val_accuracy: 0.84
85 - val_loss: 0.5339
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 15s 32ms/step - accuracy: 0.7605 - loss: 0.7331 - val_accuracy: 0.86
28 - val_loss: 0.5015
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 17s 36ms/step - accuracy: 0.7806 - loss: 0.6895 - val_accuracy: 0.86
97 - val_loss: 0.4802
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 17s 37ms/step - accuracy: 0.7919 - loss: 0.6595 - val_accuracy: 0.87
39 - val_loss: 0.4628
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 18s 39ms/step - accuracy: 0.7937 - loss: 0.6497 - val_accuracy: 0.87
74 - val_loss: 0.4472
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8011 - loss: 0.6303 - val_accuracy: 0.88
06 - val_loss: 0.4335
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8056 - loss: 0.6195 - val_accuracy: 0.88
44 - val_loss: 0.4219
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8110 - loss: 0.6032 - val_accuracy: 0.88
63 - val_loss: 0.4112
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8149 - loss: 0.5965 - val_accuracy: 0.88
95 - val_loss: 0.4010
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8223 - loss: 0.5745 - val_accuracy: 0.89
23 - val_loss: 0.3922
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8274 - loss: 0.5636 - val_accuracy: 0.89
38 - val_loss: 0.3841
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8260 - loss: 0.5556 - val_accuracy: 0.89
57 - val_loss: 0.3767
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8285 - loss: 0.5461 - val_accuracy: 0.89
83 - val_loss: 0.3698
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8309 - loss: 0.5480 - val_accuracy: 0.89
89 - val_loss: 0.3639
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8357 - loss: 0.5312 - val_accuracy: 0.90
14 - val_loss: 0.3578
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8387 - loss: 0.5178 - val_accuracy: 0.90
20 - val_loss: 0.3524
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8394 - loss: 0.5144 - val_accuracy: 0.90
36 - val_loss: 0.3473
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8450 - loss: 0.5102 - val_accuracy: 0.90
43 - val_loss: 0.3422
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8441 - loss: 0.5036 - val_accuracy: 0.90
61 - val_loss: 0.3373
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.8488 - loss: 0.4923 - val_accuracy: 0.90
65 - val_loss: 0.3328
Test loss: 0.33282503485679626
Test accuracy: 0.906499981880188
```

Train Data (w/ noise)

In [13]:

```
history2 = model.fit(x_train_noise, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test_noise, y_test))
score2 = model.evaluate(x_test_noise, y_test, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Epoch 1/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - accuracy: 0.7086 - loss: 0.8805 - val_accuracy: 0.85
53 - val_loss: 0.5371
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - accuracy: 0.7408 - loss: 0.7812 - val_accuracy: 0.86
04 - val_loss: 0.5065
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 33ms/step - accuracy: 0.7486 - loss: 0.7524 - val_accuracy: 0.86
36 - val_loss: 0.4906
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 34ms/step - accuracy: 0.7550 - loss: 0.7344 - val_accuracy: 0.86
61 - val_loss: 0.4807
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 35ms/step - accuracy: 0.7563 - loss: 0.7295 - val_accuracy: 0.86
82 - val_loss: 0.4679
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 17s 36ms/step - accuracy: 0.7606 - loss: 0.7153 - val_accuracy: 0.87
16 - val_loss: 0.4597
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 17s 37ms/step - accuracy: 0.7671 - loss: 0.6984 - val_accuracy: 0.87
23 - val_loss: 0.4517
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 18s 38ms/step - accuracy: 0.7663 - loss: 0.6945 - val_accuracy: 0.87
34 - val_loss: 0.4445
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7720 - loss: 0.6848 - val_accuracy: 0.87
55 - val_loss: 0.4371
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7731 - loss: 0.6759 - val_accuracy: 0.87
72 - val_loss: 0.4305
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7778 - loss: 0.6616 - val_accuracy: 0.87
71 - val_loss: 0.4246
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7794 - loss: 0.6557 - val_accuracy: 0.87
97 - val_loss: 0.4181
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7812 - loss: 0.6465 - val_accuracy: 0.88
06 - val_loss: 0.4124
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7848 - loss: 0.6389 - val_accuracy: 0.88
21 - val_loss: 0.4069
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7846 - loss: 0.6327 - val_accuracy: 0.88
31 - val_loss: 0.4028
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7904 - loss: 0.6241 - val_accuracy: 0.88
24 - val_loss: 0.3983
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7903 - loss: 0.6150 - val_accuracy: 0.88
42 - val_loss: 0.3929
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7932 - loss: 0.6171 - val_accuracy: 0.88
59 - val_loss: 0.3895
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7969 - loss: 0.6100 - val_accuracy: 0.88
58 - val_loss: 0.3860
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 19s 40ms/step - accuracy: 0.7943 - loss: 0.6033 - val_accuracy: 0.88
65 - val_loss: 0.3820
Test loss: 0.3820398151874542
Test accuracy: 0.8865000009536743
```

Comparison

In [14]:
```
print('Test loss:', score[0], 'Test loss with Noise:', score2[0])
print('Test accuracy:', score[1], 'Test accuracy with Noise:', score2[1])
```

```
Test loss: 0.33282503485679626 Test loss with Noise: 0.3820398151874542
Test accuracy: 0.906499981880188 Test accuracy with Noise: 0.8865000009536743
```

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

In [15]:
```
##Build Models

def noise_test(df, scale):
    noise = np.random.normal(loc=0.0, scale=scale, size=df.shape)
    df_noise = df + noise
    return np.clip(df_noise, 0., 1.)

def conv():
    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'])
    return model

def eval(x_train, y_train, x_test, y_test, scale):
    x_train_noise = noise_test(x_train, scale)
    t_test_noise = noise_test(x_test, scale)

    model = conv()
    history = model.fit(x_train_noise, y_train,
                        epochs = 20,
                        batch_size = 128,
                        validation_data=(x_test_noise, y_test),
                        verbose = 0)
    return history

#Citation: OpenAI (2024). ChatGPT. Retrieved from: openai.chatgpt.com
```

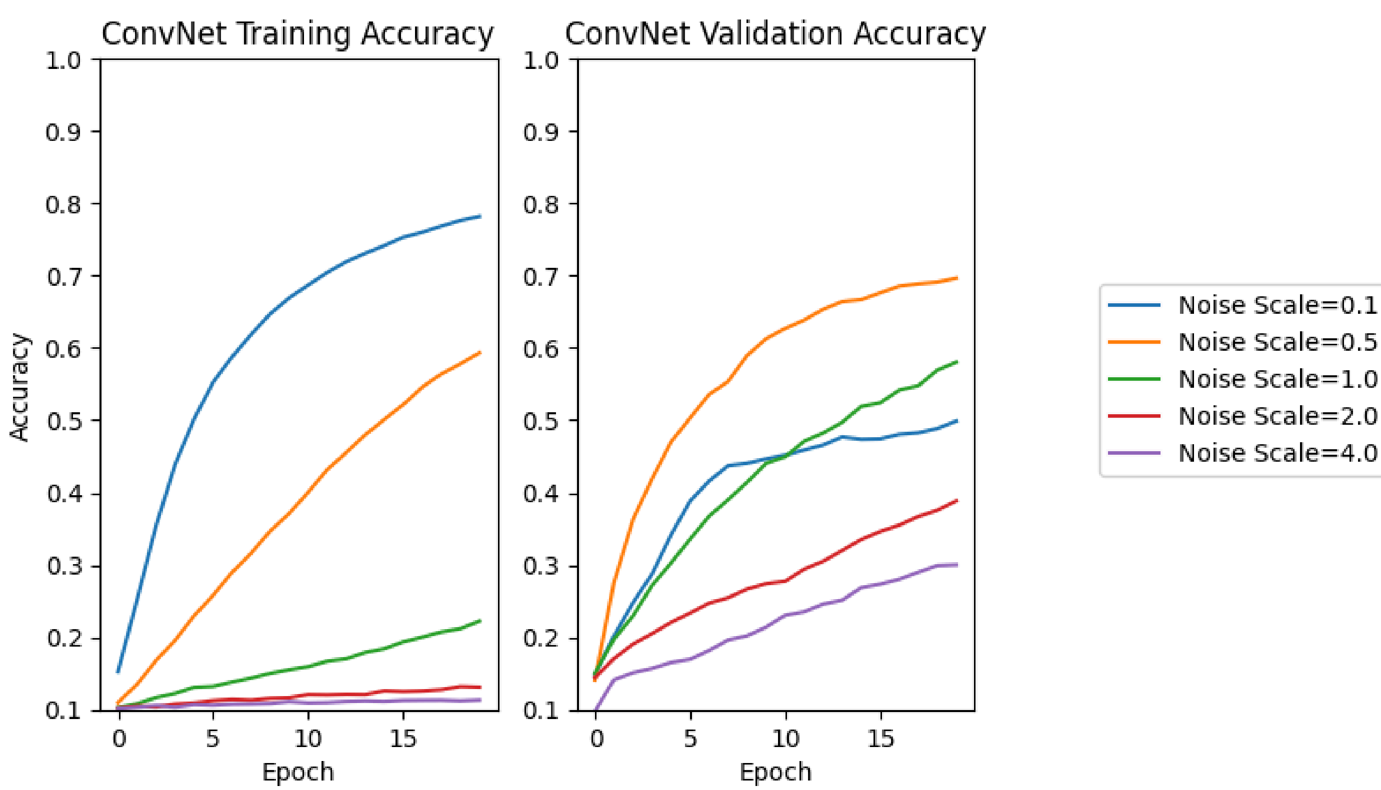
In [16]:
```
#test
noise_scale = [0.1, 0.5, 1.0, 2.0, 4.0]
histories = {}

for scale in noise_scale:
```

```
    print(f'Training Noise Scale: {scale}')
    histories[scale] = eval(x_train, y_train, x_test, y_test, scale)
```

```
Training Noise Scale: 0.1
Training Noise Scale: 0.5
Training Noise Scale: 1.0
Training Noise Scale: 2.0
Training Noise Scale: 4.0
```

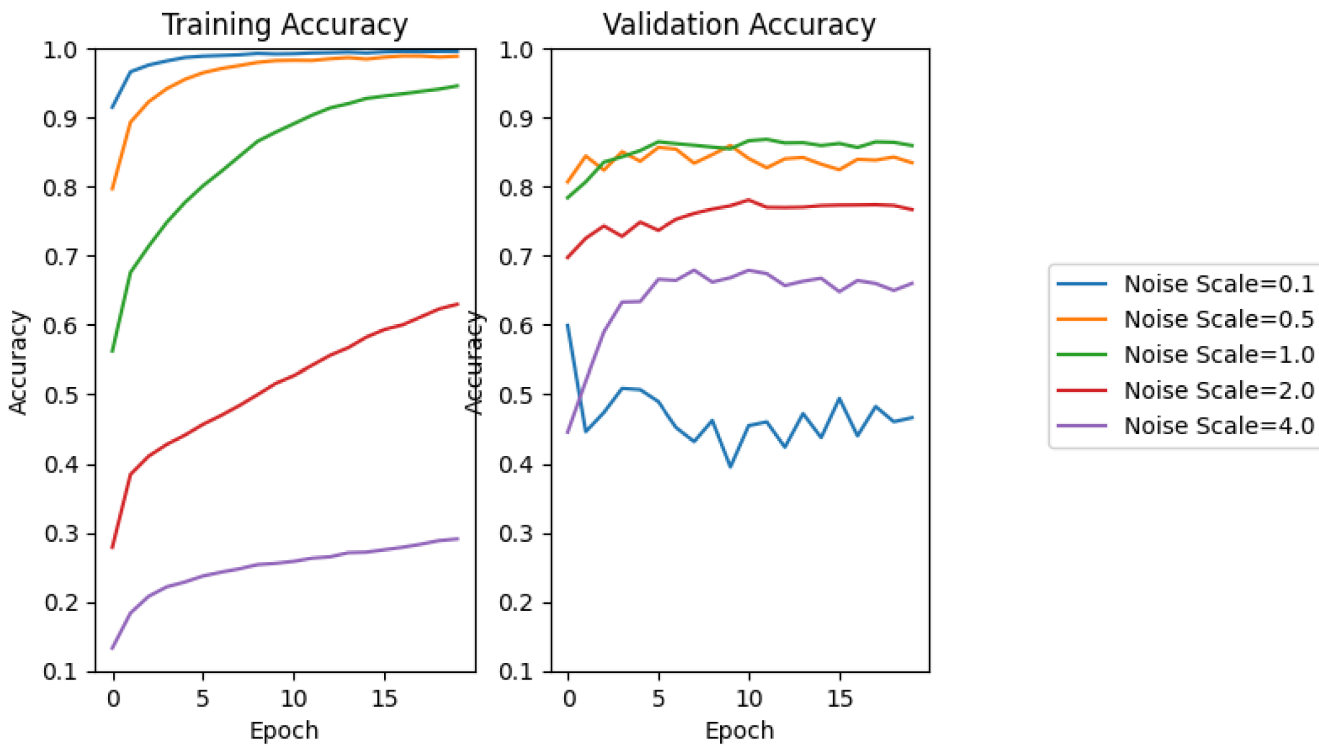
In [18]:

```
#plot for training and validation
fig, axs = plt.subplots(1, 2)

for scale, history in histories.items():
    axs[0].plot(history.history['accuracy'], label=f'Noise Scale={scale}')

axs[0].set_title('ConvNet Training Accuracy')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')

for scale, history in histories.items():
    axs[1].plot(history.history['val_accuracy'])

axs[1].set_title('ConvNet Validation Accuracy')
axs[1].set_xlabel('Epoch')

axs[0].set_ylim(0.1, 1.0)
axs[1].set_ylim(0.1, 1.0)
fig.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



4. Compare these results with the previous week where we used a MultiLayer Perceptron (this week we use a ConvNet).

Previous weeks plots below



Comparing the ConvNet to the Multilayer Perceptron, we can see visually that the perceptron model has higher training accuracy and validation accuracy at all levels of noise scales, through 20 epochs. However, we can see at Noise Scale 0.1 and 0.5, the ConvNet model is continuing to increase steadily toward 1.0 accuracy. We could expect that across a greater number of epochs, the convnet model would match the training accuracy of the perceptron model. As we look at the shape of the training curves, we also see much steeper learning for the multilayer model, especially at higher noise levels. The convnet models have both low accuracy and flat slopes at the 2.0 and 4.0 noise scales. Looking at validation accuracy, we see the perceptron models essentially remaining level through 20 epochs, but the convnet model continues to increase in validation accuracy, even at 20 epochs. We might expect it to continue to increase at greater epochs, although it might also level out (similar to the perceptron model) when the overall accuracy reaches its highest level.

One thing to note (also mentioned in the lectures) is the length of time it takes to train a convnet model is substantially longer than that of the perceptron model. That said, adjusting the batch size (smaller) and epochs (larger) would likely result in increased accuracy for the convnet model. With more computing power, adjustments could certainly be made to the convnet model to increase its accuracy.

In [ ]: