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 tensorflow.keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend
import tensorflow as tf
```

In [43]:
```
import matplotlib.pyplot as  plt
%matplotlib inline
```

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 [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()
```

In [3]:
```
x_train.shape, y_train.shape
```

Out[3]:
```
((60000, 28, 28), (60000,))
```

In [4]:
```
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 [5]:
```
x_train.shape
```

Out[5]:
```
(60000, 28, 28, 1)
```

In [6]:
```
batch_size = 128
num_classes = 10
epochs = 12

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

In [7]:
```
y_train, y_test
```

Out[7]:
```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 1., 0.]], dtype=float32),
 array([[0., 0., 0., ..., 1., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], dtype=float32))
```

In [8]:
```
x_train.shape, y_train.shape
```

Out[8]:
```
((60000, 28, 28, 1), (60000, 10))
```

In [9]:
```
# define the model and evaluate without noise
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=tf.keras.optimizers.Adadelta(),
              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])
```

```
2023-04-25 18:09:19.840728: I tensorflow/core/platform/cpu_feature_gu
ard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neur
al Network Library (oneDNN) to use the following CPU instructions in
performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appro
priate compiler flags.
2023-04-25 18:09:20.357324: I tensorflow/compiler/mlir/mlir_graph_opt
imization_pass.cc:185] None of the MLIR Optimization Passes are enabl
ed (registered 2)

Epoch 1/12
469/469 [==============================] - 48s 102ms/step - loss: 2.2
787 - accuracy: 0.1544 - val_loss: 2.2378 - val_accuracy: 0.3373
Epoch 2/12
469/469 [==============================] - 61s 129ms/step - loss: 2.2
134 - accuracy: 0.2693 - val_loss: 2.1569 - val_accuracy: 0.5303
Epoch 3/12
469/469 [==============================] - 54s 116ms/step - loss: 2.1
285 - accuracy: 0.3796 - val_loss: 2.0471 - val_accuracy: 0.6578
Epoch 4/12
469/469 [==============================] - 50s 107ms/step - loss: 2.0
106 - accuracy: 0.4728 - val_loss: 1.8952 - val_accuracy: 0.7051
Epoch 5/12
469/469 [==============================] - 51s 109ms/step - loss: 1.8
560 - accuracy: 0.5378 - val_loss: 1.6990 - val_accuracy: 0.7272
Epoch 6/12
469/469 [==============================] - 51s 110ms/step - loss: 1.6
702 - accuracy: 0.5869 - val_loss: 1.4738 - val_accuracy: 0.7552
Epoch 7/12
469/469 [==============================] - 50s 107ms/step - loss: 1.4
```

```
807 - accuracy: 0.6220 - val_loss: 1.2551 - val_accuracy: 0.7790
Epoch 8/12
469/469 [==============================] - 51s 110ms/step - loss: 1.3
038 - accuracy: 0.6542 - val_loss: 1.0672 - val_accuracy: 0.7952
Epoch 9/12
469/469 [==============================] - 50s 108ms/step - loss: 1.1
675 - accuracy: 0.6761 - val_loss: 0.9224 - val_accuracy: 0.8102
Epoch 10/12
469/469 [==============================] - 49s 105ms/step - loss: 1.0
595 - accuracy: 0.6953 - val_loss: 0.8153 - val_accuracy: 0.8199
Epoch 11/12
469/469 [==============================] - 50s 107ms/step - loss: 0.9
740 - accuracy: 0.7150 - val_loss: 0.7354 - val_accuracy: 0.8289
Epoch 12/12
469/469 [==============================] - 51s 108ms/step - loss: 0.9
126 - accuracy: 0.7272 - val_loss: 0.6754 - val_accuracy: 0.8370
Test loss: 0.6753876805305481
Test accuracy: 0.8370000123977661
```

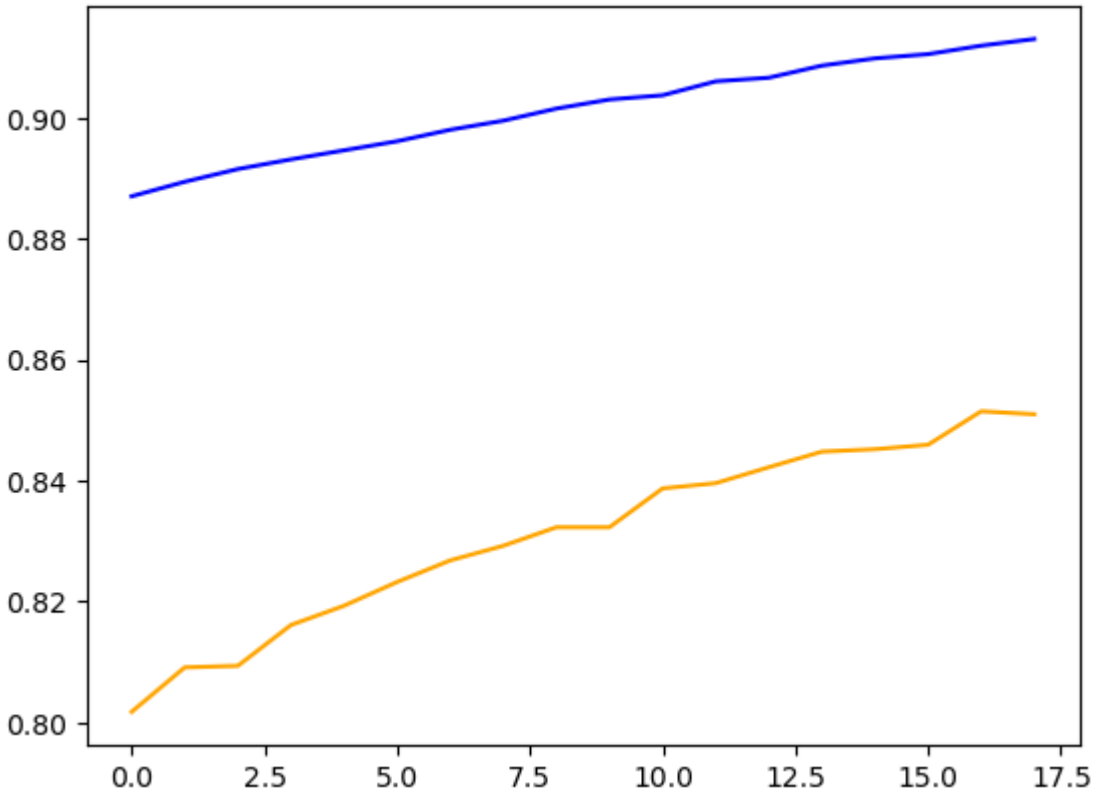
In [37]:
```
history.history['accuracy'][-1], history.history['val_accuracy'][-1]
```

Out[37]:
```
(0.8486166596412659, 0.9089999794960022)
```

In [11]:
```
score
```

Out[11]:
```
[0.6753876805305481, 0.8370000123977661]
```

In [46]:
```
plt.figure()
plt.plot(history.history['accuracy'], c = "orange")
plt.plot(history.history['val_accuracy'], c = "blue")
plt.show()
```



In [48]:
```
# define the model and evaluate without noise for a series of epochs
epochs = [10, 12, 14, 16, 18, 20]
scores = []
history_accuracy = []

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=tf.keras.optimizers.Adadelta(),
              metrics=['accuracy'])

for epoch in epochs:

    print(epoch)

    history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=epoch,
                        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])

    history_accuracy.append(history.history['accuracy'][-1])
    scores.append(score)
```

```
10
Epoch 1/10
469/469 [==============================] - 45s 95ms/step - loss: 2.27
81 - accuracy: 0.1522 - val_loss: 2.2372 - val_accuracy: 0.4221
Epoch 2/10
469/469 [==============================] - 53s 112ms/step - loss: 2.2
135 - accuracy: 0.2765 - val_loss: 2.1543 - val_accuracy: 0.5848
Epoch 3/10
469/469 [==============================] - 52s 112ms/step - loss: 2.1
251 - accuracy: 0.3900 - val_loss: 2.0404 - val_accuracy: 0.6450
Epoch 4/10
469/469 [==============================] - 53s 113ms/step - loss: 2.0
063 - accuracy: 0.4642 - val_loss: 1.8903 - val_accuracy: 0.6874
Epoch 5/10
469/469 [==============================] - 51s 109ms/step - loss: 1.8
571 - accuracy: 0.5228 - val_loss: 1.7070 - val_accuracy: 0.7177
Epoch 6/10
469/469 [==============================] - 52s 111ms/step - loss: 1.6
860 - accuracy: 0.5693 - val_loss: 1.5038 - val_accuracy: 0.7523
```

Epoch 7/10

In [49]:
```
history_accuracy, scores
```

Out[49]:
```
([0.677216649055481,
  0.7988333106040955,
  0.8416666388511658,
  0.8658666610717773,
  0.8811500072479248,
  0.8950499892234802],
 [0.8127999901771545,
  0.8827000260353088,
  0.907800018787384,
  0.9236999750137329,
  0.9319000244140625,
  0.9384999871253967],
 [[0.8744704127311707, 0.8127999901771545],
  [0.4448353946208954, 0.8827000260353088],
  [0.33696213364601135, 0.907800018787384],
  [0.2808745801448822, 0.9236999750137329],
  [0.24338044226169586, 0.9319000244140625],
  [0.21484573185443878, 0.9384999871253967]])
```

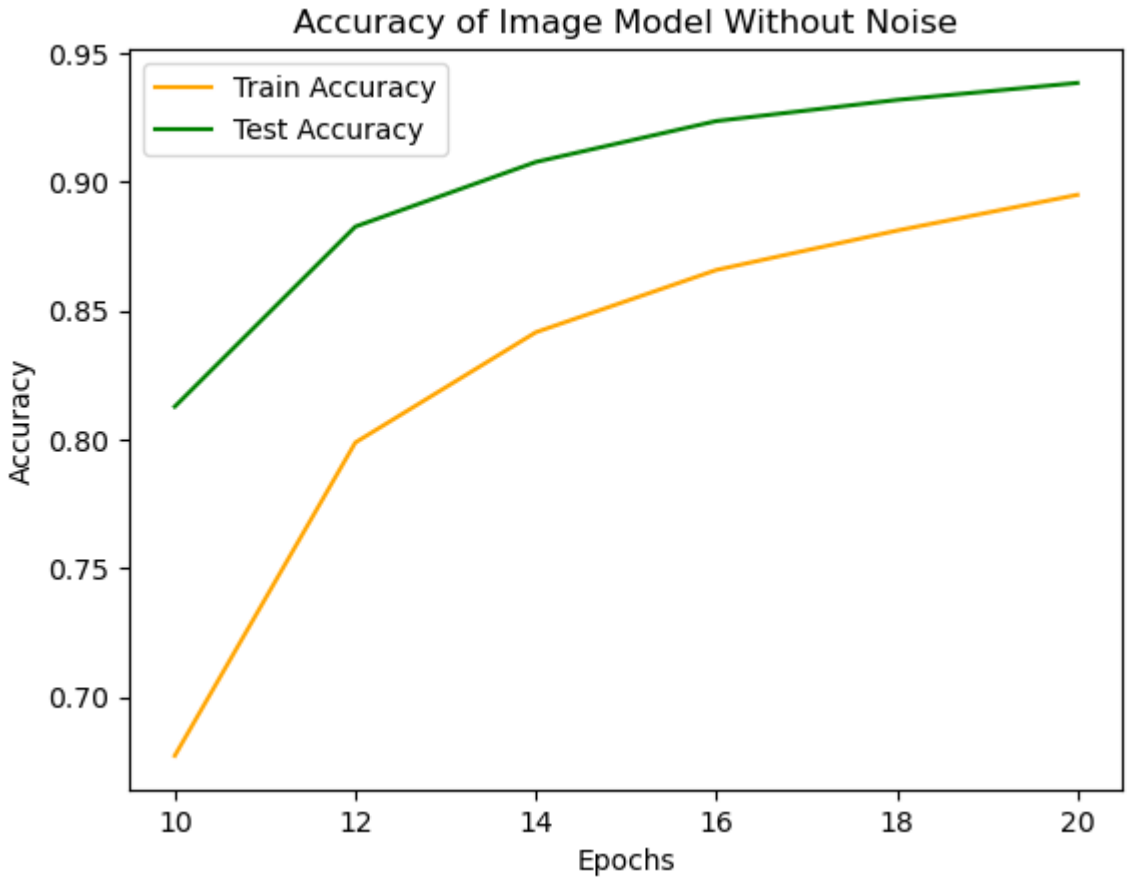
In [59]:
```
import numpy as np
import pandas as pd

scoresdf = pd.DataFrame(scores, columns=['loss','accuracy'])
scoresdf
```

Out[59]:

	loss	accuracy
0	0.874470	0.8128
1	0.444835	0.8827
2	0.336962	0.9078
3	0.280875	0.9237
4	0.243380	0.9319
5	0.214846	0.9385

In [63]:
```
plt.figure()
plt.plot(epochs, history_accuracy, label = 'Train Accuracy', c = "oran
plt.plot(epochs, scoresdf['accuracy'], label = 'Test Accuracy', c = "g
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy of Image Model Without Noise')
plt.legend()
plt.show()
```



In [51]:
```
# create noisy data at scale 2
from numpy import random

rando = random.normal(loc=1, scale=2, size=x_train.shape)
x_train_n = x_train + rando

rando = random.normal(loc=1, scale=2, size=x_test.shape)
x_test_n = x_test + rando
```

In [52]:
```
# define the model and evaluate with noise scale 2 for a series of epo
epochs = [10, 12, 14, 16, 18, 20]
scores_n = []
history_accuracy_n = []

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=tf.keras.optimizers.Adadelta(),
              metrics=['accuracy'])

for epoch in epochs:

    print(epoch)

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

    history_accuracy_n.append(history.history['accuracy'][-1])
    scores_n.append(score)
```

```
10
Epoch 1/10
469/469 [==============================] - 47s 99ms/step - loss: 2.43
19 - accuracy: 0.1017 - val_loss: 2.3045 - val_accuracy: 0.1085
Epoch 2/10
469/469 [==============================] - 52s 111ms/step - loss: 2.3
431 - accuracy: 0.1041 - val_loss: 2.2957 - val_accuracy: 0.1190
Epoch 3/10
469/469 [==============================] - 51s 109ms/step - loss: 2.3
157 - accuracy: 0.1098 - val_loss: 2.2951 - val_accuracy: 0.1306
Epoch 4/10
469/469 [==============================] - 50s 106ms/step - loss: 2.3
074 - accuracy: 0.1124 - val_loss: 2.2957 - val_accuracy: 0.1294
Epoch 5/10
469/469 [==============================] - 51s 110ms/step - loss: 2.3
015 - accuracy: 0.1138 - val_loss: 2.2949 - val_accuracy: 0.1322
Epoch 6/10
469/469 [==============================] - 52s 112ms/step - loss: 2.3
005 - accuracy: 0.1158 - val_loss: 2.2943 - val_accuracy: 0.1340
```

Epoch 7/10

In [62]:
```
scores_n
```

Out[62]:
```
[[2.288330554962158, 0.1477999985218048],
 [2.233083486557007, 0.23240000009536743],
 [2.0868451595306396, 0.3643999993801117],
 [1.8778607845306396, 0.421999990940094],
 [1.7114169597625732, 0.4494999945163727],
 [1.6221354007720947, 0.4636000096797943]]
```

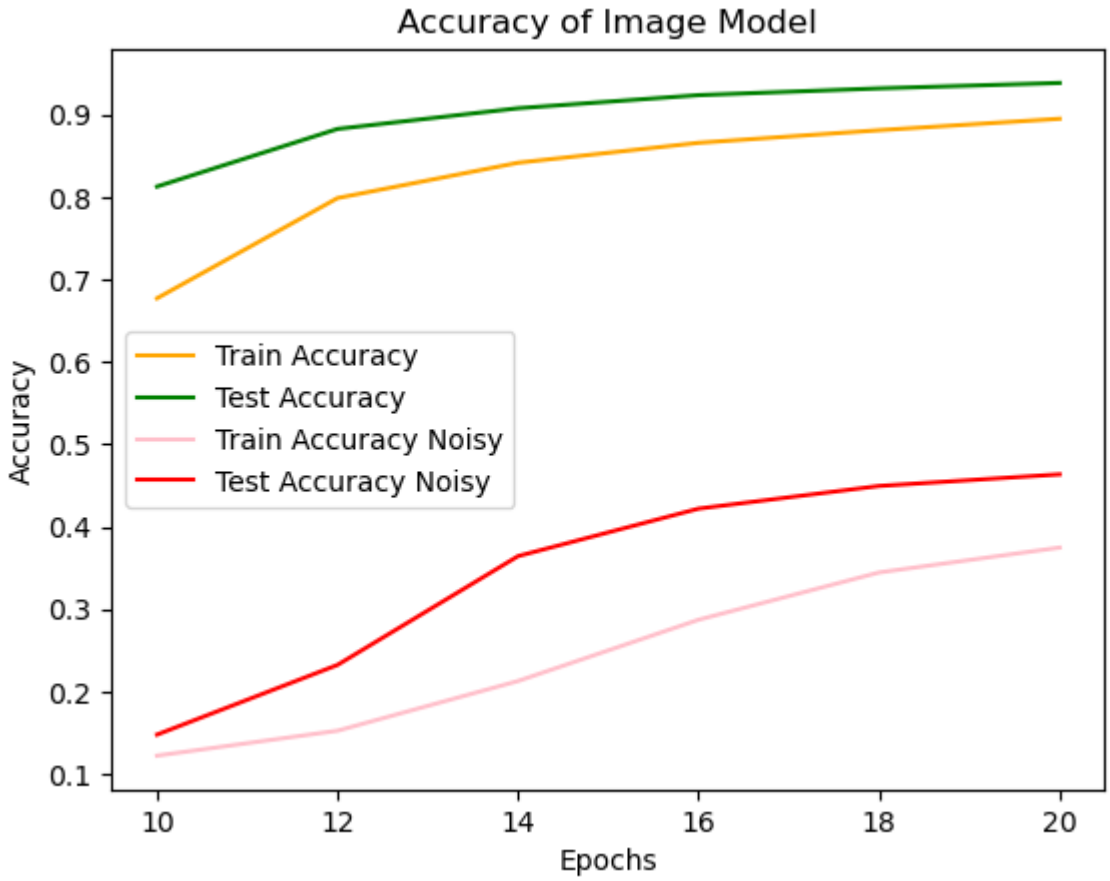
In [64]:
```
import numpy as np
import pandas as pd

scores_ndf = pd.DataFrame(scores_n, columns=['loss','accuracy'])
scores_ndf
```

Out[64]:

	loss	accuracy
0	2.288331	0.1478
1	2.233083	0.2324
2	2.086845	0.3644
3	1.877861	0.4220
4	1.711417	0.4495
5	1.622135	0.4636

In [66]:
```
plt.figure()
plt.plot(epochs, history_accuracy, label = 'Train Accuracy', c = "oran
plt.plot(epochs, scoresdf['accuracy'], label = 'Test Accuracy', c = "g
plt.plot(epochs, history_accuracy_n, label = 'Train Accuracy Noisy', c
plt.plot(epochs, scores_ndf['accuracy'], label = 'Test Accuracy Noisy'
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy of Image Model')
plt.legend()
plt.show()
```



In [68]:
```
# define the model and evaluate with noise at difference scales
scales = [.1, .5, 1.0, 2.0, 4.0]
scores_ns = []
history_ns = []
epochs = 20

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=tf.keras.optimizers.Adadelta(),
              metrics=['accuracy'])

for scale in scales:
        print(scale)

        x = random.normal(loc=1, scale=scale, size=x_train.shape)
        x_train = x_train + x

        x = random.normal(loc=1, scale=scale, size=x_test.shape)
        x_test = x_test + x

        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])

        history_ns.append(history.history['accuracy'][-1])
        scores_ns.append(score)
```

```
0.1
Epoch 1/20
469/469 [==============================] - 54s 115ms/step - loss: 2.3
247 - accuracy: 0.1020 - val_loss: 2.2950 - val_accuracy: 0.1684
Epoch 2/20
469/469 [==============================] - 55s 117ms/step - loss: 2.3
026 - accuracy: 0.1131 - val_loss: 2.2829 - val_accuracy: 0.2222
Epoch 3/20
469/469 [==============================] - 58s 123ms/step - loss: 2.2
916 - accuracy: 0.1297 - val_loss: 2.2716 - val_accuracy: 0.2140
Epoch 4/20
469/469 [==============================] - 58s 123ms/step - loss: 2.2
811 - accuracy: 0.1445 - val_loss: 2.2601 - val_accuracy: 0.2437
```

```
Epoch 5/20
469/469 [==============================] - 58s 123ms/step - loss: 2.2
731 - accuracy: 0.1553 - val_loss: 2.2481 - val_accuracy: 0.3065
Epoch 6/20
469/469 [==============================] - 59s 125ms/step - loss: 2.2
620 - accuracy: 0.1688 - val_loss: 2.2355 - val_accuracy: 0.3616
```

In [69]:
```
scores_ns, history_ns
```

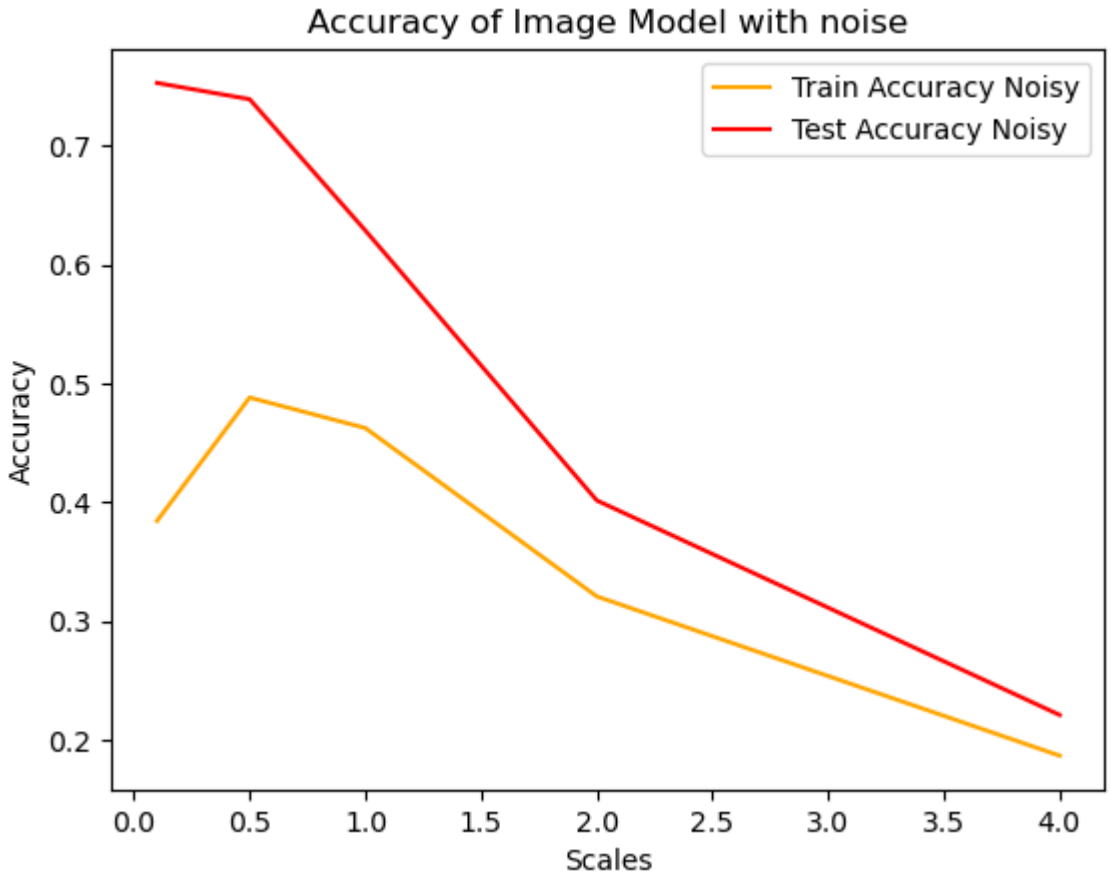
Out[69]:
```
([[1.8641937971115112, 0.7527999877929688],
  [1.3607019186019897, 0.7390999794006348],
  [1.3236926794052124, 0.628600001335144],
  [1.7738542556762695, 0.40139999985694885],
  [2.17803955078125, 0.2207999974489212]],
 [0.38420000672340393,
  0.48801666498184204,
  0.4623333215713501,
  0.3206000030040741,
  0.18653333187103271])
```

In [70]:
```
scores_nsdf = pd.DataFrame(scores_ns, columns=['loss','accuracy'])
scores_nsdf
```

Out[70]:

	loss	accuracy
0	1.864194	0.7528
1	1.360702	0.7391
2	1.323693	0.6286
3	1.773854	0.4014
4	2.178040	0.2208

In [73]:
```
plt.figure()
plt.plot(scales, history_ns, label = 'Train Accuracy Noisy', c = "oran
plt.plot(scales, scores_nsdf['accuracy'], label = 'Test Accuracy Noisy
plt.xlabel('Scales')
plt.ylabel('Accuracy')
plt.title('Accuracy of Image Model with noise')
plt.legend()
plt.show()
```



Multi level perceptron comparison

In [80]:
```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [81]:
```
#reshape the data without noise
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
```

In [82]:
```
# convert class vectors to binary class matrices
y_train = keras.utils.np_utils.to_categorical(y_train, num_classes)
y_test = keras.utils.np_utils.to_categorical(y_test, num_classes)

y_train, y_test
```

Out[82]:
```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 1., 0.]], dtype=float32),
 array([[0., 0., 0., ..., 1., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], dtype=float32))
```

In [83]:
```
# Multi level perceptron
scales = [.1, .5, 1.0, 2.0, 4.0]
epochs = 20
scores_mp = []
history_mp = []


# build model and evaluate
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=RMSprop(), me

for scale in scales:

    print(scale)

    x = random.normal(loc=1, scale=scale, size=x_train.shape)
    x_train = x_train + x

    x = random.normal(loc=1, scale=scale, size=x_test.shape)
    x_test = x_test + x

    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])

    history_mp.append(history.history['accuracy'][-1])
    scores_mp.append(score)
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_24 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_28 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_25 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_29 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
```

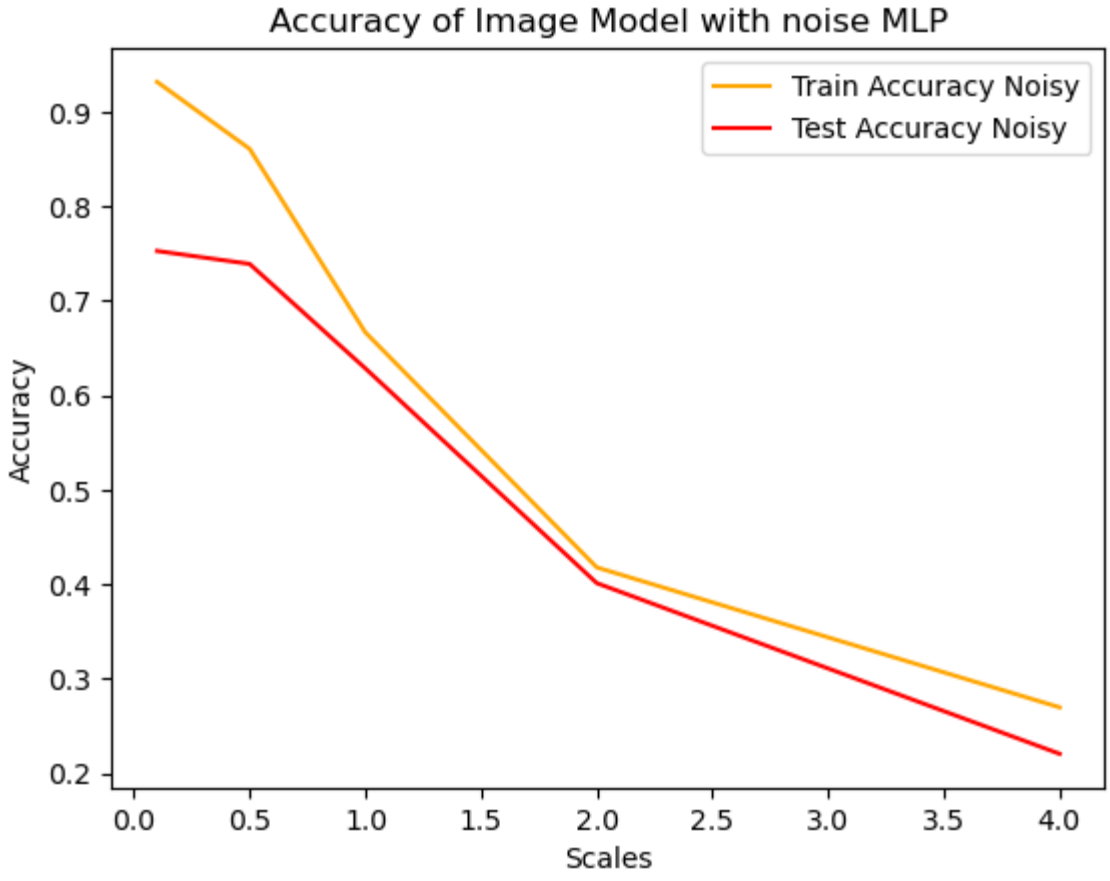
```
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
0.1
```

In [84]:
```
scores_mp = pd.DataFrame(scores_ns, columns=['loss','accuracy'])
scores_mp
```

Out[84]:

	loss	accuracy
0	1.864194	0.7528
1	1.360702	0.7391
2	1.323693	0.6286
3	1.773854	0.4014
4	2.178040	0.2208

In [85]:
```
plt.figure()
plt.plot(scales, history_mp, label = 'Train Accuracy Noisy', c = "oran
plt.plot(scales, scores_mp['accuracy'], label = 'Test Accuracy Noisy',
plt.xlabel('Scales')
plt.ylabel('Accuracy')
plt.title('Accuracy of Image Model with noise MLP')
plt.legend()
plt.show()
```



In comparison between MLP and ConvNet. It appears that MLP might have a higher accuracy at lower random noise levels but starts getting to a similar accuracy score the higher the noise scale.

In [ ]: