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.
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 [2]:
```
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
```

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 [3]:
```
# input image dimensions
img_rows, img_cols = 28, 28
num_classes = 10

# 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')

# 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)
```

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

First, I am running the network without noise added as a baseline.

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



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=1,
              validation_data=(x_test, y_test))
baseline_score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', baseline_score[0])
print('Test accuracy:', baseline_score[1])
baseline_accuracy = baseline_score[1]
```

```
Epoch 1/12
469/469 [==============================] - 28s 59ms/step - loss: 0.2552 - accuracy: 0.9234 - val_loss: 0.0526 - val_accuracy: 0.9832
Epoch 2/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0862 - accuracy: 0.9743 - val_loss: 0.0375 - val_accuracy: 0.9866
Epoch 3/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0665 - accuracy: 0.9798 - val_loss: 0.0353 - val_accuracy: 0.9890
Epoch 4/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0547 - accuracy: 0.9830 - val_loss: 0.0318 - val_accuracy: 0.9890
Epoch 5/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0459 - accuracy: 0.9856 - val_loss: 0.0280 - val_accuracy: 0.9903
Epoch 6/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0401 - accuracy: 0.9878 - val_loss: 0.0289 - val_accuracy: 0.9902
Epoch 7/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0367 - accuracy: 0.9884 - val_loss: 0.0301 - val_accuracy: 0.9910
Epoch 8/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0333 - accuracy: 0.9893 - val_loss: 0.0289 - val_accuracy: 0.9910
Epoch 9/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0293 - accuracy: 0.9908 - val_loss: 0.0289 - val_accuracy: 0.9913
Epoch 10/12
469/469 [==============================] - 27s 59ms/step - loss: 0.0279 - accuracy: 0.9909 - val_loss: 0.0281 - val_accuracy: 0.9920
Epoch 11/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0235 - accuracy: 0.9919 - val_loss: 0.0298 - val_accuracy: 0.9919
Epoch 12/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0231 - accuracy: 0.9922 - val_loss: 0.0278 - val_accuracy: 0.9924
Test loss: 0.027807746082544327
Test accuracy: 0.9923999905586243
```

In [13]:
```
baseline_accuracy
```

Out[13]:
```
0.9923999905586243
```

Below, I am printing some images and their labels from the unaltered (baseline) dataset.

In [14]:
```
x_train.shape,x_test.shape, y_train.shape, y_test.shape
```

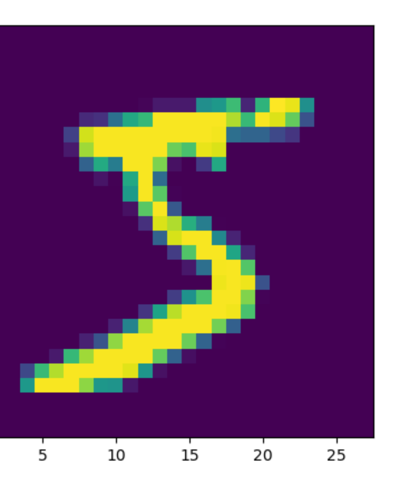
Out[14]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1), (60000, 10), (10000, 10))
```

In [15]:
```
import matplotlib.pyplot as  plt
%matplotlib inline

import numpy as np
```

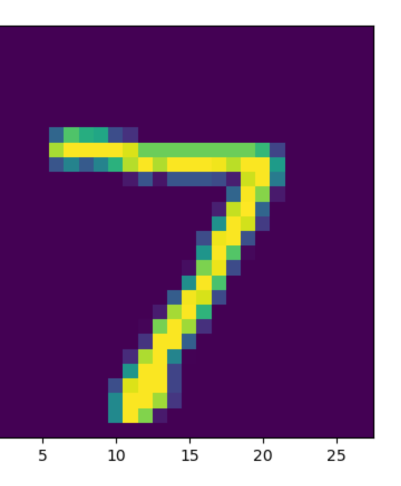
In [18]:
```
plt.imshow(x_train[0])
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x29f8a0fd0>
```



In [19]:
```
plt.imshow(x_test[0])
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x28ea98550>
```



In [20]:
```
from numpy import argmax

argmax(keras.utils.to_categorical(y_train[0], 10),axis=1)
```

Out[20]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

In [21]:
```
argmax(keras.utils.to_categorical(y_test[0], 10),axis=1)
```

Out[21]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

Adding noise here

In [24]:
```
scale = [0.1, 0.5, 1.0, 2.0, 4.0]

noise_0_train = np.random.normal(scale=0.1, size=[60000,28,28,1])
noise_1_train = np.random.normal(scale=0.5, size=[60000,28,28,1])
noise_2_train = np.random.normal(scale=1.0, size=[60000,28,28,1])
noise_3_train = np.random.normal(scale=2.0, size=[60000,28,28,1])
noise_4_train = np.random.normal(scale=4.0, size=[60000,28,28,1])

noise_0_test = np.random.normal(scale=0.1, size=[10000,28,28,1])
noise_1_test = np.random.normal(scale=0.5, size=[10000,28,28,1])
noise_2_test = np.random.normal(scale=1.0, size=[10000,28,28,1])
noise_3_test = np.random.normal(scale=2.0, size=[10000,28,28,1])
noise_4_test = np.random.normal(scale=4.0, size=[10000,28,28,1])
```

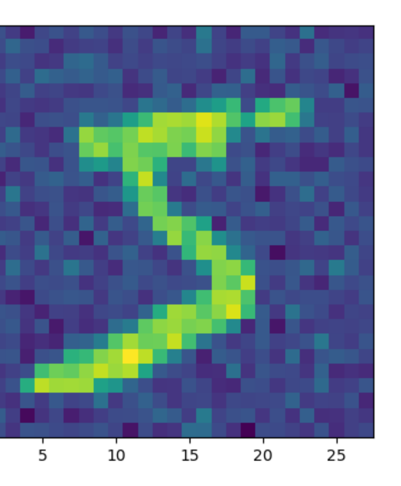
In [25]:
```
x_train_n0 = x_train + noise_0_train
x_train_n1 = x_train + noise_1_train
x_train_n2 = x_train + noise_2_train
x_train_n3 = x_train + noise_3_train
x_train_n4 = x_train + noise_4_train

x_test_n0 = x_test + noise_0_test
x_test_n1 = x_test + noise_1_test
x_test_n2 = x_test + noise_2_test
x_test_n3 = x_test + noise_3_test
x_test_n4 = x_test + noise_4_test
```

Inspecting noisy images here

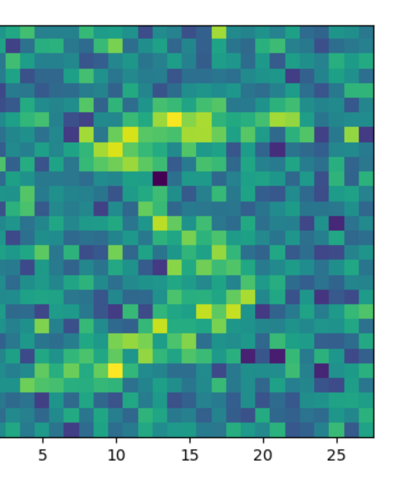
In [26]:
```
plt.imshow(x_train_n0[0])
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x29bad2b30>
```



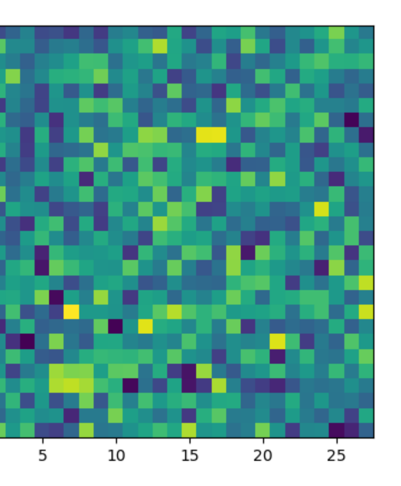
In [27]:
```
plt.imshow(x_train_n1[0])
```

Out[27]:
```
<matplotlib.image.AxesImage at 0x28b4adc30>
```



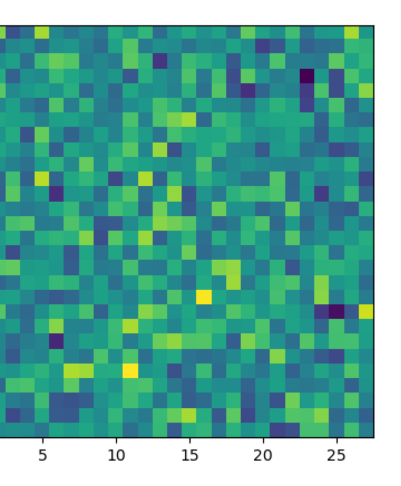
In [28]:
```
plt.imshow(x_train_n2[0])
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x2d7dd37f0>
```



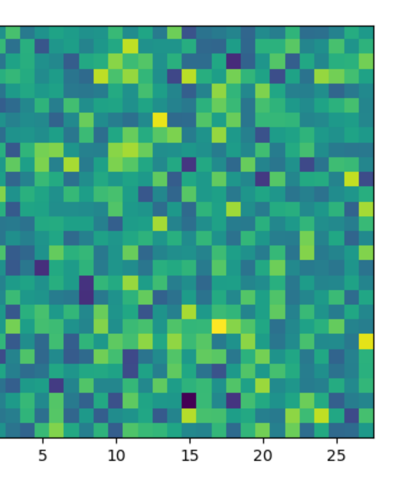
In [29]:
```
[illegible]
```

Out[29]:
```
[illegible]
```



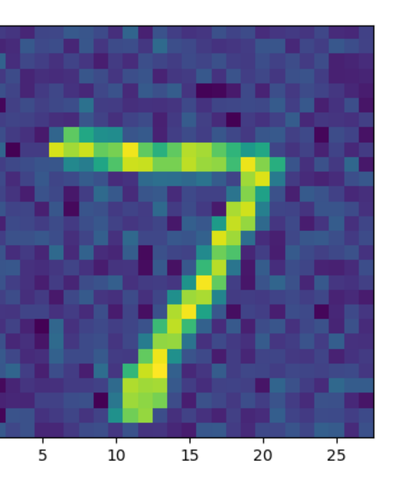
In [30]:
```
[illegible]
```

Out[30]:
```
[illegible]
```



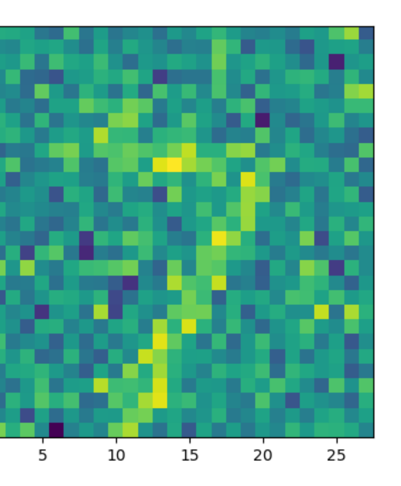
In [31]:
```
[illegible]
```

Out[31]:
```
[illegible]
```



In [32]:
```
[illegible]
```

Out[32]:
```
[illegible]
```



I will use the following code chunks to train the CNN with the different noise parameters (created above). I will also keep track of each accuracy score output, in respect to the noise parameters used during training.

In [36]:
```
[illegible]

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

    history_noisy = model_noisy.fit(x_train_noisy[i], y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(x_test_noisy[i], y_test))

    score_noisy = model_noisy.evaluate(x_test_noisy[i], y_test, verbose=0)

    noisy_scores.append(score_noisy[1])
```

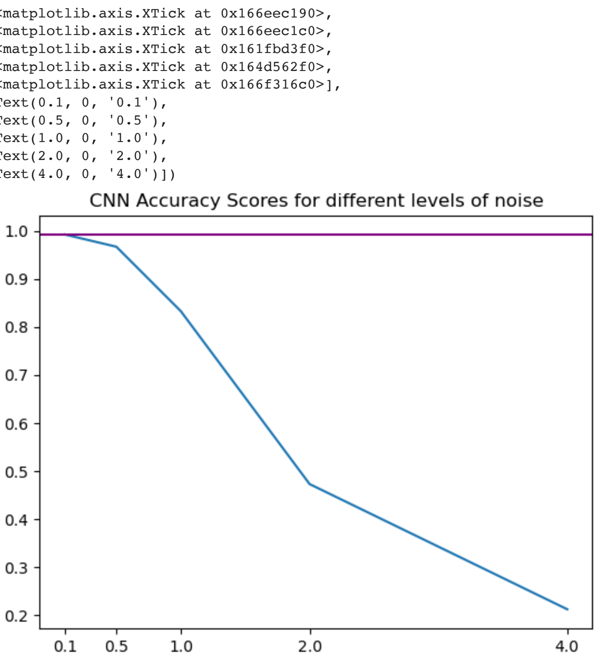
```
Epoch 1/12
469/469 [==============================] - 28s 59ms/step - loss: 0.2651 - accuracy: 0.9194 - val_loss: 0.0596 - val_accuracy: 0.9816
Epoch 2/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0974 - accuracy: 0.9711 - val_loss: 0.0459 - val_accuracy: 0.9856
Epoch 3/12
469/469 [==============================] - 28s 59ms/step - loss: 0.0711 - accuracy: 0.9783 - val_loss: 0.0430 - val_accuracy: 0.9852
Epoch 4/12
469/469 [==============================] - 28s 61ms/step - loss: 0.0603 - accuracy: 0.9816 - val_loss: 0.0368 - val_accuracy: 0.9884
Epoch 5/12
469/469 [==============================] - 29s 62ms/step - loss: 0.0492 - accuracy: 0.9852 - val_loss: 0.0327 - val_accuracy: 0.9892
Epoch 6/12
469/469 [==============================] - 29s 61ms/step - loss: 0.0447 - accuracy: 0.9856 - val_loss: 0.0317 - val_accuracy: 0.9898
Epoch 7/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0391 - accuracy: 0.9872 - val_loss: 0.0362 - val_accuracy: 0.9876
Epoch 8/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0356 - accuracy: 0.9885 - val_loss: 0.0319 - val_accuracy: 0.9901
Epoch 9/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0313 - accuracy: 0.9901 - val_loss: 0.0293 - val_accuracy: 0.9911
Epoch 10/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0311 - accuracy: 0.9898 - val_loss: 0.0279 - val_accuracy: 0.9919
Epoch 11/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0265 - accuracy: 0.9913 - val_loss: 0.0287 - val_accuracy: 0.9921
Epoch 12/12
469/469 [==============================] - 27s 58ms/step - loss: 0.0241 - accuracy: 0.9921 - val_loss: 0.0299 - val_accuracy: 0.9924
Epoch 1/12
469/469 [==============================] - 28s 59ms/step - loss: 0.6050 - accuracy: 0.8073 - val_loss: 0.1679 - val_accuracy: 0.9475
Epoch 2/12
469/469 [==============================] - 28s 60ms/step - loss: 0.2636 - accuracy: 0.9184 - val_loss: 0.1212 - val_accuracy: 0.9607
Epoch 3/12
469/469 [==============================] - 31s 66ms/step - loss: 0.2133 - accuracy: 0.9351 - val_loss: 0.1080 - val_accuracy: 0.9644
Epoch 4/12
469/469 [==============================] - 27s 58ms/step - loss: 0.1794 - accuracy: 0.9432 - val_loss: 0.1047 - val_accuracy: 0.9651
Epoch 5/12
469/469 [==============================] - 27s 59ms/step - loss: 0.1596 - accuracy: 0.9500 - val_loss: 0.1105 - val_accuracy: 0.9642
Epoch 6/12
469/469 [==============================] - 28s 61ms/step - loss: 0.1433 - accuracy: 0.9548 - val_loss: 0.1005 - val_accuracy: 0.9688
Epoch 7/12
469/469 [==============================] - 30s 65ms/step - loss: 0.1318 - accuracy: 0.9570 - val_loss: 0.1005 - val_accuracy: 0.9702
Epoch 8/12
469/469 [==============================] - 30s 64ms/step - loss: 0.1163 - accuracy: 0.9619 - val_loss: 0.1042 - val_accuracy: 0.9699
Epoch 9/12
469/469 [==============================] - 30s 63ms/step - loss: 0.1090 - accuracy: 0.9629 - val_loss: 0.0983 - val_accuracy: 0.9693
Epoch 10/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0992 - accuracy: 0.9665 - val_loss: 0.1062 - val_accuracy: 0.9681
Epoch 11/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0930 - accuracy: 0.9677 - val_loss: 0.1097 - val_accuracy: 0.9690
Epoch 12/12
469/469 [==============================] - 28s 60ms/step - loss: 0.0880 - accuracy: 0.9698 - val_loss: 0.1121 - val_accuracy: 0.9669
Epoch 1/12
469/469 [==============================] - 31s 65ms/step - loss: 1.1643 - accuracy: 0.6120 - val_loss: 0.6373 - val_accuracy: 0.8063
Epoch 2/12
469/469 [==============================] - 30s 65ms/step - loss: 0.8495 - accuracy: 0.7193 - val_loss: 0.5672 - val_accuracy: 0.8203
Epoch 3/12
469/469 [==============================] - 30s 64ms/step - loss: 0.7750 - accuracy: 0.7459 - val_loss: 0.5334 - val_accuracy: 0.8284
Epoch 4/12
469/469 [==============================] - 29s 61ms/step - loss: 0.7278 - accuracy: 0.7594 - val_loss: 0.5122 - val_accuracy: 0.8334
Epoch 5/12
469/469 [==============================] - 29s 62ms/step - loss: 0.6894 - accuracy: 0.7736 - val_loss: 0.4975 - val_accuracy: 0.8388
Epoch 6/12
469/469 [==============================] - 28s 59ms/step - loss: 0.6631 - accuracy: 0.7800 - val_loss: 0.5088 - val_accuracy: 0.8390
Epoch 7/12
469/469 [==============================] - 30s 65ms/step - loss: 0.6397 - accuracy: 0.7864 - val_loss: 0.5071 - val_accuracy: 0.8354
Epoch 8/12
469/469 [==============================] - 28s 60ms/step - loss: 0.6154 - accuracy: 0.7927 - val_loss: 0.5036 - val_accuracy: 0.8334
Epoch 9/12
469/469 [==============================] - 28s 60ms/step - loss: 0.5910 - accuracy: 0.7990 - val_loss: 0.5106 - val_accuracy: 0.8363
Epoch 10/12
469/469 [==============================] - 28s 60ms/step - loss: 0.5708 - accuracy: 0.8065 - val_loss: 0.5111 - val_accuracy: 0.8358
Epoch 11/12
469/469 [==============================] - 29s 61ms/step - loss: 0.5537 - accuracy: 0.8110 - val_loss: 0.5106 - val_accuracy: 0.8352
Epoch 12/12
469/469 [==============================] - 28s 61ms/step - loss: 0.5443 - accuracy: 0.8136 - val_loss: 0.5174 - val_accuracy: 0.8328
Epoch 1/12
469/469 [==============================] - 29s 61ms/step - loss: 2.2032 - accuracy: 0.1734 - val_loss: 1.8276 - val_accuracy: 0.3988
Epoch 2/12
469/469 [==============================] - 28s 61ms/step - loss: 1.9534 - accuracy: 0.2910 - val_loss: 1.7303 - val_accuracy: 0.4497
Epoch 3/12
469/469 [==============================] - 29s 61ms/step - loss: 1.9017 - accuracy: 0.3130 - val_loss: 1.7219 - val_accuracy: 0.4571
Epoch 4/12
469/469 [==============================] - 28s 60ms/step - loss: 1.8759 - accuracy: 0.3234 - val_loss: 1.6423 - val_accuracy: 0.4550
Epoch 5/12
469/469 [==============================] - 28s 60ms/step - loss: 1.8540 - accuracy: 0.3350 - val_loss: 1.6364 - val_accuracy: 0.4651
Epoch 6/12
469/469 [==============================] - 29s 61ms/step - loss: 1.8405 - accuracy: 0.3375 - val_loss: 1.6481 - val_accuracy: 0.4701
Epoch 7/12
469/469 [==============================] - 28s 60ms/step - loss: 1.8227 - accuracy: 0.3431 - val_loss: 1.6456 - val_accuracy: 0.4676
Epoch 8/12
469/469 [==============================] - 28s 60ms/step - loss: 1.8130 - accuracy: 0.3511 - val_loss: 1.6029 - val_accuracy: 0.4684
Epoch 9/12
469/469 [==============================] - 29s 61ms/step - loss: 1.8008 - accuracy: 0.3521 - val_loss: 1.6067 - val_accuracy: 0.4677
Epoch 10/12
469/469 [==============================] - 28s 60ms/step - loss: 1.7878 - accuracy: 0.3540 - val_loss: 1.6050 - val_accuracy: 0.4743
Epoch 11/12
469/469 [==============================] - 27s 58ms/step - loss: 1.7708 - accuracy: 0.3631 - val_loss: 1.5986 - val_accuracy: 0.4688
Epoch 12/12
469/469 [==============================] - 28s 59ms/step - loss: 1.7546 - accuracy: 0.3684 - val_loss: 1.6176 - val_accuracy: 0.4726
Epoch 1/12
469/469 [==============================] - 30s 63ms/step - loss: 2.3195 - accuracy: 0.1114 - val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 2/12
469/469 [==============================] - 29s 61ms/step - loss: 2.3013 - accuracy: 0.1124 - val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 3/12
469/469 [==============================] - 30s 64ms/step - loss: 2.3013 - accuracy: 0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
Epoch 4/12
469/469 [==============================] - 29s 61ms/step - loss: 2.3013 - accuracy: 0.1124 - val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 5/12
469/469 [==============================] - 29s 61ms/step - loss: 2.2980 - accuracy: 0.1150 - val_loss: 2.2798 - val_accuracy: 0.1379
Epoch 6/12
469/469 [==============================] - 29s 61ms/step - loss: 2.2805 - accuracy: 0.1326 - val_loss: 2.2494 - val_accuracy: 0.1732
Epoch 7/12
469/469 [==============================] - 28s 61ms/step - loss: 2.2647 - accuracy: 0.1467 - val_loss: 2.2356 - val_accuracy: 0.1885
Epoch 8/12
469/469 [==============================] - 30s 64ms/step - loss: 2.2508 - accuracy: 0.1523 - val_loss: 2.2106 - val_accuracy: 0.1933
Epoch 9/12
469/469 [==============================] - 30s 63ms/step - loss: 2.2429 - accuracy: 0.1546 - val_loss: 2.2122 - val_accuracy: 0.1993
Epoch 10/12
469/469 [==============================] - 31s 66ms/step - loss: 2.2401 - accuracy: 0.1568 - val_loss: 2.2131 - val_accuracy: 0.2119
Epoch 11/12
469/469 [==============================] - 30s 64ms/step - loss: 2.2380 - accuracy: 0.1576 - val_loss: 2.2054 - val_accuracy: 0.2060
Epoch 12/12
469/469 [==============================] - 30s 65ms/step - loss: 2.2326 - accuracy: 0.1626 - val_loss: 2.1990 - val_accuracy: 0.2118
```

In [37]:
```
baseline_accuracy, noisy_scores
```

Out[37]:
```
(0.9923999905586243,
 [0.9923999905586243,
  0.9668999910354614,
  0.8327999711036682,
  0.4726000130176544,
  0.2117999941110611])
```

In [38]:
```
plt.figure()
plt.xlabel('Scale Parameter Setting (np.random.normal)')
plt.ylabel('Accuracy Score')
plt.plot(scale, noisy_scores)
plt.axhline(baseline_accuracy, color='purple') # baseline accuracy score
plt.title("CNN Accuracy Scores for different levels of noise")
plt.xticks(ticks=scale)
```

Out[38]:
```
([<matplotlib.axis.XTick at 0x166eec190>,
  <matplotlib.axis.XTick at 0x166eec1c0>,
  <matplotlib.axis.XTick at 0x161fbd3f0>,
  <matplotlib.axis.XTick at 0x164d562f0>,
  <matplotlib.axis.XTick at 0x166f316c0>],
 [Text(0.1, 0, '0.1'),
  Text(0.5, 0, '0.5'),
  Text(1.0, 0, '1.0'),
  Text(2.0, 0, '2.0'),
  Text(4.0, 0, '4.0')])
```



The purple, horizontal line, in the plot above represents the baseline (unmodified data's) accuracy score.

In relation to the neural network from last week's assignment, this model was more accurate in all categories. The convolutional neural network:

Had a higher overall accuracy score

Experienced a slower rate of decrease in accuracy score, as noise was added to the input.

Ended up having higher accuracy scores than last week's model in all noise categories.

In [ ]: