Neural Networks image recognition - MultiLayer Perceptron

Use both MLNN for the following problem.

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) (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.

`np.random.normal`

Parameters

loc

Mean (“centre”) of the distribution.

scale

Standard deviation (spread or “width”) of the distribution. Must be non-negative.

size

Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. If size is None (default), a single value is returned if loc and scale are both scalars. Otherwise, np.broadcast(loc, scale).size samples are drawn.

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
```

Multi Layer Neural Network

Trains a simple deep NN on the MNIST dataset. Gets to 98.40% test accuracy after 20 epochs (there is *a lot* of margin for parameter tuning).

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
```

In [4]:

```
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=RMSprop(),
              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])
```

...

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)](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.* *

In [31]:

```
import numpy as np

(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')

# add noise
noise = np.random.normal(0, 1, (60000, 784))
x_train_noise = x_train + noise

noise = np.random.normal(0, 1, (10000, 784))
x_test_noise = x_test + noise

x_train /= 255
x_test /= 255
x_train_noise /= 255
x_test_noise /= 255
```

In [32]:

```
# Compare images before and after adding noise
print(f"x_train[0]:\n{x_train[0]}")
print(f"x_train_noise[0]:\n{x_train_noise[0]}")
print(f"x_test[0]:\n{x_test[0]}")
print(f"x_test_noise[0]:\n{x_test_noise[0]}")
```

```
x_train[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.          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.          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.          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.          0.
```

In [26]:

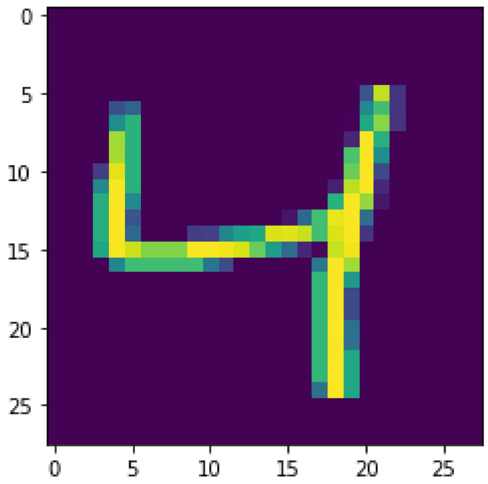
```
import matplotlib.pyplot as  plt
%matplotlib inline
```

In [33]:

```
plt.imshow(x_train[2].reshape(28, 28))
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x7f42f89e72d0>
```

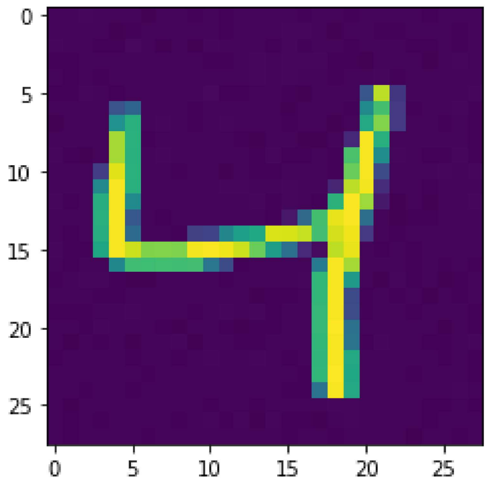


In [34]:

```
plt.imshow(x_train_noise[2].reshape(28, 28))
```

Out[34]:

```
<matplotlib.image.AxesImage at 0x7f42f89b6e90>
```



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

In [36]:

```
# with same parameters
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=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train_noise, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test_noise, y_test))
score_noise = model.evaluate(x_test_noise, y_test, verbose=0)
print('Test loss with adding noise:', score_noise[0])
print('Test accuracy with adding noise:', score_noise[1])
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 512)               401920

 dropout_8 (Dropout)         (None, 512)               0

 dense_13 (Dense)            (None, 512)               262656

 dropout_9 (Dropout)         (None, 512)               0

 dense_14 (Dense)            (None, 10)                5130

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 10s 20ms/step - loss: 0.2448 - accuracy:
0.9248 - val_loss: 0.1014 - val_accuracy: 0.9670
Epoch 2/20
469/469 [==============================] - 9s 20ms/step - loss: 0.1003 - accuracy:
```

```
0.9700 - val_loss: 0.0985 - val_accuracy: 0.9697
Epoch 3/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0746 - accuracy:
0.9774 - val_loss: 0.0960 - val_accuracy: 0.9708
Epoch 4/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0595 - accuracy:
0.9818 - val_loss: 0.0847 - val_accuracy: 0.9772
Epoch 5/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0486 - accuracy:
0.9854 - val_loss: 0.0728 - val_accuracy: 0.9808
Epoch 6/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0423 - accuracy:
0.9872 - val_loss: 0.0749 - val_accuracy: 0.9816
Epoch 7/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0380 - accuracy:
0.9888 - val_loss: 0.0740 - val_accuracy: 0.9830
Epoch 8/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0312 - accuracy:
0.9905 - val_loss: 0.0795 - val_accuracy: 0.9825
Epoch 9/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0289 - accuracy:
0.9914 - val_loss: 0.0837 - val_accuracy: 0.9838
Epoch 10/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0262 - accuracy:
0.9920 - val_loss: 0.1004 - val_accuracy: 0.9815
Epoch 11/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0279 - accuracy:
0.9921 - val_loss: 0.0851 - val_accuracy: 0.9832
Epoch 12/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0234 - accuracy:
0.9934 - val_loss: 0.0938 - val_accuracy: 0.9827
Epoch 13/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0224 - accuracy:
0.9939 - val_loss: 0.0911 - val_accuracy: 0.9839
Epoch 14/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0208 - accuracy:
0.9942 - val_loss: 0.0945 - val_accuracy: 0.9846
Epoch 15/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0208 - accuracy:
0.9941 - val_loss: 0.1121 - val_accuracy: 0.9837
Epoch 16/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0198 - accuracy:
0.9945 - val_loss: 0.0988 - val_accuracy: 0.9848
Epoch 17/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0197 - accuracy:
0.9950 - val_loss: 0.1079 - val_accuracy: 0.9840
Epoch 18/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0171 - accuracy:
0.9953 - val_loss: 0.1150 - val_accuracy: 0.9844
Epoch 19/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0178 - accuracy:
0.9955 - val_loss: 0.1219 - val_accuracy: 0.9838
Epoch 20/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0164 - accuracy:
0.9957 - val_loss: 0.1267 - val_accuracy: 0.9837
Test loss with adding noise: 0.12669143080711365
Test accuracy with adding noise: 0.9836999773979187
Test loss: 0.12676534056663513
Test accuracy: 0.9828000068664551
```

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 [42]:

```
scales = [0.1, 0.5, 1.0, 2.0, 4.0]
train_scores = [[0] for _ in range(len(scales))]
validation_scores = [[0] for _ in range(len(scales))]
for i in range(len(scales)):
    (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')

    # add noise
    noise = np.random.normal(0, scales[i], (60000, 784))
    x_train_noise = x_train + noise

    noise = np.random.normal(0, scales[i], (10000, 784))
    x_test_noise = x_test + noise

    x_train /= 255
    x_test /= 255
    x_train_noise /= 255
    x_test_noise /= 255

    # build model
    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=RMSprop(),
                  metrics=['accuracy'])

    history = model.fit(x_train_noise, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test_noise, y_test))
    train_scores[i] = history.history['accuracy']
    validation_scores[i] = history.history['val_accuracy']
    score_noise = model.evaluate(x_test_noise, y_test, verbose=0)
    print('Test loss with adding noise:', score_noise[0])
    print('Test accuracy with adding noise:', score_noise[1])
```

```
Model: "sequential_6"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_18 (Dense)            (None, 512)               401920

 dropout_12 (Dropout)        (None, 512)               0

 dense_19 (Dense)            (None, 512)               262656

 dropout_13 (Dropout)        (None, 512)               0

 dense_20 (Dense)            (None, 10)                5130

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

In [47]:

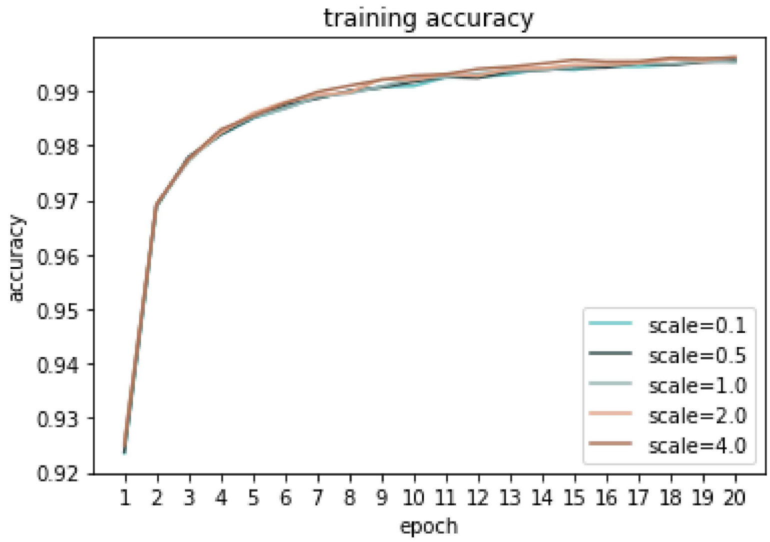
```
train_scores[0]
```

Out[47]:

```
[0.9235333204269409,
 0.9687166810035706,
 0.9776166677474976,
 0.9819166660308838,
 0.9849500060081482,
 0.9868166446685791,
 0.9888499975204468,
 0.9897500276565552,
 0.9907000064849854,
 0.9908833503723145,
 0.9924666881561279,
 0.9925833344459534,
 0.9929999709129333,
 0.9940666556358337,
 0.9938666820526123,
 0.9945166707038879,
 0.9944666624069214,
 0.9947666525840759,
 0.9956499934196472,
 0.9955000281333923]
```

In [53]:

```
# plot result
colors = ['#5EC2C2', '#324B4B', '#95B1B0', '#E2A589', '#A97157']
for i in range(len(train_scores)):
    score = train_scores[i]
    plt.plot(list(range(1, len(score)+1)), score, color=colors[i], label=f"scale={scales[i]}")
plt.legend()
plt.xlabel("epoch")
plt.xticks(list(range(1, len(score)+1)))
plt.ylabel("accuracy")
plt.title("training accuracy")
plt.show()
```



In [54]:

```
# plot result
colors = ['#5EC2C2', '#324B4B', '#95B1B0', '#E2A589', '#A97157']
for i in range(len(validation_scores)):
    score = validation_scores[i]
    plt.plot(list(range(1, len(score)+1)), score, color=colors[i], label=f"scale={scales[i]}")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.xticks(list(range(1, len(score)+1)))
plt.title("validation accuracy")
plt.show()
```