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

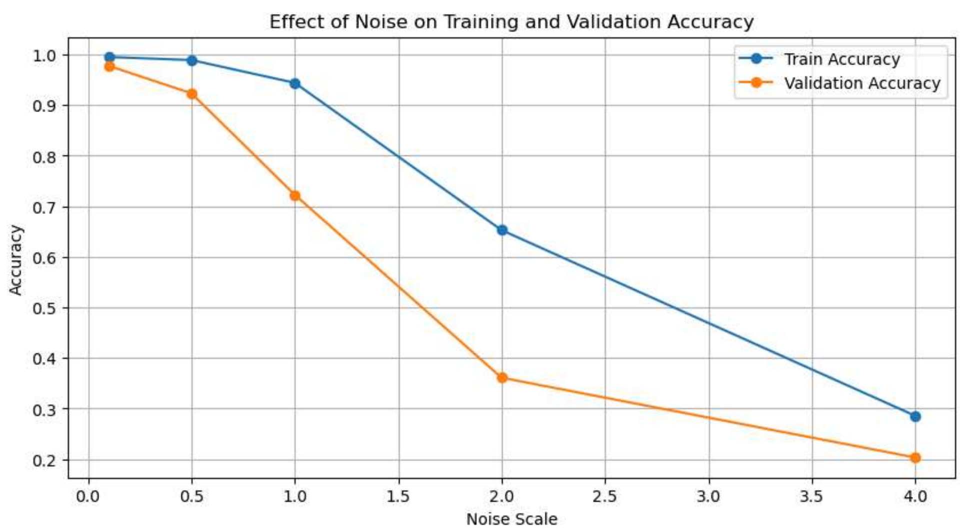
In [15]:

```
# Importing necessary packages
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# Loading and preprocessing the MNIST dataset
(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

num_classes = 10

# Converting 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)

# Creating function to add noise to the images
def add_noise(x, scale):
    noise = np.random.normal(loc=0.0, scale=scale, size=x.shape)
    x_noisy = x + noise
    x_noisy = np.clip(x_noisy, 0., 1.)
    return x_noisy

# Creating function to create the MLP model
def create_mlp_model(input_shape, num_classes):
    model = Sequential()
    model.add(Dense(512, activation='relu', input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    return model

# Creating function to plot a grid of images
def plot_images(images, titles, ncols=5, nrows=5):
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10, 10
    for ax, img, title in zip(axes.ravel(), images, titles):
        ax.imshow(img.reshape(28, 28), cmap='gray')
        ax.set_title(title)
        ax.axis('off')
    plt.subplots_adjust(wspace=0.5, hspace=0.5)
    plt.show()

# Printing out images and noisy images
def visualize_images(x, scale):
    x_noisy = add_noise(x, scale)

    # Select a subset of images to display
    indices = np.random.choice(x.shape[0], 25, replace=False)
```

```
    original_images = x[indices]
    noisy_images = x_noisy[indices]

    # Titles for visualization
    titles_original = ['Original'] * len(indices)
    titles_noisy = [f'Noisy (scale={scale:.1f})'] * len(indices)

    # Plot original images
    plot_images(original_images, titles_original, ncols=5, nrows=5)

    # Plot noisy images
    plot_images(noisy_images, titles_noisy, ncols=5, nrows=5)

# Defining noise scales for visualization
scales = [0.1, 0.5, 1.0, 2.0, 4.0]

# Visualizing images with different noise scales
for scale in scales:
    print(f'Visualizing images with noise scale {scale}...')
    visualize_images(x_train, scale)

# Training and evaluation
batch_size = 128
epochs = 20

results = []

for scale in scales:
    # Add noise to training and test data
    x_train_noisy = add_noise(x_train, scale)
    x_test_noisy = add_noise(x_test, scale)

    # Create and compile the model
    model = create_mlp_model((784,), num_classes)
    model.compile(loss='categorical_crossentropy', optimizer='adam', m

    # Train the model
    history = model.fit(x_train_noisy, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test_noisy, y_test))

    # Evaluate the model
    score = model.evaluate(x_test_noisy, y_test, verbose=0)
    print(f'Scale {scale} - Test loss: {score[0]}, Test accuracy: {sco

    # Store results
    results.append((scale, history.history['accuracy'][-1], history.hi

# Convert the results to a numpy array for easier plotting
results = np.array(results)

# Plotting the results
plt.figure(figsize=(10, 5))
plt.plot(results[:, 0], results[:, 1], marker='o', label='Train Accura
plt.plot(results[:, 0], results[:, 2], marker='o', label='Validation A
```

```
plt.xlabel('Noise Scale')
plt.ylabel('Accuracy')
plt.title('Effect of Noise on Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

```
- loss: 1.9882 - val_accuracy: 0.2033 - val_loss: 2.2195
Scale 4.0 - Test loss: 2.219526529312134, Test accuracy: 0.203
29999923706055
```




```
**Interpretation:** As the graph showcases, the accuracy score of
image detection steadily declines the more noise is added to each
image. This is further confirmed by the images becoming increasingly
unrecognizable with the addition of noise, where the numbers
presented are hardly identifiable by the human eye.
```