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]:

```
# Loading packages
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 matplotlib.pyplot as plt
```

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]:

```
# Load and preprocess the data
img_rows, img_cols = 28, 28
(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') / 255
x_test = x_test.astype('float32') / 255
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

In [3]:

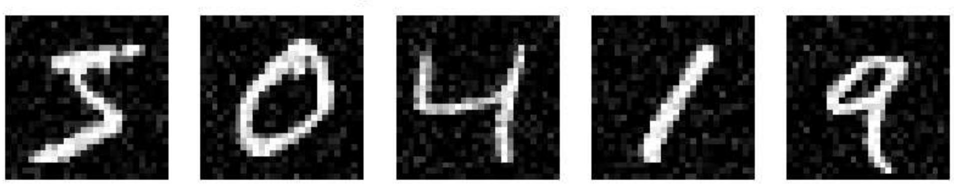
```
# Function to add random noise to images
def add_noise(images, scale):
    noise = np.random.normal(loc=0.0, scale=scale, size=images.shape)
    noisy_images = images + noise
    return np.clip(noisy_images, 0., 1.)

# Function to preview noisy images
def preview_noisy_images(images, scale, num_images=5):
    noisy_images = add_noise(images, scale)
    plt.figure(figsize=(10, 2))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(noisy_images[i].reshape(28, 28), cmap='gray')
        plt.axis('off')
    plt.suptitle(f'Images with noise scale {scale}')
    plt.show()

# Function to build and compile the ConvNet model
def build_model(input_shape, num_classes):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_
    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, optimize
    return model

# Function to train the model and return accuracy
def train_and_evaluate(model, x_train, y_train, x_test, y_test, batch_
    history = model.fit(x_train, y_train, batch_size=batch_size, epoch
    score = model.evaluate(x_test, y_test, verbose=0)
    return history, score
```

In [4]:

```python
# Parameters
batch_size = 128
num_classes = 10
epochs = 12

# Training and evaluation without noise
model = build_model(input_shape, num_classes)
history_no_noise, score_no_noise = train_and_evaluate(model, x_train,
print('Test accuracy without noise:', score_no_noise[1])

# Noise scales to test
noise_scales = [0.1, 0.5, 1.0, 2.0, 4.0]
accuracy_train = []
accuracy_val = []

for scale in noise_scales:
    x_train_noisy = add_noise(x_train, scale)
    x_test_noisy = add_noise(x_test, scale)

    # Preview noisy images
    preview_noisy_images(x_train, scale)

    model = build_model(input_shape, num_classes)
    history, score = train_and_evaluate(model, x_train_noisy, y_train,

    accuracy_train.append(history.history['accuracy'])
    accuracy_val.append(history.history['val_accuracy'])

    print(f'Test accuracy with noise scale {scale}:', score[1])
```

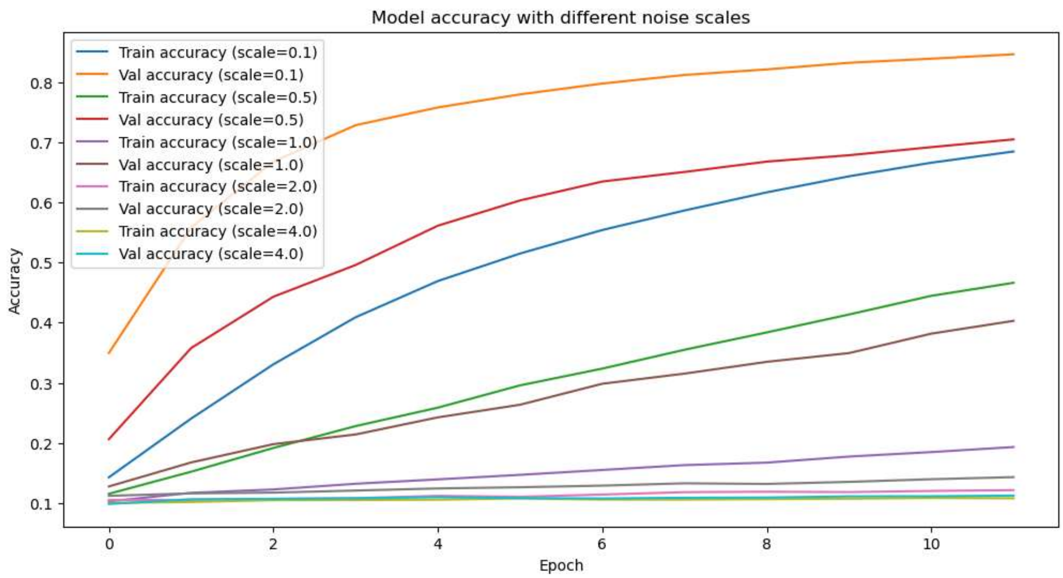
```
Epoch 12/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 48s 103ms/step - accuracy: 0.6923
- loss: 1.0086 - val_accuracy: 0.8350 - val_loss: 0.7423
Test accuracy without noise: 0.8349999785423279
```

Images with noise scale 0.1



```
Epoch 1/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 92ms/step - accuracy: 0.1234
- loss: 2.2994 - val_accuracy: 0.3495 - val_loss: 2.2624
Epoch 2/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 37s 79ms/step - accuracy: 0.2158
- loss: 2.2603 - val_accuracy: 0.5581 - val_loss: 2.2133
Epoch 3/12
```

In [5]:
```
# Plotting results
plt.figure(figsize=(12, 6))
for i, scale in enumerate(noise_scales):
    plt.plot(accuracy_train[i], label=f'Train accuracy (scale={scale})
    plt.plot(accuracy_val[i], label=f'Val accuracy (scale={scale})')
plt.title('Model accuracy with different noise scales')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```



Interpretation: Similar to last week with MultiLayer Perceptron, the ConvNet neural network suffered in performance withthe addition of more noise, with the models with the lowest amount of noise (0.1) reaching the highest accuracy (~0.80). Additionally, for most noise levels, the accuracy level improved with successive epochs. However, for stronger levels of noise, (2.0+), successive epochs made little to no improvement in overall model accuracy.

```
# PRE-SUPPLIED CODE - did not work

batch_size = 128
num_classes = 10
epochs = 12

# 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(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'])

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])
```