UNI-ID: SVW2112

1.What is one-hot encoding? Why is this important and how do you implement it in keras?

One-hot encoding is used to encode integer data into a vector where one element is one ("hot") and the other elements are zero. The use of the word hot comes from electronics where one typically represents a circuit being on (hot), and zero represents off. One-hot encoding is important because the final layer of many neural networks outputs a vector containing a probability for each class, which can then be converted to a prediction in the form of a one-hot vector by setting the element with the highest probability to one and the others to zero. In the Keras implementation, the dataset labels are converted to one-hot form using the to_categorical function.

2.What is dropout and how does it help overfitting?

Dropout layers randomly sets a certain percentage of their inputs 0 during training. This helps prevent overfitting by taking folds of the data, which results in a more diverse set of training data and reduces the risk of overfitting to any one dataset. Techniques such as bootstrapping and k-fold cross-validation also use a similar principle to reduce overfitting.

3.How does ReLU differ from the sigmoid activation function?

The ReLU and sigmoid activation functions have different output profiles, particularly for inputs near or below zero. For an input x, ReLU is defined as max(0, x), which zeros out all inputs less than zero and rises linearly for positive inputs. On the other hand, the sigmoid is defined as 1/(1 + e^-x), which has an "ess" shape that does not entirely zero out negative inputs and plateaus for higher values of x. In practice, either of these activation functions can work well overall. The sigmoid has the advantage that neurons never "die" during training, unlike ReLU where this can occur in the input falls below zero. On the other hand, ReLU is less computationally expensive than the sigmoid and often works just as well in practice.

4.Why is the softmax function necessary in the output layer?

Softmax converts the vector output by the last layer into class probabilities based on the relative scale of each value in the vector. This is done by taking the exponential of each element, and then averaging each element. The softmax function is needed to generate class probabilities from the numbers outputted in the final layer.

5.What are the dimensions of the outputs of the convolution and max pooling layers?

In [1]:
```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
import tensorflow as tf
```

In [2]:
```
# Make model
model1 = Sequential()
model1.add(Conv2D(16, (5, 5), input_shape=(100, 100, 1)))
model1.compile('adam', loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

# Generate input (100 x 100 image)
X1 = tf.convert_to_tensor(np.zeros((1, 100,100,1)))

# Output shape
y1 = model1.predict(X1)
print(y1.shape)
```

```
1/1 [==============================] - 1s 523ms/step
(1, 96, 96, 16)
```

If the input image has size (n, n, 1), the output of a convolutional layer with f filters and a kernel size of (k, k) will be (n - k + 1, n - k + 1, f). In this example, n = 100, f = 16, and k = 5, so the output is (96, 96, 16).

In [3]:
```
# Make model
model2 = Sequential()
model2.add(Conv2D(16, (5, 5), input_shape=(100, 100, 1)))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.compile('adam', loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

# Generate input (100 x 100 image)
X2 = tf.convert_to_tensor(np.zeros((1, 100,100,1)))

# Output shape
y2 = model2.predict(X2)
print(y2.shape)
```

```
1/1 [==============================] - 0s 149ms/step
(1, 48, 48, 16)
```

Inputting data of shape (q, q, f) to a (p, p) pooling layer will result in an output of size (q / p, q / p, f). In this example, q = 96, f = 16, and p = 2, so the output is (48, 48, 16).