COMP8260 AI Systems Implementation
Class 4: Exploring Convolutional Neural Networks

The purpose of this class is to design both a Convolutional and non-convolutional Neural Network and subsequently compare their relative effectiveness on a pattern classification task.

This exercise is similar to, but not the same as, examples from chapters 2 and 7 of the book Deep Learning with Python, Second Edition.

N.B. Keras reference may be found at https://keras.io/api/

Preparation

Skeleton Jupyter notebooks for the following tasks may be found on raptor at \\raptor.kent.ac.uk\exports\courses\comp8260 under a folder called class 4. Please copy this into your space on \\jupyter.kent.ac.uk\exports\home before starting. Copies may also be found on Moodle.

Study the skeleton code before proceeding. The tasks begin by constructing and investigating a non-convolutional network to identify hand written numerals, this time represented by the standard grey-scaled images of the MNIST dataset. They continue by constructing convolutional networks of various configurations to undertake the same task. The two may then be compared.

Task 1

Firstly, define a non-convolutional network with three layers by supplying code in the missing cell in the skeleton Jupyter notebook.

Begin by creating a three layer network (**Dense** layers) with the **sigmoid** activation function for the first two layers and a 10 output final layer with **softmax** activation (we want to know which one of the ten classes is most likely). The system is to be designed in **Keras** using the Functional API approach as outlined in lecture 5.

To design the system, first we need two variables called **inputs** and **outputs**.

Inputs is defined for you and represents the size of the input character images from the MNIST dataset we are using.

Outputs is defined by calling the layer definition methods of the **Layers** class for each layer in turn. For this first task, we are just going to employ fully connected **Dense** layers.

To define the first layer, call **layers.Dense** with parameters for the size of the output and activation function presenting the defined **inputs** variable as the input to the layer (it is the first layer).

The second layer is defined identically to the first layer except of course it takes the outputs of the first layer as its input.

The third layer has 10 outputs (one for each class) and uses the **softmax** activation which outputs the class with highest probability. Abstractly it appears as shown in the skeleton notebook. You need to fill in the missing sections shown by **?????????.**

Finally define the model by calling the **keras.model** method with the now defined two parameters for the **inputs** and **outputs**

keras.Model(inputs=??, outputs=??)

Now we need to follow the sequence of compiling the model, training it and making predications.

Firstly compile the model by calling **model.compile** with three arguments, defining the optimiser "**adam**" (adaptive moments) the loss function "**sparse_categorical_crossentropy**" and metric "**accuracy**") (see https://keras.io/api/ for details of these and other alternatives you may wish to try)

We next need to train the model. This is achieved by calling the **model.fit** method with the training and test images as arguments. It also requires the number epochs (number of times all training patterns are presented) and the **batch_size** (number of samples per gradient update). Suggested start values are 5 epochs and batch size of 32 but see what happens if you vary these.

Finally we can make some predictions using the test data by calling the **model.predict** method. Note that this returns an array of predictions. Each entry gives ten probability values which represents the network's opinion of how likely each test pattern is to be an example of one of the ten digits respectively. The largest value (which you can deduce by inspection or using the **argmax** method) naturally represents the class it considers the most likely.

The **evaluate** method can be used to generate the percentage accuracy of the entire test set.

Examine the accuracy as the number of **epochs** increases. What do you notice with regard to overall performance? (the accuracy is displayed by the **fit** method call)

Try removing one of the hidden layers? What is the effect?

Try replacing the **adam** optimiser with something else, such as **rmsprop**. Is there a difference?

Task 2

Let's move on and design a convolutional neural network (CNN) for the same problem.

The general principle is the same as above except this time not all the layers will be fully connected **(Dense**) layers. In fact we will have three additional layers, Convolutional, Pooling and Flattening layers as introduced in lecture 3.

Initially, let's try a network with just one convolutionl layer and one dense layer with **softmax** activation function.

Create the convolutional layer by using the **Conv2D** method with 16 filters initially (this method generates random filters which may or may not include the examples shown in lecture 4). Parameters that must be defined are the number of **filters**, **kernel size** (suggest use 3 as for the examples in lecture 3 but you may try other values) and activation function (suggest the rectified linear unit **relu**)

Before being fed into the fully connected layer, the outputs of the convolutional later will need flattening by calling the **layers.flatten** method.

Feed this into the second fully connected **Dense** layer using the **softmax** activation function which will appear as for the final layer of the network defined in Task 1.

Experiment by changing the number of filters

Task 3

For the final task, let's increase the number of convolutional layers. Modify the networks by adding an additional Conv2D layer to the model. Recall from lecture 3, that you ideally need to place a pooling layer between the two convolutional layers which you define via **layers.MaxPooling2D**. Try initially a pool size of 2. This is passed into the layer as a parameter.

Does it make a difference?

Investigate now with further convolutional layers, differing numbers of filters (suggest 64 and 128 for second and third layer respectively but again investigate) and differing pool sizes. Investigate other options given in at https://keras.io/api/ and ensure you are comfortable with the general principles of its operation.