TRINITY COLLEGE DUBLIN

School of Computer Science and Statistics

Week 8 Assignment CS7CS4/CSU44061 Machine Learning

Rules of the game:

- **Its ok to discuss with others, but do not show any code you write to others. You must write answers in your own words and write code entirely yourself. All submissions will be checked for plagiarism.**
- **Reports must be typed (no handwritten answers please) and submitted as a separate pdf on Blackboard (not as part of a zip file please).**
- **Important: For each problem, your primary aim is to articulate that you understand what you're doing - not just running a program and quoting numbers it outputs. Long rambling answers and "brain dumps" are not the way to achieve this. If you write code to carry out a calculation you need to discuss/explain what that code does, and if you present numerical results you need to discuss their interpretation. Generally most of the credit is given for the explanation/analysis as opposed to the code/numerical answer. Saying "see code" is not good enough, even if code contains comments. Similarly, standalone numbers or plots without further comment is not good enough.**
- **When your answer includes a plot be sure to (i) label the axes, (ii) make sure all the text (including axes labels/ticks) is large enough to be clearly legible and (iii) explain in text what the plot shows.**
- **Include the source of code written for the assignment as an appendix in your submitted pdf report. Also include a separate zip file containing the executable code and any data files needed. Programs should be running code written in Python, and should load data etc when run so that we can unzip your submission and just directly run it to check that it works. Keep code brief and clean with meaningful variable names etc.**
- **Reports should typically be about 5 pages, with 10 pages the upper limit (excluding appendix with code). If you go over 10 pages then the extra pages will not be marked.**
- **Note: In this assignment there is no need to use cross-validation to select hyperparameters - deep learning needs too much computing power for that to be practical.**

Assignment

In this assignment you'll take a closer look at convolutional networks.

(i) (a) Using vanilla python (no use of sklearn, keras or the like) implement a function that takes as input (i) an $n \times n$ array and (ii) a $k \times k$ kernel, convolves the kernel to the input array and returns the result

(b) Select an image (e.g. of a geometric shape, there are plenty on the internet). Make sure the image is not too large, e.g. 200 by 200 pixels (if its too big you can resize it to be smaller). In python load the image as three RGB arrays and select one of these arrays to work with. This can, for example, be done using the PIL package and numpy:

```
import numpy as np
from PIL import Image
im = Image.open('tcd.jpg')
rgb = np.array(im.convert('RGB'))
r=rgb[:,:,0] # array of R pixels
Image.fromarray(np.uint8(r)).show()
```

Now input this array to your convolution function from (a) and display the output when the following two kernels are used:

$$kernel1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad kernel2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(ii) We'll now use keras to build a convolutional network for classification of images in the CIFAR10 image dataset. This dataset contains 50K small 32×32 colour images plus labels. It's a much harder to learn dataset than the MNIST handwritten digits one we used in the lectures, achieving prediction accuracy above 60% on hold-out test data or is not so easy (the state of the art is 90% accuracy, but that needs a bigger network than we'll use here).

Download the assignment python code from `https://www.scss.tcd.ie/Doug.Leith/CSU44061/week8.py`. This uses keras to load the CIFAR10 dataset. It then builds a convolutional network, trains it and evaluates the confusion matrix of its predictions.

(a) Inspect the code you downloaded and write down the architecture of the ConvNet it uses (i.e what are the layers and for each conv layer what size of kernel is used, how many output channels are there, and so on).

(b) Run the code you downloaded and note the output. Note: training takes a while, about 1 min on my macbook with 5K training data points, so some patience is needed (if your machine has an NVIDIA/cuda GPU then you're in luck and should be able to run things faster). If it takes much longer than 1 min you should try to find a stronger machine.

(i) How many parameters does keras say this model has? Which layer has the most parameters, and why? You should see an accuracy of about 48% on the test data. How does the performance on the test data compare with the performance on the training data. Compare this performance against a simple baseline e.g. always predicting the most common label.

(ii) During training keras periodically evaluates the prediction accuracy against 10% of the training data that is held-out as test data, and this together with the accuracy on the training data is stored in the "history" variable in the code and plotted once training finishes. What diagnostics, if any, about over/under-fitting can you deduce from this plot?

(iii) Now explore the effect of increasing the amount of training data on the performance of the ConvNet after training. Train the ConvNet using 5K, 10K, 20K and 40K training data points (the CIFAR10 dataset has 50K data points in total) while keeping the number of training epochs fixed at 20. Record the time taken for each run. How does the prediction accuracy on the training and test data vary with the amount of training data? Look at the plot of the "history" variable for each run, how does it vary with the amount of training data used and what does this indicate about over/under-fitting? How does the time taken to train the network vary with the amount of training data used?
Note: If training with 20K and 40K points takes more than about 10 mins on your machine then there's no need to run these cases (this assignment will make your computer work quite hard!).

(iv) The ConvNet uses L_1 regularisation on the softmax output layer. The weight parameter of the L_1 is 0.001. Using 5K training data points vary this weight parameter (try bigger and smaller values, including 0) and discuss its effect on the prediction accuracy on the training and test data. Compare/contrast with increasing the amount of training data, which is more effective at managing overfitting?

(c) (i) The ConvNet in the code you downloaded uses strides to downsample. Modify the ConvNet to use max-pooling i.e. replace the 16 channel strided layer with a 16 channel same layer followed by a (2,2) max-pool layer, and similarly for the 32 channel strided layer. Leave the softmax output layer untouched.

(ii) Evaluate the performance of this ConvNet. How many parameters does keras say this ConvNet has? For 5K training data point how does the time taken to train the network and the prediction accuracy on the training and test data compare with that of the original network? If the training time has changed, why do you think that's happened?

(d) **Optional**. If you are interested in playing around a bit more with the ConvNet architecture, try making it thinner and deeper. For example, try a network with these layers:

```
model.add(Conv2D(8, (3,3), padding='same', input_shape=x_train.shape
    [1:], activation='relu'))
model.add(Conv2D(8, (3,3), strides=(2,2), padding='same', activation='
    relu'))
model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='
    relu'))
model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='
    relu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax',kernel_regularizer=
    regularizers.l1(0.0001)))
```

When trained for long enough on the full dataset this should achieve prediction accuracy above 70%. How does adding more layers layers affect the complex trade-off between prediction performance, over/under-fitting, the amount of training data needed and the time taken to train the network.