Deep learning lab

Course: Statistics and Prediction in the Pharmaceutical Sciences
Date: 21st May 2019

By Phil Harrison
philip.harrison@farmbio.uu.se
Dept. of Pharmaceutical bioinformatics
Uppsala University

Assisted by Staffan Arvidsson

Preamble

You are free to work in groups of 1:3 people on the exercises herein. In this lab you'll learn how to implement your own convolutional neural network (CNN) for classifying cell phenotypes. You'll be using the open-source machine learning framework `TensorFlow` (https://www.tensorflow.org) via `Keras` (https://keras.io) through the `R` interface to `Keras` (https://keras.rstudio.com/index.html). `TensorFlow` was developed by the Google Brain team and is today one of the most widely used machine learning frameworks in research and industry and `Keras` is the most popular higher-level API that runs atop `TensorFlow`.

CNNs - using convolutions instead of fully connected layers - have proven to be very successful for image recognition tasks. By convolving filters on the input layer and outputting the results to the next layer, the CNN "detects" (or learns) features at different levels of abstraction throughout the network. With lower-level abstractions (like edges and blobs) in the early layers, and higher-level abstractions (like cells) in deeper layers. Figure 1 shows the LeNet inspired CNN that you will shortly turn into `Keras` code (in the main exercise below). It has 3 convolutional layers, 3 max pooling layers, and 2 final fully connected layers.

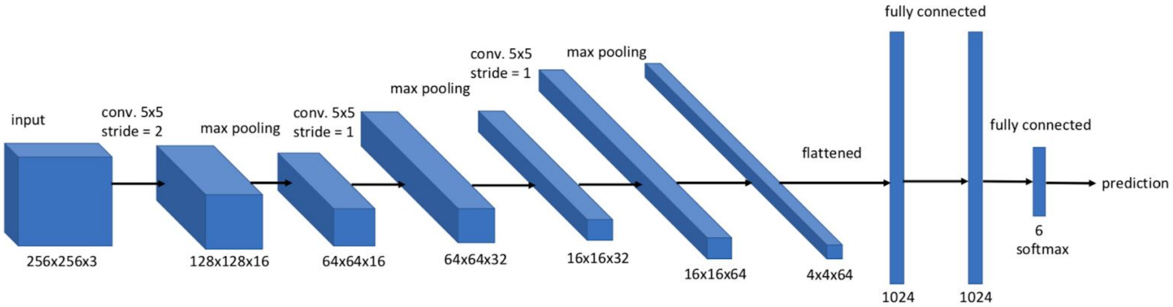

Figure 1: LeNet inspired CNN that you will implement.

If you're getting stuck or hit problems, don't hesitate to use Google or similar to search for answers. There's plenty of information out there regarding `Keras` and `TensorFlow` implementations. It's very useful to read about the different `Keras` functions and check out the examples and tutorial on the `R` interface to `Keras` link given above. I can also strongly recommend, for those of you who wish to learn more, the book ‘Deep learning with R’ (https://www.amazon.com/Deep-Learning-R-Francois-Chollet/dp/161729554X) written by the creator of `Keras` François Chollet.

Preparatory

1. download 'StatPred_2019' from github https://github.com/pharmbio/StatPred_2019/tree/master
2. once you’ve downloaded and unzipped the folder, remove the '-master' suffix from the folder's name and move the folder to an appropriate location on your computer

Warm-up exercise: Train a simple CNN on the MNIST data

In the first exercise you’ll fit a basic CNN to the MNIST data using conolutional, pooling and dropout layers. MNIST (http://yann.lecun.com/exdb/mnist/) is a database of handwritten digits containing 60,000 training images and 10,000 test images. The ten digits (0:9) have each been size-normalized and centered in a 28 x 28 pixel image. MNIST has become a classic benchmark dataset for machine learning and it comes pre-installed in `Keras`.

The code for the exercise is in `StatPred_2019_CNN_Lab_MNIST.r` . You don’t have to modify any of the code in this file, just study it and try to understand what each part is doing and think about the output that is given. Especially pay attention to how the model is defined (after the commented code line ‘`# define model`’) and how that relates to the model summary that is subsequently printed to the console. This part will help you out in the main exercise where you’ll need to write your own code for the model definition.

Main exercise: Classification of cell morphological changes with CNNs

The quantification and identification of cellular phenotypes from high-content microscopy images has proven to be very useful for understanding biological activity in response to different drug treatments. The traditional approach has been to use classical image analysis to quantify changes in cell morphology, which requires several nontrivial and independent analysis steps. Recently, CNNs have emerged as a compelling alternative, offering good predictive performance and the possibility to replace traditional workflows with a single network architecture.

In this exercise you’ll fit a CNN to a mechanisms of action (MoA) dataset. Specifically the MoA dataset you’ll be using contains MCF-7 breast cancer cell images available from the BBBC (https://data.broadinstitute.org/bbbc/BBBC021/). We used the images from this dataset that had been identified as clearly having 1 out of the 12 primary MoAs, and thus were annotated with a label, in a recently published paper (Kensert, Harrison & Spjuth, 2019). The 12 different MoAs represent a wide cross section of morphological phenotypes (Figure 2). In this paper we extracted a total of 38 compounds and 103 treatments (compound–concentration pairs), summing up to 1208 images. In this exercise, however, you will use only a subset of this data - for the 6 most common MoAs - so that the models you build will run in a reasonable timeframe on the lab CPUs. Each image is of size 256 x 256 x 3, where the 3 represents the number of image channels.

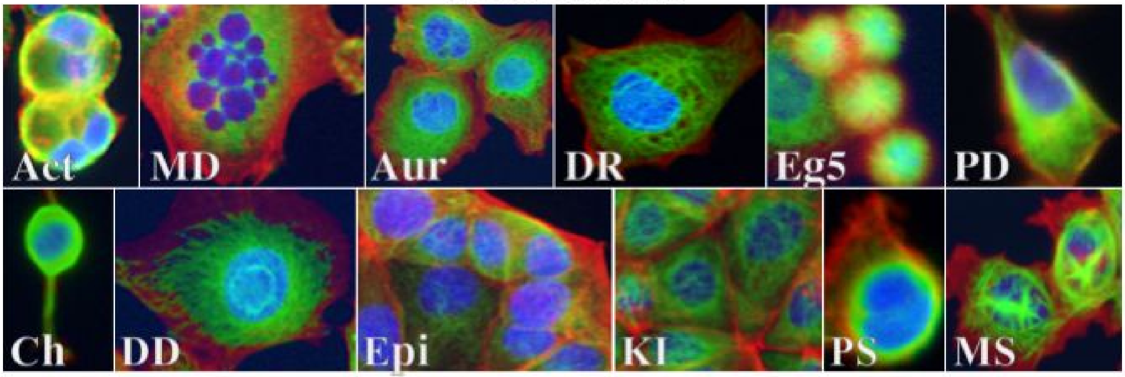

Figure 2: The different MoAs in the MoA dataset. Images were cropped and processed from the original training images. Act = actin disruption; MD = microtubule destabilization; Aur = aurora kinase inhibition; DR = DNA replication; Eg5 = Eg5 inhibition; PD = protein degradation; Ch = cholesterol lowering; DD = DNA damage; Epi = epithelial; KI = kinase inhibition; PS = protein synthesis; MS = microtubule stabilization.

The code for the exercise is in `StatPred_2019_CNN_Lab_MOA.r` . For many parts of the code we've implemented things for you, don't worry too much about those (although by all means read through them and try to get a feel for how they work/what they're doing).

Task 1: Add code after the line '`# define model`'

Your task is to define the model architecture as shown in Figure 1 above. This figure is also given in 'LeNet_inspired_CNN_for_MOA.png' so you can study it more carefully. The code to define the model in the MNIST exercise above, together with what you've learnt in the lecture, should help you with this task.

Once you feel you have accurately defined the model run the code line:

```
summary(model)
```

Do the output shapes given in the model summary correspond to those given in the figure for the model? You should have 1,121,062 parameters in the model.

In the first layer you'll see the number of parameters/weights to estimate/train is 1216. Can you work out how this number was calculated?

Compile and train your model. You're running for 30 epochs, each epoch might take 10-30 seconds depending on the speed of your computer… maybe time to go grab a cup of tea/coffee?

We did not create a hold-out test set for this exercise, although in practice you would want to do that! Here we will simply concern ourselves with the final accuracy for the validation data. What was your final validation accuracy (val_acc) on epoch 30? This was probably somewhere between 0.7 and 0.8. Hence, you will not get the same results as your friends on the computer next to you. This variability in the results is due to the inherent stochasticity in the way the weights are initialized, in the gradient descent algorithm (different runs will find different local minima in the loss surface), in the way the training and validation datasets are split and so forth. This is also a bigger problem when the datasets are small, as they are for this MOA example. We purposefully gave only a subset of the MOA data so that the models would run in a reasonable time frame on a CPU. Note if you had access to a GPU the models would run **considerably** faster!

Task 2: Change/extend the model from task 1

Try changing the model architecture in some way to see if you can get improved validation performance. You could try adding data augmentation and/or dropout for example. Data augmentation can be done 'on the fly' as the images are read in by the 'flow_images_from_directory' function, all you need to do is extend the 'image_data_generator' to perform augmentations in addition to the rescaling to 1/255 that it's already doing (see https://keras.rstudio.com/reference/image_data_generator.html). For dropout, you already saw how to code that in the MNIST example.

There are also other - plenty of other - changes you could try to make a model that performs better including changes to the number of layer, the number of kernels, the kernel sizes, the size of strides to take, alternate regularization techniques (besides data augmentation and dropout), different optimizers, etc. In different circumstances different solutions can work better than others. Unfortunately there are not so many hard and fast rules in deep learning. Experimentation, imagination and even luck can make developing deep learning models feel more like an art than a science sometimes...

Note that to be sure if a change you implemented really improved the accuracy you would need to run all the models multiple times and take the average accuracy across the runs. This process again is made much simpler with a GPU. For small datasets cross-validation is also a preferred method for evaluation. Using the entire MoA dataset (12 classes), transfer learning, data augmentation and more advanced network architectures (specifically the ResNet and Inception architectures introduced during the lectures) a test accuracy of 95-97% can be achieved (Kensert, Harrison & Spjuth, 2019).

Reference

Kensert, A., Harrison, P.J., and Spjuth, O. (2019). Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes. SLAS DISCOVERY: Advancing Life Sciences R&D 2472555218818756.