Lab on Convolutional NNs (CNN)

[ credits to: A. Geron, “Hands-On Machine Learning With Scikit-Learn and Tensorflow” ]
Convolutional Neural Networks (CNNs)

CNNs emerged from the study of the brain’s visual cortex

- not a new technique: used in image recognition since the 80’s
- only recently: exponential increase in computational power + amount of training data
- \(\Rightarrow\) boost for training DNNs!

CNNs have objectively managed to achieve superhuman performance on some complex visual tasks. They power:

- image search services
- self-driving cars
- automatic video classification systems
- ... and more

Moreover, CNNs are not restricted to visual perception:

- they are also successful at many other tasks, such as voice recognition and natural language processing (NLP)
A couple of typical (visual) tasks

Example:

- **object detection** (classifying multiple objects in an image and placing bounding boxes around them)

- **semantic segmentation** (classifying each pixel according to the class of the object it belongs to)
Receptive fields and hierarchy of neurons

Crucial insights into the structure of the visual cortex from a series of experiments by Hubel and Wiesel in 1958 and 1959, on cats (and a few years later on monkeys)

- Nobel Prize in Physiology or Medicine in 1981 for their work

In brief, most relevant observations:

- “neurons in general get active on small visual areas”: many neurons in the visual cortex have a small local “receptive fields”, i.e. they react only to visual stimuli located in a limited region of the visual field (see next). The receptive fields of different neurons may overlap, of course, and altogether they tile the whole visual field.

- “neurons get active on patterns”: some neurons react only to images of horizontal lines, while others react only to lines with different orientations (two neurons may have the same receptive field - previous bullet - but react to different line orientations).

- “some neurons work on larger visual areas: they have larger receptive fields, and they react to more complex patterns that are combinations of the lower-level patterns.”

These observations led a classification of higher-level and lower-level neurons, and to the idea that higher-level ones are based on the outputs of neighboring lower-level ones. This powerful architecture is able to detect all sorts of complex patterns in any area of the visual field.
More brain modules, increased complexity

In summary:

• most biological neurons in the visual cortex respond to specific visual patterns in small regions of the human visual field (“receptive fields”)

• as the visual signal makes its way through consecutive brain modules, neurons respond to more complex patterns in larger receptive fields

the local receptive fields of 4 neurons are represented by the red dashed circles

notice that each neuron is connected only to a few neurons from the previous layer
CNN vs fully connected

Before continuing…

Why not simply use a deep neural network (from the MLP model) with fully connected layers for image recognition tasks?

Unfortunately, although this works fine for small images, it breaks down for larger images, as a huge number of parameters would be required

• e.g. MNIST has 28x28 images. A 100×100 image has 10,000 pixels, so with a NN with 1,000 neurons in the first layer and beyond, one easily get to millions of connections, and that’s just the first layer

• CNNs solve this problem using partially connected layers and weight sharing
Convolutional layers

The neurons in the **first convolutional layer** are not connected to every single pixel in the input image, but only to pixels in their receptive fields

- before CNN, it was like the former, instead

In turn, each neuron in the **second convolutional layer** is connected only to neurons located within a small area in the first layer

Let’s go for a pictorial view.
Input layer

Conv layer 1
A pictorial summary of CNNs

Convolutional layers extract ("filters") characteristics from the images

ogni gatto, ogni posa, ogni angolazione o porzione..
 Used for “computer vision” (but not only)
Even in this simplified and intuitive explanation, one can grasp that this architecture allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on.

- This hierarchical structure is common in real-world images, which is one of the reasons why CNNs work so well for image recognition.

There is MUCH more, but this might be enough to give it a try in colab..

10 minutes | Time to code!
Coding a simple CNN model

```python
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

# create model
model = Sequential()
# add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

The model type that we will be using is based on the Keras Sequential API (i.e. the easiest way to build a model in Keras)

- it allows you to build a model layer by layer, sequentially
- use the `add()` function to add layers to the model
Our first 2 layers are **Conv2D** layers (ingest 2D matrices as input)

- first Conv2D layer has **64** nodes, second has **32** nodes

  - The # of nodes in a layer can be adjusted to be higher or lower, depending on the size of the dataset and of the type of problem to solve.

- **kernel_size** = the size of the filter matrix for our convolution (kernel\_size = 3 → 3x3 filter matrix)

- **activation** = the activation function for the layer → Rectified Linear Unit, proven to work just fine in most NN applications

- Note that the 1st layer (and only that one) also takes in an input shape, i.e. the shape of each input image (28,28,1 as seen earlier).
In-between the Conv2D layers and the Dense layer, there is a Flatten layer

- it serves as a connection between the convolution and dense layers.

Dense is the layer type we will use for the output layer

- it is a standard layer type that is used in many cases for NNs
- Note we have 10 nodes in our output layer, not unexpectedly given the problem: one for each possible outcome (0–9)!
- softmax = activation function used in the Dense layer
  - it makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability
Compiling the CNN model

Compiling the model takes 3 hyper-parameters: optimizer, loss and performance metric.

• the optimizer controls the learning rate. We will be using adam as our optimizer
  ❖ The adam optimizer adjusts the learning rate throughout training, and it is generally a good choice to use for many cases

• the categorical_crossentropy will be our loss function
  ❖ this is the most common choice for classification. A lower score will indicate that the model is performing better

• to make things easier to interpret here, we will just use the `accuracy` performance metric in this example, to see the accuracy score on the validation set when we train the model
CPU vs GPU exercise

Run it on Colab!
Make prediction

Check the actual predictions that the model can make for new data by using the `predict` function

- it will give an array with 10 numbers, i.e. the probabilities that the input image represents each digit (0–9). The sum of each array equals 1 (since each number is a probability). The array index with the highest number represents the model prediction.

❖ NOTE: no new data here, so we run this on some test data, just for education..

```python
[28] model.predict(X_test[:4])

array([[2.35129143e-08, 6.70223035e-12, 1.03976021e-07, 3.48299295e-06,
        6.79778331e-11, 1.17970811e-09, 9.66348230e-12, 9.99999990e-01,
        5.97975287e-08, 4.69683698e-07],
       [2.16053422e-06, 1.40790490e-09, 9.9948025e-01, 2.70847875e-11,
        1.82843241e-09, 1.04299194e-10, 4.99568699e-09, 1.88103331e-13,
        5.62497375e-11, 5.1334412e-12],
       [2.12416380e-05, 9.9125657e-01, 1.34323782e-04, 9.80237260e-07,
        7.28716422e-03, 2.93072898e-05, 2.19629565e-05, 1.21397148e-04,
        9.83650261e-06, 9.83650261e-06],
       [9.99932005e-01, 4.35965153e-11, 2.20277272e-07, 7.44720818e-09,
        2.90047712e-08, 2.12868567e-09, 2.70689338e-06, 2.69808230e-11,
        8.72395862e-08, 3.71255999e-06]], dtype=float32)
```

The model predicts 7, 2, 1, 0 for the first four images.

Let’s compare this with the actual labels (the truth):

```python
y_test_OHE[:4]

array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```
That’s it, for our Lab on CNNs