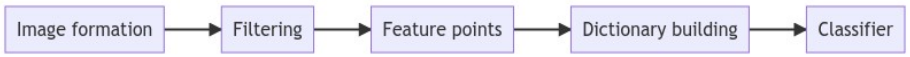
Advanced Topics on ML - 23/09/2022

Convolutional Neural Networks

Tradicional ML approach for image-related tasks

Typical image-related task pipeline:



Note that all these intermediate steps are hand designed.

Why sequential NNs are bad for image tasks?

- There is a lot of variability in data
- Since they need 1D inputs, the image needs to be collapsed into a 1D vector and this destroys spatial information
- The model matches features in the training set, but won't match the same feature when it appears translated/scaled since the spatial information is lost
- RGB information is really scrambled, since usually the 1D input will have N blue values, followed by N green values etc. Thus, the color information, which should be local, has been spread in the input vector

Convolutional layers

Concept

In order to use spatil structure, we can connect whole patches of the input image to a given unit. The chosen node becomes related only to a single image region.

This unit is usually called a filter, **and they will be learned**. The type of features they extract will be chosen **automatically**, by the training process.

In order to extract features for a given image, we **convolve** it with the filters.

Note that what is called convolution here is not really a convolution in the mathematical sense. It is essentially a **cross-correlation**.

Convolution

For each convolution step:

1. Select a filter
2. Parse image, multiply each patch element-wise with the filter, **integrate result** *
3. Next filter, repeat

$*$ - By integration, we mean sum every entry of the resulting array

Note that if we just parse a $K \times K$ filter with unit stride an $N \times N$ image, the result will be $(N - K + 1) \times (N - K + 1)$ image.

Padding is extending (with some given rule) the initial image so that the filter fits enough times during parsing such that results mantains $N \times N$ size.

What if image has many channels e.g RGB?

The filter has also many channels i.e its dimension should match the number of input channels.

But since we integrate in the end, the result is always "flat" (1 channel)

Example: RGB input, 3×3 filter becomes $3 \times 3 \times 3$ cube.

This also is important for the deeper convolutional layers, since a given convolutional layer output total number of channels is the same as its number of filters (**we have one channel for each filter**).

What are the trainable parameters? Think of each filter as a unit in the context of MLPs.

Each $K \times L$ filter has KL values to be trained (**remember, the filters are learned**), but after each convolution step we can also add **a scalar bias** to the whole output.

Also, after each convolution the result **is mapped element-wise by an activation function.**

Hyperparameters

- The number of filters
- The **kernel size** (dimension of filter)
- The **stride** (step during convolution)
- **Padding** input with a border of zeros (in order to keep output size after convolution)

Exercise

- Calculate output volume size (width, height, channels) of a convolutional layer composed of 10 filters of size 3×3, without padding, stride of 1...

  ...when applied to color input image of size 6×6

Answer: output after convolution of 1 filter must be $(W - 3 + 1) \times (H - 3 + 1) = 4 \times 4$, since there is no padding. Taking into account that the input are **color** images, each filter must have 3 channels. Since we have 10 filters total, the output will be $4 \times 4 \times 10$, one channel for each applied filter.

- How many parameters does this layer have?

Answer: for each filter, we have $3 \times 3 \times 3$ values to train, plus a bias term. Thus, we have $27 + 1 = 28$ parameters per filter. Since we have 10 filters, the layer amounts to 280 parameters.

Pooling

Concept

In order to condense the information and reduce dimensionality, we can apply pooling, i.e applying a **commutative** operation to patches of the image that assigns to each patch a single value (it's usefull to **think of it as downsampling**)

By pooling at difference locations we gain robustness to the exact spatiallocation of image features. We lose some spatial information, but we are **increasing the semantical value** of what remains.

Types of pooling

- Max pooling: trade patch for its maximum value
- Average pooling: trade patch for its average value

CNN Overall Architecture

Stack Convolutional layers and feed their flattened output to a fully-connected NN.

We don't face the problem of losing spatial information anymore, since it was already condensed into the learned features.

The **receptive field** is the region in input space that a particular CNN feature examines (i.e is activated by), and **the deeper the network is the larger the receptive field becomes** (each feature becomes richer, and probes a larger region of input space)

This kind of network is excellent for tasks dealing with images, or other data that has **strong spatial/temporal correlation**.

They **need a lot of hyperparameter tuning**, and usually their **training needs a large amount of data**.

CNNs are **highly modular** and often **can be transfered from one task to another**. We often use **pre-trained models as feature extractors**!

Exercise

- Given the task of classifying different species of fish, what would be your strategy for training a model?

Discussion: It would be better to begin with a pre-trained model using ImageNet, than to use a fresh model with randomly initialized parameters. Then,

we would use it to extract features from fish dataset. These features will be used as inputs in our own model, which then should be finetuned to perform the specific task it was given. This is called **transfer learning**.

- What if we want to identify cancer in brain MRI images?

Discussion: Since ImageNet has nothing to do with MRI images, maybe it would be wise to try with and without pre-trained models.