50.039 – Theory and Practice of Deep Learning

Alex

Week 05: Data Augmentation

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Takeaway points

at the end of this lecture you should be able to:

- be able to give suggestions how to use images of varying sizes in a neural network with a fixed input size
- be able to use a neural network with a different input size than the one used at training time
- explain some example data augmentations
- explain how to choose a data augmentation parameter.

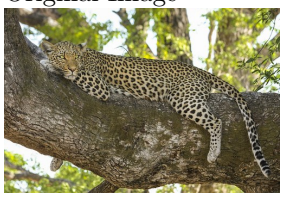
There are two big important things which you are not going to learn when trying to get experience on MNIST and CIFAR, but which matter a lot in practice: data augmentation and transfer learning in a narrow sense.

The goal of this lecture is to give an introduction to data augmentation.

- **Problem 0: Input normalization** to match how the network was trained
- **Problem 1: The input dimensionality** of a neural network may not fit the dimensionality of the input data. We discuss ways to deal with it.
- **Problem 2: use neural network with a different than the specified input size** – yes one can!!
- **Problem 3: Data augmentation at Testing time**
  allow the neural net to see *multiple views* of the same sample. One creates multiple views of one sample. The final prediction will be an average of the prediction scores, averaged over all views. "Look at all aspects to understand a thing."

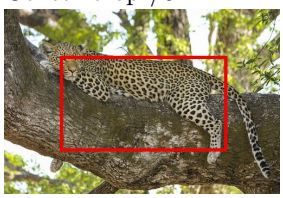
Example in this class: Do not input an image as a whole, but input 5 crops: a cropped image from the center, and 4 corner crops.

Original Image

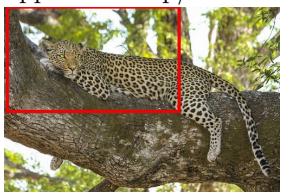


compute the prediction as an average over crops:

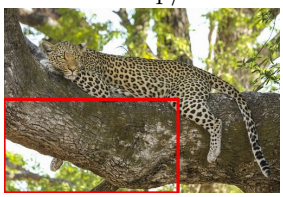
prediction =
Center crop /5



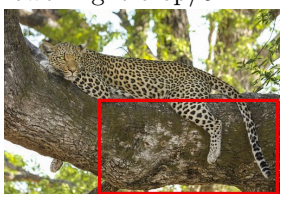
+
upper left crop/5



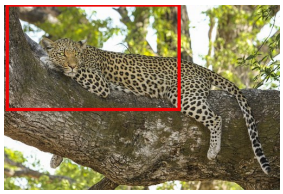
+
lower left crop/5



+
lower right crop/5



+
lower right crop/5



- **Problem 4: Data augmentation at Training time:** allow the neural net to be trained with more data samples. Data augmentation increases the data size by creating many similar samples from one sample.
- Example: training with slight photometric deformations of a sample (e.g. darker or brighter images) will allow the network to be able to recognize similarly brighter or darker images (relative to your original training images) when deployed at testing time.

  This is very important, in particular for finetuning over small datasets.
- **Problem 5: Getting the scale of relevant objects right:** another example of a fixed, non-randomized augmentation for images (will see that in a later lecture):
  make the image size such that relevant structures (e.g. cats, cars) can be identified well.

  A neural network for images has convolutional kernels with a fixed size. During training their weights are learned and adapt to some structures.

  The sizes of input images at test time must be such that ... the structures to be identified at test time (e.g. cats, cars) – have similar size as – structures used at training time.

  If your training set contains only screen filling cat faces,

  then dont complain that you will not be able to predict cat correctly on this:

Your localized convolutional kernels were simply not trained to find structures on such small scales!
but data augmentation (or training a bounding box detector) can help with that – you can classify over many small windows.

- Above are not image-specific problems. The same problems (normalization, input dimensionality, multiple views, slightly augmented input data) arise when using a neural network e.g. for text processing, as the number of words is varying. Later we will see recurrent neural networks which can deal with inputs being sequences of arbitrary lengths (however even then it can make a big difference to deal with dimensionality properly).
- **Coding:** You will use a pretrained deep neural net to compute predictions for images.

0.1 Problem 0: Input normalization:

What inputs does a neural network expect ? As for images, common image readers produce subpixels in $[0, 255]$. the output of `PIL.Image` into numpy is usually of shape $(height, width, channels)$.

A neural network expects inputs often in $[0, 1]$ with shape $(batchsize, channels, height, width)$ with afterwards subtracted the mean of the training dataset.
For the conversion to $[0, 1]$ - in pytorch and mxnet `transforms.ToTensor()` takes care of that.

Often one has to do **further preprocessing**. The reason is: you have to do the same preprocessing steps at test time, as what was done at training time, otherwise train and test data have different distributions in input space. This usually results in heavily reduced performance.

Neural nets are usually trained with images, from which the training dataset mean is subtracted (that is: for every pixel). It was observed that training with samples which have on average over the dataset a zero mean results in faster convergence.
See `http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf` for an explanation.
Furthermore, often input dimensions are also normalized by dividing them over a standard deviation (either channel-wise or pixel-wise) estimated over the training set.

$$subpixel[channel] = subpixel[channel] - trainset\_mean[channel]$$
$$subpixel[channel] = \frac{subpixel[channel] - trainset\_mean[channel]}{trainset\_std[channel]}$$

In keras, that hosts many different modes, one can see at least three different modes:

`https://github.com/keras-team/keras-applications/blob/master/keras_applications/imagenet_utils.py`

One of them does not do division by standard deviation by default (`else:` – the caffe standard)

...

The rule:
Check what preprocessing was used at training time for the network you are going to use. Check what of that makes sense to be used at test time. Fail here = you generate random numbers on a GPU.

1 Problem 1: data input size vs nn input size without changing the network input

The default resnet deep neural net has an input size of 224×224 for images (inception 299×299). Your images usually are not of that size.
What can one do? One can resize the neural network for every image. We will talk later how to do that. But reinitializing the NN is ultra-slow, in particular when one wants to process a larger set of images.
Alternative is to adapt the image. One may resize the image to 224×224 with giving up the aspect ratio (height to width). Unusual – aspect ratio.more common:

...

- resize the image while preserving the aspect ratio such that the smaller size s is $s \geq 224$ pixels, and then take a crop of size 224×224
- the crop can have multiple approaches: a **center crop** or a **random crop** of 224×224 with random position.

The same problem affects sequence analysis with 1d-cnns.

Recurrent neural networks can deal with sequences of arbitrary lengths, however training them with subsampled sequences of fixed length can be considered for improving performance.

2 Problem 2: use neural network with a different than the specified input size

Learning goal: you can most of the time use a pretrained neural network with a different input size than its original input size.

You want to classify over a larger input than 224×224, say 330×360 ? Yes you can.
The important thing to understand is: if you change the input size, then this does **NOT** affect the number of parameters in any convolutional layer, only its output sizes.
It can break at subsequent fully connected layers - but often it can be fixed. So if there is a fully connected layer, with a pooling or flatten layer before it, then one must change the parameters of the pooling or flatten layer to become a global pooling with a fixed output size.

...

- Resizing the input of a neural network changes the sizes of outputs (show it), but for convolution layers not their number of parameters.
- Resizing the input of a neural network changes the number of parameters in fully connected layers. In order to ensure the same number of parameters in ully connected layers with changed size as with original input size, one needs to ensure that the number of inputs to fully connected layers stays the same. If the fully connected layer is preceded by a pooling layer, the pooling layer can be modified.
- Global pooling/adaptive pooling are pooling methods where the output size is fixed and instead the pooling kernel size is changing dependent on the input to match the desired output size.

Lets look at a resnet 18 to understand this: `https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py` in `class ResNet(nn.Module):` and `def resnet18(pretrained=False, **kwargs):`
Consider how the forward pass works `def forward(self,x):`

$(224, 224) \rightarrow$ many conv layers $\rightarrow (512, 7, 7) \rightarrow$ Pooling by `self.avgpool`
$\rightarrow (512, 1, 1) \rightarrow x = x.view(x.size(0), -1) \rightarrow (512) \rightarrow self.fc- > (1000)$

```
self.avgpool = nn.AvgPool2d(7, stride=1)
```

In the meantime it was fixed it already to `nn.AdaptiveAvgPool2d`, but you may encounter the problem with other networks created by other people.

The same problem affects sequence analysis with 1d-cnns.

Recurrent neural networks: same comment as in last section.

3 Problems 3 and 4: Data augmentation

One does not simply predict on a raw sample. One does not simply train with a raw sample.
Using data augmentation is a fundamental standard in deep learning. Unfortunately, many tutorials skip this.

Examples of data augmentation at test time for images:

- random crops of a fixed size of the image
- corner and center crops (see above)
- mirroring along the vertical axis

Test time data augmentation

At test time data augmentation is often used in an averaging sense. Let x be an input sample, $f(\cdot)$ a predictor, and A_a the a-th augmentation (e.g. the a-th crop of an image), then the prediction is computed as an average over a set of chosen augmentations:

$$f(x) = \frac{1}{n_A} \sum_{a=1}^{n_A} f(A_a(x))$$

Here the average is computed over augmentations of the same input sample x.

The idea of data augmentation at test time is to predict on an average of multiple views of the same input.

Test time data augmentation

The idea of data augmentation at training time is to extend the training data available by creating instances

- which can be derived from the training data set
- which are plausible to have the same ground truth label as their origin
- which are plausible to be seen in the test set.
- which sometimes are used to encode invariances that one wants to have in the learning process

Thus to increase the training dataset.

Examples of data augmentation at training time for images:

- geometric distortions:

  – rotations - they make sense sometimes up to a certain degree, some imaging problems are suitable for arbitrary rotations

  – mirroring

  – distort an MNIST digit by e.g. shear. Not used often outside of MNIST or similar shape recognition tasks.

- photometric distortions: modify brightness, and contrast/gamma of an image. for some scenario (e.g. medical stainings) one can think also to play with hues (colors).

  Contrast change (e.g. `https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/`) we assume that each subpixel lies in $[0, 255]$.

  $$sub = C \times (sub - 128) + 128, C > 0$$

  For randomized image augmentation purposes the value C is drawn for every image and minibatch from a random distribution such as

  $$C \sim Uniform[a, b]$$

  The random distribution has usually parameters, here a, b which need to be selected on a validation dataset, but not on the test set. Brightness change:

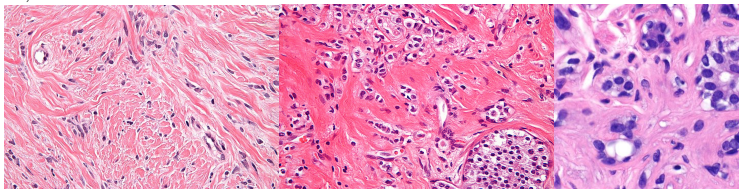
  $$sub = C + sub$$

  Gamma change:

  $$sub = 255 * \left(\frac{sub}{255}\right)^{C}, C > 0$$

  The hue is a value for the color in HSV colorspace.

  An example where playing with hues a little bit may make sense: Below is the same stain Haematoxylin and Eosin, of the same tissue. Different lab people create different color intensities!

  For one quick intro on photometric distortions : Andrew Howard, Some Improvements on Deep Convolutional Neural Network Based Image Classification `https://arxiv.org/pdf/1312.5402.pdf`. See also the googlenet paper.

- distortions depends typically on some parameter. One needs to train with multiple choices of the parameter, and find the best parameter by looking at validation dataset errors. Once all parameters have been selected, then one does one final evaluation on the test data set – in order to check that one did not overfit by optimizing parameters by looking at the validation error.
- generic option: add gaussian noise
- generic option: add gaussian noise in feature space (e.g. after fourier transform)
- riskier options: generative methods like seq2seq learning and cycleGANs
- Examples of data augmentation at training time for text sentences?
- cycle-gan inspired (Jigsaw toxic comment classification):
  - text → other language → translate back
- Noising in NLP: `https://arxiv.org/pdf/1703.02573.pdf`, ICLR 2017
- use a language model to suggest augmentations: `http://aclweb.org/anthology/N18-2072`
- Audio: modify the speed mildly `https://openreview.net/pdf?id=HyaF53XYx`
- Audio: intrapolation vs extrapolation (not sure if this result holds!) `https://openreview.net/forum?id=HyaF53XYx`

Train time data augmentation

Data augmentation is used at training time in a randomized fashion – because mild randomization often helps against overfitting.

- at training time: *randomized transforms* – The randomization depends on the transformation. Example: For image brightness, one choose a random add within some range, like $[-0.2, 0.2]$, every time an image is loaded, so for one minibatch one draws a vector of random numbers).
- at training time: often one keeps an image as it is with p probability, and applies randomized transforms with $1 - p$ probability. Every time an image batch is loaded, one chooses a new parameter from a random distribution.
- **one does usually NOT apply these distortions on the validation or test set !! (except one wants averaging over multiple views)**

important:
The probability distribution of the distortion parameter has hyperparameters, that needs to be validated on the validation dataset.

$$\text{argmin}_w \frac{1}{n} \sum_{i \in \text{ minibatch}} L(f_w(A_i(x_i, \text{params})), y_i)$$

where $A_i(\cdot, \text{params})$ is a randomized augmentation, e.g. brightness modification with a randomly drawn value, which varies for every image during every training epoch.

Compare to the augmentation at test time – for which the average is computed over different augmentations of the same input sample x.

Augmentation can be also applied to the ground truth, adding some noise to the labels! Example: adding noise to regression ground truth, moving boundary boxes of detections, moving segmentation boundaries