Deep Learning for Computer Vision
Assignment Sheet 4 - Due 28.01.18, 23:59

Next exercise groups: 16.01.18, 13:30 and 30.1.18, 13:30.

Exercise 4.1: the transpose of convolution in practice (5 + 10 + 5 points)
We will think a bit about the output shapes of strided convolutions and their transpose. This will help you with building encoder-decoder networks in the next exercise.

(a) The goal of this exercise is to fit parameters of two convolution layers, which are defined with a following code snippet, to the desired size of the output:

```
x = tf.placeholder(np.float32, [None,64,48,17])
W1 = tf.Variable(tf.zeros([3,3,x1,x2]))
layer1 = tf.nn.conv2d(x, W1, strides=[1, x3, x4, 1], padding='SAME')
W2 = tf.Variable(tf.zeros([3,3,x5,x6]))
layer2 = tf.nn.conv2d(layer1, W2, strides=[1, x7, x8, 1], padding='SAME')
```

We assume that the input has 64 pixels in height, 48 pixels in width and 17 channels, the desired output has height 11, width 4, and 13 channels. Missing numbers in the code above are depicted in red. Find correct values of the numbers such that the layers fit together and to the desired input/output shape.

While you can do this by just experimenting in Tensorflow, I suggest you try to find a correct result with pen and paper first (note that the solution is not unique).

(b) Give a code snippet for defining the decoder path which corresponds to the strided convolutions above, such that the output of your decoder has exactly the same shape as the input, and also the corresponding intermediate layers in encoder and decoder have the same shape. Use `tf.nn.conv2d_transpose()` to define the transpose convolution layers.

(c) The layer `tf.nn.conv2d_transpose()` requires you to define the output shape, in contrast to a normal convolutional layer. Explain why.

Exercise 4.2 Training an encoder-decoder network (5 + 20 + 20 + 5 + 10 points)
Some countries use a Cyrillic alphabet. Unfortunately, Cyrillic does not have a bijective mapping to the Latin alphabet, but some glyphs can be understood by a sound expressed as a single Latin glyph. The goal of this exercise is to build an encoder-decoder network which translates the subset of Cyrillic handwritten glyphs for which there is a unique correspondence into the matching Latin sounds, expressed as handwritten glyphs again. We prepared a dataset which consists of labeled handwritten glyphs in both alphabets. Each alphabet contains many labeled $(28, 28)$ images of the handwritten glyphs and each datum's label associates the datum with the Latin equivalent.

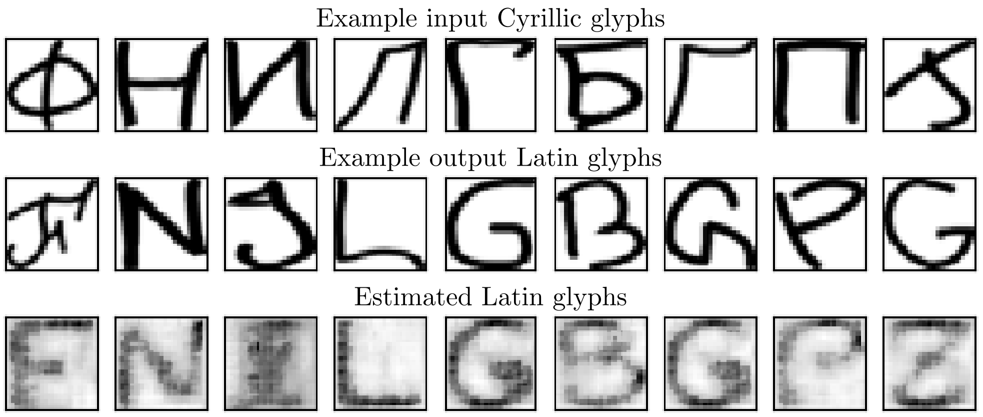

Figure 1: The example of input images in Cyrilic alphabet with its corresponding possible handwritten versions in Latin glyphs.

(a) Normalize images in Latin and Cyrillic input datasets such that each image has zero mean and unit standard deviation.

(b) Build an encoder-decoder network which consists of a sequence of strided convolutions for downsampling and strided transpose convolutions for the corresponding upsampling overations. Size of input and output should of course match. Remember to also use non-linear activations.

(c) Propose a training strategy and loss function to learn the transformation from a Cyrillic glyph to a corresponding latin glyph, making best use of all available training data. Also try to use some of the tricks you learned in the lecture for better performance.

(d) Minimize the network loss and optimize until you get a meaningful output. This will probably not be particularly beautiful - look at the image in the last row of Fig. 1 where you can see an example of what you probably should expect.

(e) Perform optimization on the space of latent codes (the layer at the bottom of the encoder), and find a code for which the MSE over all training examples in the Latin alphabet is minimized. It is somewhat similar to Exercise 2.4, but instead of finding the most likely input for given class, we are looking for the most likely code to represent a given class.

Exercise 4.3: backpropagation through a convolutional layer (5 + 5 points)
Let $g : \mathbb{R}^m \to \mathbb{R}$ be a differentiable function, $A \in \mathbb{R}^{m \times n}$ a matrix.
We consider the function $E(x) := g(Ax)$ on $\mathbb{R}^n$.

(a) Compute $\nabla E(x)$ in terms of A and the gradient of g.

(b) Explain why you need the transpose convolution for backpropagation through a convolutional layer.

Exercise 4.4: PCA vs. auto-encoder (Bonus, 5 + 10 + 5 points)

Warning: for this (non-easy) bonus exercise, you should be quite familiar with linear algebra, and already know about principal component analysis from previous lectures. The following is just a brief reminder of the basic ideas.

Reminder: The Principal Component Analysis (PCA) is one of the most popular feature representation methods. Consider n observations $x_1, \ldots, x_n \in \mathbb{R}^d$, arranged as rows of a matrix $X \in \mathbb{R}^{n \times d}$. PCA finds an orthogonal transformation $V \in \mathbb{R}^{d \times d}$ of the data space such that when applying the transformation $\hat{X} = XV$ to the dataset, the components of the transformed observations are statistically decorrelated. Mathematically, V can be found by computing the Eigendecomposition $X^T X = V \Lambda V^T$, where Λ is a diagonal matrix of the Eigenvalues $\lambda_1, \ldots, \lambda_d$ (usually arranged in decreasing order), and the columns of V are the Eigenvectors. In particular, the Eigenvectors capture the directions of maximum variance of the data in decreasing order. Thus, a popular way for dimensionality reduction is to project the data onto the subspace spanned by the first $k < d$ Eigenvectors, as this will minimize the reconstruction error in the remaining dimensions.

Tasks:

(a) Give a formula for the projection of a vector $x \in \mathbb{R}^d$ onto the subspace spanned by the first k principal components (the result should thus still live in $\mathbb{R}^d$).

(b) Consider a fully-connected auto-encoder with just one hidden layer and identity activation functions. Thus, the encoding transformation is just $z = Ax \in \mathbb{R}^k$ with a matrix A, the decoding transformation $x' = Bz \in \mathbb{R}^n$ with a matrix B. Suppose the auto-encoder is trained with L^2-loss on the data X. Write down the loss function explicitly and find one particular solution for A and B of the resulting minimization problem in terms of the *PCA*. Do not forget to proof that this is indeed a minimizer.

(c) Will you get exactly this solution if you minimize the loss with stochastic gradient descent? If not, then how are the two different solutions still related?