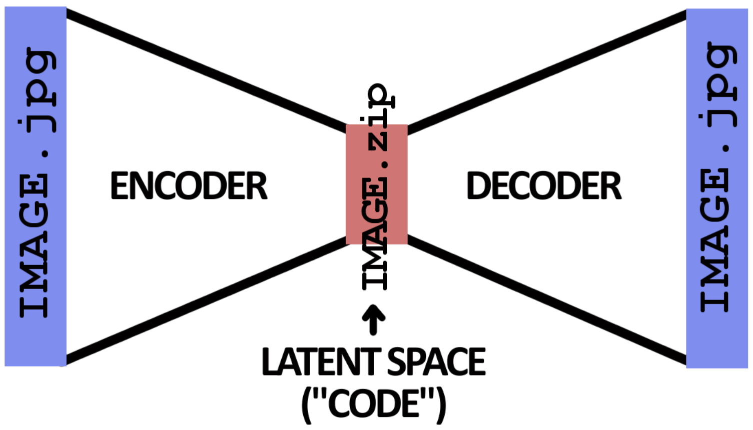
Advanced D. L. for Physics

Exercise 2

In this exercise you will build a convolutional auto encoder to encode and reconstruct velocity fields from a mantaflow simulation. You can find a re-cap of the autoencoder basics in the box below. To do so you will have to modify both a mantaflow scene script and a tensorflow training script, so make sure you have familiarized yourself with them! If you have any problems or questions, please use the Moodle forum.

Autoencoders:
Used to create a method to encode/compress the input data. In order to retrieve the original data, the encoded data also has to be decoded again. Just like compressing a file to a .zip and unzipping it again.
To achieve this, the network is simply asked to output its input. By creating a bottleneck in the middle of the network, we force it to encode and finally decode the data. Because the input data is too large to fit through the bottleneck, every network layer **before** the bottleneck has to **encode** the data, so it can pass. Every layer **after** the bottleneck has to **decode** the data, so it becomes the input data again - which we initially wanted as an output. Thereby two network parts are created that can be used individually: The encoder and the decoder.



The error of the network is calculated by comparing the network output to the input.

As simple as they are, autoencoders are used for more than just efficient compression. In order to compress, the encoder network does not e.g. save every pixel of an eye but only the location of the eye. The decoder then draws an eye at this specific location. With this, by sampling latent space points and inserting them into a well trained decoder, it is possible to generate new data sets that are variations of the original input data.

Once you're done with the assignments below, hand in your exercise by sending a zip archive via email to adl.phys.ex@gmail.com by **May 17th 2018, before midnight** (email arrival time will count)**.**

Name the zip file "**xx_lastname1_lastname2_lastname3.zip**" (xx being the group number) with **no subfolders,** please make sure to use **meaningful** filenames. The subject of the email should be [ADL Exercise02 GroupXX].

Exercise 2.1 – *Saving and Loading Training Data*

Create the training dataset using a mantaflow scene.
The autoencoder should encode and reconstruct velocity fields. Because of this you first need to adapt the mantaflow scene script from exercise 1 (*manta_genSimSimple.py*) so that it saves the velocity data of each frame instead of densities. This dataset will later be used to train the autoencoder.

Modify your training script to load the velocity data.
You can base your autoencoder training script on the *tf_simple.py* script from the previous exercise (i.e. make a copy of it and modify it). For data loading have a look at lines 38 - 65 in particular. Modify the script to load the velocity data you saved before instead of loading densities. Since the autoencoder should work with 2D velocities, you need to discard the z-component of the loaded velocities here.

In the end your data should be in a numpy array of shape [#Frames, 64, 64, 2]. This must then be split up into training and validation data. Note that running the training script now will lead to errors as you also have to adapt the network structure to your new input data (see exercise 2.2)!

Hints:

- To easily discard the z-value of the velocity, use *numpy array slicing*
  https://www.tutorialspoint.com/numpy/numpy_indexing_and_slicing.htm
  You can use negative indices too! E.g. [:-1] will give you all but the last element.

- Use *numpy.reshape* to get the numpy array containing your data into the correct shape so it can be fed into the network.
  (https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html)

- To check that you discarded the z-values and reshaped the data correctly, you can *print()* the numpy arrays before and after each operation and compare if they contain the same values at the desired positions.

- Note on Normalization: It is usually highly recommended to normalize the velocities since they can contain values smaller than -1 or larger than 1. This could e.g. be done by dividing each velocity by the standard deviation of all velocities. Afterwards, network's predictions then need to be de-normalized. For this exercise, however, you can omit normalization since the velocity values are largely within [-1, 1] for the *manta_genSimSimple.py* simulation.

Exercise 2.2 – *First Network Architecture*

Build a network consisting of 2 convolutional and 1 fully connected layers.
As a first step, build a network with the following structure:

1. Convolutional Layer (Filter Size: 12x12, Stride: 4, Channels: 2)
2. Convolutional Layer (Filter Size: 6x6, Stride: 4, Channels: 4)
   → Here your intermediate result, your "latent space" vector, should have the shape [4, 4, 4], i.e., 64 values.
3. Fully Connected Layer ([64 * 64 * 2])

For the convolutions you can use the following helper function:

```
def convolution2d(input, biases, weights, strides, padding_kind='SAME'):
        input = tf.nn.conv2d(input, weights, strides, padding=padding_kind)
        input = tf.nn.bias_add(input, biases)
        input = tf.nn.leaky_relu(input)
        return input
```

Each convolutional layer should have as many bias values as the number of output filters used. To store your Tensorflow variables in an organized fashion you can use a dictionary. You can also declare the variables within the convolution2d function instead of passing them as parameters.

Add a function to compute the degrees of freedom.
When building neural networks you should always be aware of the degrees of freedom - the parameters that are being optimized, i.e. the weights and biases. You should therefore write a function which automatically evaluates how many degrees of freedom (DoF) your network has, and prints it to the command line.

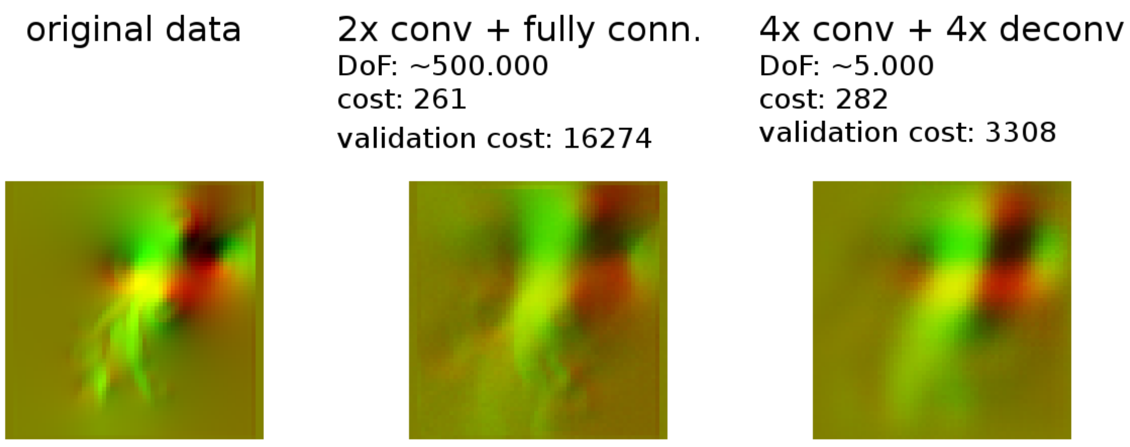
Train your network and save its predictions.
You should now be able to train your network and have a first look at its predictions! This should not take longer than a few minutes. To save a test velocity field and the network prediction corresponding to it you can use *scipy.misc.toimage* (as in *tf_simple.py*, lines 118 and 199) and the following helper function to convert the two components of the velocity field into red and green channels of an RGB image:

```
def velocityFieldToPng(frameArray):
        """ Returns an array that can be saved as png with scipy.misc.toimage
        from a velocityField with shape [height, width, 2]."""
        outputframeArray = np.zeros((frameArray.shape[0], frameArray.shape[1], 3))
        for x in range(frameArray.shape[0]):
        for y in range(frameArray.shape[1]):
        # values above/below 1/-1 will be truncated by scipy
        frameArray[y][x] = (frameArray[y][x] * 0.5) + 0.5

        outputframeArray[y][x][0] = frameArray[y][x][0]
        outputframeArray[y][x][1] = frameArray[y][x][1]

        return outputframeArray
```

Here you can find an example velocity input, and outputs of the two network architectures:



Exercise 2.3 – *Deconvolutions*

Modify your network to consist of 4 convolutional layers and 4 deconvolutional layers.
Choose the filter size, stride and amount of channels yourself. As a start, you can re-use the convolutional layers from 2.2 with a smaller stride.

Use a network with the following structure:

1. Convolutional Layer (e.g. Filter Size: 12x12, Stride: 2, Channels: 2)
2. Convolutional Layer (e.g. Filter Size: 6x6, Stride: 2, Channels: 4)
3. Convolutional Layer
4. Convolutional Layer
   → Here your latent space should have the same size as in 2.2, e.g. [2, 2, 16]
5. Deconvolutional Layer
6. Deconvolutional Layer
7. Deconvolutional Layer
8. Deconvolutional Layer

For the deconvolutions (= transpose convolution) you can use the following helper function:

```
def deconvolution2d(input, weights, outputShape, strides, padding_kind='SAME'):
        # needed for dynamic shape with deconvolution
        dynamicBatchSize = tf.shape(input)[0]
        deconvShape = tf.stack([dynamicBatchSize, outputShape[1], outputShape[2], outputShape[3]])

        input = tf.nn.conv2d_transpose(input, weights, deconvShape, strides, padding=padding_kind)
        input = tf.nn.bias_add(input, biases)
        input = tf.nn.leaky_relu(input)
        return input
```

Compare your new DoF count with the DoFs from 2.2.
You should use fewer DoF now than in 2.2 to achieve a similar result. If you have too many DoF consider reducing the size of filters and amount of channels, especially in the deeper layers.

Train your network, iteratively improve it and save its predictions.
Train your network and compare the results to those from 2.2. Then try to iteratively improve them and get something about as accurate as your previous version. You can compare the results visually, but also consider the L_2 residual, i.e., the validation cost.

Exercise 2.4 – *Activation Function*

You have so far been using leaky ReLU as the activation function. Why is this not optimal for the type of data we are using? What activation function would be more suitable and why? Implement it for your network from 2.3.

Hints:

- http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html, Section 5
- Consider the value range of your output!

Submission summary for exercises 2.1. to 2.4:

Hand in your exercise by sending a zip archive via email to adl.phys.ex@gmail.com by **May 17. 2018, before midnight** (email arrival time will count).

Name the zip file “**xx_lastname1_lastname2_lastname3.zip**” (xx being the group number) with **no subfolders,** please make sure to use **meaningful** filenames. The subject of the email should be [ADL Exercise02 GroupXX].

Files to include in the zip file:

- Mantaflow Data Generation Scene Script (modified *manta_genSimSimple.py*)
- Training Script (*tf_simple.py*, final version from 2.3 is fine)
- 3 predictions of your trained network from ex. 2.2 and 2.3
  - Choose an arbitrary frame and provide the network’s input and prediction
  - Name your files “frame_XXX_input.png” and “frame_XXX_prediction.png” (where XXX represents the frame’s number)
- A text file containing the computed DoF for exercises 2.2 and 2.3 as well as your answer for exercise 2.4
  - Name this file “answers.txt”

To verify that we got your submission, you can check the exercise overview.