Deep Learning Exercises [DL E]
Exercise 0
Numpy Tutorial

L Mill N Ravikumar
T Würfl K Breininger
S Gündel M Hoffmann
F Thamm F Denzinger
October 17, 2018

LME

Numpy Tutorial

We will use numpy functions to create three different types of patterns, and implement an image generator class to read in, and synthetically augment the data using rigid transformations.

1 Array Manipulation Warmup

1.1 Exercise Skeleton

Each pattern should be implemented as a separate class in separate files and the should provide the following functions: **__init__()**, **draw()** and **show()**.
A main script which imports and calls all these classes should also be implemented. There are **no loops** needed for the first three tasks in this exercise! Try to prevent them since they will severely impact the performance.

Task:

Create a skeleton with the classes **Checker**, **Spectrum** and **Circle** in the files: ”checkers.py”, ”spectrum.py” and ”circle.py”.
Also create a file ”main.py” which imports all other classes.

- Import numpy for calculation and matplotlib.pylab for visualisation in all classes using
  import numpy as np and
  import matplotlib.pyplot as plt.
  This is the most common way to import those packages.
- Implement public members:
  - The class **Checker** needs two integers **resolution** and **tile_size**
  - The class **Circle** needs two integers **resolution**, **radius** and a list **position**
  - The class **Spectrum** needs a integer **resolution**

  Moreover, create a member **output** of type np.ndarray for all classes.
- Create the methods **draw()** and **show()** for each class

Hints:
__init__() is the constructor of the class.
Following functions from the cheatsheet might be useful: **np.tile()**, **np.arange()**, **np.zeros()**, **np.ones()**, **np.concatenate()** and **np.expand_dims()**

L Mill N Ravikumar
T Würfl K Breininger
S Gündel M Hoffmann
F Thamm F Denzinger
October 17, 2018

1.2 Checkerboard

The first pattern to implement is a checkerboard pattern with adaptable tile size and resolution. You might want to start with a fixed tile size and adapt later on. For simplicity we assume that the resolution is divisable through the tilesize.

Task:

- Implement the method **draw()** which creates the checkerboard pattern as a numpy array using the public members **resolution** and **tile_size**.
  Helpful functions for that can be found on the **Deep Learning Cheatsheet** provided.
- Implement the method **show()** which shows the checkerboard pattern with for example **plt.imshow()**.

Hints:
Verify your implementation by creating an object of this class in your main script and calling the object's functions.

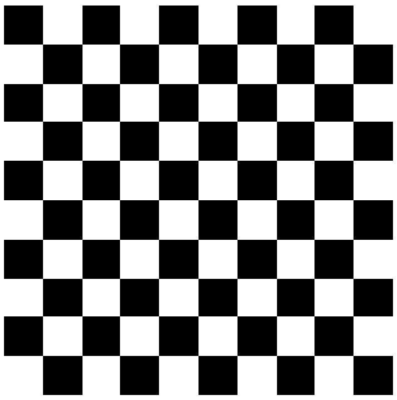

Figure 1: Checkerboard example.

1.3 Color Spectrum

The second pattern to implement is a RGB color spectrum.

Task:

- Implement the method **draw()** which creates the spectrum as a numpy array using the public members **resolution**.
  Remember that RGB images have 3 channels and that a spectrum consists of rising values across a specific dimension
- Implement the method **show()** which shows the RGB spectrum with for example **plt.imshow()**.

Hints:
Verify your implementation by creating an object of this class in your main script and calling the object's functions.

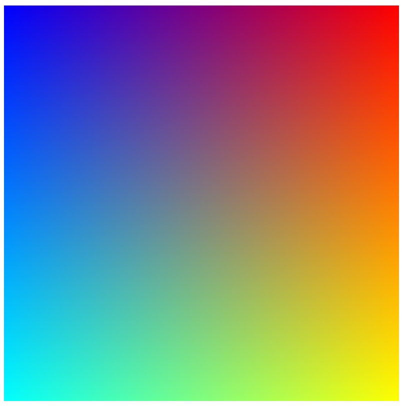

Figure 2: RGB spectrum example.

1.4 Circle

The second pattern to implement is a binary circle.

Task:

- Implement the method **draw()** which creates a binary image of a circle as a numpy array using the public members **resolution**, **radius** and **position**.
  Think of a formula describing the circle.

- Implement the method **show()** which shows the circle with for example **plt.imshow()**.

Hints:
Verify your implementation by creating an object of this class in your main script and calling the object's functions.

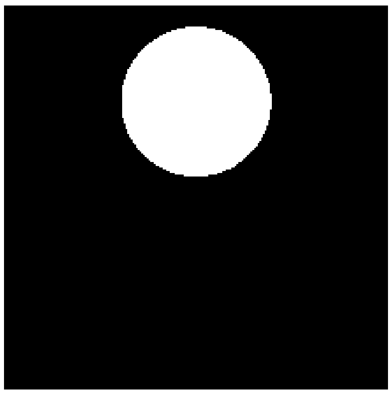

Figure 3: Binary circle example.

L Mill N Ravikumar
T Würfl K Breininger
S Gündel M Hoffmann
F Thamm F Denzinger
October 17, 2018

LME

2 Data Handling Warmup

One of the most important tasks for machine learning is adequate pre-processing and data handling.

2.1 Image Generator

Task:

- Implement an image generator class to read in images and their associated labels (stored as a JSON file), and provide the method **next()**, which returns one batch of the provided dataset when called.
- The public members used by the generator should include: batch size, image size, and optional flags for rotation, mirroring, shuffling and the folder containing the data
- Implement the following functionalities for data manipulation and augmentation: mirroring, shuffling and 90° rotations.
- The generator class should include a method **show()** which generates a batch and plots it.
- It should also include a method **class_name()**, which returns a vector of the class labels/names corresponding to the images generated in the most recent batch. This should be called and used as the title for the image plots generated using method **show()**, as shown in Fig. 4. .
- Not all samples provided have the same image size, consequently, a check and resizing option should be included within the **next()** method.
- A for loop can be used within the method **show()** to create sub-plots for images in each batch.
- The script "main.py" created for the previous task should also import the image generator class.

Hints:
__init__() is the constructor of the class as previously. Sample shuffle needs to be included within the constructor as well. Create an object of this class in the main script and call **show()**.

L Mill N Ravikumar
T Würfl K Breininger
S Gündel M Hoffmann
F Thamm F Denzinger
October 17, 2018

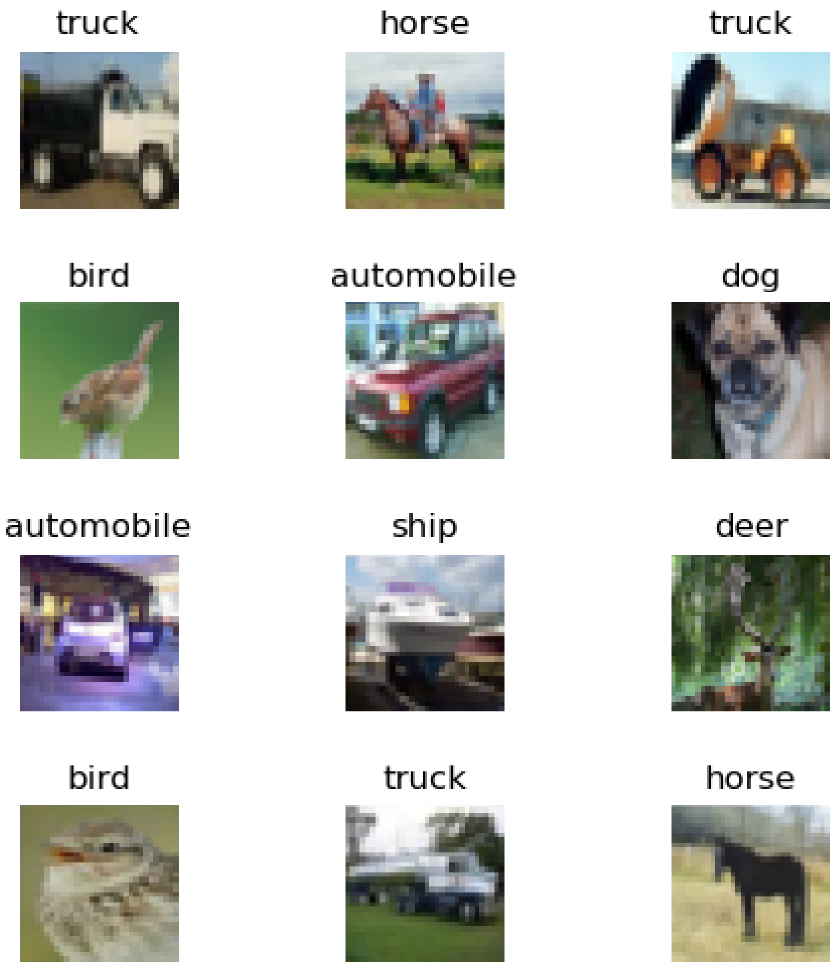

Figure 4: Example image generator output.