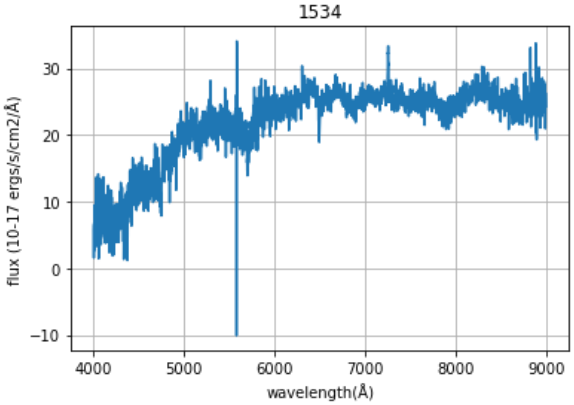
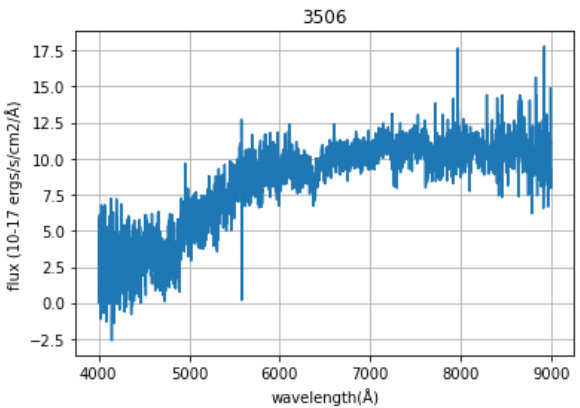
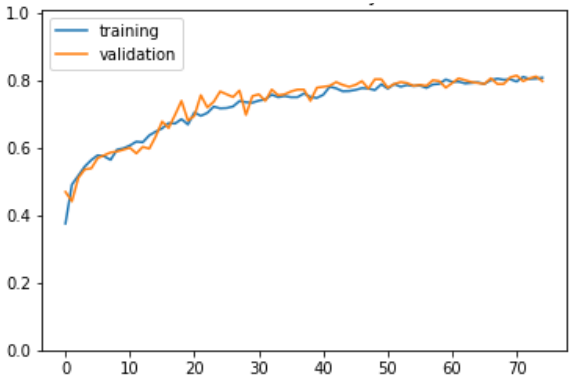
Deep learning for astronomy & astrophysics – An introduction, with Keras







Caroline Heneka, Hamburg Observatory

Outline of this lecture

Part 1: Hands-on tutorial (ca. 60 min)

1) Short motivation and basics

2) Classification of SDSS spectra

Break! 5-10min

Part 2: Current astrophysical applications (ca. 30 min)

Outline of this lecture

Part 1: Hands-on tutorial (ca. 60 min)

1) Short motivation and basics

2) Classification of SDSS spectra

Please start binder image here:
https://mybinder.org/v2/gh/csheneka/ML-for-Astro-tutorial/HEAD
-> open spectral_classifier.ipynb

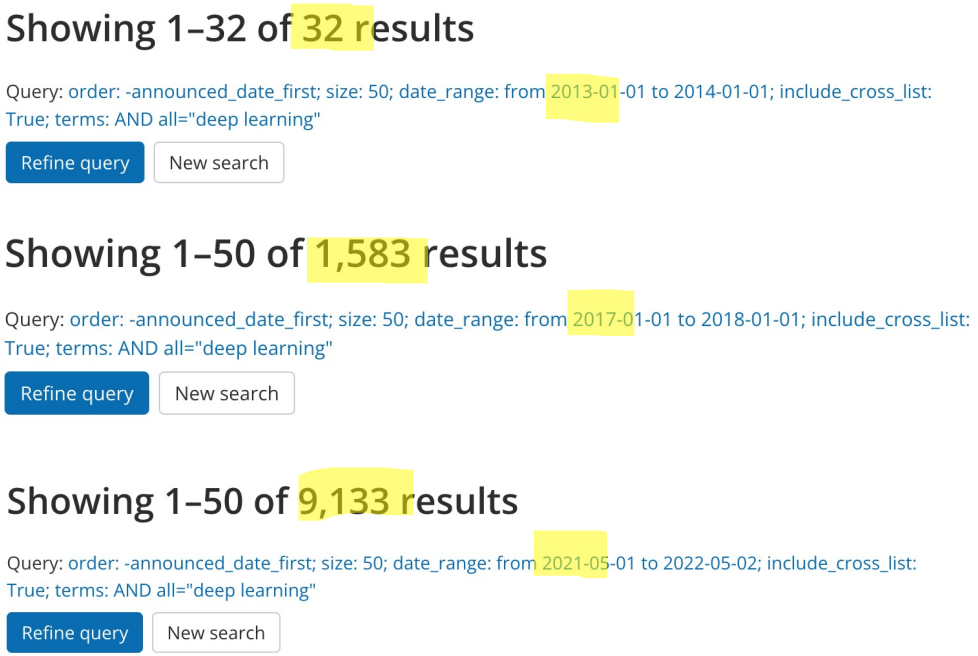
OR via git here: https://github.com/csheneka/ML-for-Astro-tutorial

Full data: https://cloud.hs.uni-hamburg.de/s/bc9s3Z6mpnHYEDK

Motivation and basics

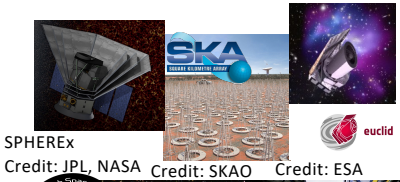
Accelerating on arXiv.org

and in
astronomy & astrophysics



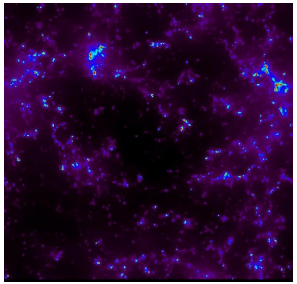
Motivation and basics

Driven by large-scale + high-resolution surveys



SPHEREx
Credit: JPL, NASA
Credit: SKAO
Credit: ESA
Credit: NASA, ESA, CSA
Credit: ESO

Example: Intensity Mapping



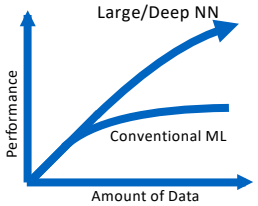
Lyα emission < 1 billion years after Big Bang

Extract more & less biased (?) information
Data mining

Efficient data reduction
Automation

Complementarity to
previous probes + methods

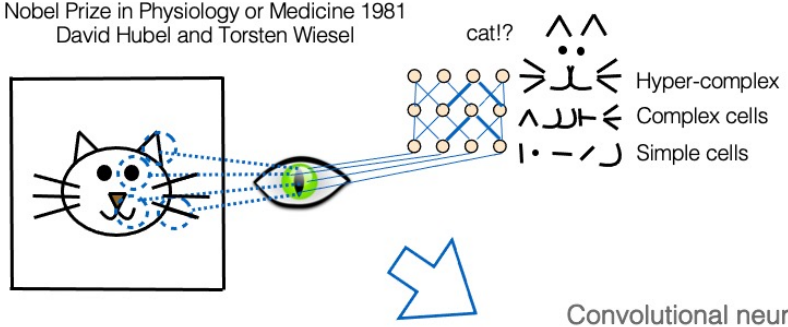
Example: Deep Learning
Driven by ability to improve with large datasets



Motivation and basics

Neuron response to visual stimuli

Nobel Prize in Physiology or Medicine 1981
David Hubel and Torsten Wiesel

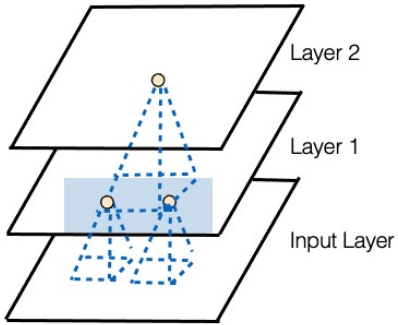


Representation learning

- Feature detection and assembly
- Local detection
- Overlapping fields

Convolutional neural network

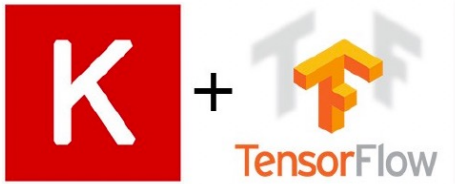
'Neocogniton' Kunihiko Fukushima 1979



Our setup today

The tutorial provides examples of neural network models written in Python, using the Keras library and TensorFlow tensor ordering convention.

Keras provides a high level API to create deep neural networks and train them using numerical tensor libraries (backends) such as TensorFlow, CNTK or Theano.



Credit: A. Boucaud

Our setup today

Today we work with binder-images, usually e.g. TensorFlow needs to be installed.

See environment.yml file

Standard package:

pip install tensorflow

or

conda install tensorflow

For execution on GPU
– older versions only (<=1.15):

pip install tensorflow-gpu

or

conda install tensorflow-gpu

GPU support

TensorFlow code runs without additional add-ons
on CUDA-able graphic cards:

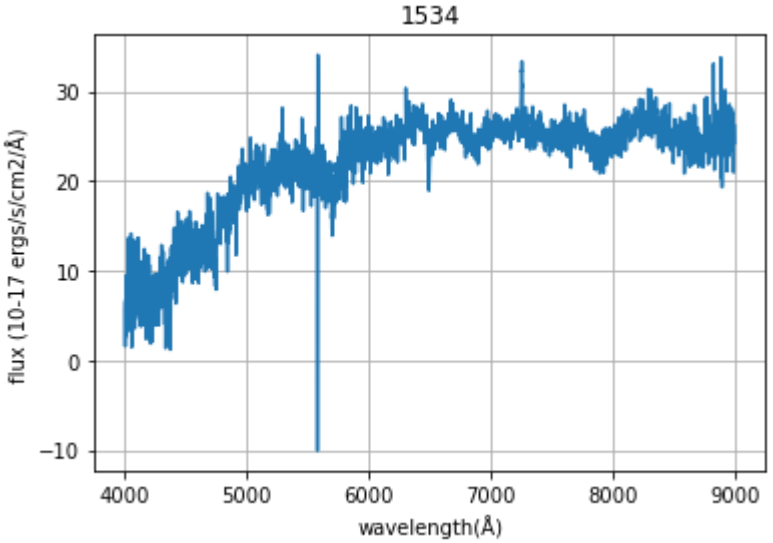
- NVIDIA GPU of generation G8x or newer
- All cards of GeForce, Quadro and Tesla series

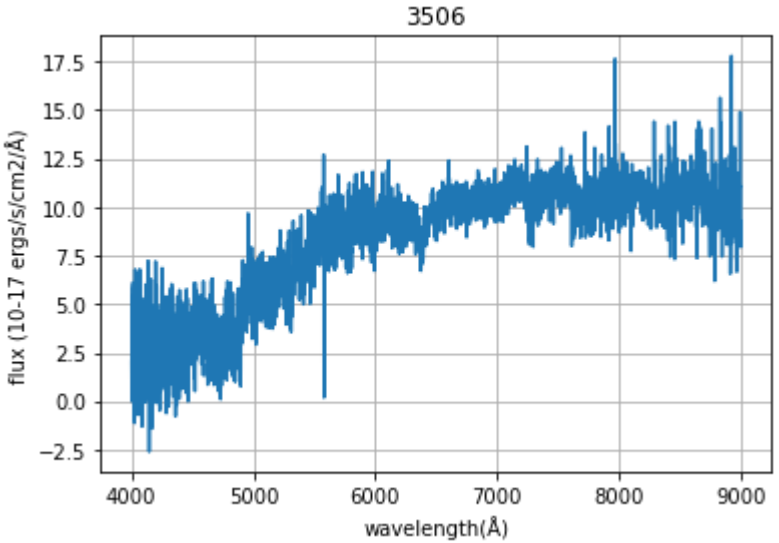
Otherwise available CPU are used (slow)

The SDSS database

- Sloan Digital Sky Survey (SDSS)
- Since 2000
- Full-sky survey in 5 wavelength bands (u, r, g, i, z)
- Telescope with 2.5m main mirror at Apache Point Obversatory, New Mexiko
- For example 12th data release contains > 4 million spectra
- Public access to data via Science Archive Server (SAS) or ‚Skyserver‘
  Check out some galaxies: https://data.sdss.org/sas/
  http://skyserver.sdss.org/dr12/

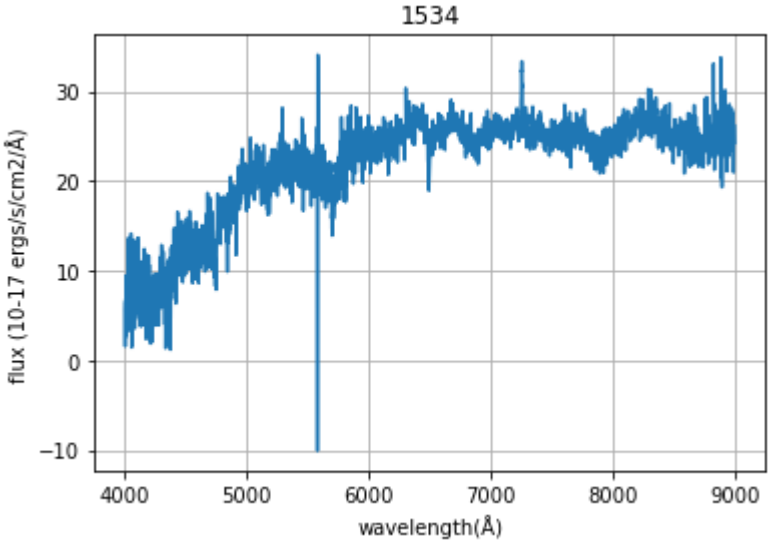
Classification of SDSS spectra



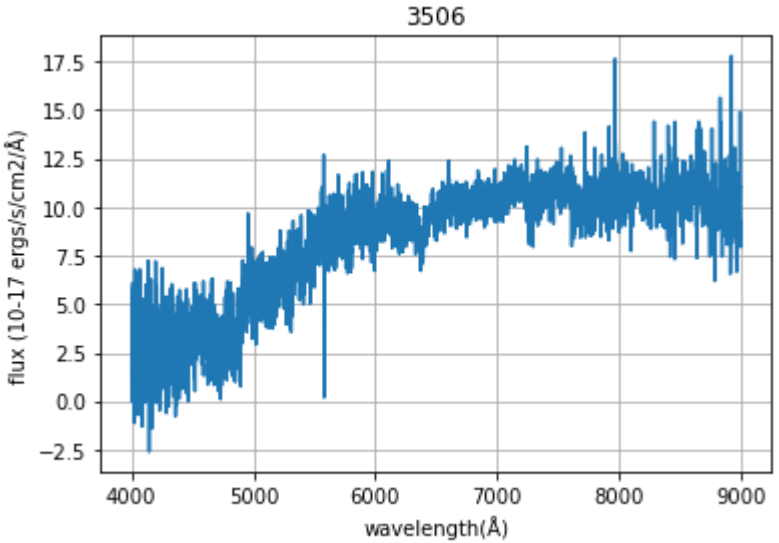


Which is a star and which is a galaxy?

Classification of SDSS spectra



Class: Galaxy



Class: Star

DR12 science archieve server (SAS)

- Access via Advanced Optical Spectra Search
- There is a database online search to find spectra and download data
- Here search masks can be allied, such as:

  Plate ID

  Date (MJD)

  Fiber on plate

  Object ID

Download from Skyserver

- Skyserver gives access to all public data (spectra, imaging - FITS-files)

- Access via browser-based navigation tool or SQL search

FITS = Flexible Image Transport System

Standardized, multi-dimensional arrays, tables + metadata

Download from Skyserver

- Example SQL search:

```
SELECT top 10 ra, dec, targetObjID, class
FROM SpecObj WHERE ra < 10
AND ra > 5 and class = 'star'
```

- Select the first 10 stellar spectra with coordinates 5 > RA > 10

Classification of SDSS spectra

Download from Skyserver

Easiest to use the astroquery package:

```
import sys
import os
import subprocess
import astropy.io.fits as pyfits
from astroquery.sdss import SDSS
from astropy.coordinates import SkyCoord, ICRS
import astropy.units as u
import requests

sdss_path = 'https://data.sdss.org/sas/dr16/sdss/spectro/redux/26/spectra/'
```

Example for download on github:
https://github.com/csheneka/ML-for-Astro-tutorial

Gold data set

- Almost half of the downloaded spectra had a zwarning flag other than 0
- We use a ‚gold sample' that we trust better, containing only spectra with zwarning = 0 in our SQL search
- As expected, improved performance

Save data in numpy arrays

- Downloaded FITS-files can be read and saved as numpy arrays

- In our example spectra of each of 4 classes (star, galaxy, AGN and QSO) were downloaded and saved as .npy:

Name: `data.npy` `labels.npy` `wavelengths.npy`

Full data: https://cloud.hs.uni-hamburg.de/s/bc9s3Z6mpnHYEDK

Classification of SDSS spectra

Setting things up

- Load the data

```
data = np.load(data_path + "data.npy")
labels = np.load(data_path + "labels.npy")
wavelengths = np.load(data_path + "wavelengths.npy")
```

- Shuffle the data

```
z = list(zip(data, labels, numbers))
random.shuffle(z)
data_shuffled, labels_shuffled, numbers_shuffled = zip(*z)
```

Setting things up

- Divide in training and test data

```
data_training = np.asarray(data_shuffled[:split_index])
data_test = np.asarray(data_shuffled[split_index:])

labels_training = np.asarray(labels_shuffled[:split_index])
labels_test = np.asarray(labels_shuffled[split_index:])

numbers_training = numbers_shuffled[:split_index]
numbers_test = numbers_shuffled[split_index:]
```

Classification of SDSS spectra

Setting things up

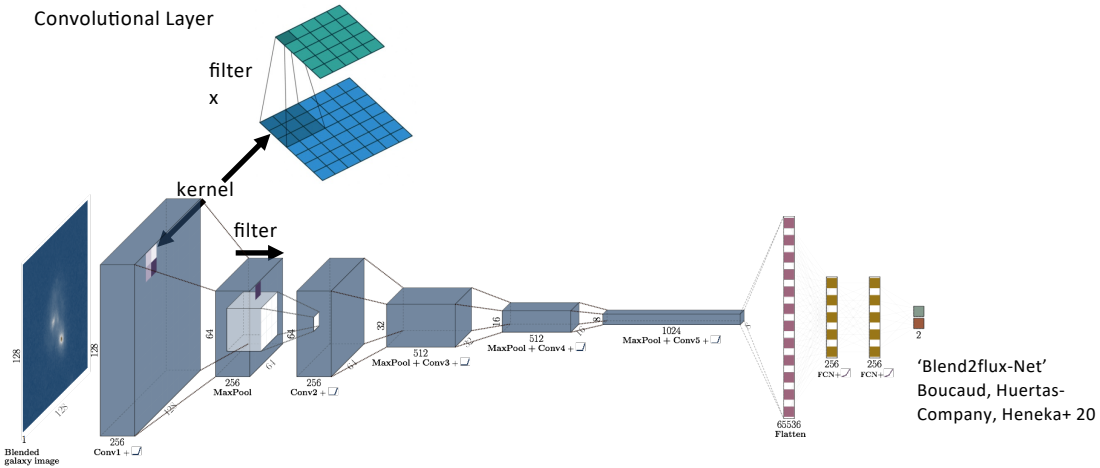
- Re-shaping for input to convolutional layer

```
input_shape = (3522,1)
data_training_r = np.reshape(data_training, newshape=(len(data_training),
        input_shape[0], input_shape[1]))
data_test_r  = np.reshape(data_test, newshape=(len(data_test),
        input_shape[0], input_shape[1]))
```

- Now we create the network model (-> run net)

Notes on: Convolutional neural networks

Crucial properties of a neural network



'Blend2flux-Net'
Boucaud, Huertas-Company, Heneka+ 20

Convolutional Layer

- pass on tensors (e.g. images!)
- convolve with kernels
- kernels are optimised

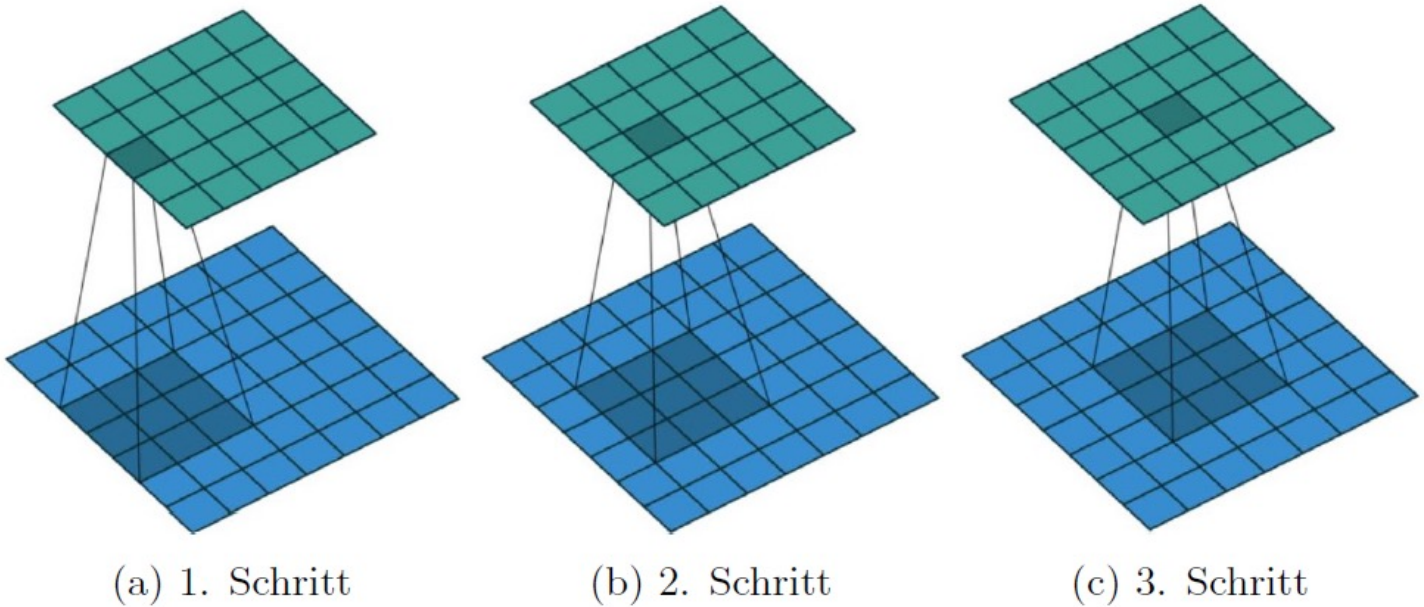
Basics

- **Convolutional Layer**: Layer that convolves data inputted
- Convolution matrix is called the **Kernel**
- Example for the creation of a convolutional 2D layer:

```
model = Sequential()
model.add(Conv2D (1,(3,3),
                  input_shape=(7,7,1)))
```

Convolutional network

How it works



Credits: https://aboucaud.github.io/adaix-ml-tutorial/slides/hands-on-deep-learning/#10

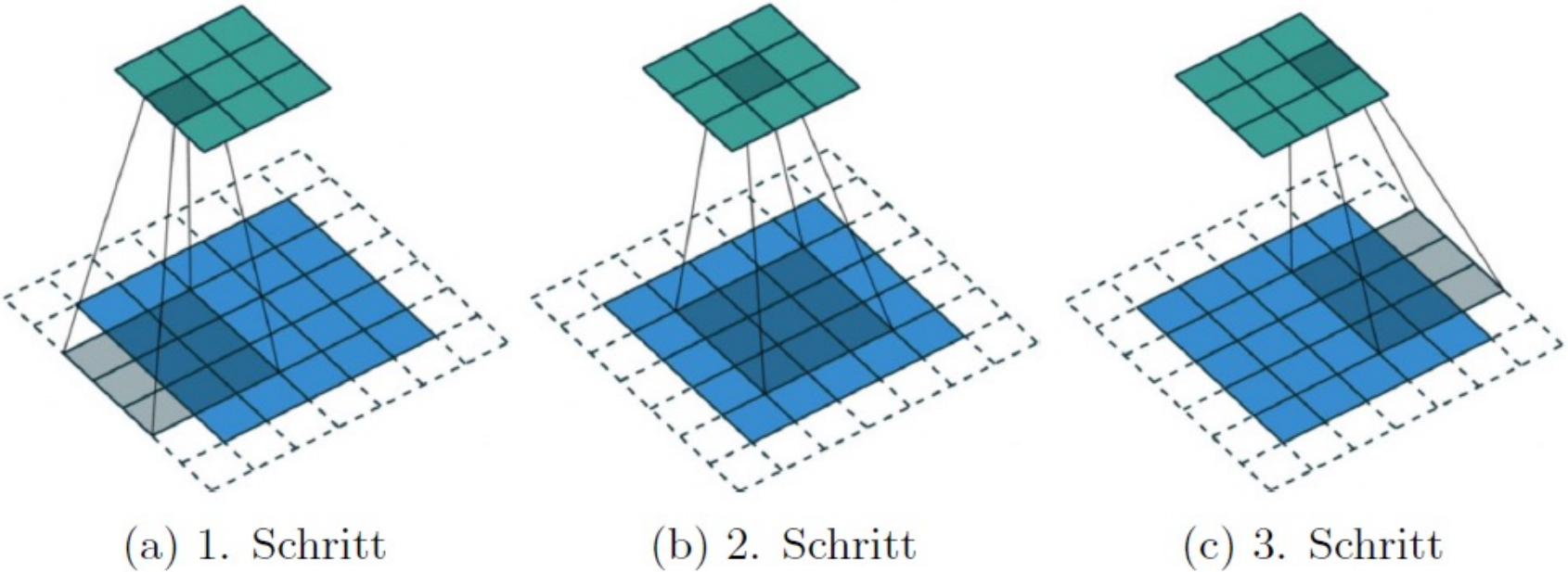
Basics 2

- **strides**: dertermines the step-size of the kernel
- **padding**: how the kernel treats the boundaries of the layer input
- For example:

```
model = Sequential()
model.add(Conv2D(1,(3,3),
                 strides=2,
                 padding='same',
                 input_shape=(7,7,1)))
```

Convolutional network

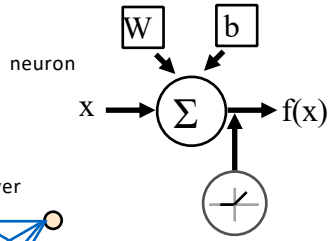
Basics 2



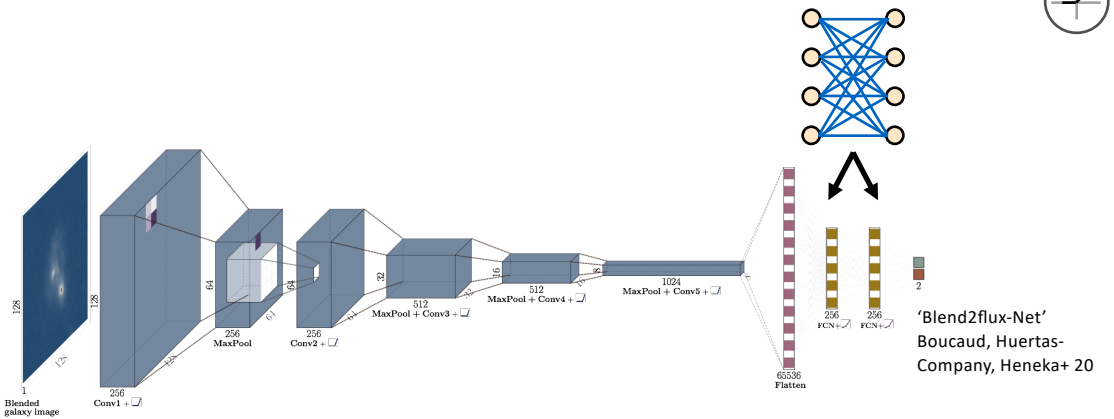
Credit: https://aboucaud.github.io/adaix-ml-tutorial/slides/hands-on-deep-learning/#10

Notes on: Convolutional neural networks

Crucial properties of a neural network



Fully connected layer



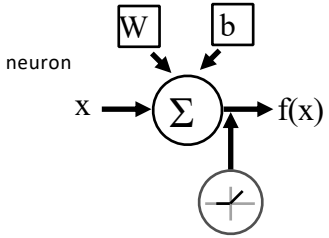
'Blend2flux-Net'
Boucaud, Huertas-Company, Heneka+ 20

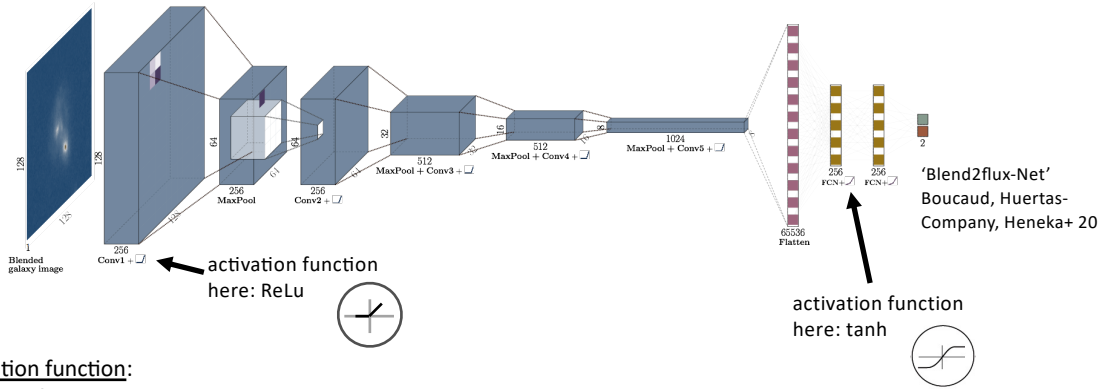
Fully connected layer

- pass on vectors (e.g. time series)
- between 2 layers, each neuron is interconnected
- optimise weights

Notes on: Convolutional neural networks

Crucial properties of a neural network

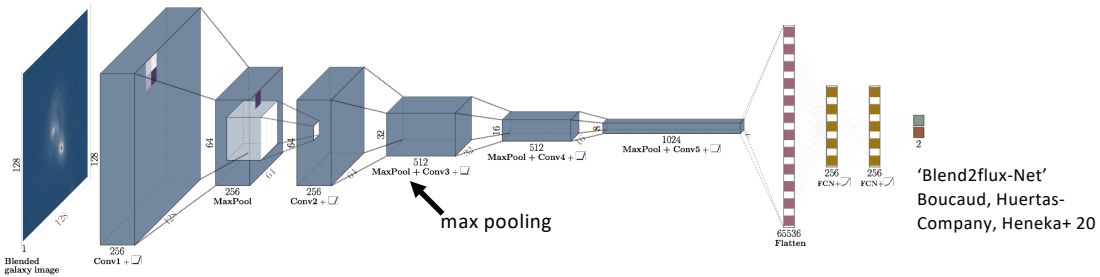




Activation function:
adds **non-linearity**

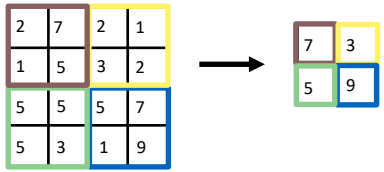
Notes on: Convolutional neural networks

Crucial properties of a neural network



'Blend2flux-Net'
Boucaud, Huertas-Company, Heneka+ 20

Avoidance of over-fitting:
max pooling,
average pooling,
drop-out



Classification of SDSS spectra

Creating the network model

```
model = Sequential([
    Conv1D(filters=64, kernel_size=80, strides=10,
           activation='relu', input_shape=(3522,1)),
    MaxPooling1D(3),
    Dropout(0.35),
    Conv1D(filters=128, kernel_size=40, strides=10, activation='relu'),
    MaxPooling1D(3),
    Dropout(0.35),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dropout(0.35),
    Dense(units=4, activation='softmax')
])
```

$$\max(0, x)$$

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Compilation and training

- First we compile the network model…

```
model.compile(optimizer='Adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- …and then we can train it

```
x_train = tf.keras.utils.normalize(data_training_r, axis=1)
x_test = tf.keras.utils.normalize(data_test_r, axis=1)

y_train = labels_training
y_test = labels_test

history = model.fit(x_train, y_train,
```

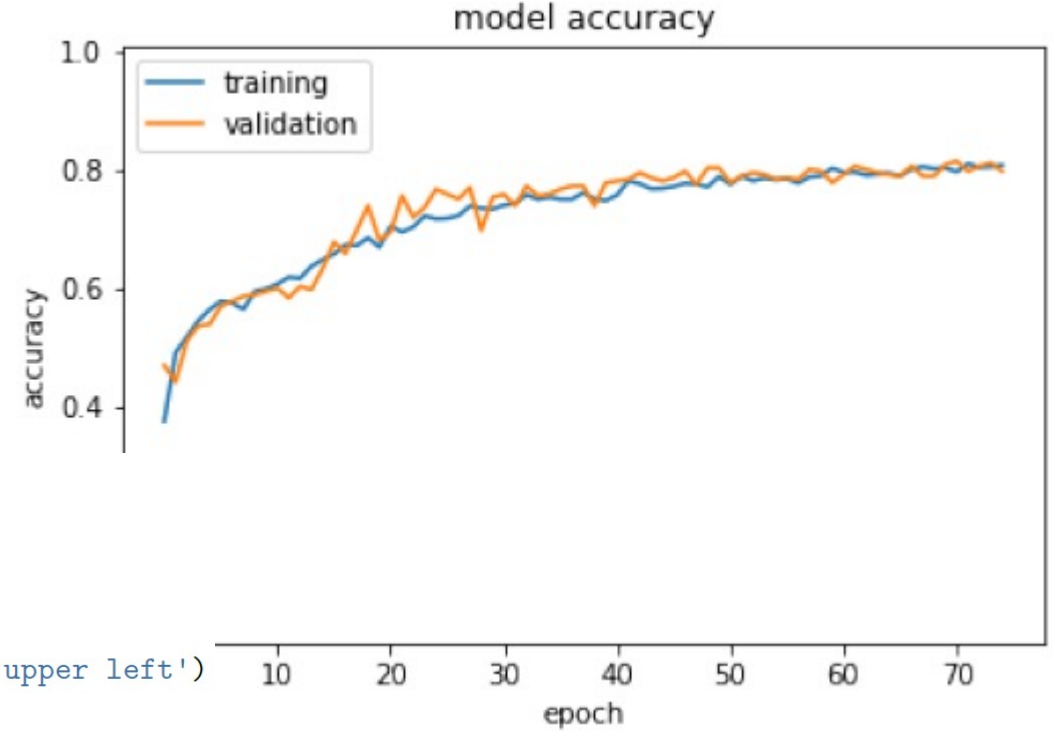
Network training

- We divide the training data into training and **validation** sets
- The **batch-size** determines how many data are used simultaneously in training
- The **history** object saves the training progress
- **Epochs** designate the number of training steps with the full training set

```
history = model.fit(train_samples, train_labels, epochs=30,
                    validation_split=0.1, batch_size=120, shuffle=True)
```

Classification of SDSS spectra

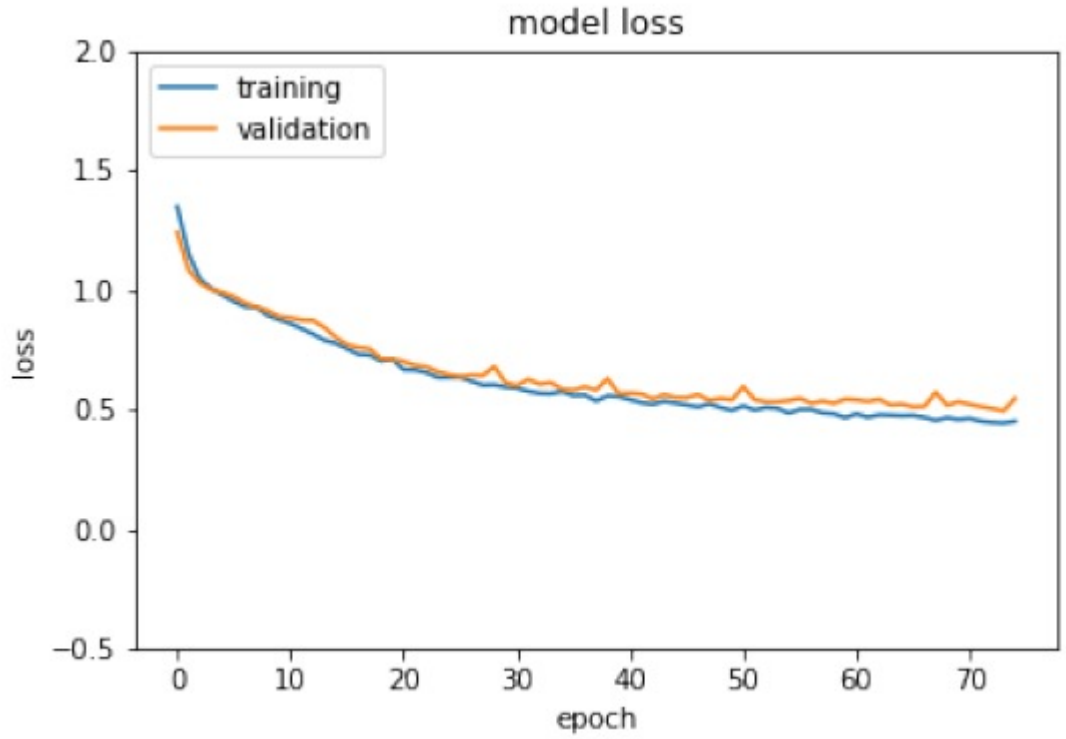
Accuracy




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper left')
plt.show()
```

Classification of SDSS spectra

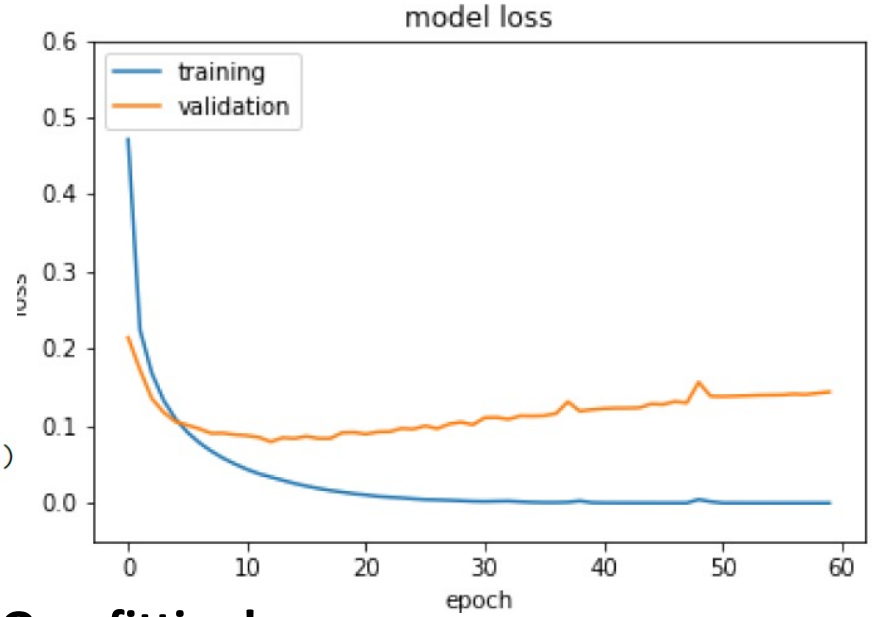
Model loss



Loss function

Enforces training and evaluates training progress

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper left')
plt.show()
```



Training does not work optimally! Overfitting!

Network predictions

- The **predict-Funktion** derives network predictions:

```
predictions = model.predict(x=test_samples)
```

- Get the most probable numbers:

```
rounded_predictions = np.argmax(predictions, axis=-1)
```

Accuracy of predictions

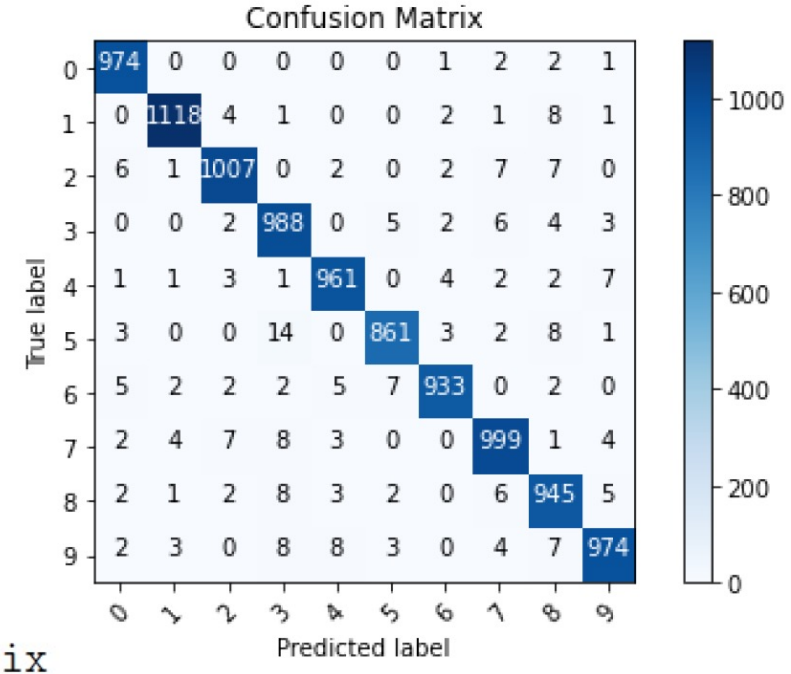
- Your can check the accuracy of predictions:

```
accuracy = Accuracy()
accuracy.update_state(y_true=test_labels, y_pred=rounded_predictions)
accuracy.result().numpy()
```

- This of course is only valid in supervised training for a test set (not used in training) with known labels.

Confusion matrix

- Tool to judge performance
- Which classes where miss-classified?




```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true=test_labels, y_pred=rounded_predictions)
```

Save and load networks

1.) model.save **Saves structure, weights (+bias), training configuration and compiler status**

- Save:

```
model.save('digit_recognizer_model.h5')
```

- Load:

```
from tensorflow.keras.models import load_model
model1 = load_model('digit_recognizer_model.h5')
```

Save and load networks

2.) model.save_weights **Only saves weights**

- Save:

```
model.save_weights('my_weights.h5')
```

- Load:

```
model3.load_weights('my_weights.h5')
```

Outline of this lecture

Part 1: Hands-on tutorial (ca. 60 min)

1) Short motivation and basics

2) Classification of SDSS spectra

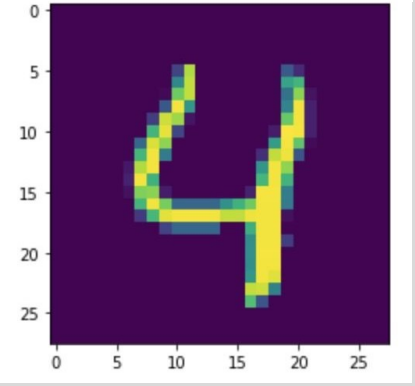
Break! 5-10min

Part 2: Current astrophysical applications (ca. 30 min)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Machine Learning

CLUSTER OF EXCELLENCE
QUANTUM UNIVERSE

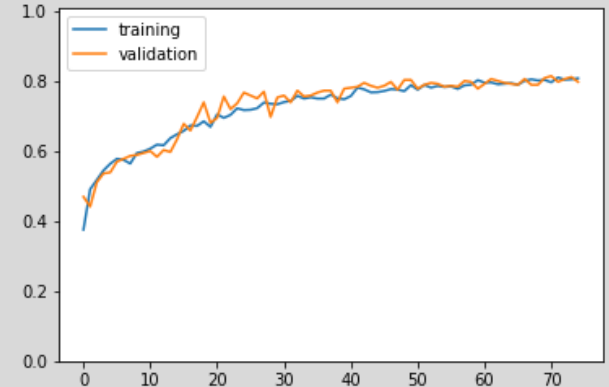


Documentation, Code and Slides:
https://gitlab.rrz.uni-hamburg.de/bay8647/ml-for-astro-tutorial-sternwarte

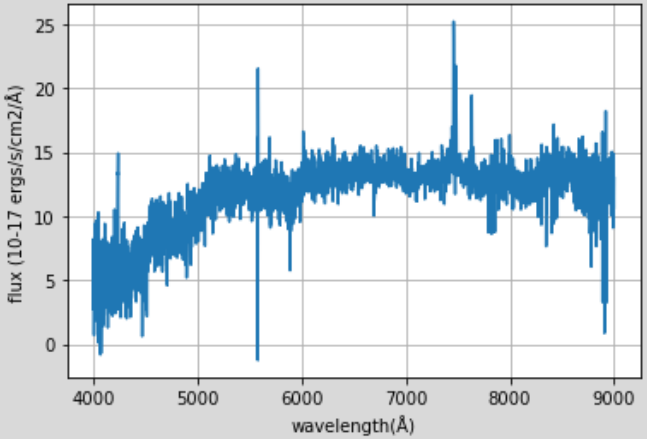
Contact:
caroline.heneka@hs.uni-hamburg.de

Joshua Roshlaub (contact: joshuaroschlaub1999@gmail.com)

1 Keras basics



2 Getting to know the HPC Cluster



3 Classification of SDSS Spectra