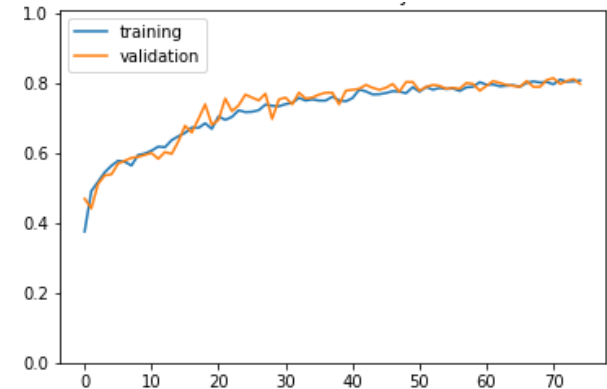
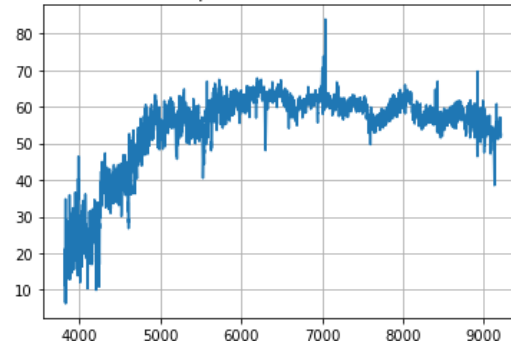
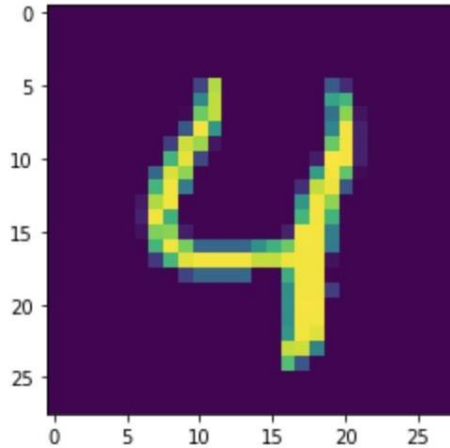


MACHINE LEARNING

EINE EINFÜHRUNG IN KERAS



GLIEDERUNG

- 1) Das erste eigene Netzwerk
- 2) Convolutional Layer
- 3) Das HPC Rechencluster
- 4) Klassifizierung von SDSS Spektren

INSTALLATION VON TENSORFLOW

Normales Paket:

pip install tensorflow

oder

conda install tensorflow

Zur Ausführung auf gpu's:

pip install tensorflow-gpu

oder

conda install tensorflow-gpu

GPU UNTERSTÜTZUNG

TensorFlow Code läuft ohne zusätzlichen Befehle auf CUDA-fähigen Grafikkarten:

- NVIDIA GPU der G8x-Generation oder neuer
- Alle Karten der GeForce, Quadro und Tesla Reihen

Ansonsten wird die CPU verwendet

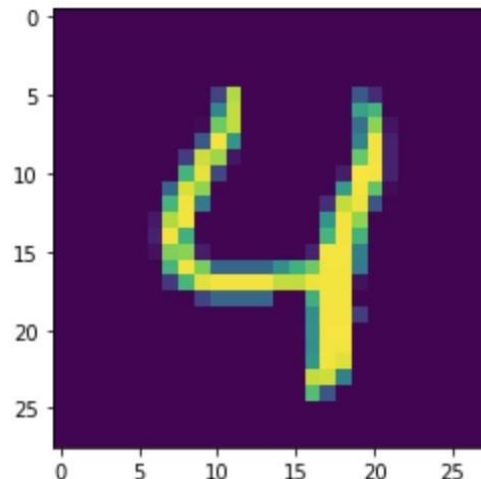
LADEN DER DATENSÄTZE

MNIST-Datensatz:

- 60.000 Trainings-Bilder
- 10.000 Test-Bilder

```
import tensorflow as tf
from tensorflow import keras
```

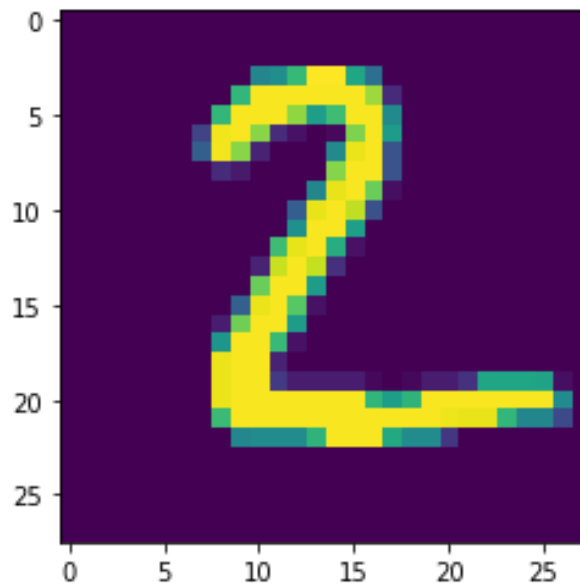
```
mnist = keras.datasets.mnist
(train_samples, train_labels) = mnist.load_data()[0]
(test_samples, test_labels) = mnist.load_data()[1]
```



ANZEIGEN DER BILDER

Die ersten 10 Einträge:

```
for i in range(10):  
    plt.imshow(test_samples[i])  
    plt.show()
```



NORMIERUNG DER DATEN

- Eine Normierung der Daten verbessert häufig das Training
- Zum Beispiel auf eine Größe zwischen 0 und 1:

```
train_samples = tf.keras.utils.normalize(train_samples, axis=1)
test_samples = tf.keras.utils.normalize(test_samples, axis=1)
```

ERSTELLUNG DES NETZWERKES

Wir benötigen folgende Module:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten,
                                   Conv2D, MaxPool2D
from tensorflow.keras.metrics import Accuracy
```

Eine simple Architektur:

```
model = Sequential([
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=10, activation='softmax')
])
```

ERSTELLUNG DES NETZWERKES

Vor dem Training muss noch die **compile-Funktion** aufgerufen werden:

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

- Der Optimizer **Adam** ist häufig eine gute Wahl
- „**sparse**“ bedeutet, die Labels sind durchnummerierte integer

TRAINIEREN DES NETZWERKES

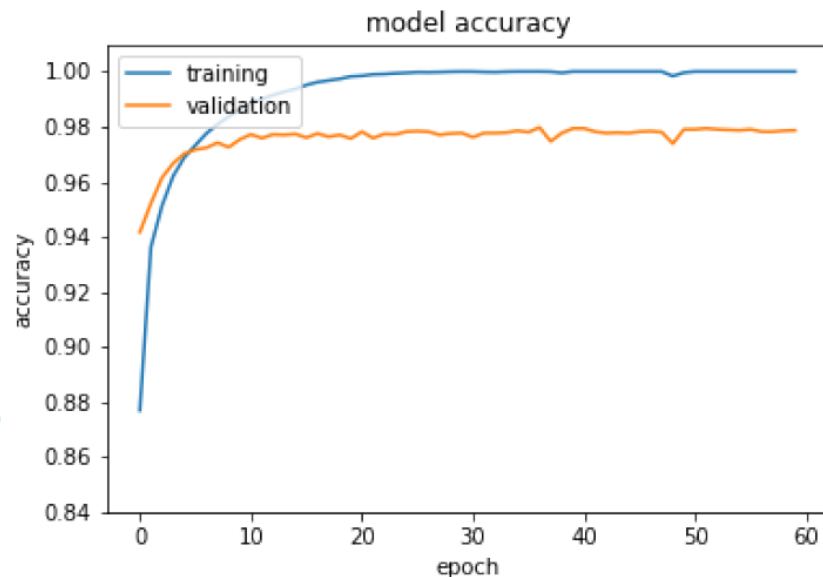
- Der Trainings-Datensatz wird in Trainings- und **Validation**-Daten aufgeteilt
- Die **batch-size** legt fest, mit wie vielen Daten zeitgleich trainiert wird
- In dem Objekt **history** wird der Trainingsfortschritt gespeichert
- Die **Epochen** geben an, wie häufig mit dem gesamten Datensatz trainiert wird

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

TREFFERGENAUIGKEIT

Wie gut „rät“ das Netzwerk

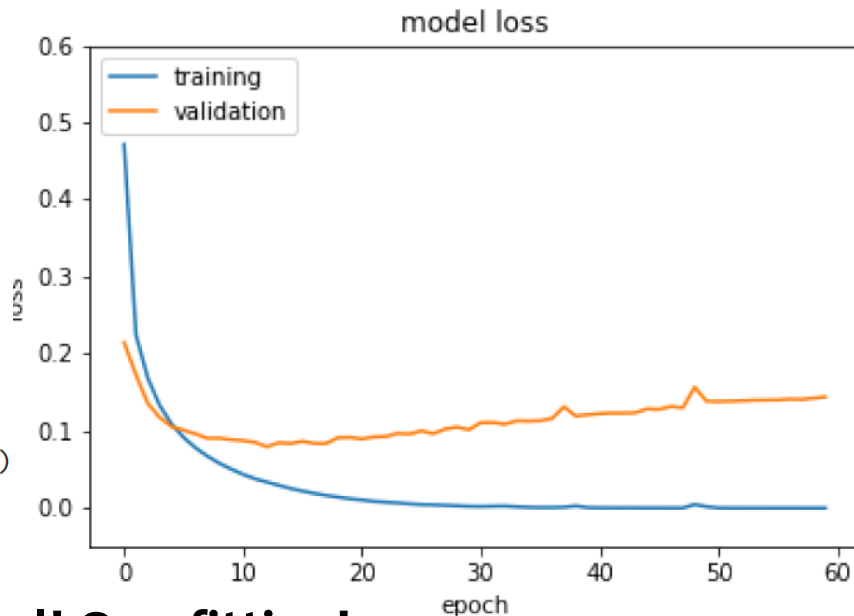
```
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['training', 'validation'], loc='upper left')  
plt.show()
```



LOSS-FUNKTION

Zeigt den Trainingsfortschritt

```
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 funktioniert nicht optimal! Overfitting!

NETZWERK VORHERSAGEN

- Mit der **predict-Funktion** kann das Netzwerk Vorhersagen treffen:

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

- Ziffern mit größter Wahrscheinlichkeit:

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

TREFFERGENAUIGKEIT VON VORHERSAGEN

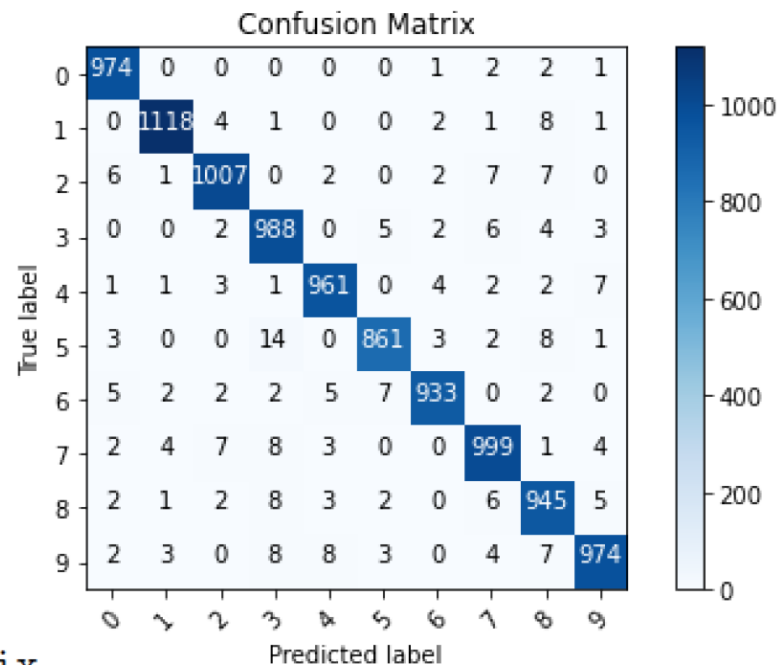
- Die Treffergenauigkeit kann überprüft werden:

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

- Das geht natürlich nur, wenn die wahren Labels bekannt sind

CONFUSION MATRIX

- Wichtiges tool zur Analyse eines Netzwerks
- Welche Klassen wurden richtig und welche falsch geraten?



```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_true=test_labels, y_pred=rounded_predictions)
```

NETZWERKE SPEICHERN UND LADEN

1.) `model.save` Speichert Struktur, Gewichte,
Trainingskonfiguration und Status des compilers

- Speichern:

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

- Laden:

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

NETZWERKE SPEICHERN UND LADEN

2.) `model.to_json` Speichert nur Struktur des Netzwerks

- Speichern:

```
json_string = model.to_json()
```

- Laden:

```
from tensorflow.keras.models import model_from_json  
model2 = model_from_json(json_string)
```

NETZWERKE SPEICHERN UND LADEN

3.) `model.save_weights` Speichert nur die Gewichte

- Speichern:

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

- Laden:

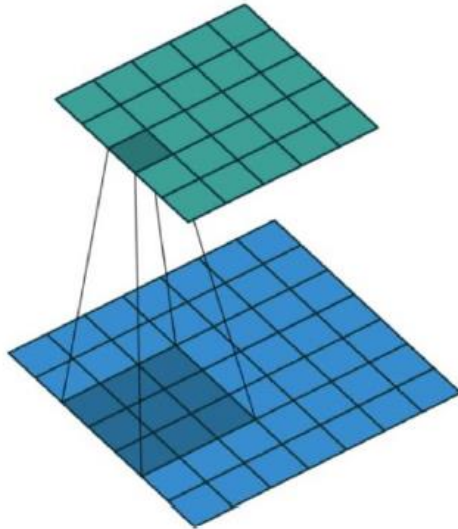
```
model3.load_weights('my_weights.h5')
```

GRUNDLAGEN

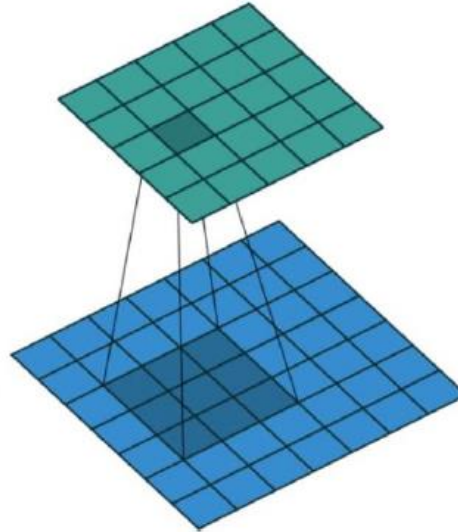
- **Convolutional Layer:** Ebene an der Eingabe Daten gefaltet werden
- Faltungsmatrix wird als **Kernel** bezeichnet
- Beispiel für die Erstellung einer Conv2D-Ebene:

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

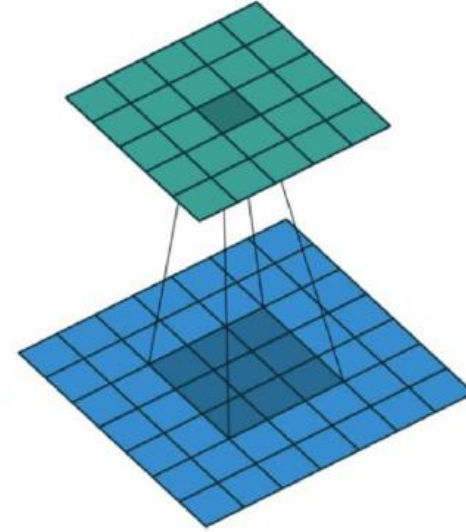
FUNKTIONSWEISE



(a) 1. Schritt



(b) 2. Schritt



(c) 3. Schritt

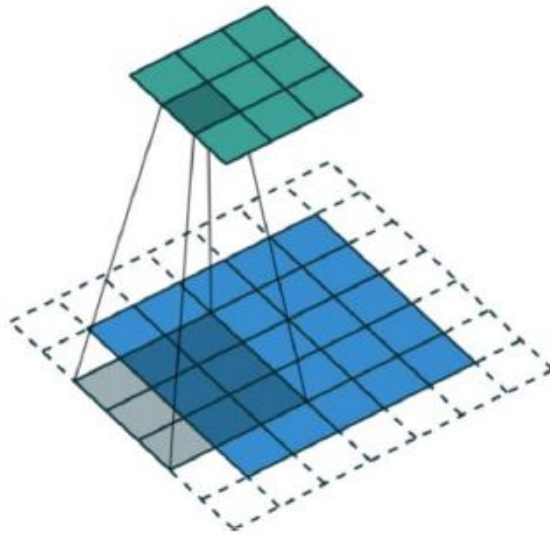
Übernommen von: <https://aboucaud.github.io/adaix-ml-tutorial/slides/hands-on-deep-learning/#10>

GRUNDLAGEN 2

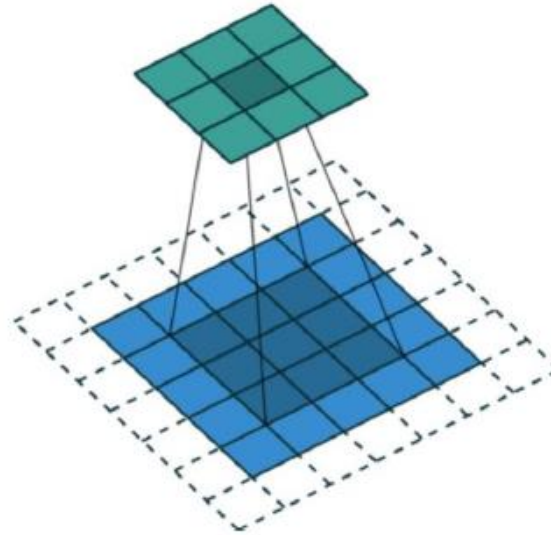
- **strides**: Bestimmt die Schrittweise, mit welcher sich der Kernel bewegt
- **padding**: Legt fest, ob sich der Kernel auch über den Rand hinaus bewegt
- Zum Beispiel:

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

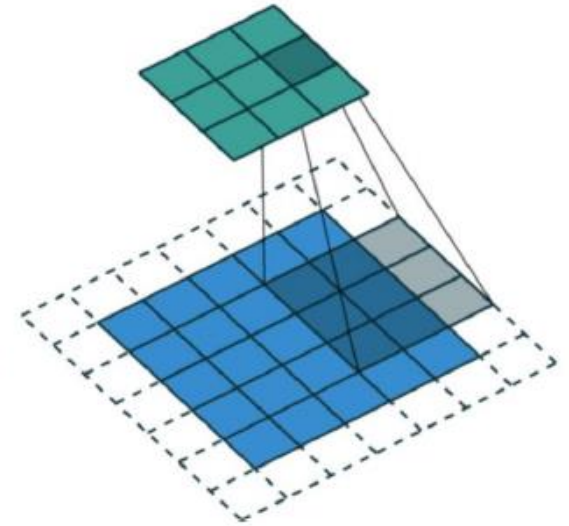
FUNKTIONSWEISE 2



(a) 1. Schritt



(b) 2. Schritt



(c) 3. Schritt

Übernommen von: <https://aboucaud.github.io/adaix-ml-tutorial/slides/hands-on-deep-learning/#10>

ANWENDUNG AUF MNIST DATEN

- Die Daten benötigen eine extra Dimension:

```
train_samples = train_samples.reshape(len(train_samples),28,28,1)
test_samples  = test_samples.reshape(len(test_samples),28,28,1)
```

- Stellt bei Farbbildern den Farbkanal da (rgb)

ANWENDUNG AUF MNIST DATEN

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3,3), activation='relu',
           input_shape=(28,28,1)),
    Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    MaxPool2D(pool_size=(2,2)),
    Dropout(0.25),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dropout(0.5),
    Dense(units=10, activation='softmax')
])
```

ANWENDUNG AUF MNIST DATEN

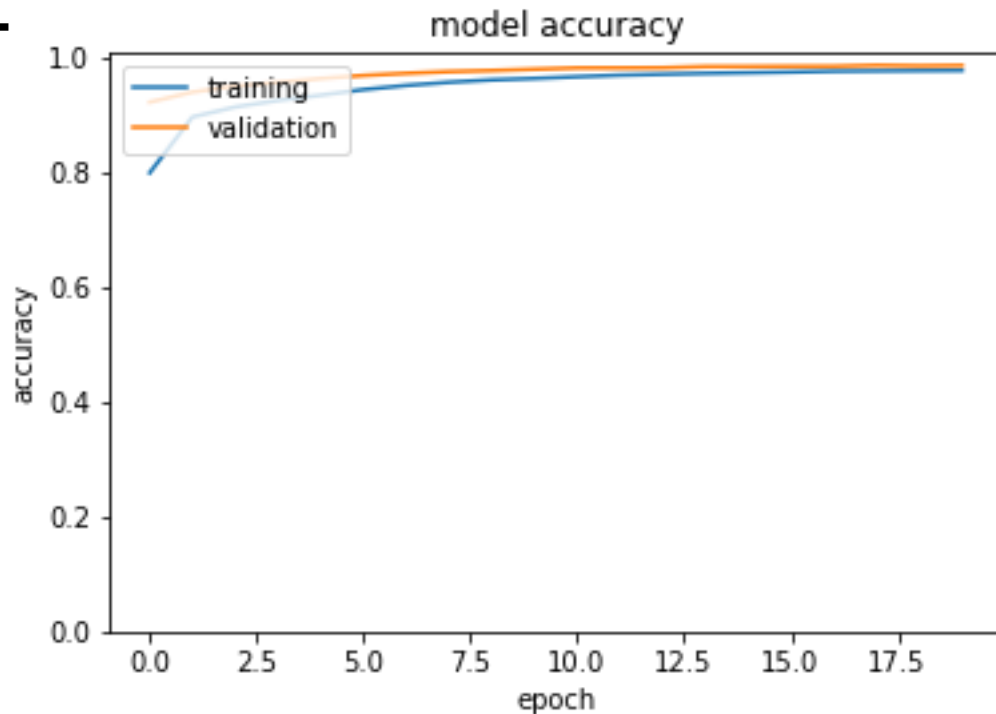
- Aufrufen der compile-Funktion:

```
model.compile(optimizer=keras.optimizers.Adadelta(learning_rate=0.1),  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

- **Optimizer:** Adadelta mit einer Learning-Rate = 0.1 (Standard = 0.01)

NEUE TREFFERGENAUIGKEIT

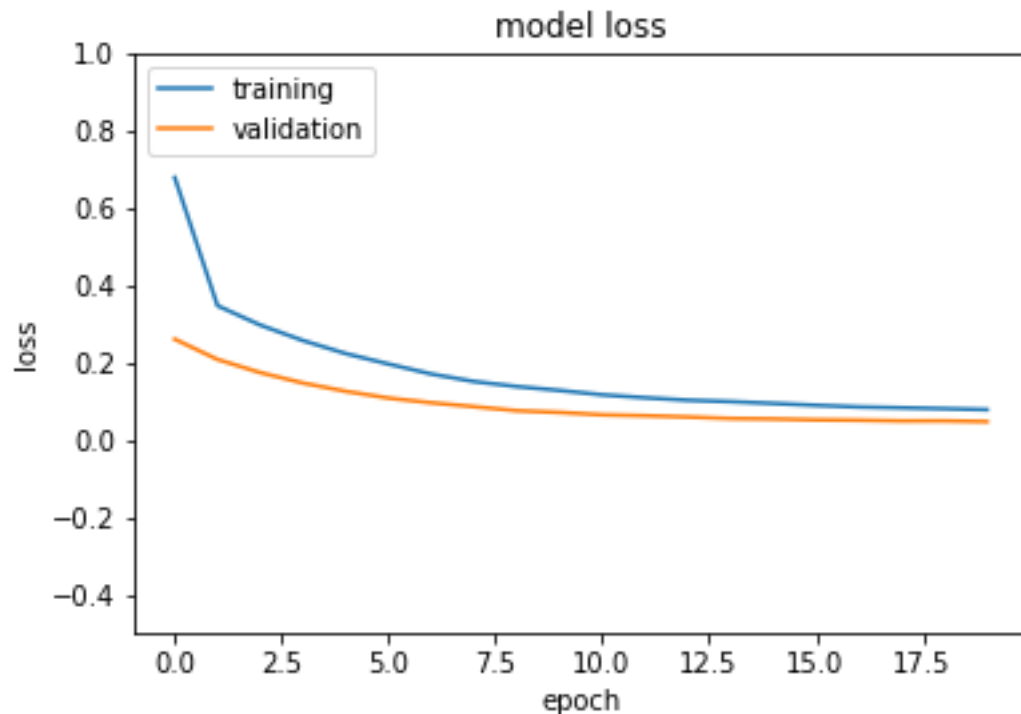
- Viel besseres Ergebnis!
- Training und Validation nähern sich an
- Kein Overfitting mehr



NEUE LOSS-FUNKTION

- Auch hier bestätigt sich:

**Das Convolutional Network
klassifiziert die Ziffern
deutlich besser!**



ALLGEMEINES

„Das Linux-Cluster „Hummel“ am RRZ wurde als Forschungsgroßgerät für wissenschaftliche Projekte mit großem parallelen Rechenbedarf beschafft.“

Seit **2009** im Betrieb



Übernommen von: <https://www.rrz.uni-hamburg.de/services/hpc.html>

ZUGANG

Notwendig:

- VPN Zugang der UHH (siehe Website)
- Freischaltung durch hpc@uni-hamburg.de
- Austausch eines öffentlichen SSH Schlüssels

```
ssh-keygen -t rsa -b 4096 -f $HOME/.ssh/id_rsa_hummel  
ssh -i $HOME/.ssh/id_rsa_hummel
```

Generierung einer Schlüsselpaars im Home Ordner und Annahme der Identität

ZUGANG

- Zwei Login-Gateways

`hummel1.rrz.uni-hamburg.de`

`hummel2.rrz.uni-hamburg.de`

- Ermöglichen Zugriff auf die \$WORK und \$HOME Ordner
- Verbindung zu Front-End-Knoten per ssh

`ssh front1`

`ssh front2`

- Zusammengefasst:

```
ssh -t Stine-Kennung@hummel1.rrz.uni-hamburg.de ssh front1
```

INSTALLATION VON ANACONDA + TENSORFLOW

- Download des [Anaconda Installer](#)
- Installer muss per scp in den \$WORK Ordner kopiert werden

```
scp home/Pfad/zu/Anaconda3-2021.05-Linux-x86_64.sh  
Stine-Kennung@hummel1.rrz.uni-hamburg.de:/work/Stine-Kennung/
```

- Installation von Anaconda mit

```
bash Anaconda3-2021.05-Linux-x86_64.sh
```

INSTALLATION VON ANACONDA + TENSORFLOW

- Pfad zu conda.sh in .bashrc Datei (im \$HOME Ordner) hinterlegen

```
/work/Stine-Kennung/anaconda3/etc/profile.d/conda.sh
```

- Aktivierung der Installation mit

```
source ~/.bashrc
```

- Überprüfung der Installation mit `conda list`

INSTALLATION VON ANACONDA + TENSORFLOW

- Einrichtung einer neuen Umgebung

```
conda create --name my_env python=3  
conda activate my_env
```

- Installation von Tensorflow-gpu und matplotlib

```
conda install tensorflow-gpu  
conda install matplotlib
```

INSTALLATION VON ANACONDA + TENSORFLOW

In einer neuen Sitzung können Anaconda und die neue Umgebung wieder geladen werden:

```
source ~/.bashrc  
conda activate my_env
```

BATCH-VERARBEITUNG

- Für einen compute-job auf einem der Rechenknoten muss das Batch-System verwendet werden
- Auf Hummel wird SLURM verwendet

(SIMPLE LINUX UTILITY FOR RESOURCE MANAGEMENT)

- Dokumentation: [SLURM Website](#)
- Jobs werden mit einem Job-Skript gestartet

BEISPIELJOB

```
#!/bin/bash
#SBATCH --job-name=keras_example
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --tasks-per-node=16
#SBATCH --time=00:10:00
#SBATCH --output=keras_example_output
#SBATCH --error=keras_example_error
#SBATCH --mail-user=name@studium.uni-hamburg.de
```

BEISPIELJOB

```
#SBATCH --mail-type=ALL
```

```
set -e
```

```
module switch env env/2020Q3-gcc-openmpi
```

```
source ~/.bashrc
```

```
conda activate my_env
```

```
module load cuda
```

```
python $HOME/scripts/run_example.py
```

DIE WICHTIGSTEN SLURM-BEFEHLE

- Um ein Job-Skript hochzuladen: `$ sbatch script.sh`
- Um alle eigenen laufenden Jobs anzuzeigen: `$ squeue -u $USER`

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
1461514	std	name	abc	R	1:37	2	node269

- Detaillierte Informationen: `$ scontrol show job JOBID`

DIE WICHTIGSTEN SLURM-BEFEHLE

- Um einen Job abzubrechen: `$ scancel JOBID`
- Um einen Job bei einem automatischen Abbruch nicht erneut zu starten:

```
$ sbatch --no-requeue script.sh
```

- Oder im Skript:

```
#Sbatch --no-requeue
```

CPU PARTITIONEN

- Bis auf die Spezial-Knoten: Alle Compute-Knoten 2 CPUs vom Typ Intel Xeon E5-2630v3 – Je CPU:
 - **8 RECHENKERNE**
 - **GRUNDFREQUENZ 2,4 GHZ**
 - **L3-CACHE 20 MBYTE**
- Es sind 3 unterscheidliche CPU-Partitionen zur Verfügung

CPU PARTITIONEN

1. Standard-Partition

316 Knoten (node1 bis node 316)

jeweils 64 Gbyte Hauptspeicher

2. Große Partition

24 Knoten (node371 bis node 394)

jeweils 256 Gbyte Hauptspeicher

3. Spezial-Partition

2 Knoten (node395 bis node 396)

jeweils 1024 Gbyte Hauptspeicher

GPU PARTITION

- 4992 NVIDIA CUDA-Cores
- 24 Gbyte GDDR5 Speicher
- Speicher-Bandbreite von 480 Gbyte/s
- Grudfrequenz 562 MHz
- Boost-Frequenz 824 MHz

GPU Partition

54 Knoten (node317 bis node 370)

Jeweils 64 Gbyte Hauptspeicher

GPU PARALLELISIERUNG

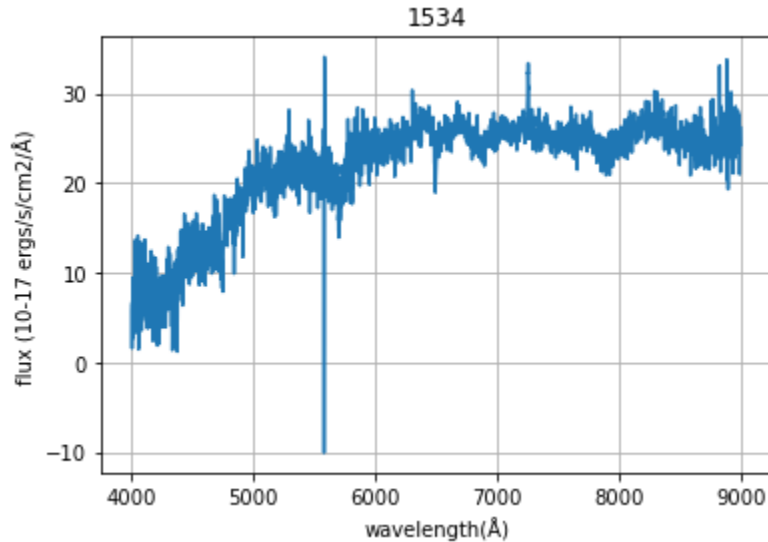
- Parallelisierung der GPUs wird mithilfe von OpenMPI gesteuert
- Zunächst muss die Umgebung mit OpenMPI und dem Cuda-Modul geladen werden

```
module switch env env/2020Q3-gcc-openmpi  
module load cuda
```

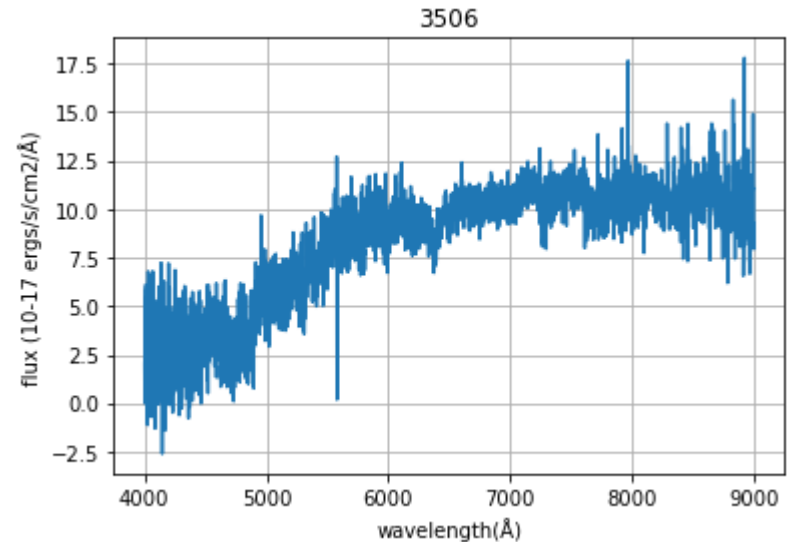
DER SDSS KATALOG

- Sloan Digital Sky Survey (SDSS)
- 2004 begonnenes Projekt
- Vermessung des Himmels in 5 Wellenlängenbereichen (u, r, g, i, z)
- Teleskop mit 2.5m Hauptspiegeldurchmesser am Apache Point Observatory in New Mexiko
- 12. Datenrelease beinhaltet mehr als 4 Millionen Spektren
- Zugriff auf Daten z.B. über den Science Archive Server (SAS) oder den Skyserver

KLASSIFIZIERUNG VON SDSS SPEKTRALDATEN



Klasse: Galaxie



Klasse: Stern

DR12 SCIENCE ARCHIEVE SERVER (SAS)

- Zugriff über die [Advanced Optical Spectra Search](#)
- Über eine Suchfunktion können Spektren gefunden und heruntergeladen werden
- In verschiedenen Reitern können die wichtigsten Parameter eingegrenzt werden, zum Beispiel:
 - Platten-Identifikationsnummer (Plate ID)
 - Julianische Datum (MJD)
 - Platten-Faser (fiber)
 - Identifikationsnummer des Objekts (Thing_ID)

DOWNLOAD VOM SKYSERVER

- Der Skyserver bietet Zugriff auf alle öffentlich verfügbaren Daten (Spektren, fits-Dateien, jpg-Bilder, usw.)
- Zugriff bsw. über das [Navigations-Tool](#) oder eine [SQL-Suche](#)
- Der [Schema-Browser](#) listet alle durchsuchbaren Tabellen auf

DOWNLOAD VOM SKYSERVER

- Beispiel einer SQL-Suche:

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

- Auswahl der ersten 10 Spektren mit $5 > ra > 10$

DOWNLOAD VOM SKYSERVER

- Ausgewählte Daten können heruntergeladen werden:

```
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/'
```

KLASSIFIZIERUNG VON SDSS SPEKTRALDATEN

```
sdss = SDSS.query_sql(queries[i])    ← SQL-Suche
speclist = open('speclist.txt', 'w')
```

```
for plate, mjd, fiberid in zip(sdss['plate'],sdss['mjd'],sdss['fiberid']):
    speclist.write("%04d/spec-%04d-%d-%04d.fits \n" %(plate, plate, mjd, fiberid))
speclist.close()
```

```
with open('speclist.txt', 'r') as f:
    names = f.readlines()
```

Beispiel: spec-0266-51630-0013.fits

```
for item in names:
    name = item[:-2]
    url = sdss_path + name
    r = requests.get(url)
    target_file = 'F:\data\spectral_fits\\' + class_names[i] + '\\\ ' + name[5:]

    with open(target_file,'wb') as f:
        f.write(r.content)
```

ABSPEICHERUNG IN NUMPY ARRAYS

- Die heruntergeladenen fits-Dateien können als numpy Arrays eingelesen und abgespeichert werden
- In einem Beispiel wurden jeweils 1000 Spektren der 4 Klassen Star, Galaxy, AGN und QSO heruntergeladen und in 3 Arrays abgespeichert:

Name:	<code>data.npy</code>	<code>labels.npy</code>	<code>wavelengths.npy</code>
Form:	<code>(4000, 3522)</code>	<code>(3522,)</code>	<code>(4000,)</code>

ERSTELLUNG DES NETZWERKS

- Laden der Daten

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

- Mischen der Daten

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

ERSTELLUNG DES NETZWERKS

- Aufteilung in Trainings- und Testdaten

```
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:]
```

ERSTELLUNG DES NETZWERKS

- Re-shaping für 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]))
```

- Nun kann das Netzwerk erstellt werden

ERSTELLUNG DES NETZWERKS

```
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')
```

ERSTELLUNG DES NETZWERKS

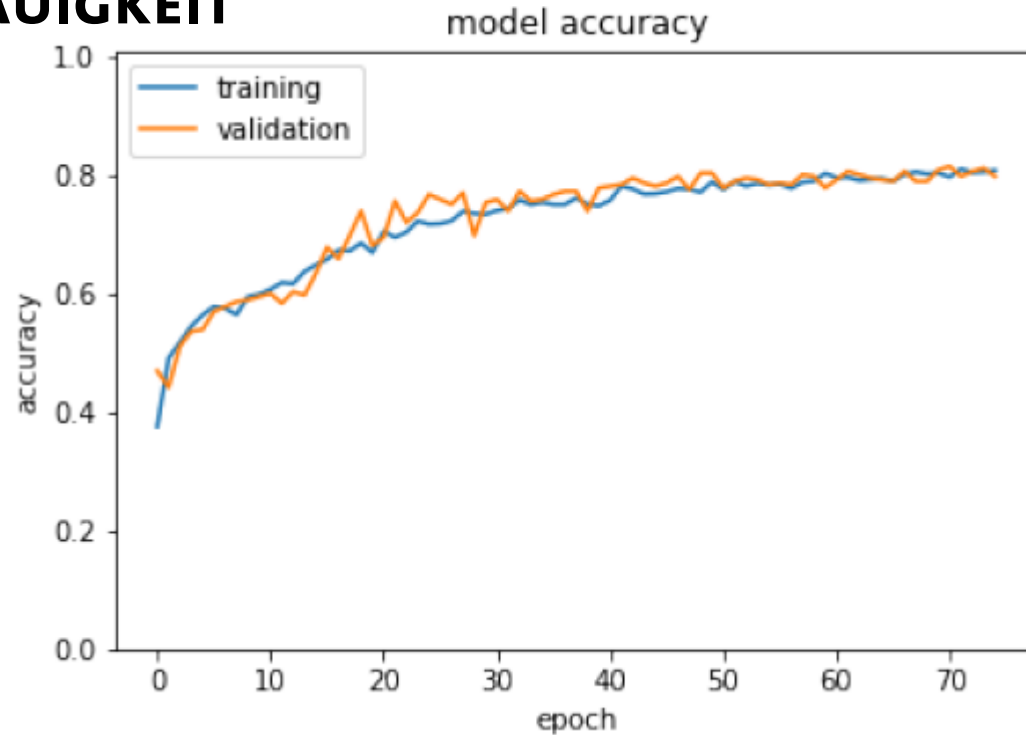
- Netzwerk kann compiliert...

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

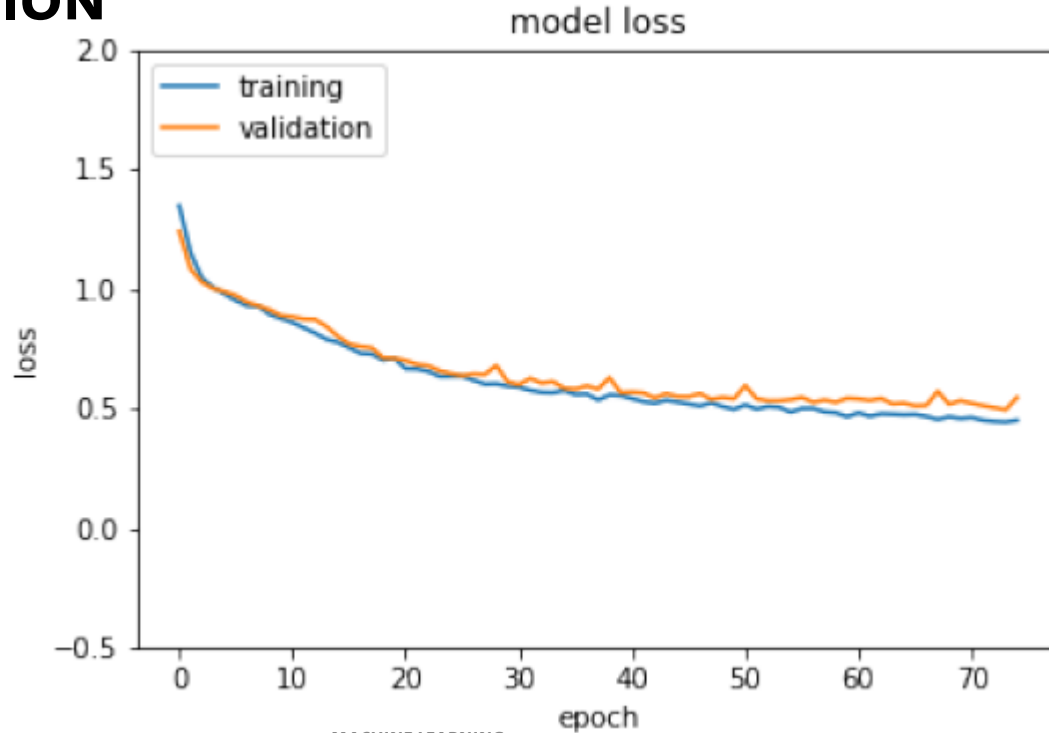
- ...und trainiert werden

```
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,
```

TREFFERGENAUIGKEIT



LOSS-FUNKTION



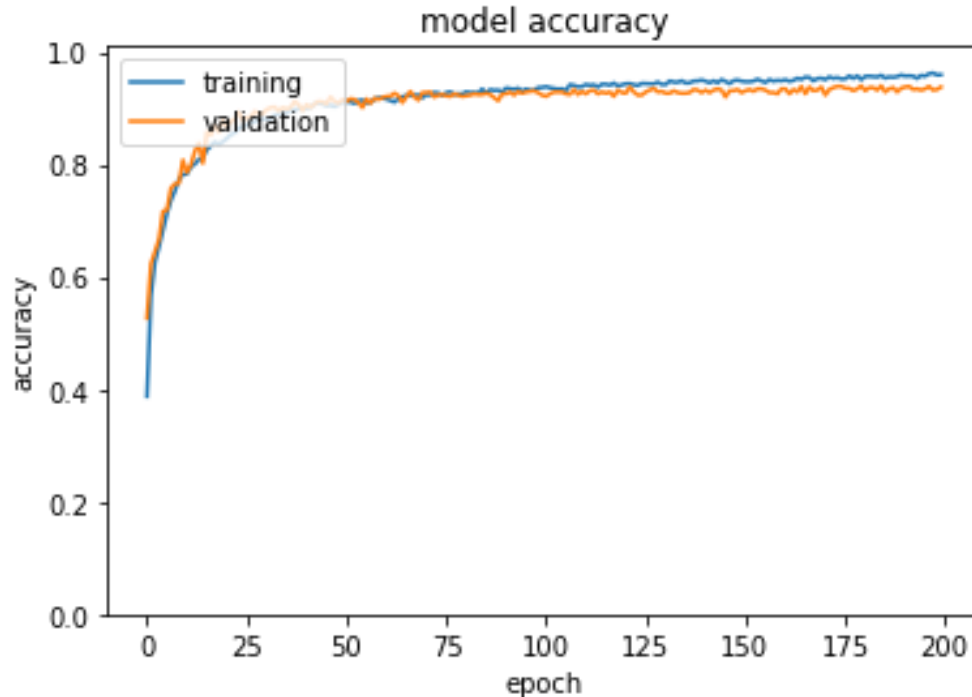
ZWARNING FLAGS

zWarning-Flag	Name	Bedeutung
0	OK	Keine bekannten Probleme
1	LITTLE_COVERAGE	Zu kleine Wellenlängen Abdeckung
2	SMALL_DELTA_CHI2	Chi-Quadrat des besten Fits ist zu nah an dem des zweitbesten Fits
3	NEGATIVE_MODEL	Synthetische Spektrum ist negativ (nur für stars und QSO)
4	MANY_OUTLIERS	Teil der Messpunkte mehr als 5 sigma weg vom besten Modell
5	Z_FITLIMIT	Chi-Quadrat Minimum an der Kante der redshift fitting range
6	NEGATIVE_EMISSION	Eine QSO-Linie exponiert negative Strahlung
7	UNPLUGGED	Die Detektor-Faser war während des Messvorgangs nicht eingesteckt

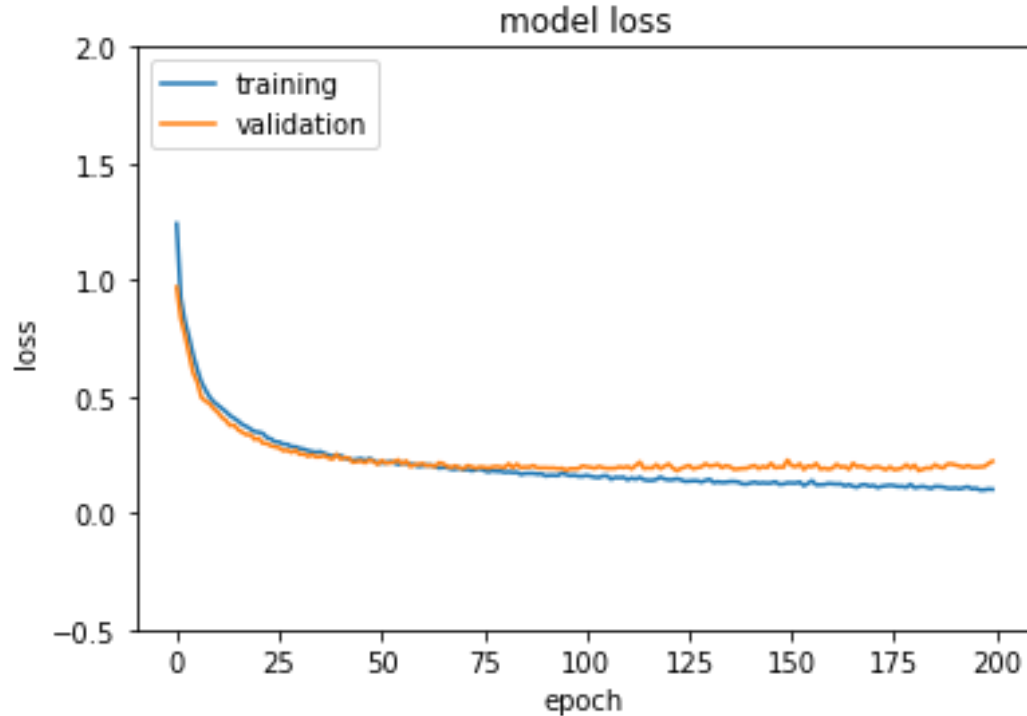
GOLDENER DATENSATZ

- Fast die Hälfte der falsch Klassifizierten Daten hatten keinen zWarning-Flag = 0
- Neuer goldener Datensatz mit ausschließlich Spektren mit zWarning-Flag = 0 in SQL-Suche
- Datensatz mit 10.000 Spektren
- Verbessert Training deutlich!

TREFFERGENAUIGKEIT MIT GOLDENEM DATENSATZ



LOSS-FUNKTION MIT GOLDENEM DATENSATZ





Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Machine Learning

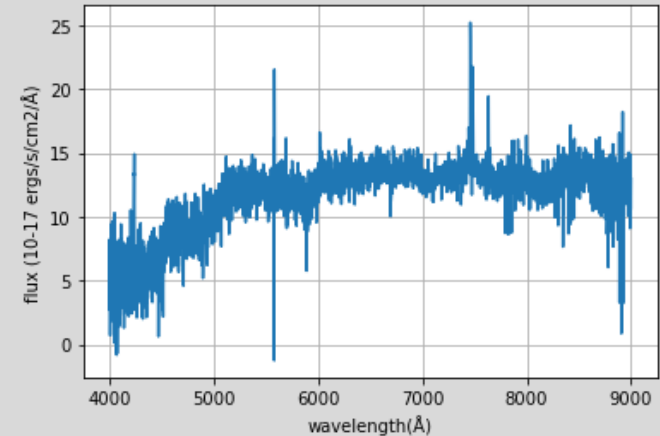
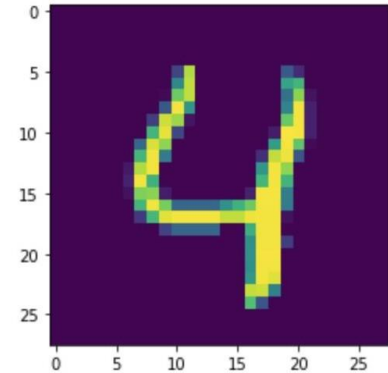


2 Getting to know the HPC Cluster

Dokumentation, Code und Folien:

[github.com/joshuaroschlaub/
Keras_Einfuehrung](https://github.com/joshuaroschlaub/Keras_Einfuehrung)

1 Keras basics



3 Classification of SDSS Spectra

