Quasar Spectra Analysis

Chengliang Su
Guozhen Ma
Miao Shang
Sachin Wong

14/06/2023
Applied Machine Learning 2023

Outline

- Tensorflow regressor to predict the quasar redshifts
- Gated Recurrent Unit (GRU) to predict redshifts
- CNN to classify whether DLAs (Damped Lyman-alpha absorber) appear
- CNN to locate where DLAs are

Background

Quasar, also known as QSO (quasi-stellar object)

A kind of extremely luminous active galactic nucleus (AGN)

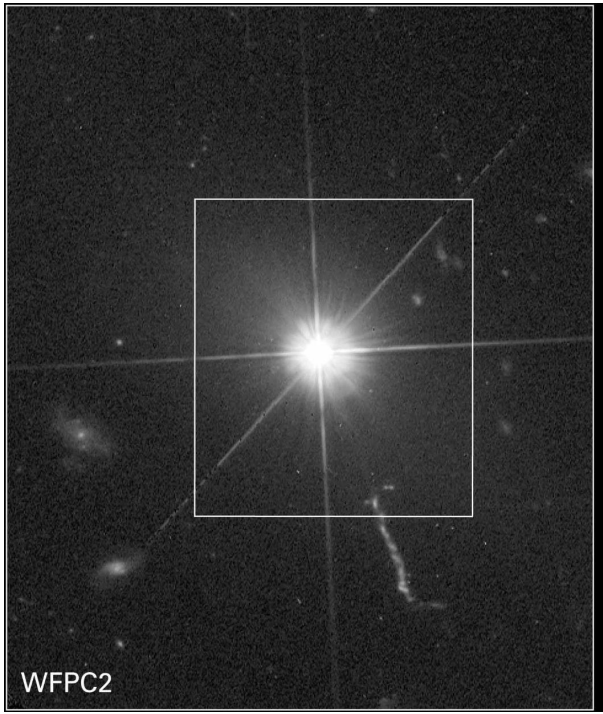
- Supermassive black hole

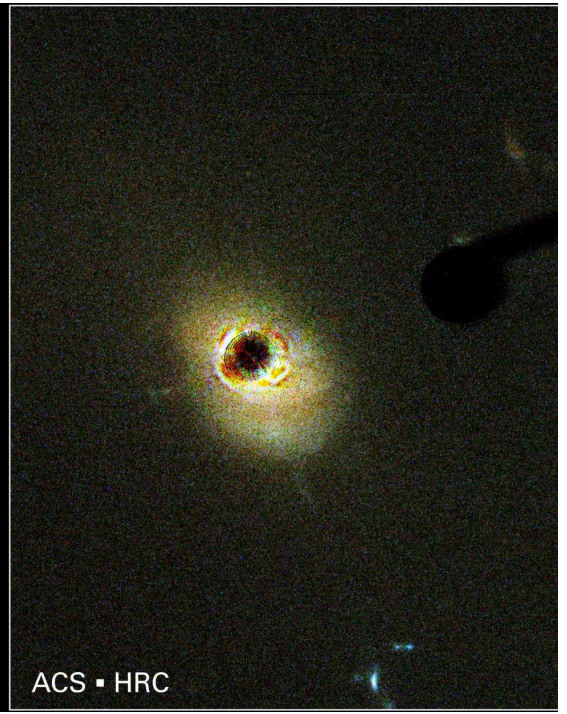
The nearest known one is about 600 million light-years away from Earth.

- Markarian 231 (UGC 8058)

Source: Wikipedia

Background - imaging

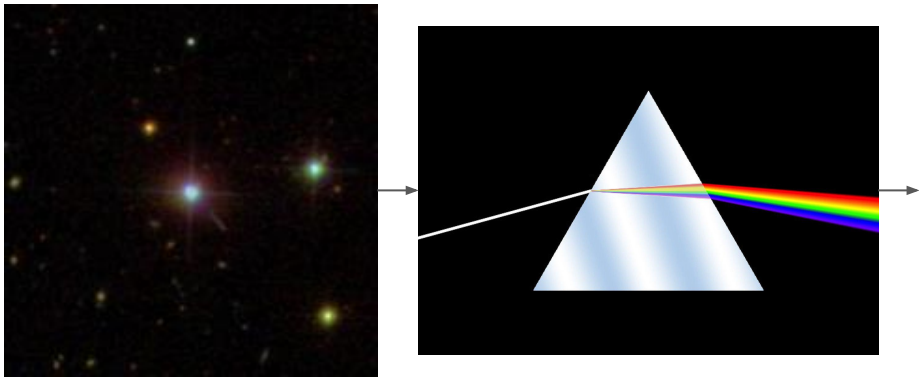




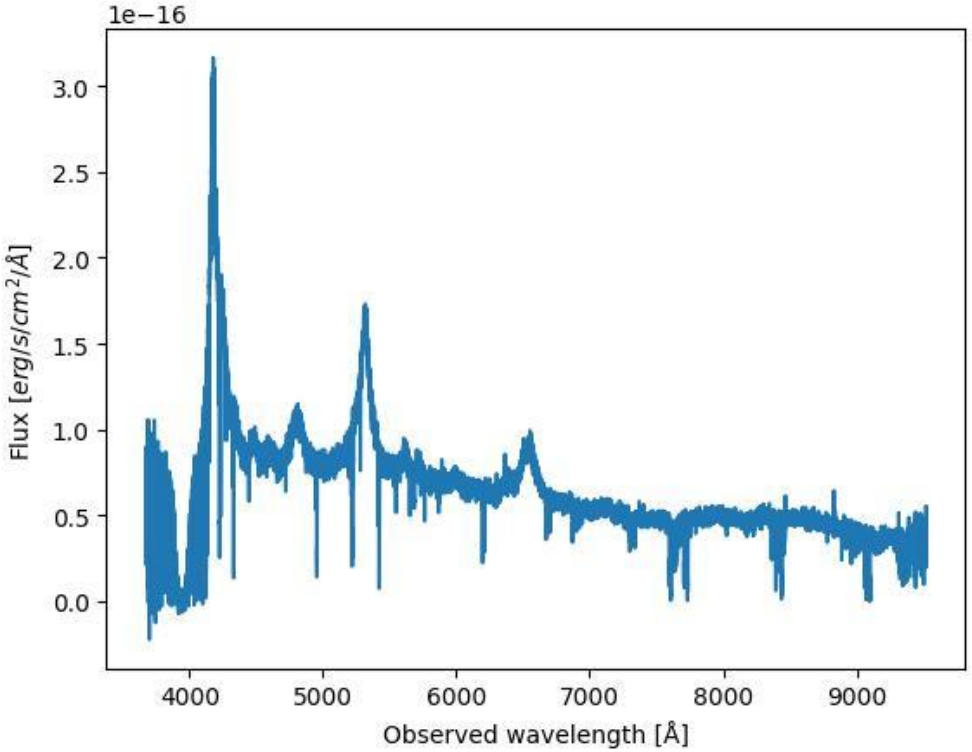
Source: Wikipedia

3C 273

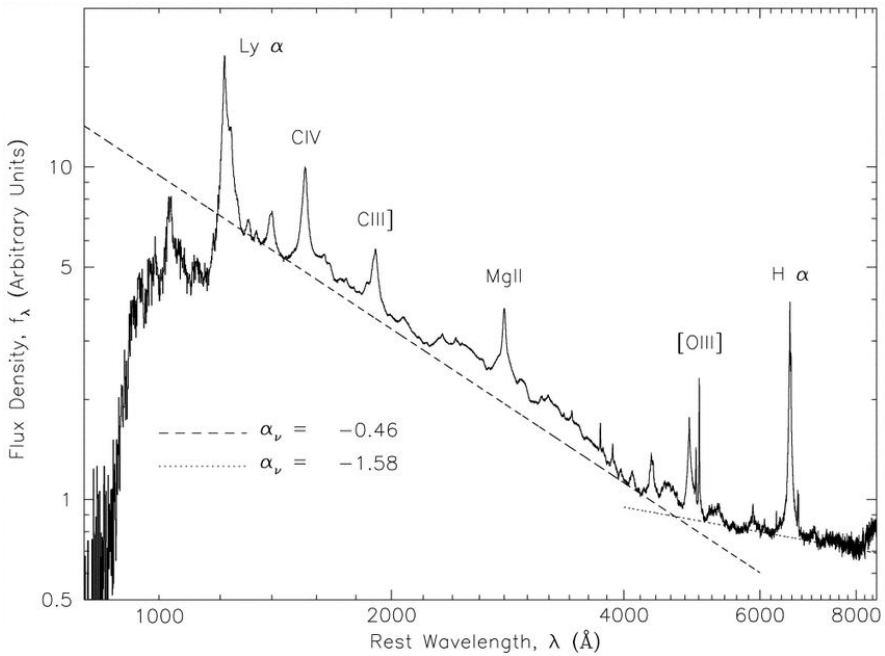
Background - spectroscopy

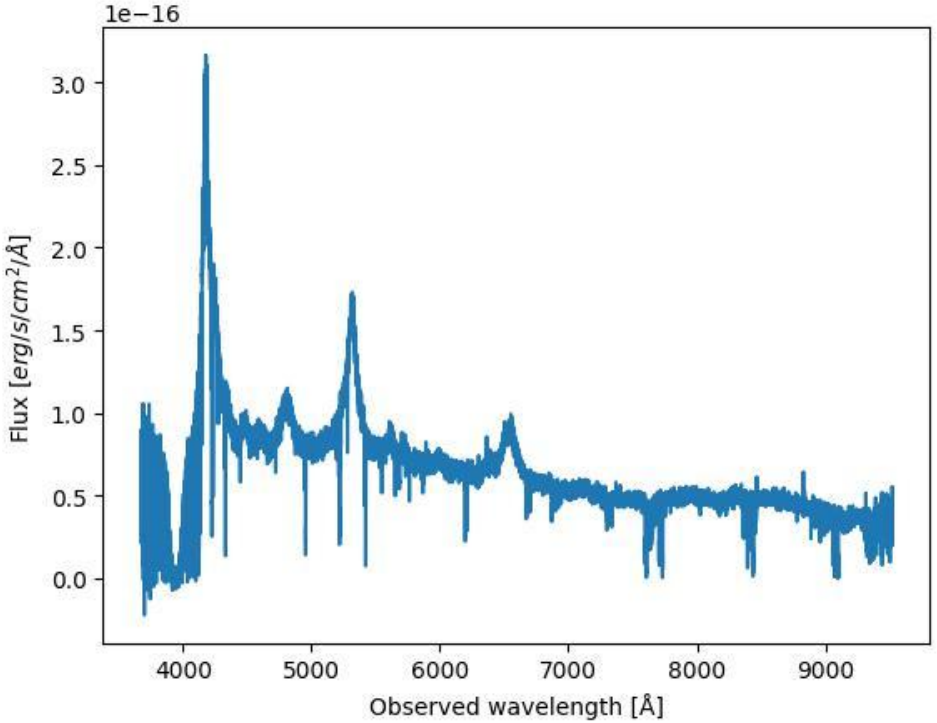


Source:
https://www.thoughtco.com/introduction-to-spectroscopy-603741



Background - redshift (z)

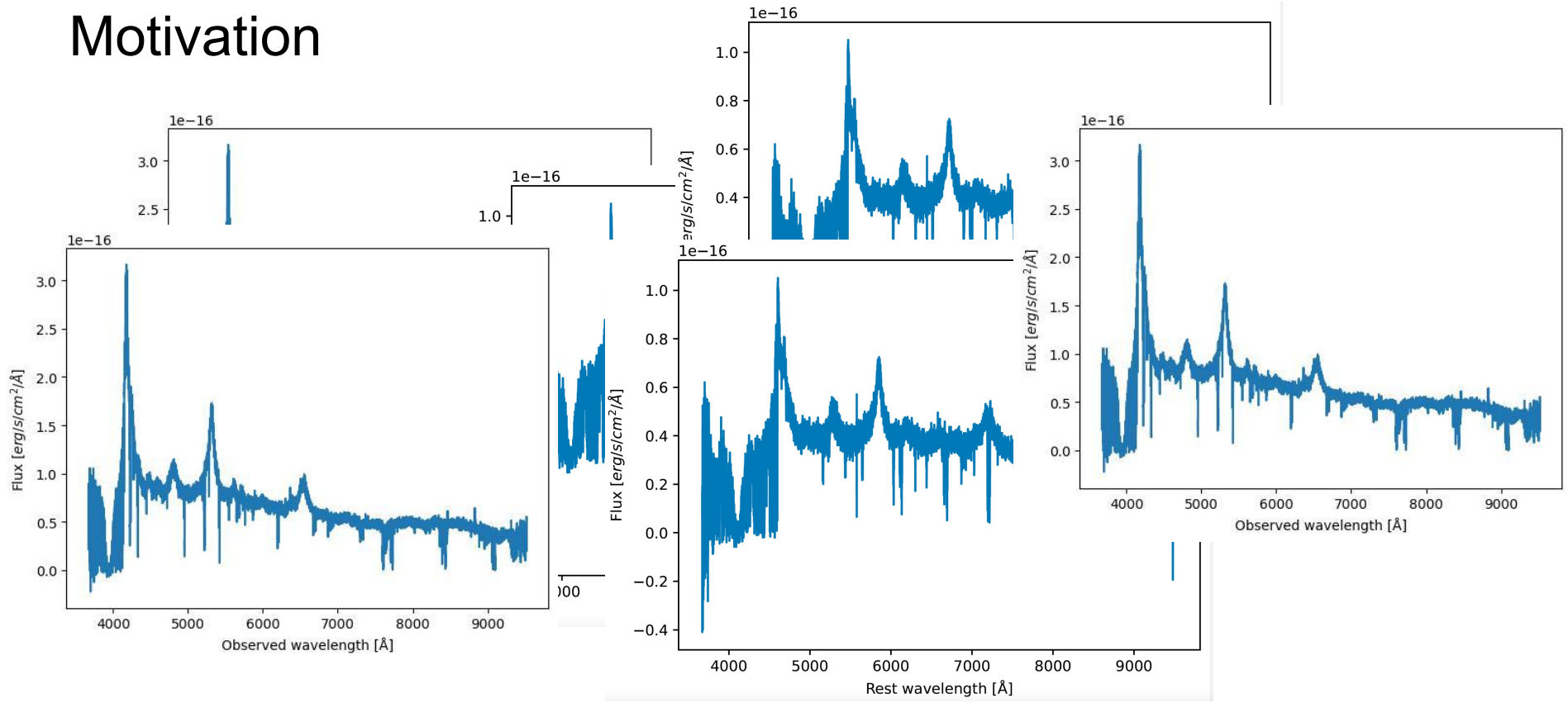




z = λ_obsv / λ_rest - 1

Source: Vanden Berk et al. (2001).

Motivation



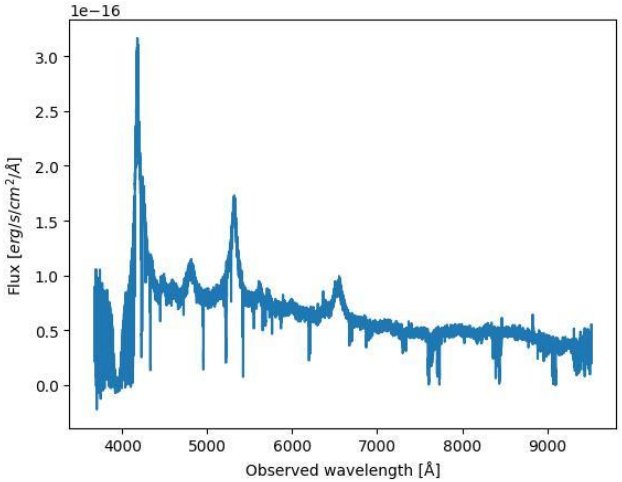
4MOST - (4G–PAQS) - The survey will observe nearly 250,000 quasar candidates

Randomly simulated data

Z_range ∈ [2,4], Mag_range ∈ [18, 20.5]

Data size: 10000 spectra (~ 20 hrs for generation)

Data shape: (10000, 23411, 1) - (N_spec, N_wav_bins, input_features - Flux)



Wavelength range: 3600 ~ 9600 Å

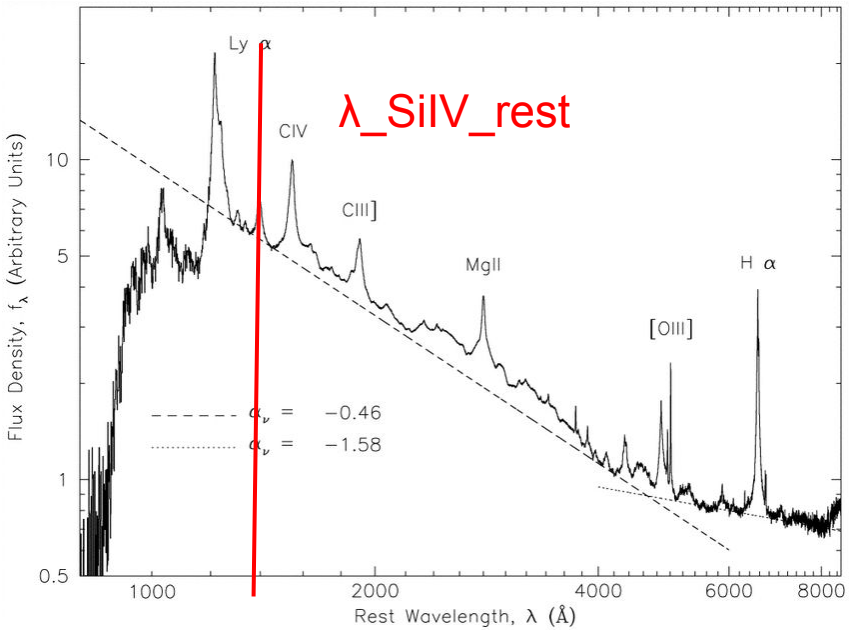
Labels/Truth

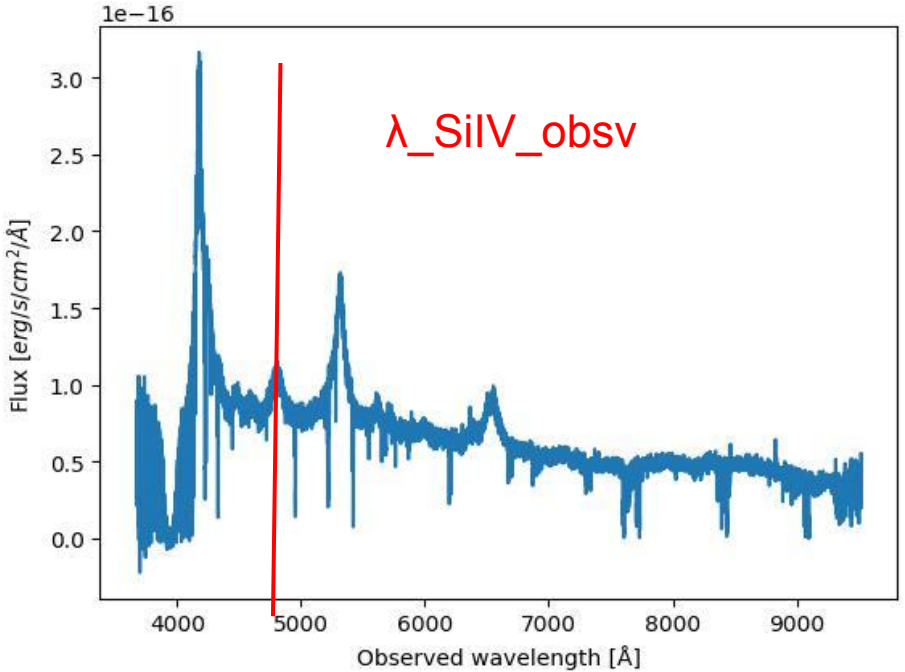
list_templates_10000

FILE	Z_QSO	N_ABS	N_DLA
output/abs_templates/PAQS_absorber_000001.fits	2.2085301947956900	2	0
output/abs_templates/PAQS_absorber_000002.fits	3.486421113996830	14	1
output/abs_templates/PAQS_absorber_000003.fits	2.4346230456855200	5	1
output/abs_templates/PAQS_absorber_000004.fits	3.8531780612472500	12	1
output/abs_templates/PAQS_absorber_000005.fits	3.6273264733007900	8	1
output/abs_templates/PAQS_absorber_000006.fits	3.499137413692190	14	1
output/abs_templates/PAQS_absorber_000007.fits	3.00276328984965	6	0
output/abs_templates/PAQS_absorber_000008.fits	3.407262869730870	5	1
output/abs_templates/PAQS_absorber_000009.fits	3.0702350849661300	8	0
output/abs_templates/PAQS_absorber_000010.fits	2.132028557337740	5	0
output/abs_templates/PAQS_absorber_000011.fits	2.159184245191250	1	0
output/abs_templates/PAQS_absorber_000012.fits	3.8764656374040300	10	1

Finding the redshift(z) - TensorFlow regressor

The simplest idea - calculate redshift by a certain peak

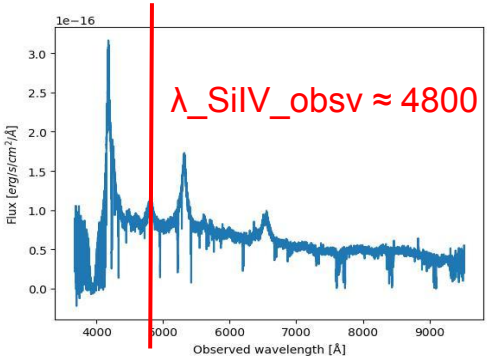


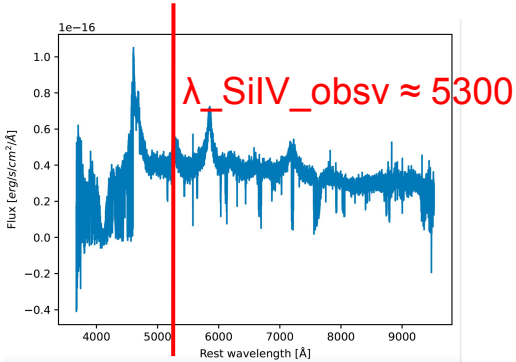


z = λ_SiIV_obsv / λ_SiIV_rest -1

Source: Vanden Berk et al. (2001).

The part that is not simple - how do we know which peak is SiIV?





We human - compare neighboring peaks by eyes…

Or…let Regressor figure out its own way!

Model structure:

Learning rate determined by Adam optimizer

n_epochs = 8 batch_size = 50 n_samples=10000

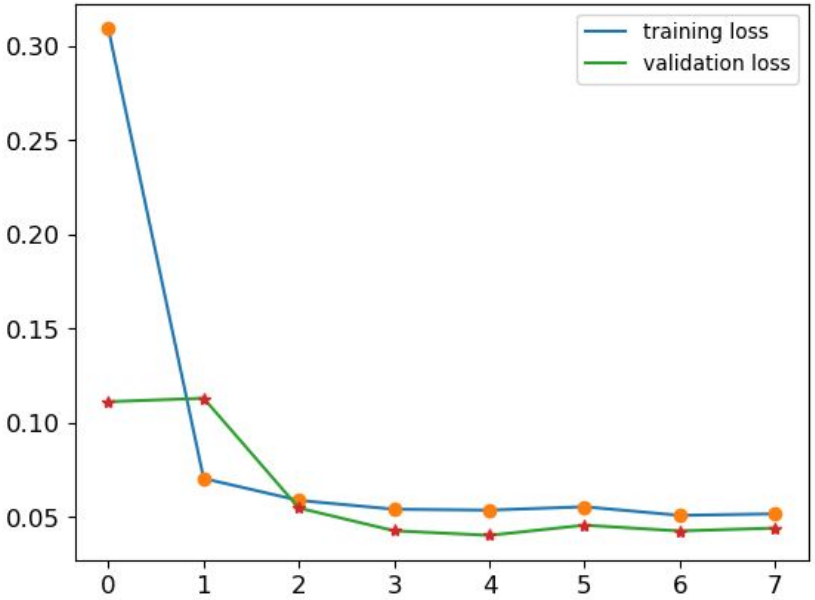
Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 23411)	548098332
dense_6 (Dense)	(None, 10)	234120
batch_normalization_3 (BatchNormalization)	(None, 10)	40
dense_7 (Dense)	(None, 10)	110
batch_normalization_4 (BatchNormalization)	(None, 10)	40
dense_8 (Dense)	(None, 10)	110
batch_normalization_5 (BatchNormalization)	(None, 10)	40
dense_9 (Dense)	(None, 1)	11

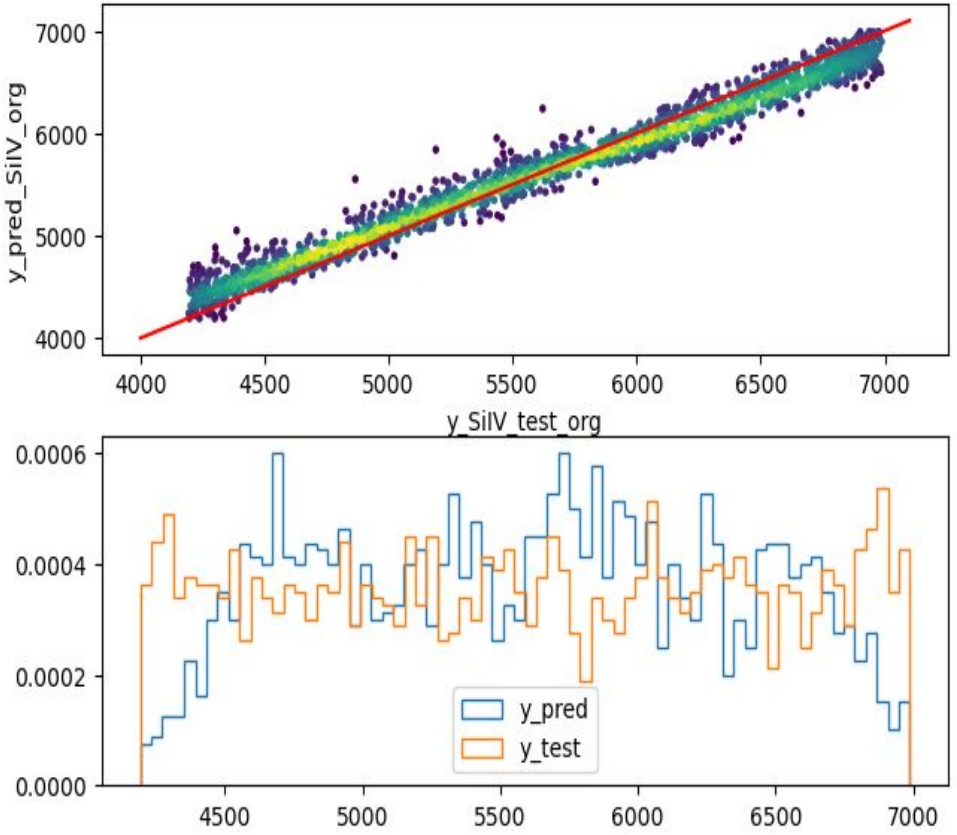
Total params: 548,332,803
Trainable params: 548,332,743
Non-trainable params: 60

Results & Evaluation

Loss history* (mean absolute error)



Time used: 6775.952 s (failed to use GPU)



* Data is normalized by qt.transform

Results & Evaluation

It is a very simple ML model, but shows not bad accuracy

What are bad?

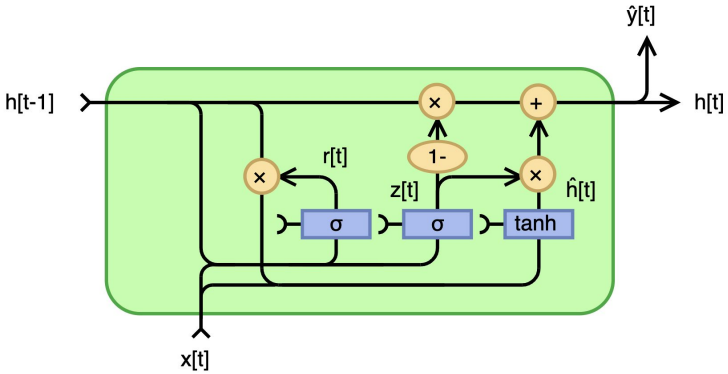
Loss remains unchanged in the last several epochs

It tends to predict higher values in small redshift, and lower values in large redshift

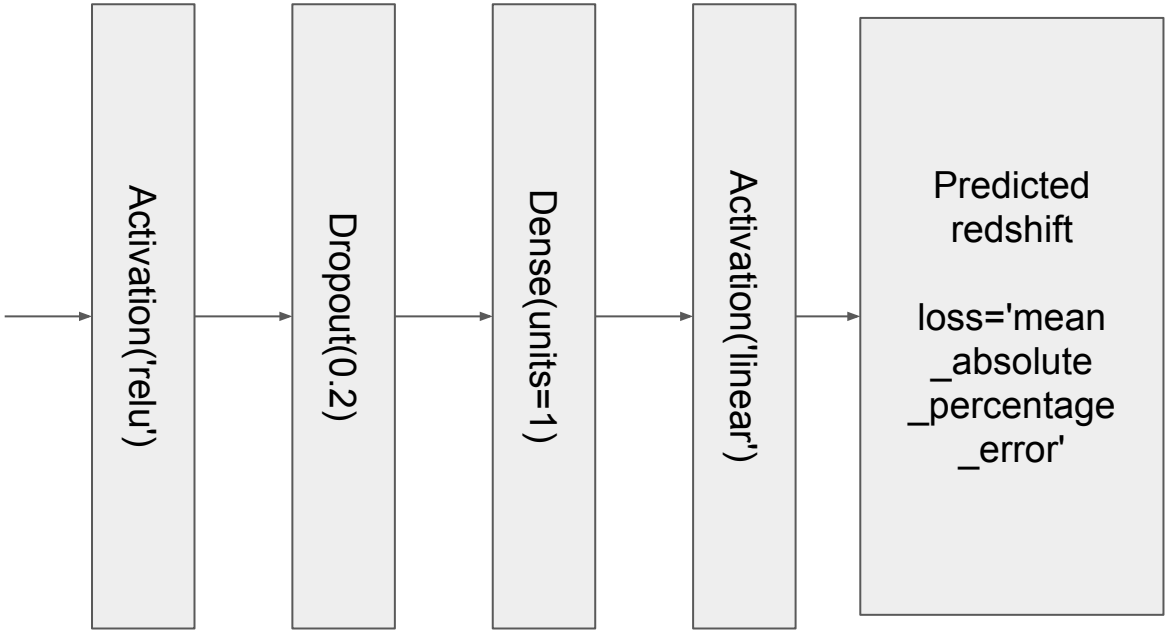
So it performs badly near the end points of the redshift range

Too long training time (~2 hours)

RNN - Gated Recurrent Unit (GRU) - Tensorflow

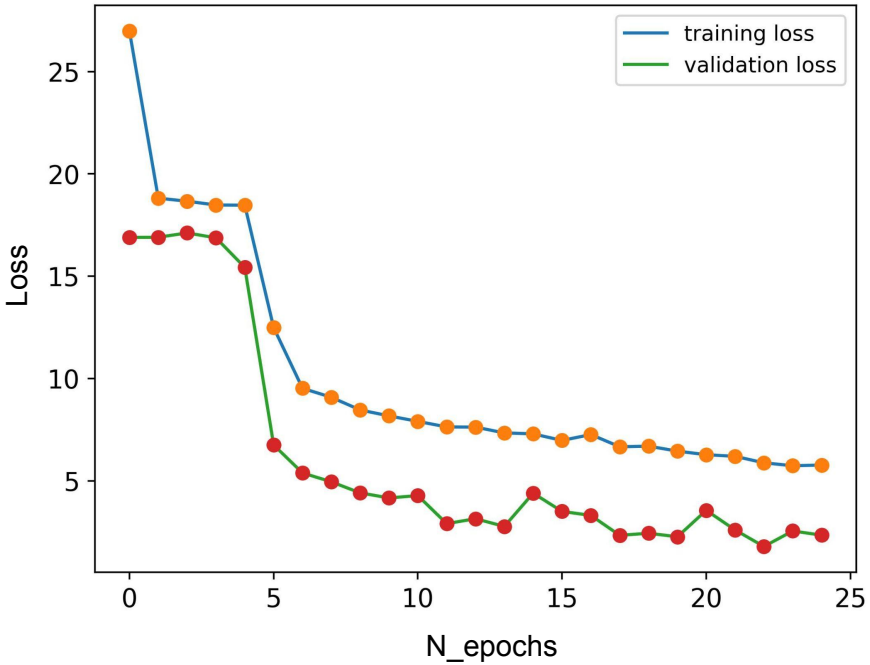


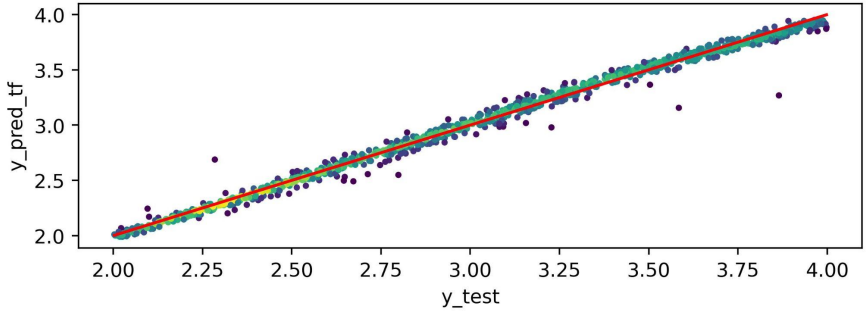
units=100, input_shape=(23411, 1)

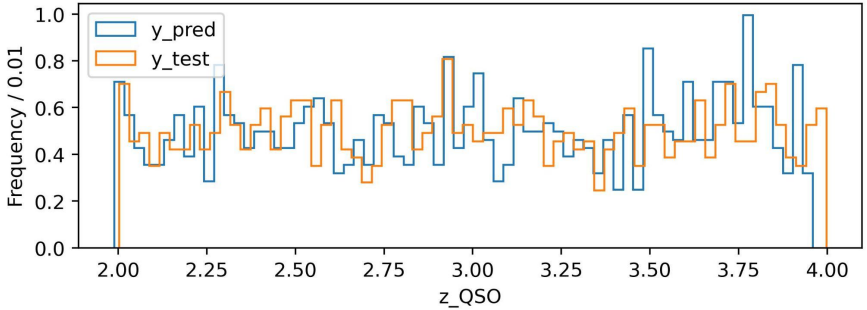


batch_size = 16,
epochs = 25, (+25+7)
with GPU acceleration thanks to Colab

RNN - Gated Recurrent Unit (GRU) - Tensorflow

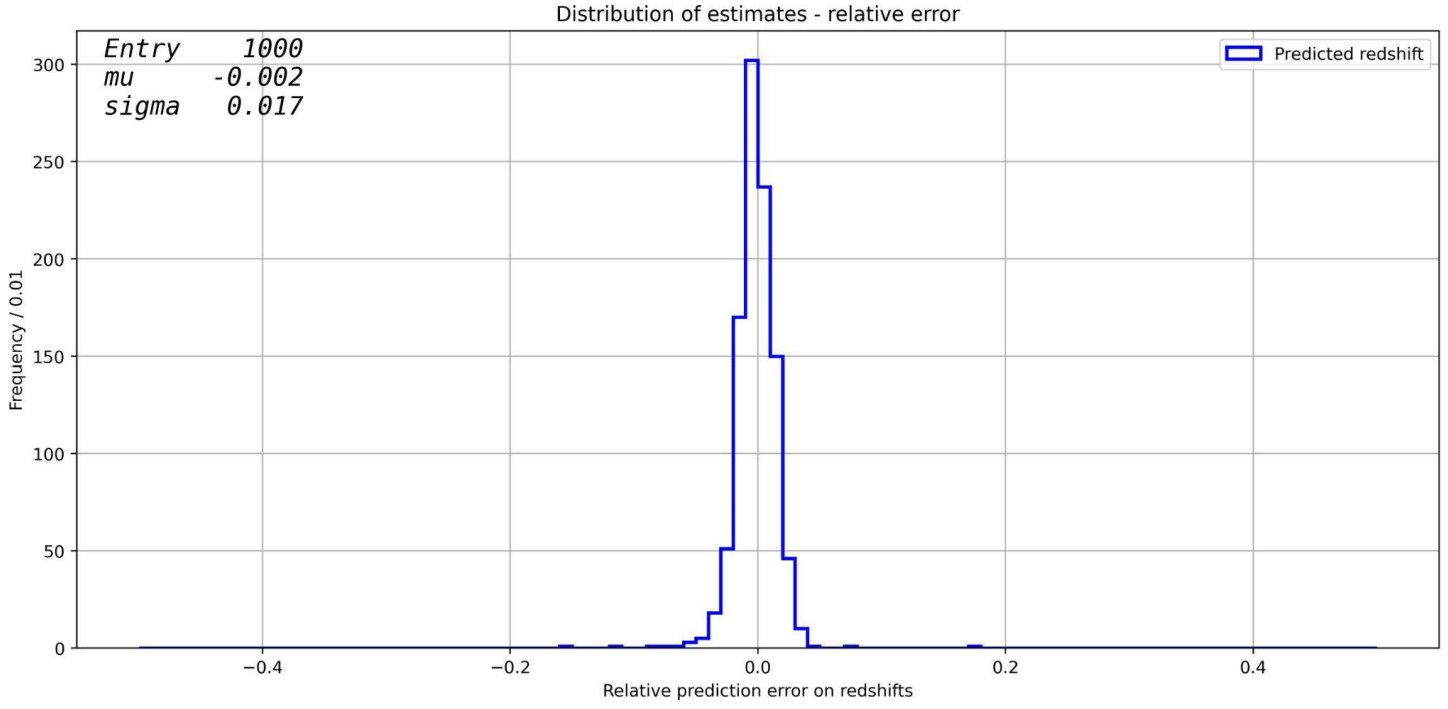






Test loss: 1.14%;
Time used by TensorFlow: 7547.7 s

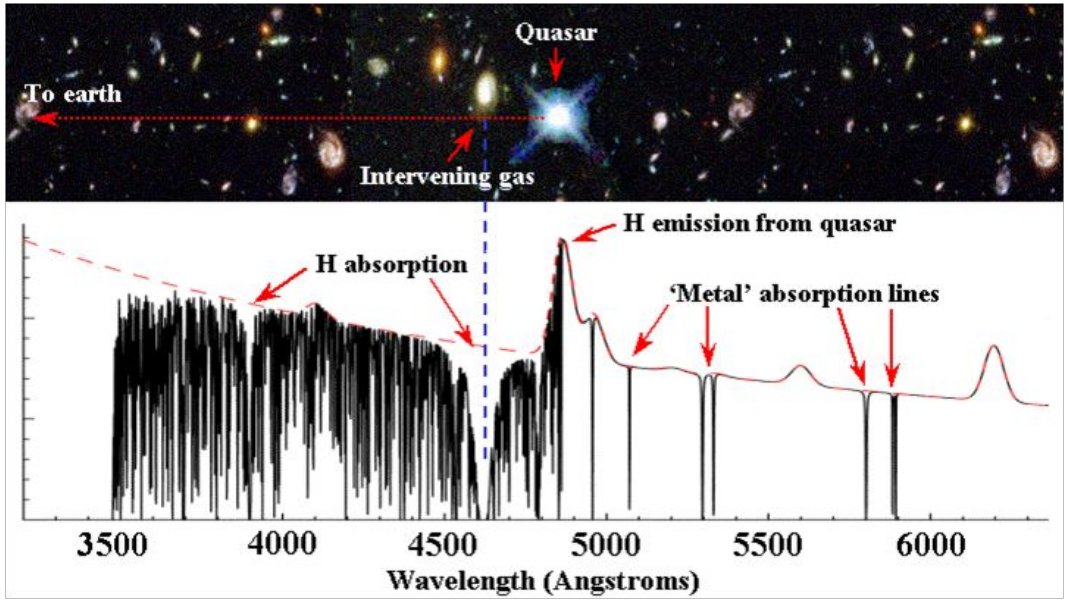
RNN - Gated Recurrent Unit (GRU) - Tensorflow



Test loss: 1.14%;
Time used by TensorFlow: 7547.7 s

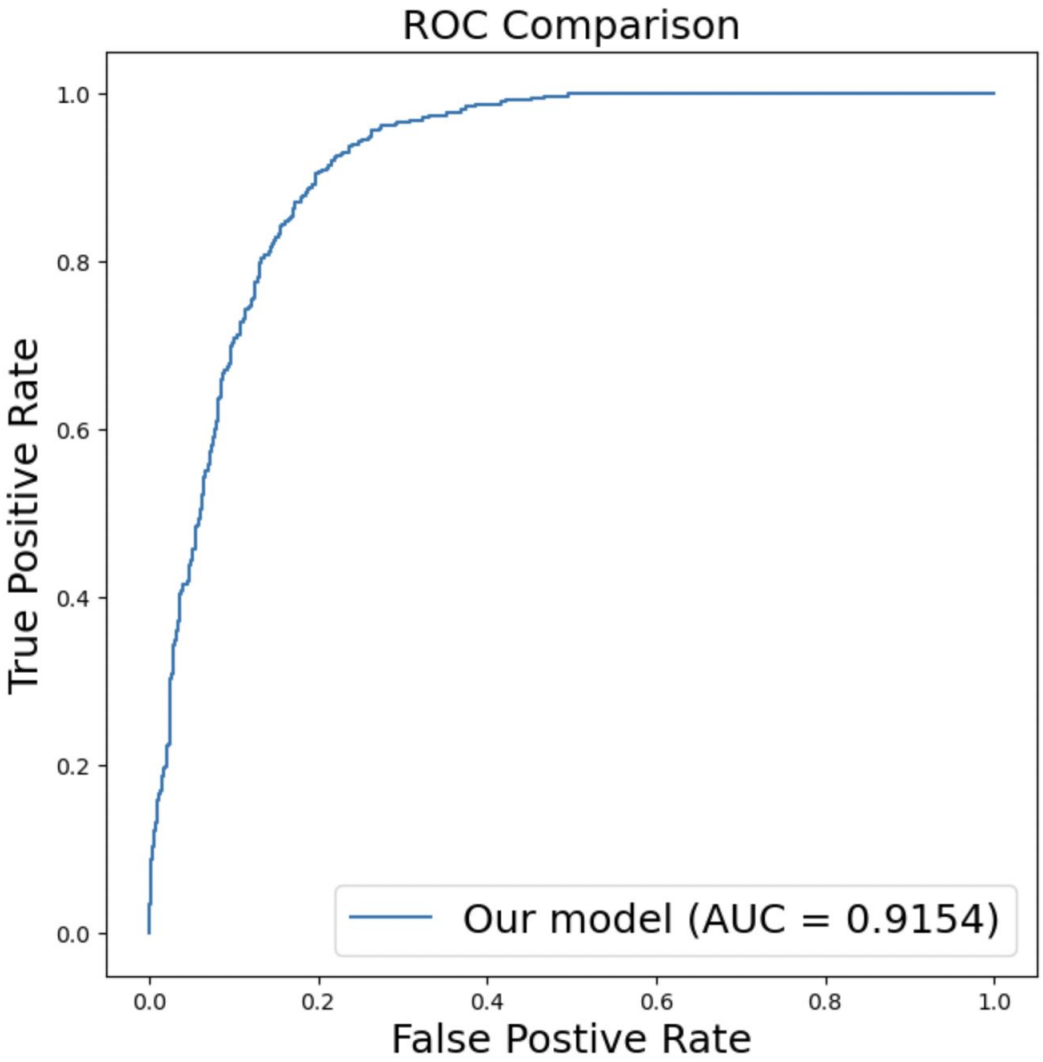
CNN - Classification - Tensorflow

- What is DLA(Damped Lyman-alpha Absorbers)?
- Why DLA?
- Why CNN?
- My work

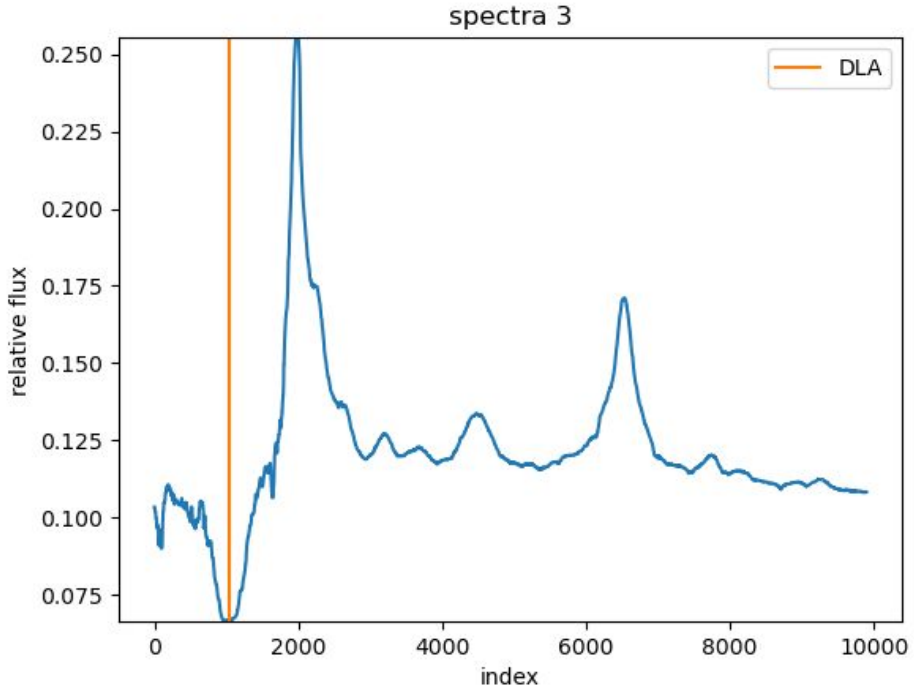


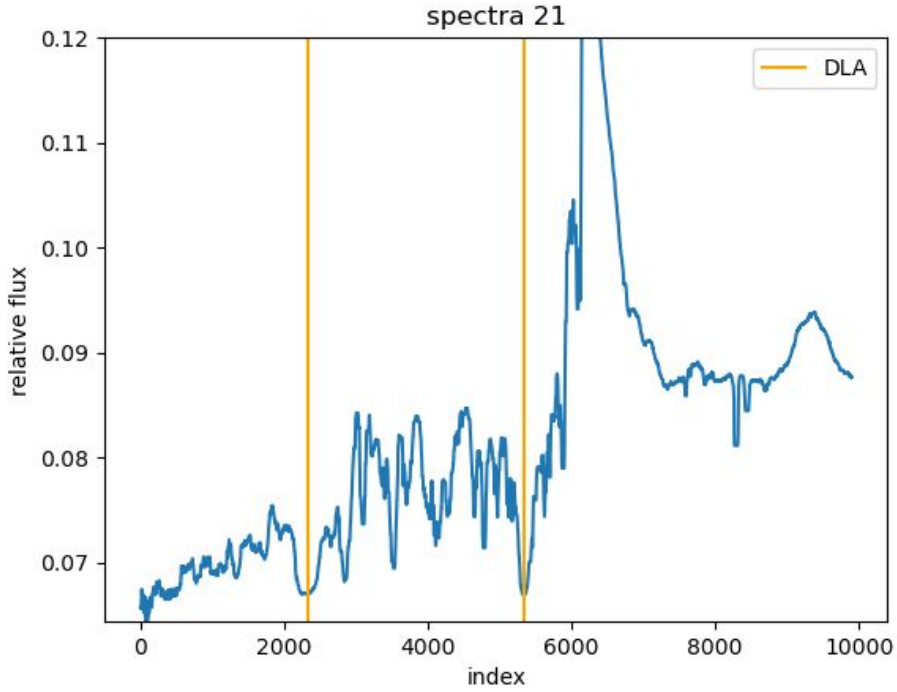
Source: http://www.nat.vu.nl/~wimu/FundConst-Notes.html

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 13598, 32)         128

 conv1d_1 (Conv1D)           (None, 13596, 64)         6208

 batch_normalization (BatchN  (None, 13596, 64)        256
 ormalization)

 max_pooling1d (MaxPooling1D  (None, 13596, 64)        0
 )

 conv1d_2 (Conv1D)           (None, 13592, 64)         20544

 conv1d_3 (Conv1D)           (None, 13588, 128)        41088

 global_average_pooling1d (G  (None, 128)              0
 lobalAveragePooling1D)

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 84,865
Trainable params: 84,737
Non-trainable params: 128
_________________________________________________________________
```

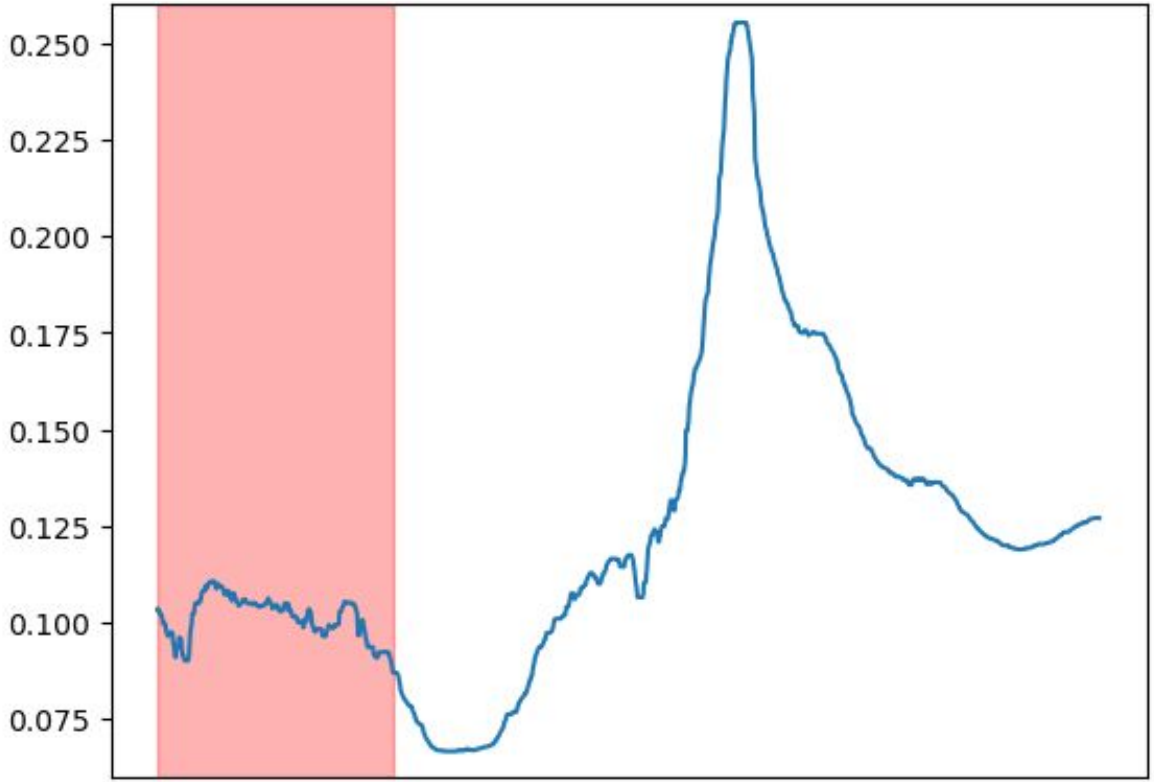

ROC Comparison
True Positive Rate
False Postive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Our model (AUC = 0.9154)

CNN - Locating Damped Lyman Alpha Absorptions

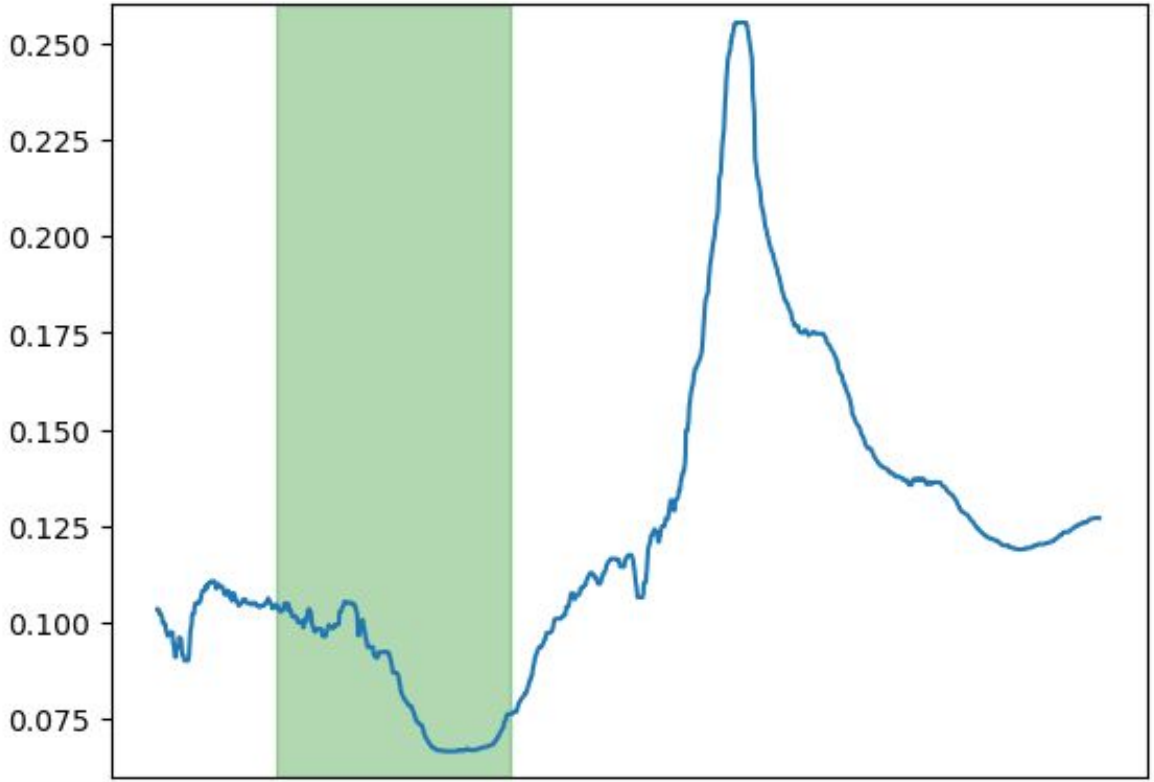




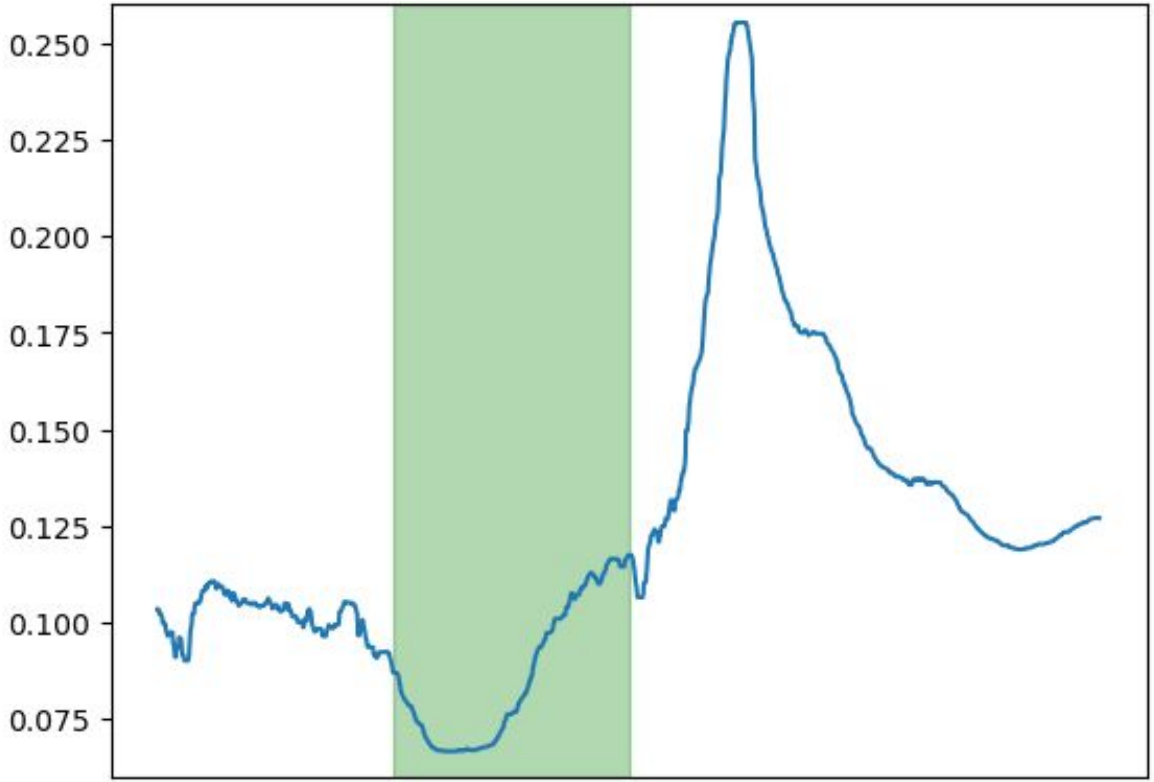
CNN - Preparing Data + Labels



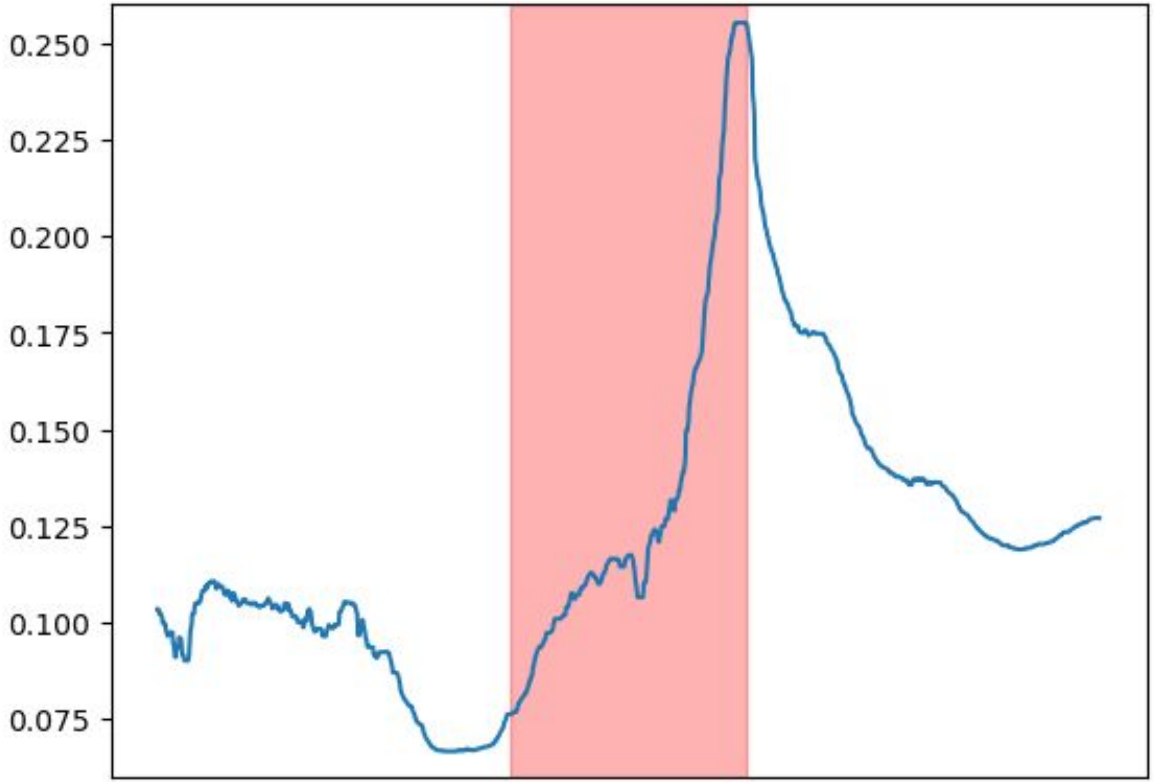
CNN - Preparing Data + Labels



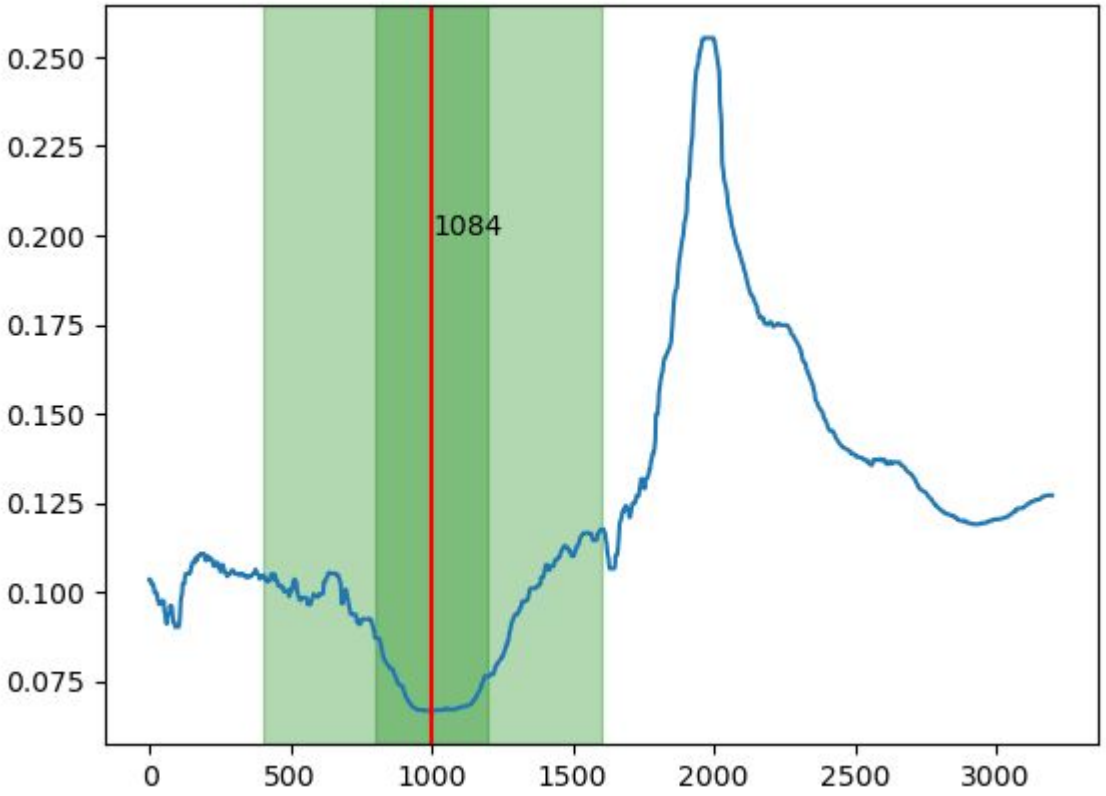
CNN - Preparing Data + Labels



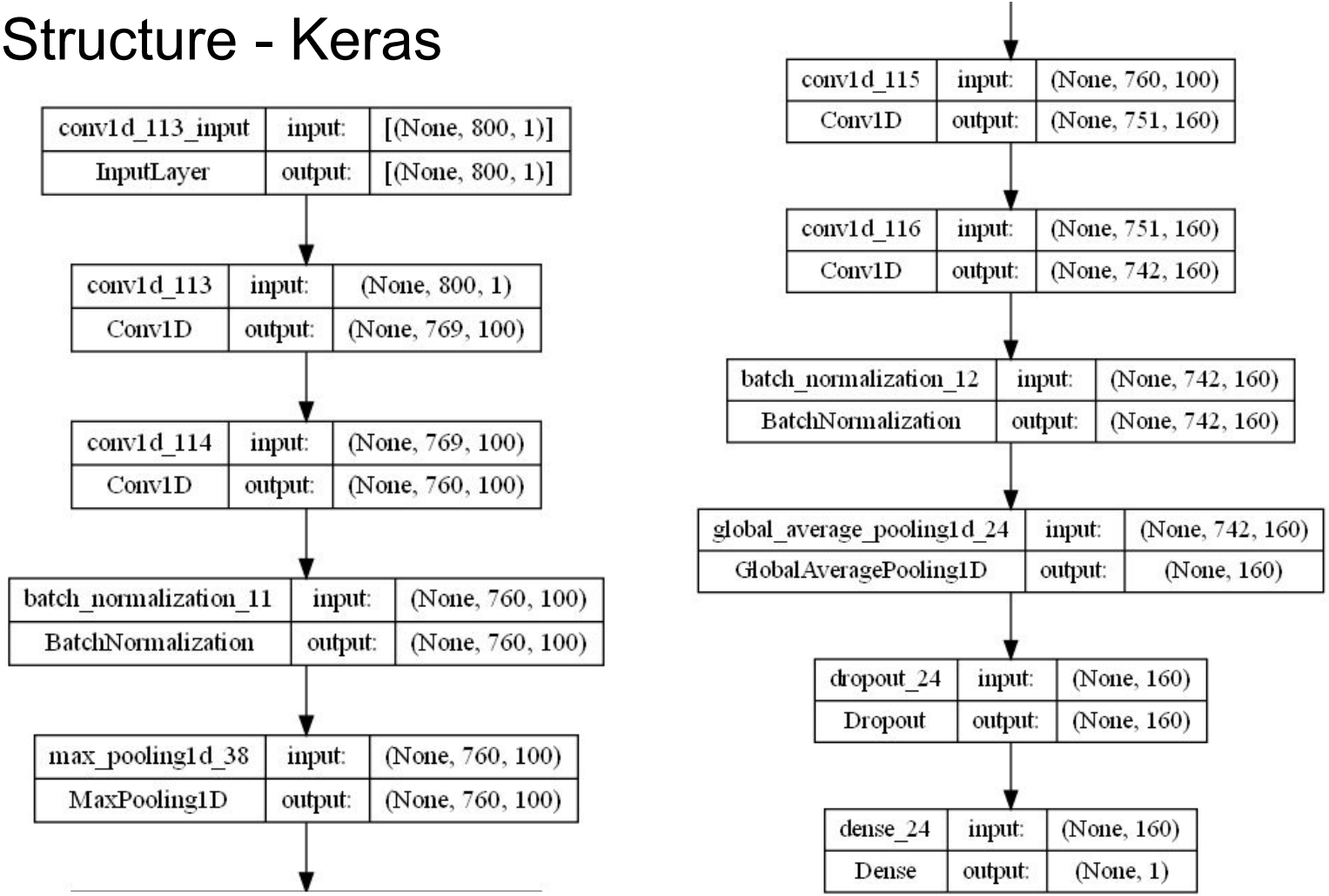
CNN - Preparing Data + Labels



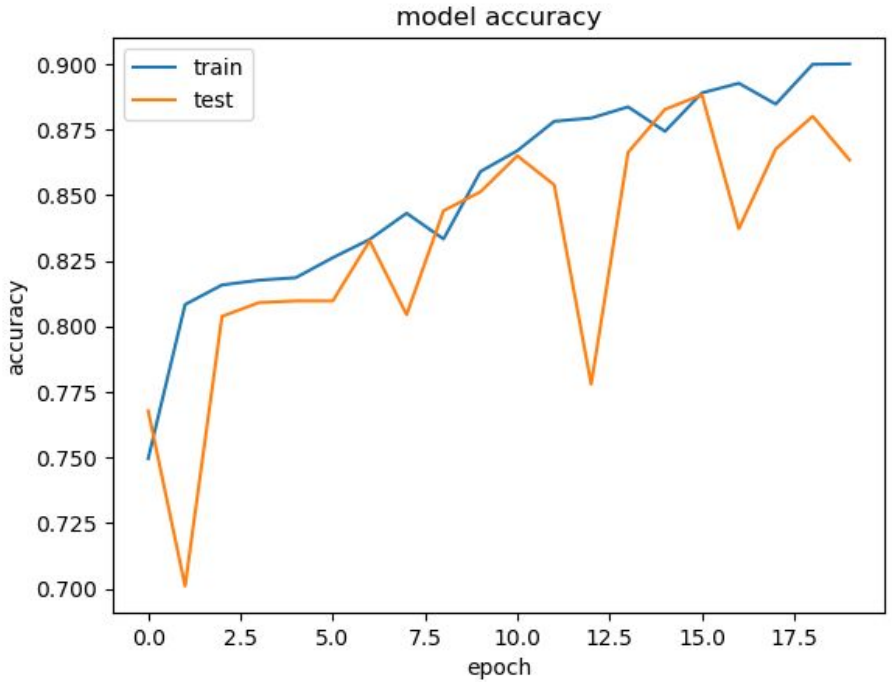
CNN - Locating Damped Lyman Alpha Absorptions

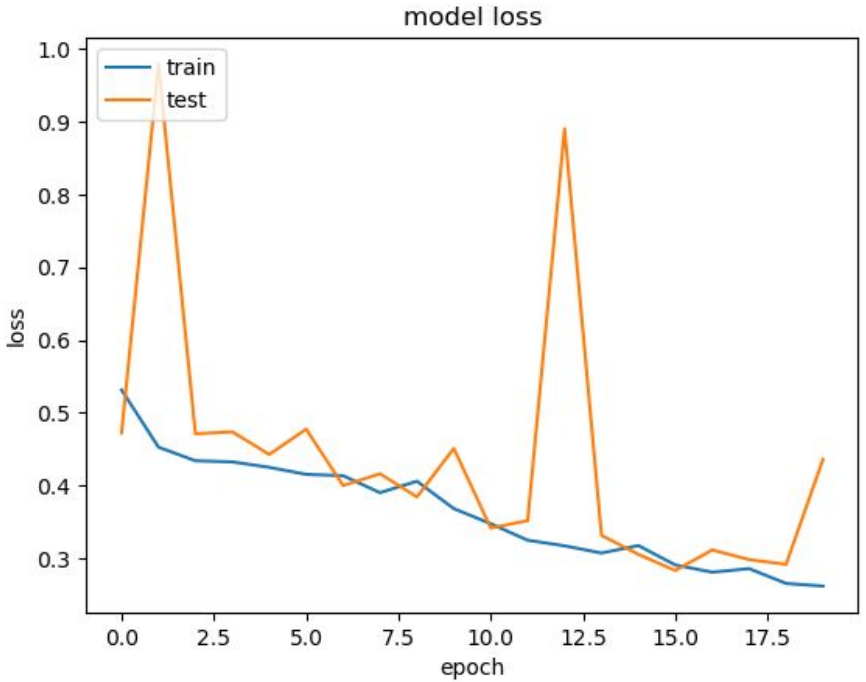


CNN Structure - Keras



Results


model accuracy
train
test
accuracy
epoch
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
loss
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Conclusion

- Performance of our models
  - Tensorflow regressor: test loss = 0.0441
  - GRU: test loss = 1.14%
  - CNN for classifying DLAs: AUC = 0.9154
  - CNN for locating DLAs: accuracy ~ 90%
- Did we reach our objective?
- What could we improve?
  - Try to implement GPU acceleration on our own machine or get access to larger clusters.
  - More training data.

Thank you :)

Any questions?

Appendix

Appendix - TensorFlow Regressor

Features

Our samples are spectra, and the features are the flux of each spectrum. In each spectrum, the wavelength ranges from 3671.5 to 9524 and a wavelength bin is 0.25 wide, i.e. there are 23411 wavelength bins in each spectrum. So each sample has 23411 features (a very large parameter space).

Data preprocessing

Before qt.transform and training, I preprocess the truth data (the location of SiIV peak) by

$$y\_truth = (\lambda\_SiIV - 3671.5) / 0.25$$

The idea is that, we only input the flux data so the machine know nothing about the wavelength. Then it might be hard for it to learn how a wavelength value like 9524 jumps out. But the machine does know the label of flux, in other words, the label of wavelength bins. Hence, the above transformation convert 9524 to (9524 - 3671.5) / 0.25 = 23410 = the label of the last flux, something the machine should know.

Optimization

Due to the long training time I only try to optimize the batch size. It turns out that the smaller batch size gives better results.

Appendix - GRU

- Reasons for choosing to use GRU:
  - Spectrum data is in series and it is very similar to time series data. The spectrum is a flux function with respect to wavelength, and every point is highly linked with the one before and after, so we immediately wanted to try to analyse the spectra with RNN. LSTM is better for analysing time series, so we choose to use GRU since the spectrum is not really time series.
  - GRU does not make use of the cell state and instead uses the hidden state to transfer information. It also only has two gates, a reset gate and update gate. GRUs have fewer tensor operations; therefore, they are a little speedier to train than LSTM.

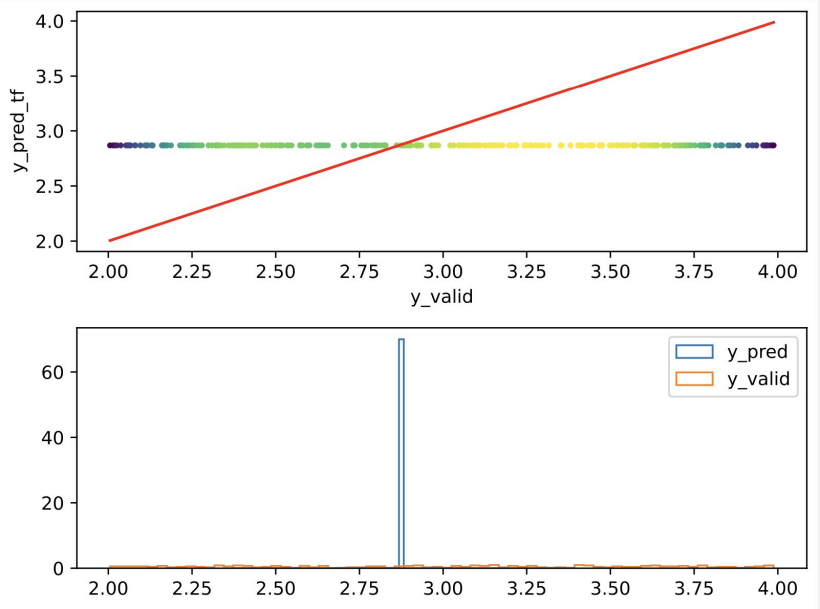
Appendix - GRU

- Hyper parameter optimization:
  - It seems that more units in GRU always help, but it also increase the code running time significantly. By considering both the performance of the code and the running time, we choose to use 100 in the end.
  - We add a drop-out layer after the GRU to avoid over-fitting.
  - A smaller batch-size is better, and it takes longer time, which is different from the common sense. (Batch-size gives the size of samples which is the amount of training samples to consider at a time for updating the network weights) - size = 32 -> Test loss: 3.68%, t: 1670s / size = 64 -> Test loss: 6.66%, t: 936 s.
  - We used a loss function of mean_absolute_percentage_error, since our true values are between 2 and 4 and the mean_absolute_error will be a too small value (0.02 or lower), so we chose the percentage error.

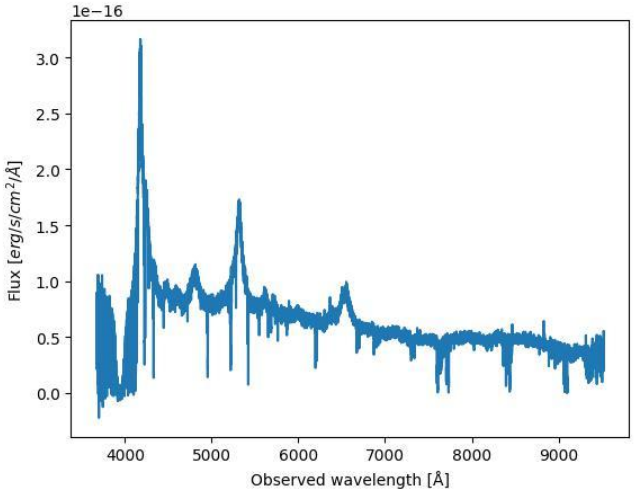
Appendix - GRU

- Preprocessing is very necessary:

Before:

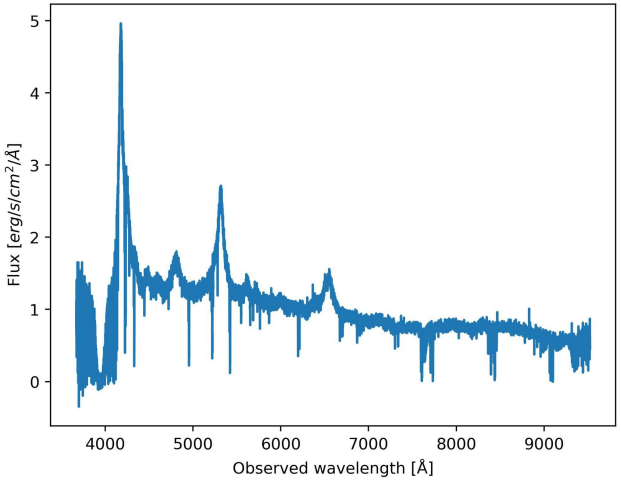


Before:
(~ 1e-16)



Flux/mean(Flux)

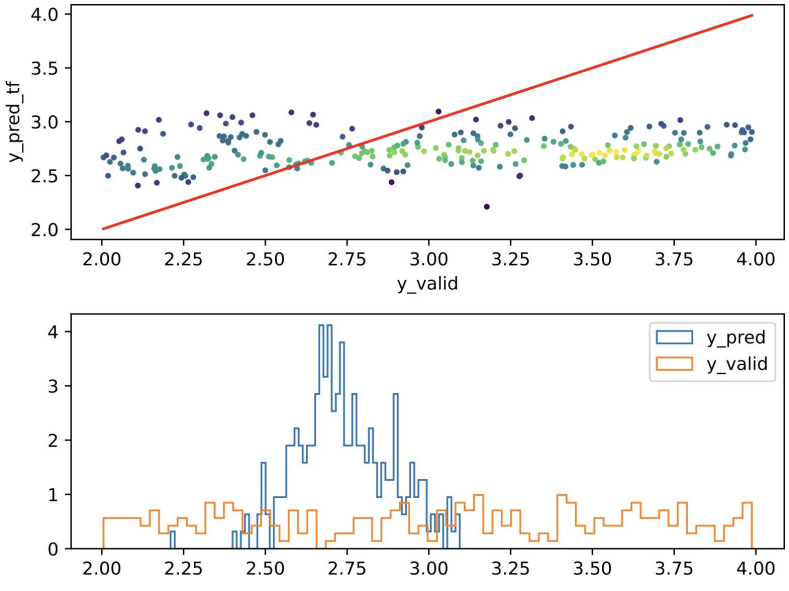
After:
(~ 1)



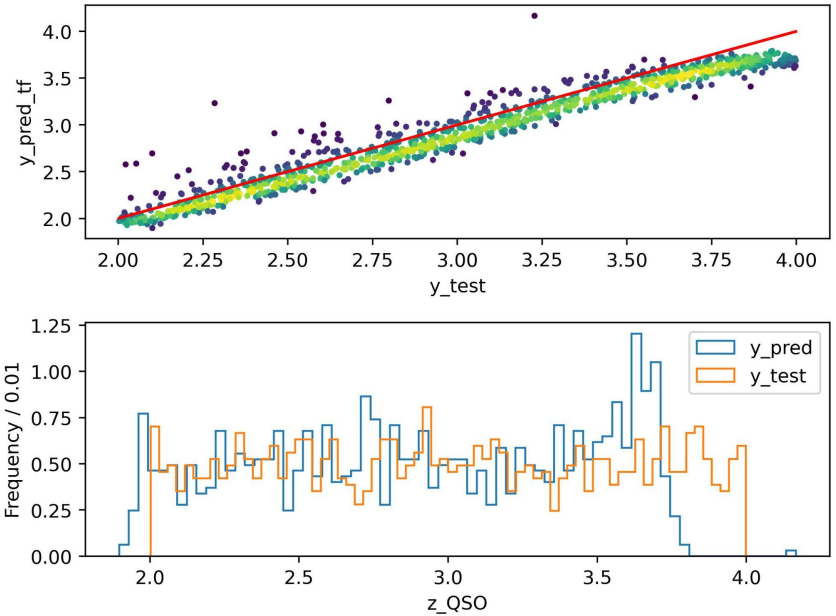
Appendix - GRU

- Can improve significantly with larger sample size:

With using 1000 samples to train:



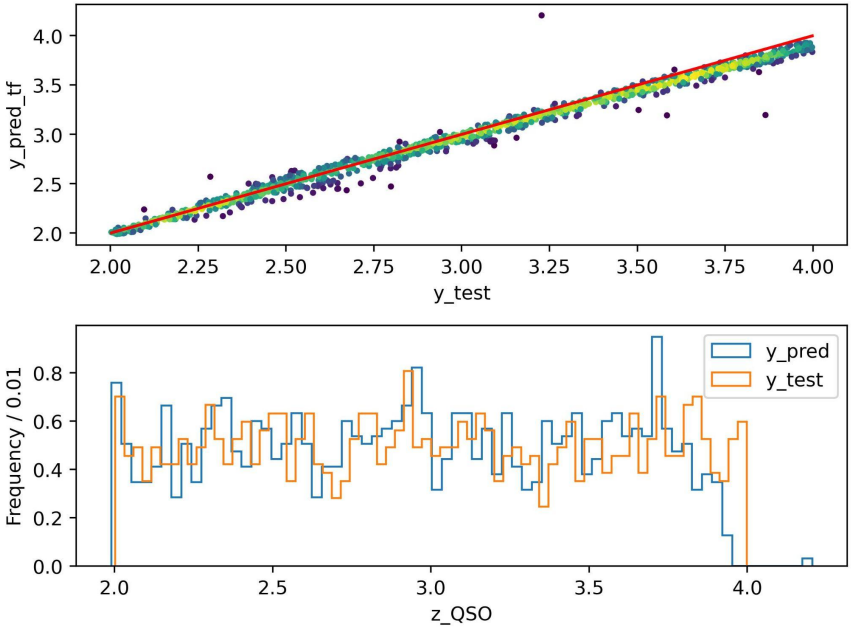
With using 4000 samples to train:



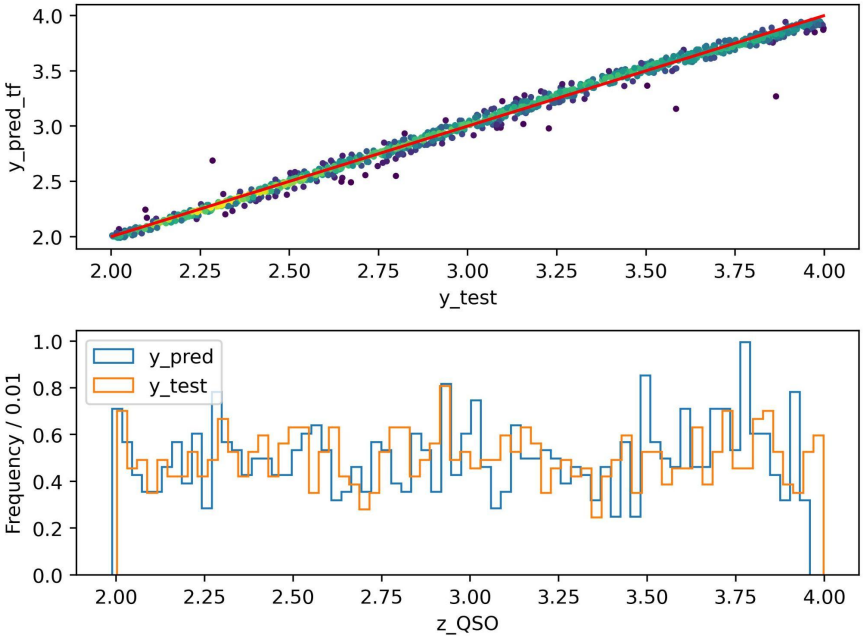
Appendix - GRU

- Can improve significantly with larger sample size:

With using 8000 samples to train:



With using 10000 samples to train:



Appendix - CNN classifying DLAs

Preprocessing the data:

- The flux value can be too small for computer to recognize and instead of extracting the features from the spectra image, we use an one dimensional array, so we have to normalize the data.
- Transform the wavelength into pixels, the dataset size would increase and we can use human knowledge of where DLAs approximately appear to select a sub dataset which will save memory and raise efficiency.

Appendix - CNN Locating DLAs

Reasons for choosing CNN:

- Convolutional NNs used for analysing images and visual features
- Spectra are basically 1D images, and we’re looking for a specific visual feature (DLAs)

Appendix - CNN Locating DLAs

Preprocessing data:

- We know that DLAs only show up between a given range of wavelength, so we could reduce the size of the data down to what was more important (not necessarily too useful for my method but decreased the amount of segments I had to create)
- Normalising the data was very helpful, especially normalising the flux to give the relative flux
- Reduced the effect of noise by using the rolling medium for each spectra (improved the CNN's accuracy)
- For the input data, made sure to balance the number of true and false segments as to avoid the model from just training to output "0" (as number of segments w/o DLA >>> number of segments with DLA)
- Using the segment method gave 10 000 segments w/ DLAs and then we took 10 000 without, for a total of 20 000 in the input data (split into train/val/test etc.)

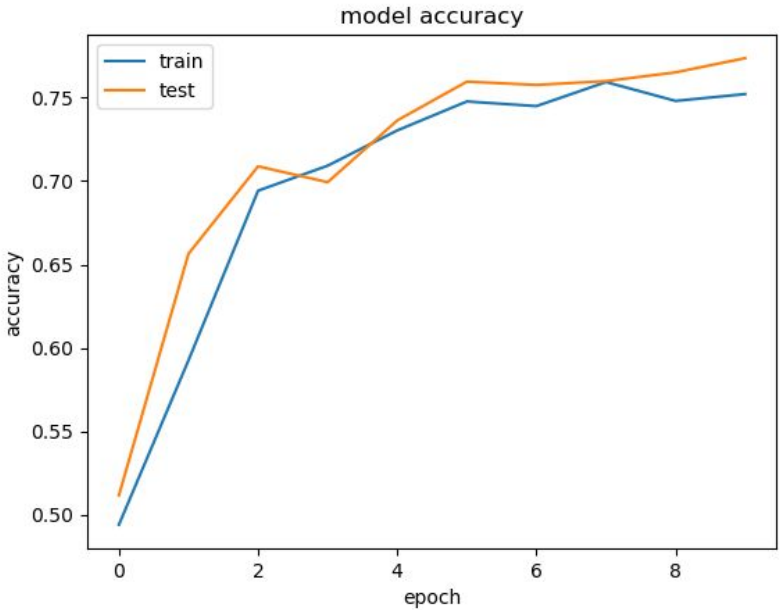
Appendix - CNN Locating DLAs

Hyperparameter Optimisation:

- Used a learning rate scheduler, as I noticed after ~epoch 3 the accuracy and validation accuracy started diverging (overtraining)
- Also implemented early stopping, but I was too lenient on the patience as the model starts to overtrain a little near the end
- Important hyperparameters were the learning rate (of course), the segment size and the overlap between each segment. I think that the labelling process (how many “1” values in a segment to label the entire segment “1”) as well as the range of what I consider the DLA is also an important hyperparameter
- Found that using a larger kernel for the input layer helped
- Played with the size of the Conv1D filters to get a good result without too long a run time or significant overfitting

Appendix - CNN Locating DLAs

Batch Normalisation was very important! This + normalising the data improved the accuracy a lot



Accuracy of previous model where these were not applied. Accuracy improved by ~10% in the final model, and loss improved significantly also

Appendix - CNN Locating DLAs

Further improvements:

- Could have used regression to find the location of the DLA inside the detected segment (e.g. +/- the width of the segment) = more accurate, rather than just looking at the range of the DLA segments and estimating an average - this would reduce the error in the final value of redshift calculated
- Would have liked to experiment with the hyperparameters more, and try to optimise them. Was not feasible to do a lot of systematic optimization with the time left due to long runtimes
- Could have accounted for other absorptions, which probably would have improved the accuracy as these are somewhat similar to the DLA absorptions and most likely were a source of false positives
- Could have tried to use an autoencoder perhaps? (tried to implement one to denoise the spectra since I thought that would be interesting, but then I realised I could just apply the rolling median to basically get the same effect)
- Potentially use weight decay with AdamW?