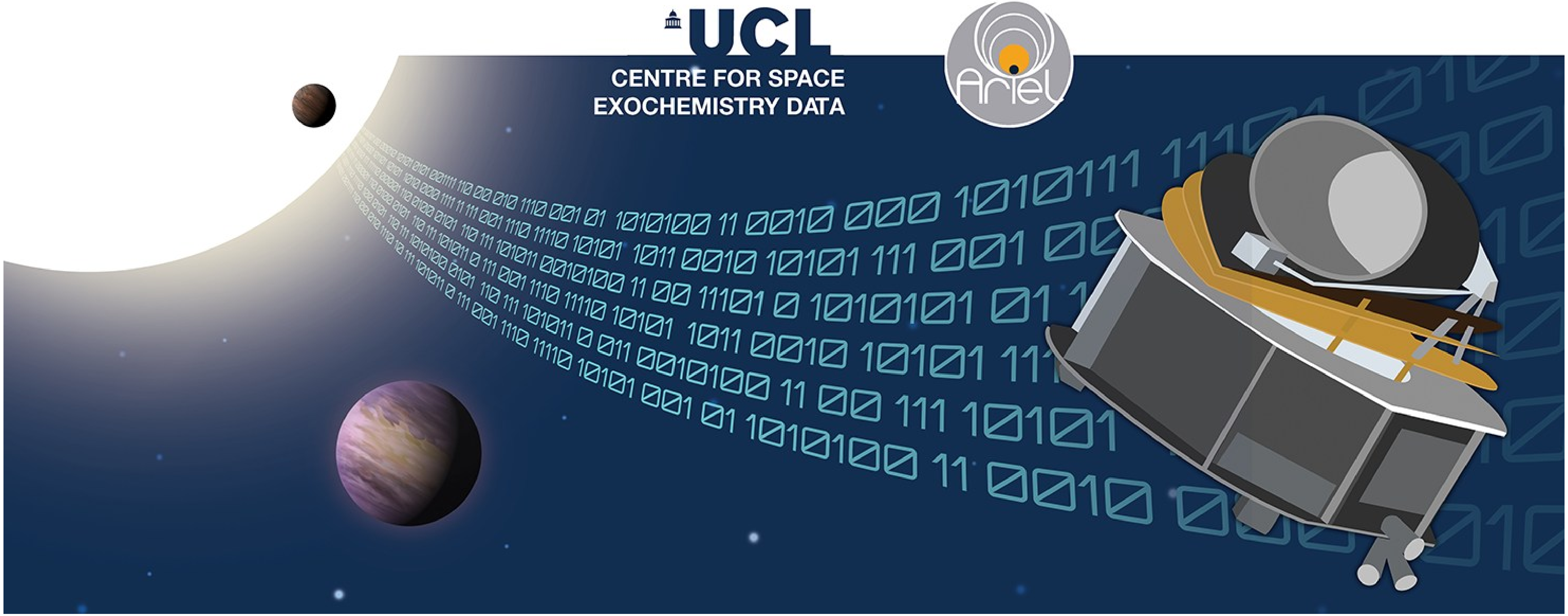


Lecture 2: Official Solution Walkthrough

Tutorials: [NeurIPS - Ariel Data Challenge 2024]

Presenter: kaggle君-sakura（bili_sakura@zju.edu.cn）

Date: October 9, 2024

Image Credit: NeurIPS - Ariel Data Challenge 2024 | Kaggle

Outline

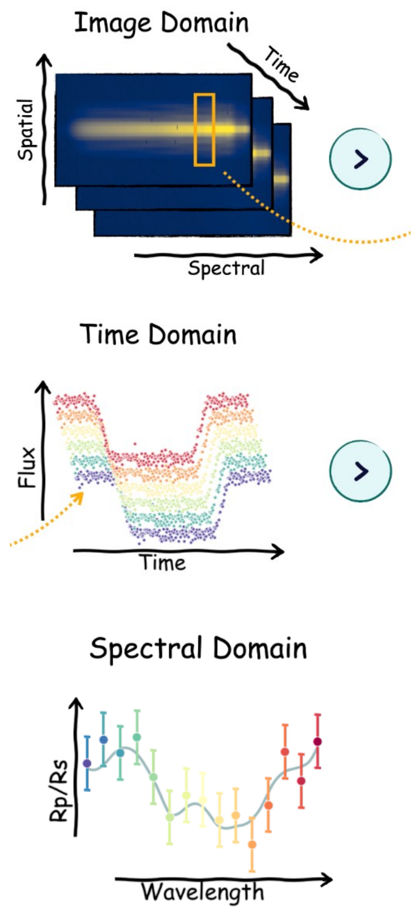
- ■ Recall the Task for Ariel Data Challenge
- ■ Calibration Details
- ■ Fundamentals
- ■ Explanation on Official Baseline Solution
  - □ Data Preparation
  - □ 1D-CNN for mean transit depth
  - □ 2D CNN for atmospheric features
  - □ Results
- ■ Insights (See Lecture 3)
  - □ Possible Tricks for Baseline Improvement
  - □ DL-Tech1 : End-to-End Transformer-Based Solution
  - □ DL-Tech2 : Consider 3D Convolutional Module

Recall the Task for Ariel Data Challenge

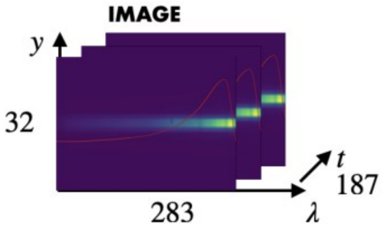
The challenge's primary objective is to process these exposures to produce a single, clean spectrum for each exoplanet, summarizing the r_p/r_s values across all wavelengths.
挑战的主要目标是处理这些曝光数据，为每个系外行星生成一个干净的单一光谱，汇总所有波长下的 r_p/r_s值。

The exposure are subject to noises and the images or spectrum are not perfect. The Jitter noise has a complex signature that the ML model should recognize to produce a better spectra.
图像或光谱并不完美。抖动噪声具有复杂的特征，机器学习模型需要识别这些特征以生成更好的光谱。

Different techniques are possible and are up to the participant imagination to produce a novel (and hopefully better) solution to this task.
这些曝光数据受到噪声的影响，可以使用不同的技术，参与者可以发挥想象力，提出一种新颖（并且希望更好）的解决方案来完成这一任务。



Calibration

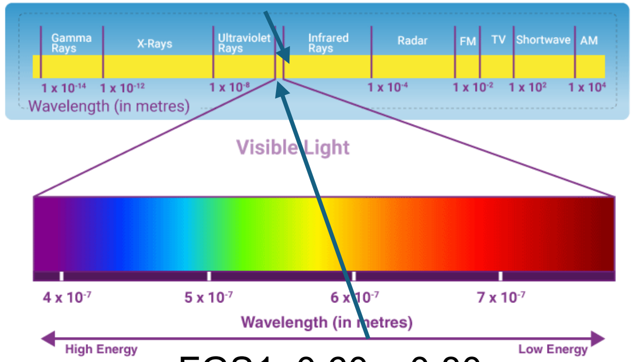


For each sample:

AIRS-CH0_signal.parquet: (11250, 356, 32) -> (187, 282, 32) (time, spectral, spatial)

FGS1_signal.parquet: (135000, 32, 32) -> (187, 32, 32) (time, spectral, spatial)

AIRS-CH0: 1.95 - 3.90 μm



FGS1: 0.60 – 0.80 μm

Calibrating and Time Binning Astronomical Data 校准和时间分箱天文数据

- Step 1: Analog-to-Digital Conversion 模拟到数字转换
- Step 2: Mask hot/dead Pixel 屏蔽热像素/坏像素
- Step 2: Linearity Correction 线性校正
- Step 3: Dark Current Subtraction 暗电流扣除
- Step 4: Get Correlated Double Sampling (CDS) 获取相关双重采样
- Step 5 (Optional): Time Binning 时间分箱
- Step 6: Flat Field Correction 平场校正

Q1: How does dimension of time decrease?

A1:

- The observations are first conducted CDS.

AIRS: (**11250**,**356**,**32**) ->(**5625**,356,32) ; FGS1: (**135000**,32,32) ->(**67500**,32,32)

- Then they are binned in time by group of 30 frames for AIRS and 360 frames for FGS1.

AIRS: (5625,**356**,**32**) ->(**187**,356,32) ; FGS1: (**135000**,32,32) ->(**187**,32,32)

	A	B	C	D	E	JS	JT	JU	JV	JW
1	wl_1	wl_2	wl_3	wl_4	wl_5	wl_279	wl_280	wl_281	wl_282	wl_283
2	0.705	1.95176	1.96061	1.96945	1.97827	3.87503	3.88006	3.88506	3.89006	3.89504

Q2: How does dimension of spectral in AIRS decrease?

A2: The images are cut along the wavelength axis between pixels 39 and 321, so that the 282 pixels left in the wavelength dimension match the last 282 targets' points, from AIRS*.

I think this mapping by cropping is confusing. We would find a way to match $N_{spectral} = 356 \rightarrow N_{spectral} = 282$

* https://www.kaggle.com/code/gordonyip/update-calibrating-and-binning-astronomical-data/comments#2953798

Image Credit: https://byjus.com/physics/visible-light/

Calibration

There are some things you should pay attention to/modify in official calibration notebook:

Step 1: Analog-to-Digital Conversion
第 1 步：模数转换

The Analog-to-Digital Conversion (adc) is performed by the detector to convert the pixel voltage into an integer number. We revert this operation by using the gain and offset for the calibration files 'train_adc_info.csv'.

模数转换 （adc） 由检测器执行，将像素电压转换为整数。我们通过使用校准文件 'train_adc_info.csv' 的增益和偏移来恢复此操作。

[6]:

```
def ADC_convert(signal, gain, offset):
    signal = signal.astype(np.float64)
    signal /= gain
    signal += offset
    return signal
```

Note we divide gain as the gain provided is an inversion factor of a standard gain.
See * for more explanation.

*https://www.kaggle.com/code/gordonyip/update-calibrating-and-binning-astronomical-data/comments#2951175

Calibration

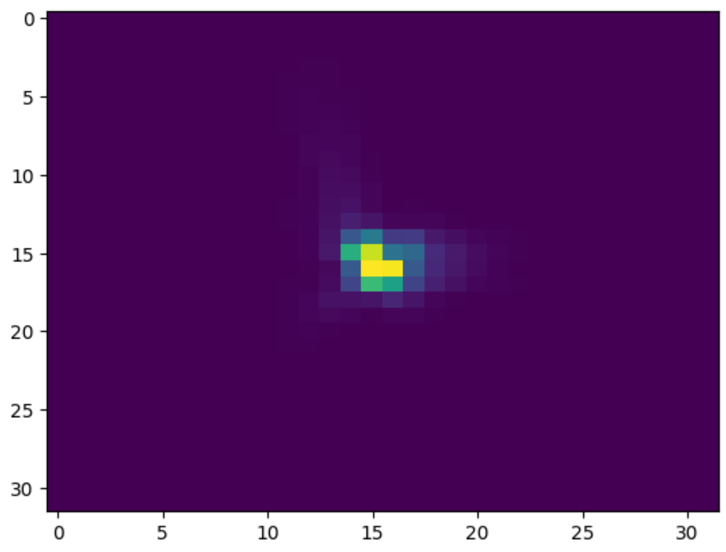
There are some things you should pay attention to/modify in official calibration notebook:

```
files = glob.glob(os.path.join(path_folder + 'train/', '*/*'))

## 48 is hardcoded here but please feel free to remove it if you want to do it for the entire dataset
index = get_index(files[:22],CHUNKS_SIZE)

train_adc_info = pd.read_csv(os.path.join(path_folder, 'train_adc_info.csv'))
train_adc_info = train_adc_info.set_index('planet_id')
axis_info = pd.read_parquet(os.path.join(path_folder,'axis_info.parquet'))
DO_MASK = True
DO_THE_NL_CORR = False
DO_DARK = True
DO_FLAT = True
TIME_BINNING = True

cut_inf, cut_sup = 39, 321
l = cut_sup - cut_inf

for n, index_chunk in enume
    AIRS_CH0_clean = np.zeros((CHUNKS_SIZE, 11250, 32, l))
    FGS1_clean = np.zeros((CHUNKS_SIZE, 135000, 32, 32))

    for i in range (CHUNKS_SIZE) :
        df = pd.read_parquet(os.path.join(path_folder,f'train/{index_chunk[i]}/AIRS-CH0_signal.parquet'))
        signal = df.values.astype(np.float64).reshape((df.shape[0], 32, 356))
```

1. Default CHUNKS_SIZE=1, enlarge it for acceleration.
2. Replace "[:22]" with "[:]" to process all data in "train" folder. If you want to do a subset, the index number should be 4 × sample_number. In other words, "[:]" works the same as "[:4*673]" where 673 is the number of samples in "train" folder.
3. As mentioned previously, this cropping for AIRS_CH0 data in spectrum dimension is confusing. Use other way instead.

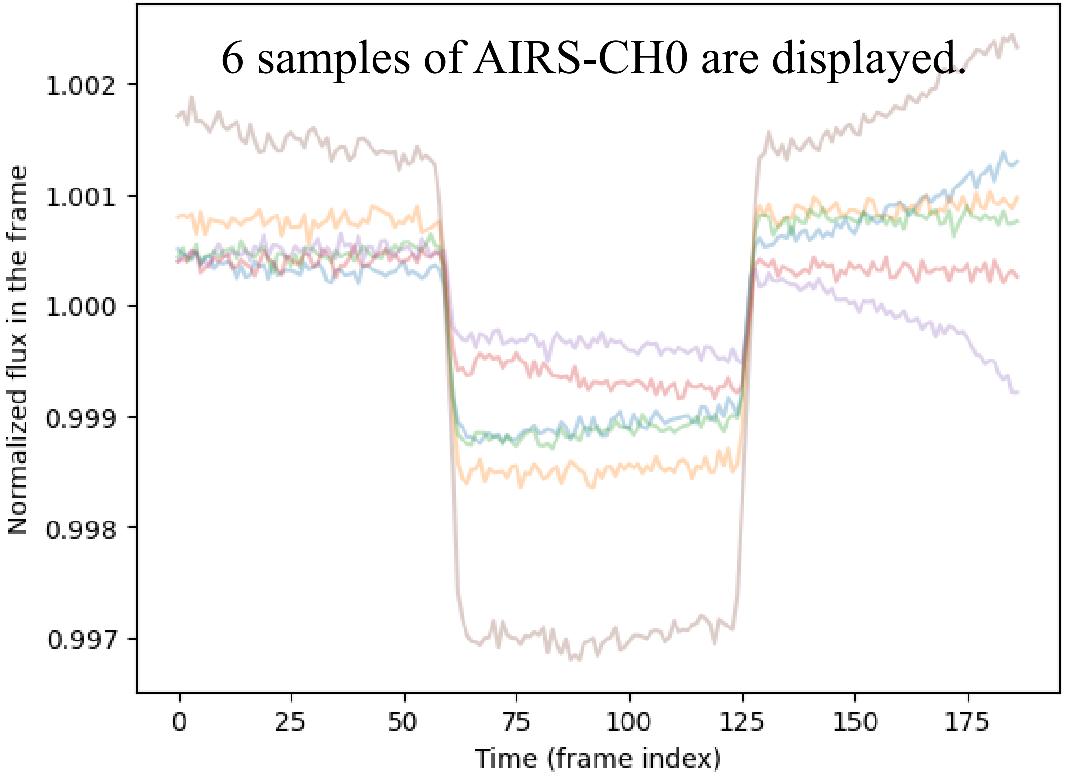
Calibration

```
plt.imshow(data_train_FGS[-1,50,:,:].T, aspect = 'auto')
```



An image-like slice from FGS1 sample data, where time_frame=50. X-axis denotes spectral dimension while Y-axis denotes spatial dimension.

For each sample:
AIRS-CH0_signal.parquet: $(11250, 356, 32)$ -> $(187, 282, 32)$ (time, spectral, spatial)
FGS1_signal.parquet: $(135000, 32, 32)$ -> $(187, 32, 32)$ (time, spectral, spatial)



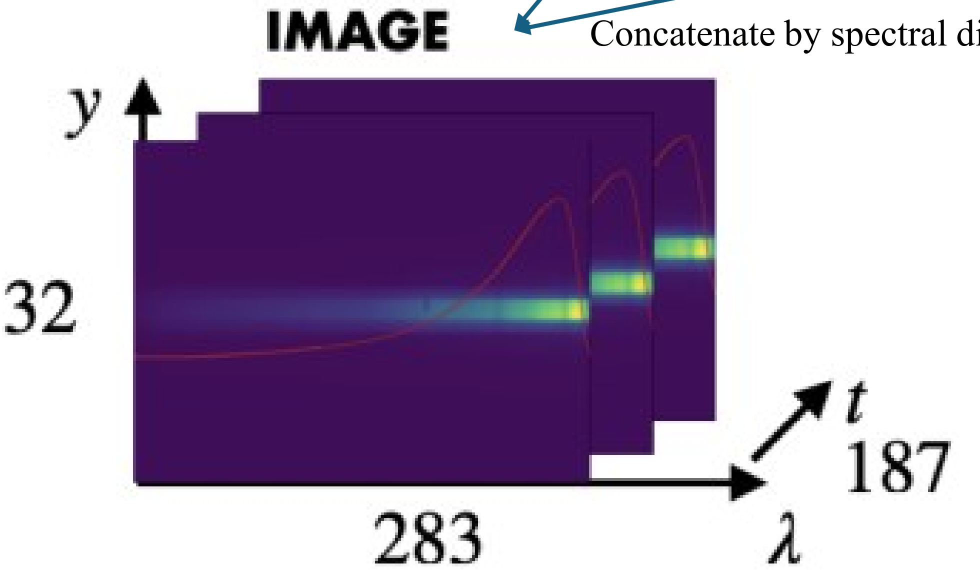
Visualization of calibrated data.

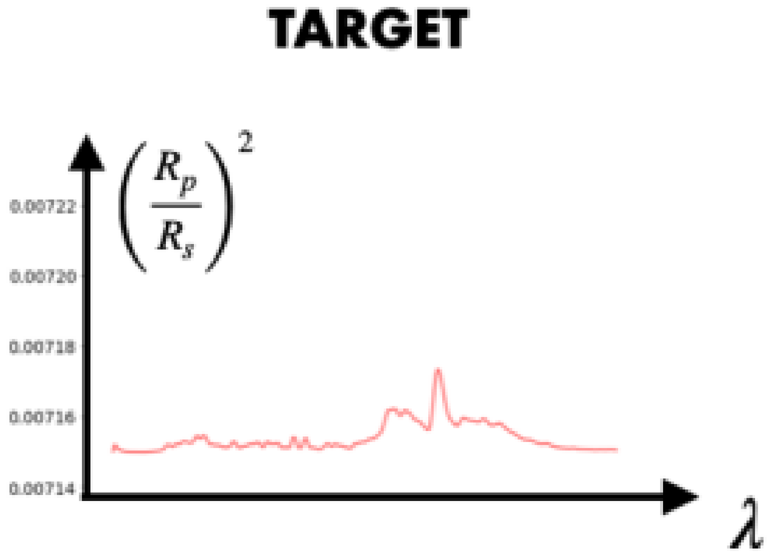
Data Preparation

For FGS1 spectral dimension, the official solution seems to use average 32 channel, which ranges from 0.6 μm to 0.8 μm, to match the target wavelength i.e. 0.705 μm. This is also confusing. Check it on your own, and you may refer to comments or other people's solution to find a better way out.

For each sample:
AIRS-CH0_signal.parquet: $(11250, 356, 32)$ -> $(187, 282, 32)$ (time, spectral, spatial)
FGS1_signal.parquet: $(135000, 32, 32)$ -> $(187, 32, 32)$ (time, spectral, spatial) → $(187, 1, 32)$

Concatenate by spectral dimension





Fundamentals of Transits

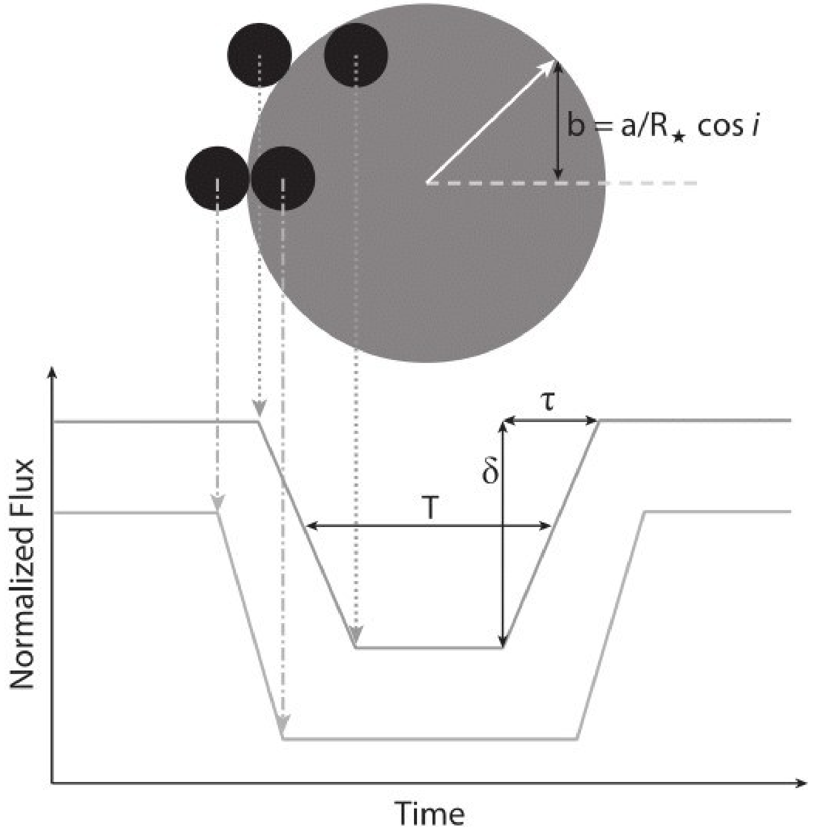
Following are astrophysics theorem only.

1. **Flux Drop During Transit**: When an exoplanet passes in front of its host star (a transit event), the star's light is partially blocked. This causes a measurable decrease in the star's observed flux. The amount of flux reduction is directly related to the size of the planet relative to the star, i.e., the r_p/r_s ratio.
2. **Flux Reduction Equation**: The change in flux ΔF during a transit can be approximated by the following equation:

$$\Delta F \propto \left(\frac{r_p}{r_s}\right)^2$$

Here, r_p/r_s is the ratio of the planet's radius to the star's radius. This shows that the flux decrease is proportional to the square of this ratio. Larger planets (larger r_p) block more light, resulting in a greater flux drop.

在系外行星的凌星过程中，恒星的光通量（flux）与恒星半径 r_s和行星半径 r_p 的比值 r_p/r_s 有着直接关系。这种关系是凌星光度法中推断行星大小的重要基础。
此处，光通量即对y轴（空间轴）像素值（光强）求和得到。



Theoretical transiting exoplanet light curve. This image shows the transit depth (δ), transit duration (T), and ingress/egress duration (τ) of a transiting exoplanet relative to the position that the exoplanet is to the star.

Fundamentals of Transits

3. **Transit Depth**: The transit depth, or the percentage drop in flux during the transit, is a measure of the planet's size relative to the star. It is given by:

凌星深度： 凌星深度，或在凌星期间光通量下降的百分比，可以表示为：

F_{out}:测量凌星外的光通量
F_{in}:测量凌星期间的光通量

$$\delta = \frac{F_{out} - F_{in}}{F_{out}} = \left(\frac{r_p}{r_s}\right)^2$$

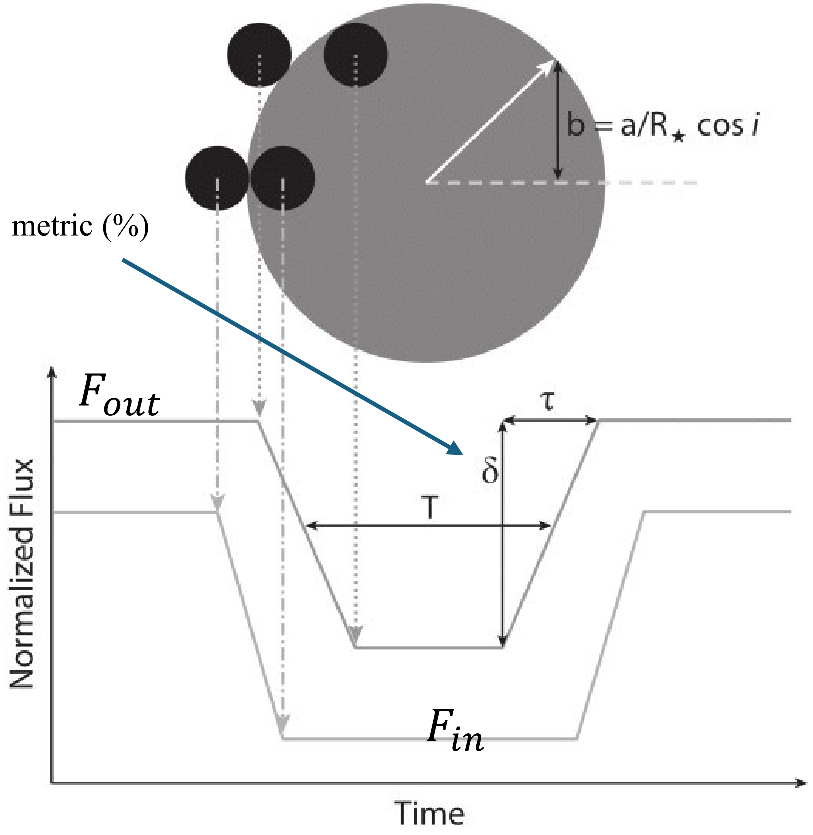
For example, if a planet is 10% the size of its star, the flux will drop by approximately 1% during the transit.

4. **Inferring Planetary Radius**: By measuring the amount of flux decrease during a transit and knowing the star's radius (r_s), we can calculate the planetary radius (r_p) using the relationship:

推断行星半径： 通过测量凌星期间的光通量下降量并已知恒星的半径 r_s，可以使用以下关系计算出行星的半径 r_p：

$$r_p = r_s \times \sqrt{\delta}$$

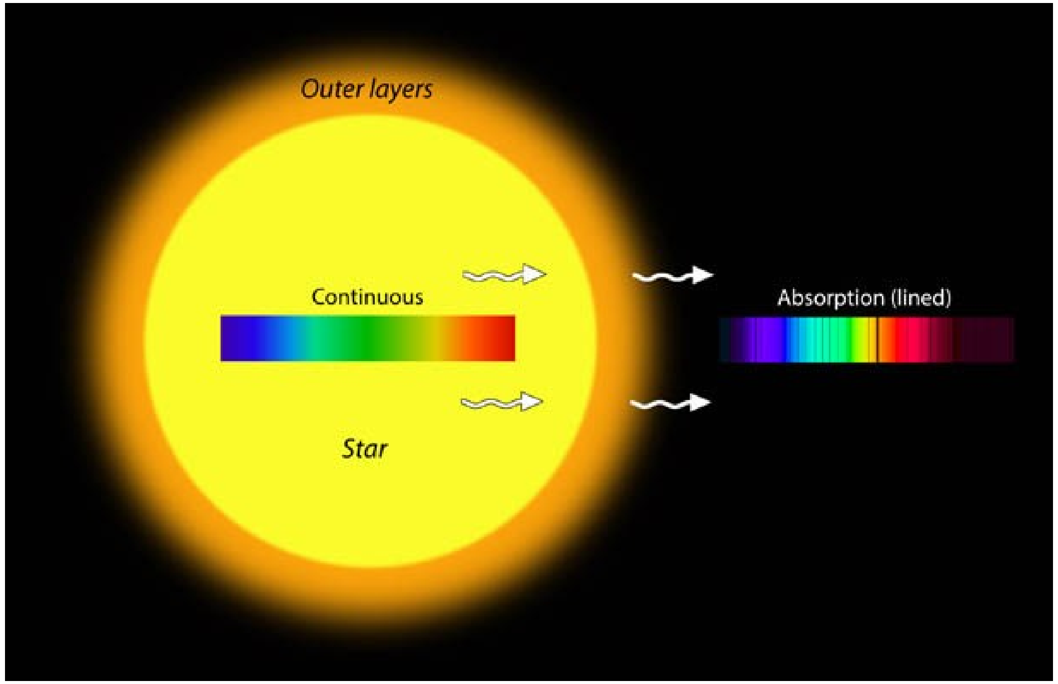
物理模型的基础： 该模型还假设行星和恒星之间的距离相对较大，以至于我们可以将行星和恒星视为两个圆盘（视角内的圆）。在这种假设下，行星的投影面积和恒星的投影面积的比值决定了凌星期间光通量的变化。
理想情况假设： 这种物理模型假设恒星是均匀发光的，即恒星表面每个区域的亮度相同。在现实中，恒星的亮度分布可能会因为**边缘昏暗效应**（Limb Darkening）而略有变化，但这个模型提供了一个很好的近似。

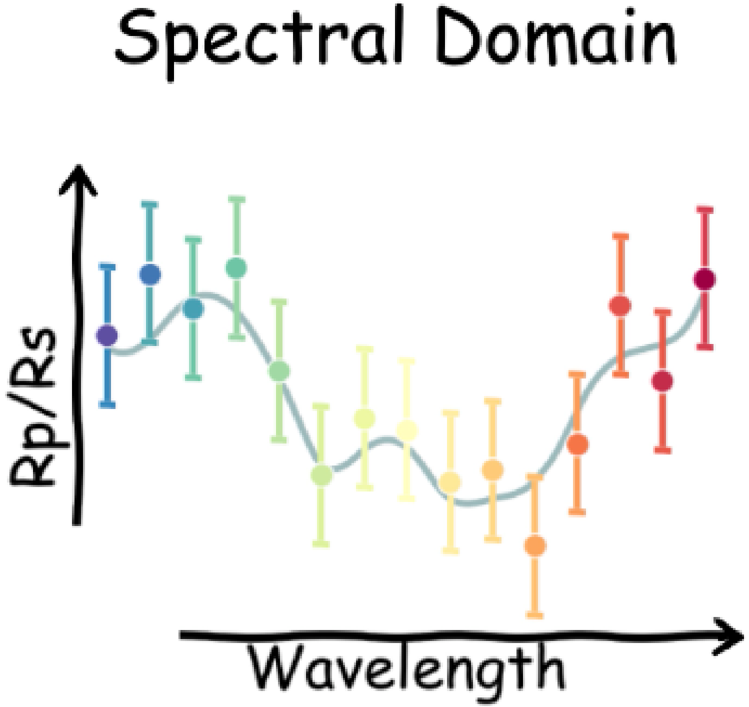


Theoretical transiting exoplanet light curve. This image shows the transit depth (δ), transit duration (T), and ingress/egress duration (τ) of a transiting exoplanet relative to the position that the exoplanet is to the star.

Fundamentals of Transits

- The size of a certain planet (r_p) is a constant.
- The size of a certain star (r_s) is various. You should specify the spectral (wavelength) to denote which r_s is talking about.





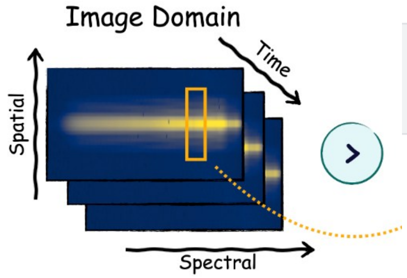
Data Preparation

We highly commend to preprocess data on your own again with corrected configurations in official notebook, as we have found several mistakes in the official notebook.

Setup Paths and Read Data

```
data_folder = '/kaggle/input/binned-dataset-v3/' # path to the folder containing the data
auxiliary_folder = '/kaggle/input/ariel-data-challenge-2024/' # path to the folder containing the train targets and wavelength
```

```
data_train = np.load(f'{data_folder}/data_train.npy')
data_train_FGS = np.load(f'{data_folder}/data_train_FGS.npy')
```



We create a directory to save the outputs of this notebook, and define the hyperparameters of the model

```
output_dir = './output'

SEED = 42

do_the_mcdropout_wc = True
do_the_mcdropout = True

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"Directory {output_dir} created.")
else:
    print(f"Directory {output_dir} already exists.")
```

Input

+ Add Input
Upload

COMPETITIONS

- NeurIPS - Ariel Data Challenge 2024

DATASETS

- baseline-img
- binned-dataset-v3
  - data_train.npy
  - data_train_FGS.npy

Output (80KiB / 19.5GiB)

- /kaggle/working

Table of contents

Data Preparation

1D-CNN for mean transit depth

FGS1 AIRS-CH0

	A	B	C	D	E	JS	JT	JU	JV	JW
1	wl_1	wl_2	wl_3	wl_4	wl_5	wl_279	wl_280	wl_281	wl_282	wl_283
2	0.705	1.95176	1.96061	1.96945	1.97827	3.87503	3.88006	3.88506	3.89006	3.89504

```
train_solution = np.loadtxt(f'{auxiliary_folder}/train_labels.csv', delimiter = ',', skiprows = 1)
targets = train_solution[:,1:]
# used for the 1D-CNN to extract the mean value,
# only AIRS wavelengths as the FGS point is not used in the white curve
targets_mean = targets[:,1:].mean(axis = 1)
N = targets.shape[0]
```

Shape: (N_sample,N_wavelength)

Calculate mean without FGS1 i.e. wl_1

So, the sequence a bit differ from train_labels.csv.

data > train_labels.csv > data

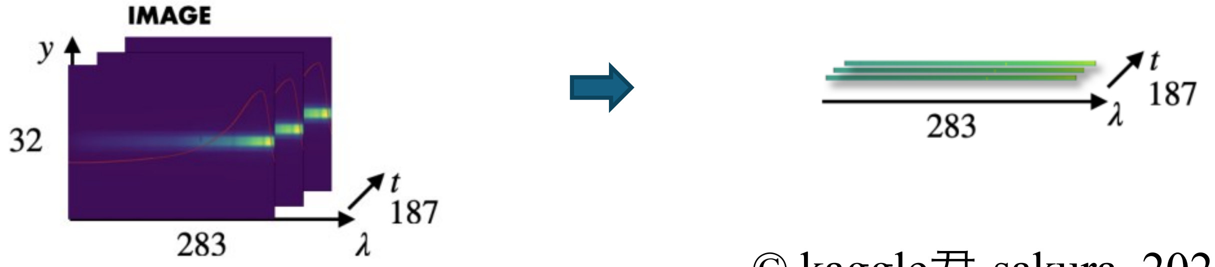
```
1  planet_id,wl_1,wl_2,wl_3,wl_4,wl_5,wl_6,wl_7,wl_8
2  785834,0.0010857421046721,0.0011374878153765,0.00
3  14485303,0.0018350194388515,0.0018348189852079,0.0
4  17002355,0.0027918368327131,0.0028136573643522,0.0
5  24135240,0.0012942004606281,0.0013081888097258,0.0
```

We create the dataset by adding the FGS frame, crushed in one column, **at the end** of the AIRS data cube.

```
signal_AIRS_diff_transposed_binned, signal_FGS_diff_transposed_binned  = data_train, data_train_FGS
FGS_column = signal_FGS_diff_transposed_binned.sum(axis = 2)
dataset = np.concatenate([signal_AIRS_diff_transposed_binned, FGS_column[:,:, np.newaxis,:]], axis = 2)
```

We sum up the pixels on the **y-axis** to transform the data into 2D images

```
dataset = dataset.sum(axis=3)
```



Data Preparation

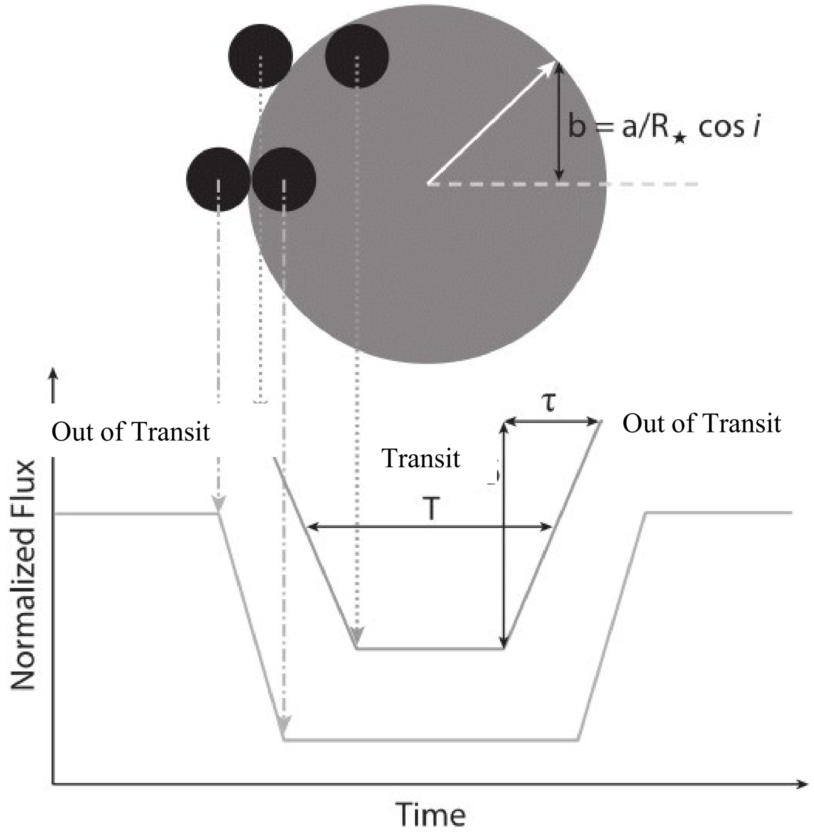
We divide the images by the star flux assuming the **first and last 50 instants belong to the out of transit**.

The images are normalized using the star spectrum extracted from the images themselves.

```
def create_dataset_norm(dataset1, dataset2) :
    dataset_norm1 = np.zeros(dataset1.shape)
    dataset_norm2 = np.zeros(dataset1.shape)
    dataset_min = dataset1.min()
    dataset_max = dataset1.max()
    dataset_norm1 = (dataset1 - dataset_min) / (dataset_max - dataset_min)
    dataset_norm2 = (dataset2 - dataset_min) / (dataset_max - dataset_min)
    return dataset_norm1, dataset_norm2


def norm_star_spectrum (signal) :
    img_star = signal[:,:50].mean(axis = 1) + signal[:,-50:].mean(axis = 1)
    return signal/img_star[:,np.newaxis,:]

dataset_norm = norm_star_spectrum(dataset)
dataset_norm = np.transpose(dataset_norm,(0,2,1))
```

Create_dataset_norm function seems to be not used in the notebook. Check it.

Star Spectrum Flux
不发生凌日时，恒星自己的光谱特征



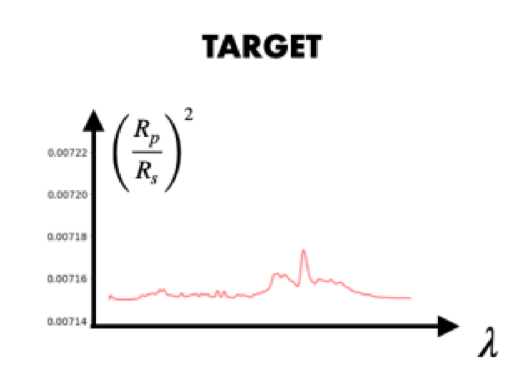
Data Preparation

Actually, if you are convinced with your model, you do not need test set (this notebook do not split test set either). Just run your code on the test set officially provided for scoring.

We start by computing a "white curve", that is actually the sum of the signal over the all image, as a function of time. We split the data and normalize the train/valid/test data.

```
# we have previously cut the data along the wavelengths to remove the edges,
# this is to match with the targets range in the make data file
cut_inf, cut_sup = 39, 321
l = cut_sup - cut_inf + 1
wls = np.arange(l)


def split (data, N) :
    list_planets = random.sample(range(0, data.shape[0]), N_train)
    list_index_1 = np.zeros(data.shape[0], dtype = bool)
    for planet in list_planets :
        list_index_1[planet] = True
    data_1 = data[list_index_1]
    data_2 = data[~list_index_1]
    return data_1, data_2, list_index_1

N_train = 8*N//10

# Validation and train data split
train_obs, valid_obs, list_index_train = split(dataset_norm, N_train)
train_targets, valid_targets = targets[list_index_train], targets[~list_index_train]
```

Again, as mentioned before, this part is confusing. Try your own.



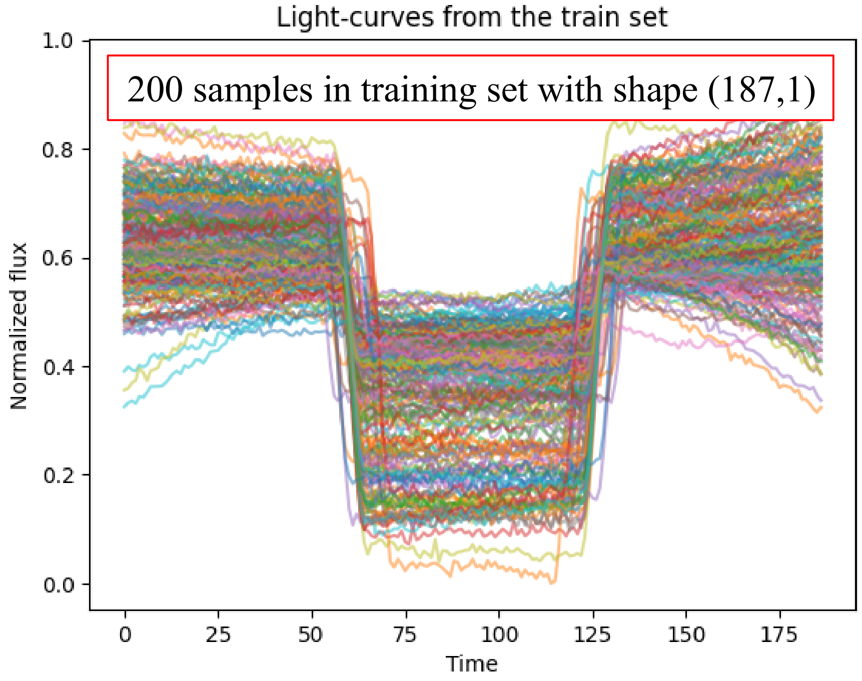
Data Preparation

```python
signal_AIRS_diff_transposed_binned = signal_AIRS_diff_transposed_binned.sum(axis=3)
wc_mean = signal_AIRS_diff_transposed_binned.mean(axis=1).mean(axis=1)
white_curve = signal_AIRS_diff_transposed_binned.sum(axis=2)/ wc_mean[:, np.newaxis]

def normalise_wlc(train, valid) :

    wlc_train_min = train.min()
    wlc_train_max = train.max()
    train_norm = (train - wlc_train_min) / (wlc_train_max - wlc_train_min)
    valid_norm = (valid - wlc_train_min) / (wlc_train_max - wlc_train_min)

    return train_norm, valid_norm

def normalize (train, valid) :
    max_train = train.max()
    min_train = train.min()
    train_norm = (train - min_train) / (max_train - min_train)
    valid_norm = (valid - min_train) / (max_train - min_train)
    return train_norm, valid_norm, min_train, max_train

# Split the light curves and targets
train_wc, valid_wc = white_curve[list_index_train], white_curve[~list_index_train]
train_targets_wc, valid_targets_wc = targets_mean[list_index_train], targets_mean[~list_index_train]

# Normalize the wlc
train_wc, valid_wc = normalise_wlc(train_wc, valid_wc)

# Normalize the targets
train_targets_wc_norm, valid_targets_wc_norm, min_train_valid_wc, max_train_valid_wc = normalize(train_targets_wc, valid_targets_wc)
```

Data Preparation

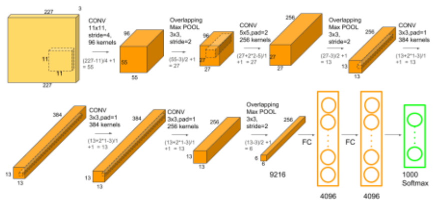
In 1D CNN pipeline, the spectral dimension is normalized into a single dimension which is closely related the conception of flux.

```
plt.figure()
for i in range (200) :
    plt.plot(train_wc[-i], '-', alpha = 0.5)
plt.title('Light-curves from the train set')
plt.xlabel('Time')
plt.ylabel('Normalized flux')
plt.show()
```



Train 1D CNN

In this notebook, CNNs are much similar to AlexNet (2012), where they pass convolutional layer and then FC layers.



```
from keras.layers import Input, Conv1D, MaxPooling1D, Flatten, Dense, Dropout, BatchNormalization, Concatenate,AveragePooling1D
from keras.models import Model, load_model
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import LearningRateScheduler, ModelCheckpoint


input_wc = Input((187,1))
x = Conv1D(32, 3, activation='relu')(input_wc)
x = MaxPooling1D()(x)
x = BatchNormalization() (x)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D()(x)
x = Conv1D(128, 3, activation='relu')(x)
x = MaxPooling1D()(x)
x = Conv1D(256, 3, activation='relu')(x)
x = MaxPooling1D()(x)
x = Flatten()(x)

x = Dense(500, activation='relu')(x)
x = Dropout(0.2)(x, training = True)
x = Dense(100, activation='relu')(x)
x = Dropout(0.1)(x, training = True)
output_wc = Dense(1, activation='linear')(x)

model_wc = Model(inputs=input_wc, outputs=output_wc)
model_wc.summary()
```

Model: "functional"

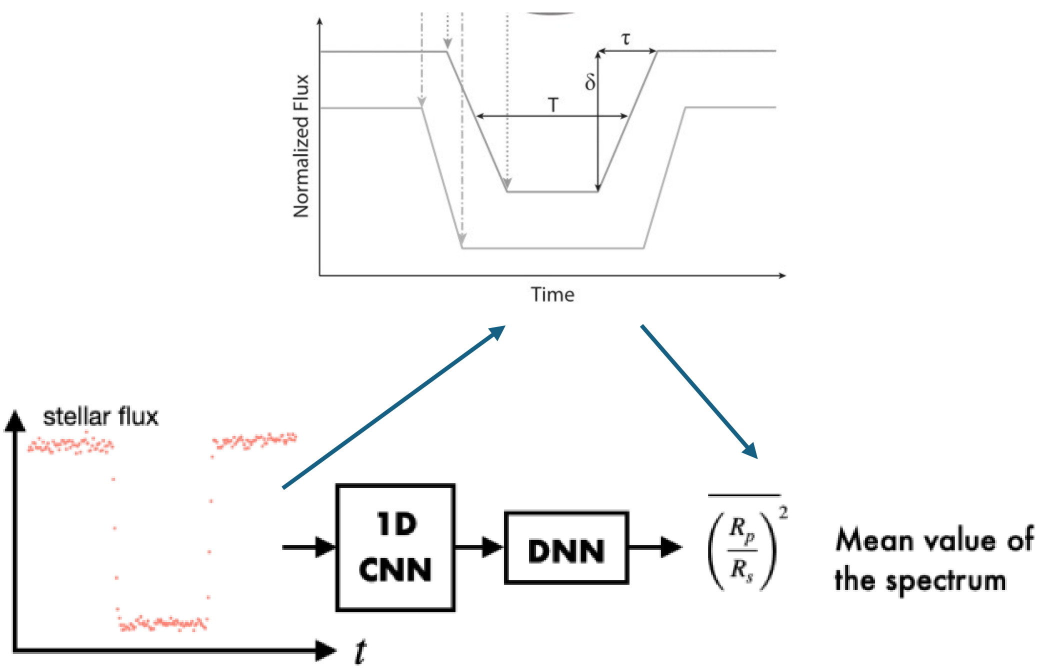
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 187, 1)	0
conv1d (Conv1D)	(None, 185, 32)	128
max_pooling1d (MaxPooling1D)	(None, 92, 32)	0
batch_normalization (BatchNormalization)	(None, 92, 32)	128
conv1d_1 (Conv1D)	(None, 90, 64)	6,208
max_pooling1d_1 (MaxPooling1D)	(None, 45, 64)	0
conv1d_2 (Conv1D)	(None, 43, 128)	24,704
max_pooling1d_2 (MaxPooling1D)	(None, 21, 128)	0
conv1d_3 (Conv1D)	(None, 19, 256)	98,560
max_pooling1d_3 (MaxPooling1D)	(None, 9, 256)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 500)	1,152,500
dropout (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 100)	50,100
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 1)	101

Total params: 1,332,429 (5.08 MB)
Trainable params: 1,332,365 (5.08 MB)
Non-trainable params: 64 (256.00 B)

Train 1D CNN

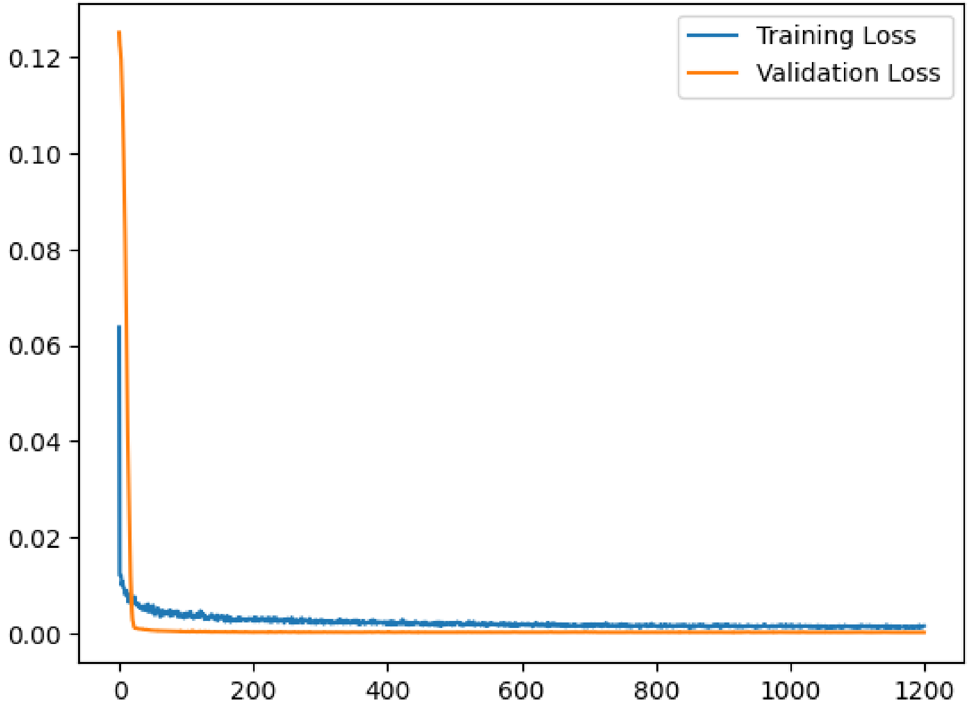
For training, what we do is to filter the noise to get a unnoised F_{out} and F_{in}. Thus, we can calculate $\delta=\frac{F_{out}-F_{in}}{F_{out}}=\left(\frac{r_p}{r_s}\right)^2$ theoretically.

```
def scheduler(epoch, lr):
    decay_rate = 0.2
    decay_step = 200
    if epoch % decay_step == 0 and epoch:
        return lr * decay_rate
    return lr

optimizer = SGD(0.001)
model_wc.compile(optimizer=optimizer, loss='mse', metrics=[MeanAbsoluteError()])
callback = LearningRateScheduler(scheduler)
checkpoint_filepath = 'output/model_1dcnn.keras'
model_ckt = ModelCheckpoint(
    checkpoint_filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode="min",
    save_freq="epoch",
)

print('Running ...')
history = model_wc.fit(
    x = train_wc,
    y = train_targets_wc_norm,
    validation_data = (valid_wc, valid_targets_wc_norm),
    batch_size=16,
    epochs= 1200,
    shuffle=True,
    verbose=0,
    callbacks=[model_ckt]
    )
print('Done.')
```



1D CNN Inference

```
model_wc = load_model(checkpoint_filepath)

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```



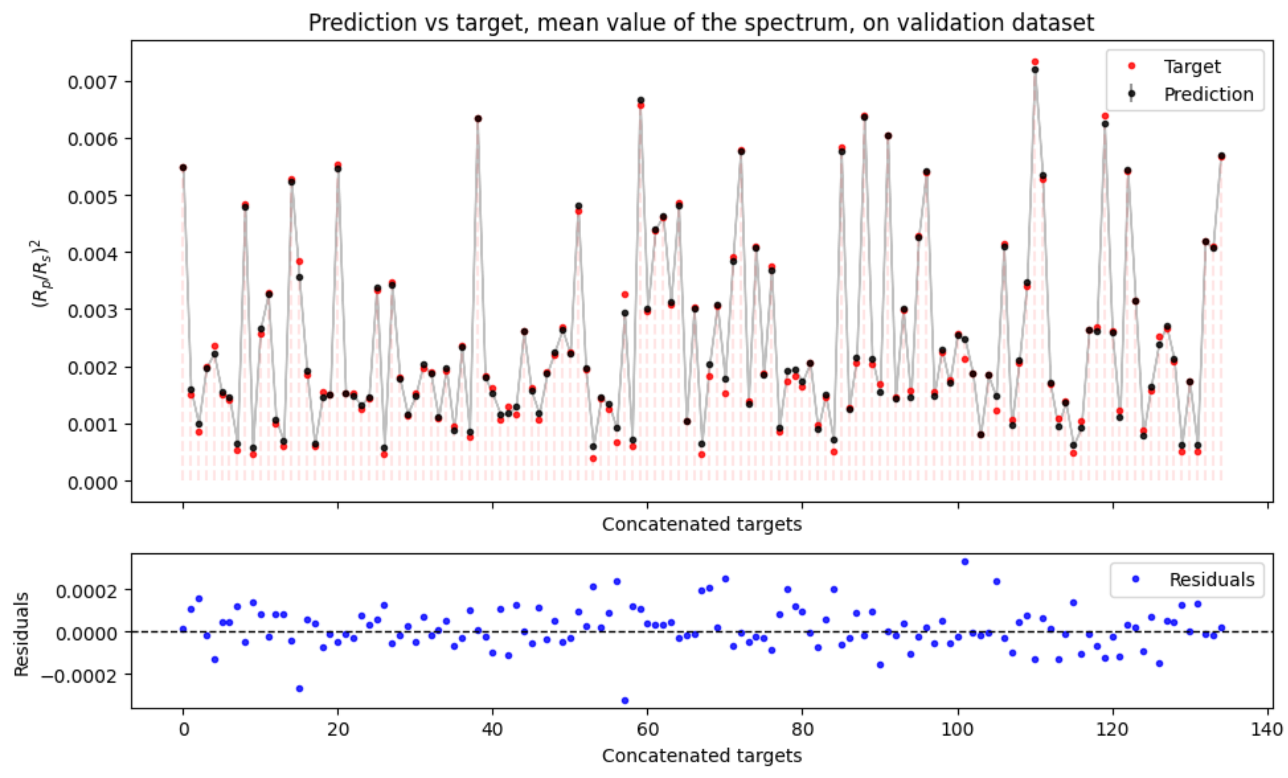
1D CNN Inference

Then, we perform the MC Dropout to obtain the mean prediction and the uncertainty associated. We choose to compute 1000 instances.

```
nb_dropout_wc = 1000

def unstandardizing (data, min_train_valid, max_train_valid) :
    return data * (max_train_valid - min_train_valid) + min_train_valid

def MC_dropout_WC (model, data, nb_dropout) :
    predictions = np.zeros((nb_dropout, data.shape[0]))
    for i in range(nb_dropout) :
        predictions[i,:] = model.predict(data, verbose = 0).flatten()
    return predictions

if do_the_mcdropout_wc :
    print('Running ...')
    prediction_valid_wc = MC_dropout_WC(model_wc, valid_wc, nb_dropout_wc)
    spectre_valid_wc_all = unstandardizing(prediction_valid_wc, min_train_valid_wc, max_train_valid_wc)
    spectre_valid_wc, spectre_valid_std_wc = spectre_valid_wc_all.mean(axis = 0), spectre_valid_wc_all.std(axis = 0)
    print('Done.')

else :
    spectre_valid_wc = model_wc.predict(valid_wc).flatten()
    spectre_valid_wc = unstandardizing(spectre_valid_wc, min_train_valid_wc, max_train_valid_wc)
    spectre_valid_std_wc = 0.1*np.abs(spectre_valid_wc)
```

1D CNN Inference

```
residuals = spectre_valid_wc - valid_targets_wc
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), sharex=True,
                               gridspec_kw={'height_ratios': [3, 1]})

ax1.errorbar(x = np.arange(len(spectre_valid_wc)),
             y = spectre_valid_wc, yerr =spectre_valid_std_wc, f
ax1.fill_between(np.arange(len(spectre_valid_wc)),
                 spectre_valid_wc - spectre_valid_std_wc, spectr
ax1.vlines(np.arange(len(spectre_valid_wc)),ymin=0, ymax=spectre
ax1.plot(valid_targets_wc, 'r.', label='Target', alpha=0.8)
ax1.set_xlabel('Concatenated targets')
ax1.set_ylabel('$(R_p/R_s)^2$')
ax1.set_title('Prediction vs target, mean value of the spectrum,
ax1.legend()

ax2.plot(residuals, 'b.', label='Residuals', alpha=0.8)
ax2.set_xlabel('Concatenated targets')
ax2.set_ylabel('Residuals')
ax2.axhline(0, color='black', linestyle='--', linewidth=1)
ax2.legend()

plt.tight_layout()
plt.show()
```



1D CNN Inference

```
residuals = valid_targets_wc - spectre_valid_wc
print('MSE : ', np.sqrt((residuals**2).mean())*1e6, 'ppm')
```

```
MSE :  100.05598082843308 ppm
```

```
# np.save(f'{output_dir}/pred_valid_wc.npy', spectre_valid_wc)
# np.save(f'{output_dir}/targ_valid_wc.npy', valid_targets_wc)
# np.save(f'{output_dir}/std_valid_wc.npy', spectre_valid_std_wc)
```

Preprocessing for 2D CNN

2D CNN for atmospheric features

We now remove the mean value (transit depth) of the spectra to keep the atmospheric features only

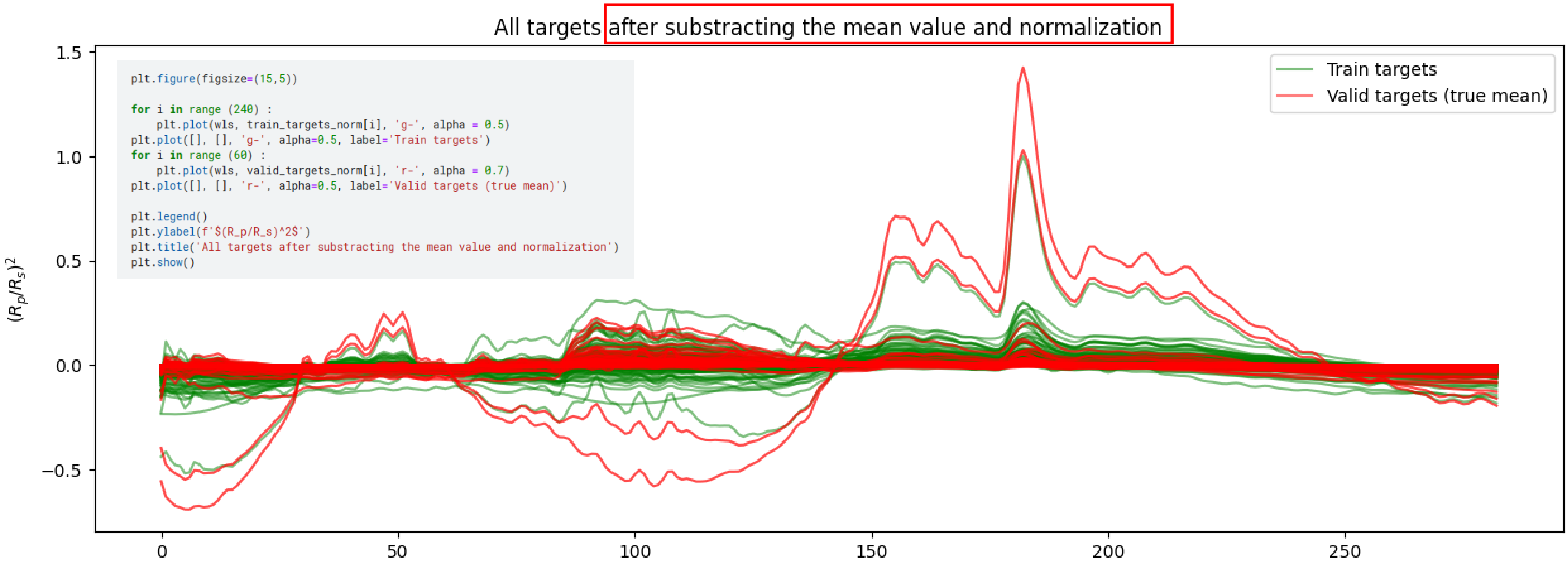
```
def suppress_mean(targets, mean) :
    res = targets - np.repeat(mean.reshape((mean.shape[0], 1)), repeats = targets.shape[1], axis = 1)
    return res
train_targets, valid_targets = targets[list_index_train], targets[~list_index_train]

train_targets_shift = suppress_mean(train_targets,  targets_mean[list_index_train])
valid_targets_shift = suppress_mean(valid_targets,  targets_mean[~list_index_train])
```

Note: we still sum spatial dimension.

We normalize the targets so that they range between -1 and 1, centered on zero

```
##### normalization of the targets ###
def targets_normalization (data1, data2) :
    data_min = data1.min()
    data_max = data1.max()
    data_abs_max = np.max([data_min, data_max])
    data1 = data1/data_abs_max
    data2 = data2/data_abs_max
    return data1, data2, data_abs_max

def targets_norm_back (data, data_abs_max) :
    return data * data_abs_max

train_targets_norm, valid_targets_norm, targets_abs_max = targets_normalization(train_targets_shift, valid_targets_shift)
```

Preprocessing for 2D CNN



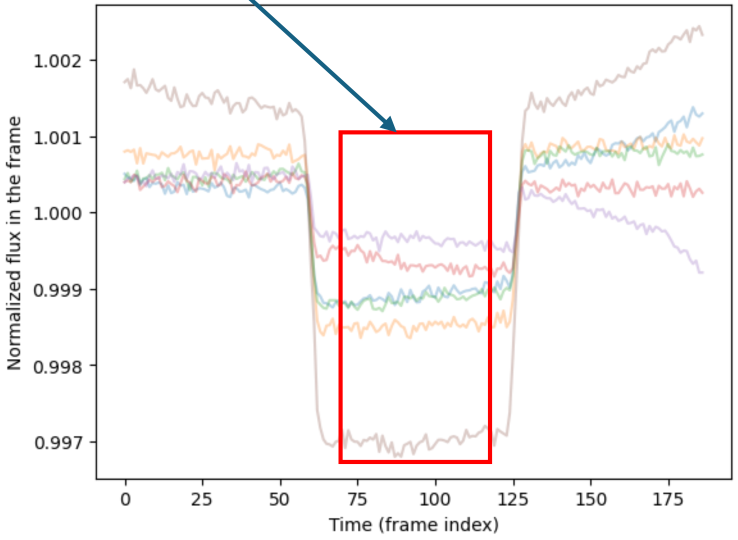
Preprocessing for 2D CNN

```
###### Transpose #####
train_obs = train_obs.transpose(0, 2, 1)
valid_obs = valid_obs.transpose(0, 2, 1)
print(train_obs.shape)
```

(538, 187, 283) $(N_{training\ samples}, D_{time}, D_{spectrel})$

We cut the transit to keep the in-transit. We assume an arbitrary transit duration of 40 instants with a transit occuring between 75 and 115.

```
##### Substracting the out transit signal #####
def suppress_out_transit (data, ingress, egress) :
    data_in = data[:, ingress:egress,:]
    return data_in

ingress, egress = 75,115
train_obs_in = suppress_out_transit(train_obs, ingress, egress)
valid_obs_in = suppress_out_transit(valid_obs, ingress, egress)
```



Preprocessing for 2D CNN

```
###### Substract the mean #####
def substract_data_mean(data):
    data_mean = np.zeros(data.shape)
    for i in range(data.shape[0]):
        data_mean[i] = data[i] - data[i].mean()
    return data_mean

train_obs_2d_mean = substract_data_mean(train_obs_in)
valid_obs_2d_mean = substract_data_mean(valid_obs_in)
```

We remove the mean value of the in-transit to get relative data like the targets

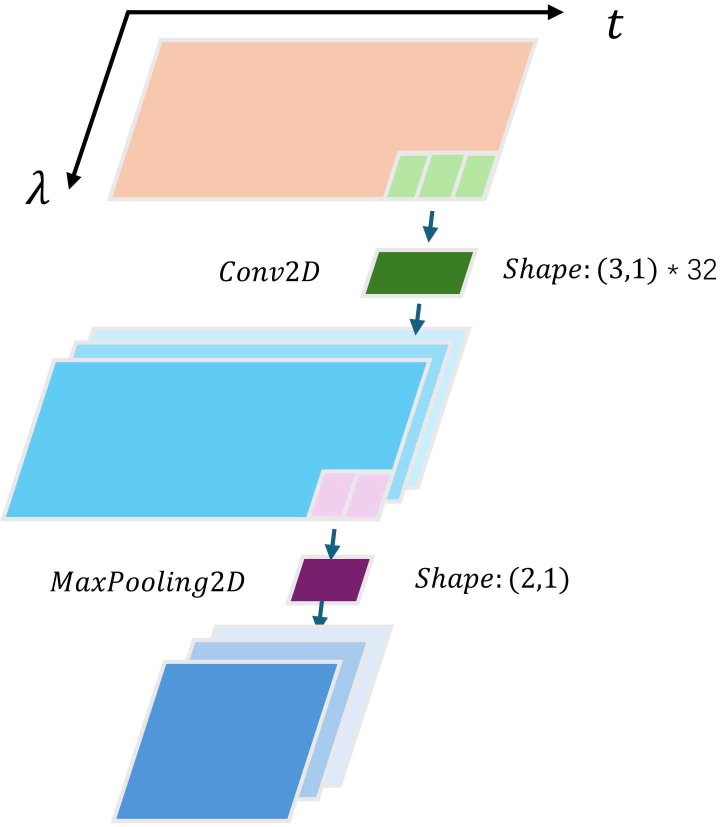
```
##### Normalization dataset #####
def data_norm(data1, data2):
    data_min = data1.min()
    data_max = data1.max()
    data_abs_max = np.max([data_min, data_max])
    data1 = data1/data_abs_max
    data2 = data2/data_abs_max
    return data1, data2, data_abs_max


def data_normback(data, data_abs_max) :
    return data * data_abs_max

train_obs_norm, valid_obs_norm, data_abs_max = data_norm(train_obs_2d_mean, valid_obs_2d_mean)
```

We use the same normalization as for the targets, i.e. between -1 and 1 centered on zero

Train 2D CNN

```python
from tensorflow import keras
from keras.layers import
{Input, Conv2D, MaxPooling2D, Flatten, Dense,
 Concatenate, Reshape, Dropout, BatchNormalization, AveragePooling2D}
from keras.models import Model
import tensorflow as tf
import numpy as np

## CNN 2 global normalization data
input_obs = Input((40,283,1))
x = Conv2D(32, (3, 1), activation='relu', padding='same')(input_obs)
x = MaxPooling2D((2, 1))(x)
x = BatchNormalization() (x)
x = Conv2D(64, (3, 1), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 1))(x)
x = Conv2D(128, (3, 1), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 1))(x)
x = Conv2D(256, (3, 1), activation='relu', padding='same')(x)
x = Conv2D(32, (1, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((1, 2))(x)
x = BatchNormalization() (x)
x = Conv2D(64, (1, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((1, 2))(x)
x = Conv2D(128, (1, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((1, 2))(x)
x = Conv2D(256, (1, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((1, 2))(x)
x = Flatten()(x)
# DNN
x = Dense(700, activation='relu')(x)
x = Dropout(0.2)(x, training = True)
output = Dense(283, activation='linear')(x)

model = Model(inputs=[input_obs], outputs=output)

checkpoint_filepath = 'output/model_2dcnn.keras'
model_ckt2 = ModelCheckpoint(
    checkpoint_filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode="min",
    save_freq="epoch",
)
model.compile(optimizer=Adam(0.001), loss='mse', metrics=[MeanAbsoluteError()])
model.summary()
```

```python
history = model.fit(
     x = train_obs_norm,
     y = train_targets_norm,
     validation_data = (valid_obs_norm, valid_targets_norm),
     batch_size=32,
     epochs= 200,
     shuffle=True,
     verbose=0,
    callbacks=[model_ckt2]
 )
```



Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 40, 283, 1)	0
conv2d (Conv2D)	(None, 40, 283, 32)	128
max_pooling2d (MaxPooling2D)	(None, 20, 283, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 20, 283, 32)	128
conv2d_1 (Conv2D)	(None, 20, 283, 64)	6,208
max_pooling2d_1 (MaxPooling2D)	(None, 10, 283, 64)	0
conv2d_2 (Conv2D)	(None, 10, 283, 128)	24,704
max_pooling2d_2 (MaxPooling2D)	(None, 5, 283, 128)	0
conv2d_3 (Conv2D)	(None, 5, 283, 256)	98,560
conv2d_4 (Conv2D)	(None, 5, 283, 32)	24,608
max_pooling2d_3 (MaxPooling2D)	(None, 5, 141, 32)	0
batch_normalization_2 (BatchNormalization)	(None, 5, 141, 32)	128
conv2d_5 (Conv2D)	(None, 5, 141, 64)	6,208
max_pooling2d_4 (MaxPooling2D)	(None, 5, 70, 64)	0
conv2d_6 (Conv2D)	(None, 5, 70, 128)	24,704
max_pooling2d_5 (MaxPooling2D)	(None, 5, 35, 128)	0
conv2d_7 (Conv2D)	(None, 5, 35, 256)	98,560
max_pooling2d_6 (MaxPooling2D)	(None, 5, 17, 256)	0
flatten_1 (Flatten)	(None, 21760)	0
dense_3 (Dense)	(None, 700)	15,232,700
dropout_2 (Dropout)	(None, 700)	0
dense_4 (Dense)	(None, 283)	198,383

Total params: 15,715,019 (59.95 MB)
Trainable params: 15,714,891 (59.95 MB)
Non-trainable params: 128 (512.00 B)

Postprocessing and Visualisation

We obtain uncertainties on the predictions by computing a MCDropout.

```
nb_dropout = 5

def NN_uncertainity(model, x_test, targets_abs_max, T=5):
    predictions = []
    for _ in range(T):
        pred_norm = model.predict([x_test],verbose=0)
        pred = targets_norm_back(pred_norm, targets_abs_max)
        predictions += [pred]
    mean, std = np.mean(np.array(predictions), axis=0), np.std(np.array(predictions), axis=0)
    return mean, std


if do_the_mcdropout :
    spectre_valid_shift, spectre_valid_shift_std = NN_uncertainity(model, [valid_obs_norm], targets_abs_max, T = nb_dropout)

else :

    pred_valid_norm = model.predict([valid_obs_norm])
    pred_valid = targets_norm_back(pred_valid_norm, targets_abs_max)
    spectre_valid_shift = pred_valid
    spectre_valid_shift_std = spectre_valid_shift*0.1
```

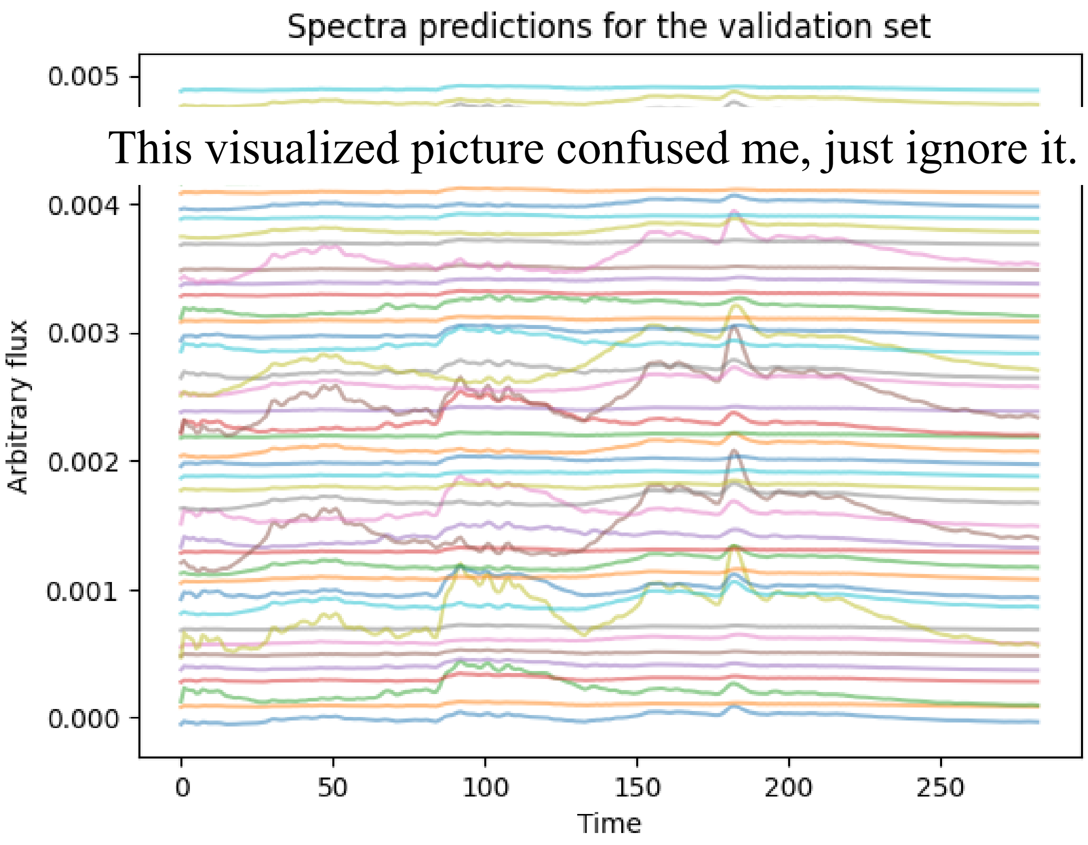
Postprocessing and Visualisation

```
residuals = valid_targets_shift - spectre_valid_shift
print('MSE : ', np.sqrt((residuals**2).mean())*1e6, 'ppm')
```

```
MSE :  33.371626933191784 ppm
```

```
# np.save(f'{output_dir}/pred_valid_shift.npy', spectre_valid_shift)
# np.save(f'{output_dir}/targ_valid_shift.npy', valid_targets_shift)
# np.save(f'{output_dir}/std_valid_shift.npy', spectre_valid_shift_std)
```

```
plt.figure()
for i in range (50) :
    plt.plot(spectre_valid_shift[-i]+0.0001*i, '-', alpha = 0.5)
plt.title('Spectra predictions for the validation set')
plt.xlabel('Time')
plt.ylabel('Arbitrary flux')
plt.show()
```



This visualized picture confused me, just ignore it.

Postprocessing and Visualisation

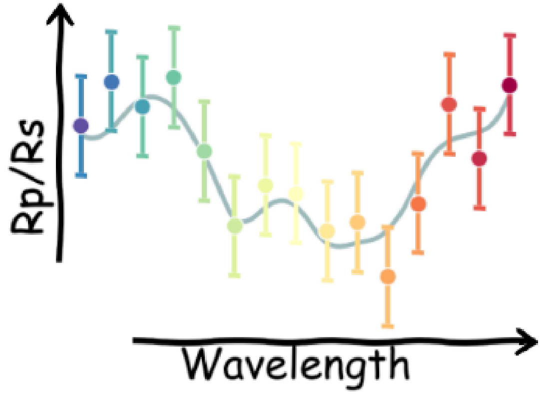
The value in this plot is not a standard $\left(\frac{r_p}{r_s}\right)^2$, we need norm back and add mean later.

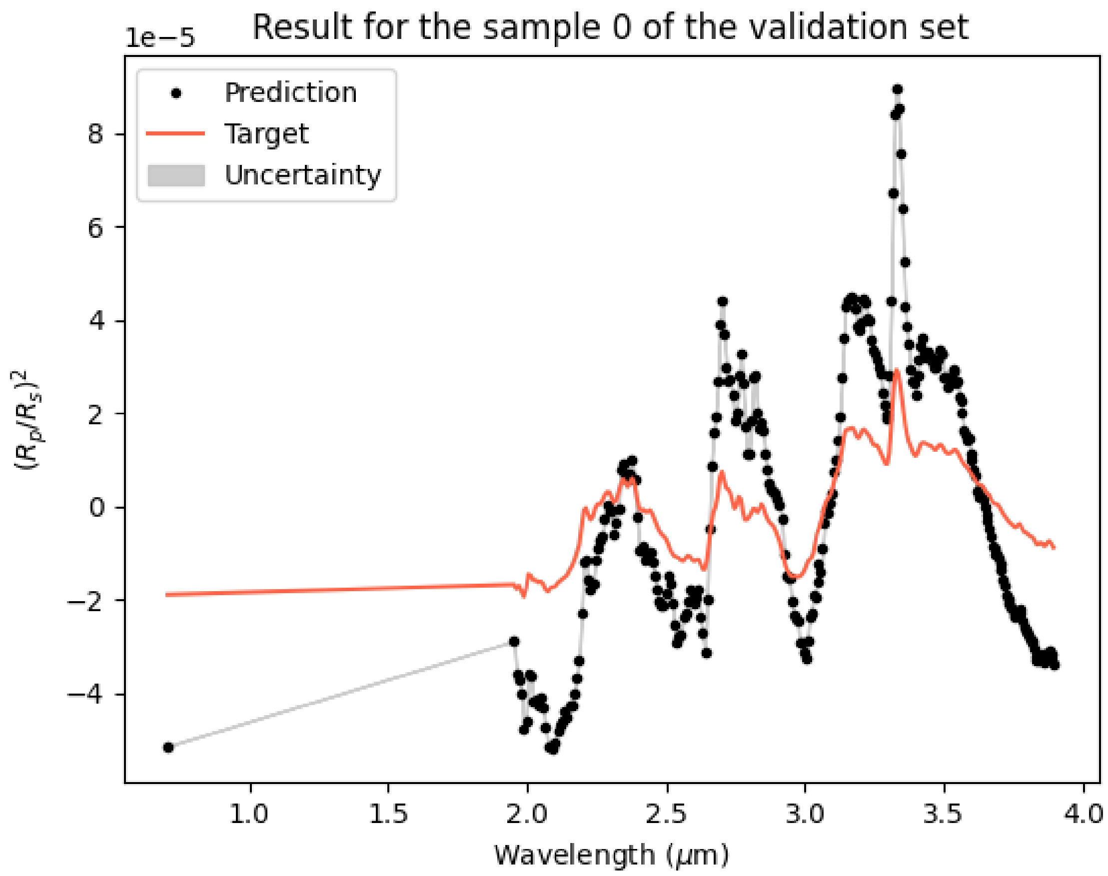
```
list_valid_planets = [0, 12, 35, 60, 70]
wavelength = np.loadtxt('/kaggle/input/ariel-data-challenge-2024/wavelengths.csv', skiprows=1, delimiter = ',')
uncertainty = spectre_valid_shift_std
for i in (list_valid_planets):
    plt.figure()
    plt.title('Result for the sample {} of the validation set'.format(i))
    plt.plot(wavelength, spectre_valid_shift[i], '.k', label = 'Prediction')
    plt.plot(wavelength, valid_targets_shift[i], color = 'tomato', label = 'Target')
    plt.fill_between(wavelength,
                     spectre_valid_shift[i] - spectre_valid_shift_std[i],
                     spectre_valid_shift[i] + spectre_valid_shift_std[i],
                     color='silver', alpha = 0.8, label = 'Uncertainty')
    plt.legend()
    plt.ylabel(f'$(R_p/R_s)^2$')
    plt.xlabel(f'Wavelength ($\mu$m)')
    plt.show()
```

→ np.arrange(len(wavelength))

Spectral Domain

Uncertainty should be displayed as box style like this example.



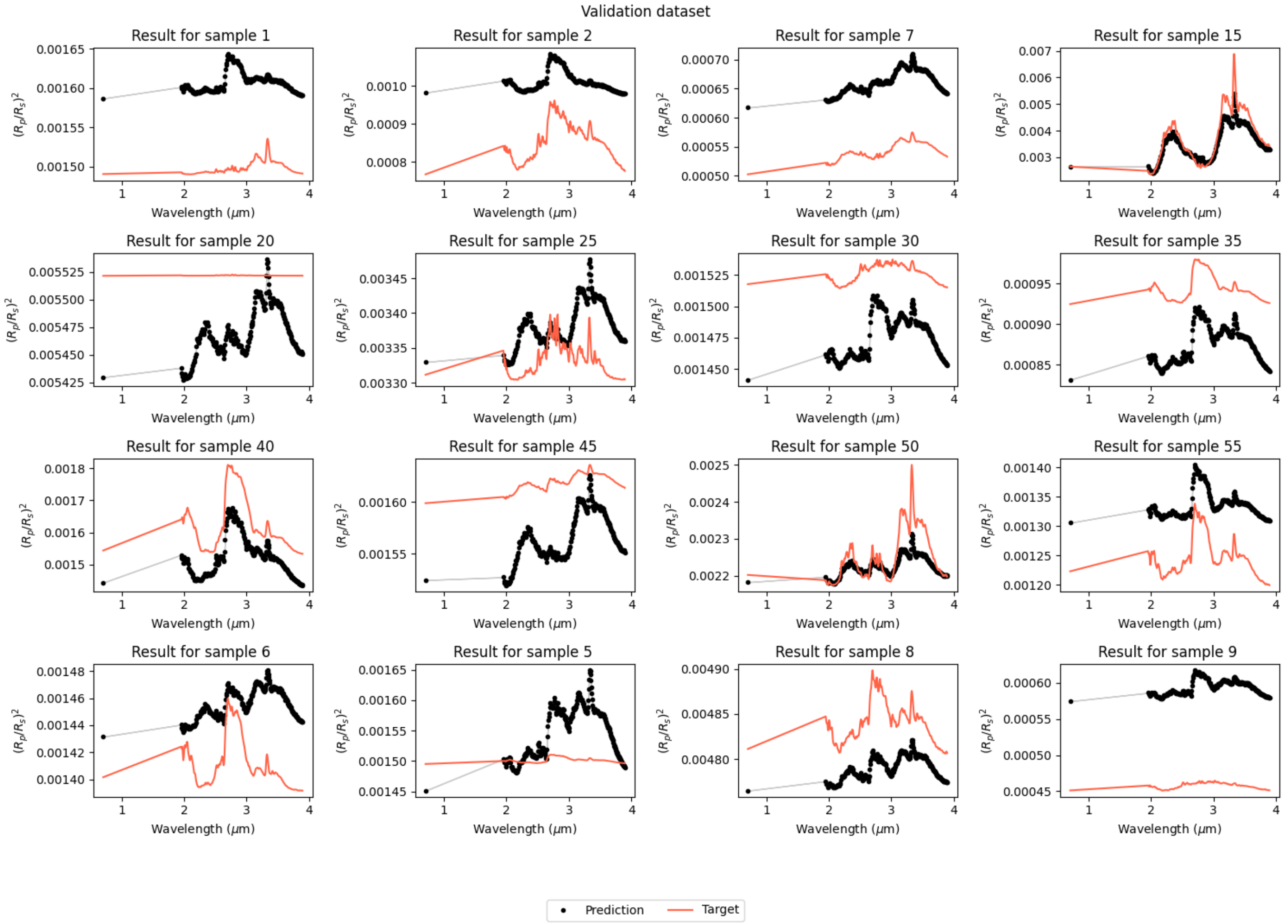


Results

```
######## ADD THE FLUCTUATIONS TO THE MEAN ########
def add_the_mean (shift, mean) :
    return shift + mean[:,np.newaxis]

predictions_valid = add_the_mean(spectre_valid_shift,spectre_valid_wc)
predictions_std_valid = np.sqrt(spectre_valid_std_wc[:,np.newaxis]**2 + spectre_valid_shift_std**2)
```

```
uncertainty = predictions_std_valid

def plot_one_sample_valid(ax, p):
    ax.set_title(f'Result for sample {p} ')
    line1, = ax.plot(wavelength, predictions_valid[p], '.k', label='Prediction')
    line2, = ax.plot(wavelength, valid_targets[p], color='tomato', label='Target')
    ax.fill_between(wavelength,
                    predictions_valid[p, :] - uncertainty[p],
                    predictions_valid[p, :] + uncertainty[p],
                    color='silver', alpha=0.8, label='Uncertainty')
    ax.set_ylabel(f'$(R_p/R_s)^2$')
    ax.set_xlabel(f'Wavelength ($\mu$m)')
    return line1, line2


num_samples = 16
rows, cols = 4, 4

fig, axs = plt.subplots(rows, cols, figsize=(15, 10))
samples = [1, 2, 7, 15, 20, 25, 30, 35, 40, 45, 50, 55, 6, 5, 8, 9]
lines = []

for i, ax in enumerate(axs.flat):
    lines.extend(plot_one_sample_valid(ax, samples[i]))

fig.legend(lines[:2], ['Prediction', 'Target'],
          loc='upper center', ncol=3, bbox_to_anchor=(0.5, -0.05))
fig.suptitle('Validation dataset')
plt.tight_layout()
plt.show()
```

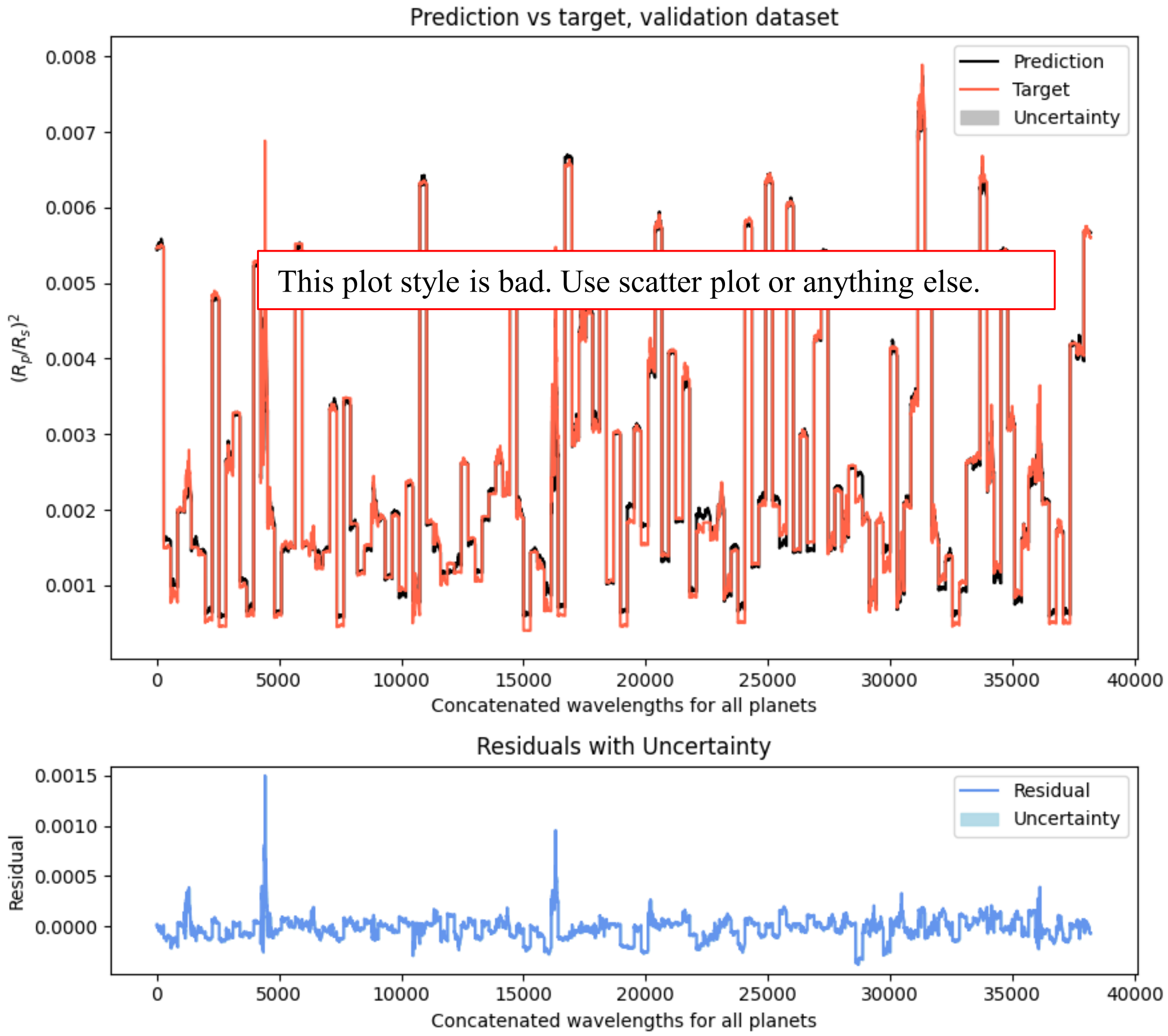
wavelength → np.arrange(len(wavelength))



Results

```python
######## PLOTS THE RESULT ########
predictions = predictions_valid
targets_plot = valid_targets
std = predictions_std_valid

predictions_concatenated_plot = np.concatenate(predictions, axis=0)
wls_concatenated = np.arange(predictions_concatenated_plot.shape[0])
targets_concatenated_plot = np.concatenate(targets_plot, axis=0)
spectre_valid_std_concatenated = np.concatenate(std, axis=0)
residuals = targets_concatenated_plot - predictions_concatenated_plot
uncertainty = spectre_valid_std_concatenated

fig, axs = plt.subplots(2, 1, figsize=(9, 8), gridspec_kw={'height_ratios': [3, 1]})


axs[0].plot(wls_concatenated, predictions_concatenated_plot, '-', color='k', label="Prediction")
axs[0].plot(wls_concatenated, targets_concatenated_plot, '-', color='tomato', label="Target")
axs[0].fill_between(np.arange(len(wls_concatenated)),
                    predictions_concatenated_plot - uncertainty,
                    predictions_concatenated_plot + uncertainty,
                    color='silver', alpha=1, label='Uncertainty')
axs[0].set_xlabel('Concatenated wavelengths for all planets')
axs[0].set_ylabel(f'$(R_p/R_s)^2$')
axs[0].set_title('Prediction vs target, validation dataset')
axs[0].legend()

axs[1].plot(wls_concatenated, residuals, '-', color='cornflowerblue', label="Residual")
axs[1].fill_between(np.arange(len(wls_concatenated)),
                    residuals - uncertainty,
                    residuals + uncertainty,
                    color='lightblue', alpha=0.9, label='Uncertainty')
axs[1].set_xlabel('Concatenated wavelengths for all planets')
axs[1].set_ylabel('Residual')
axs[1].set_title('Residuals with Uncertainty')
axs[1].legend()

plt.tight_layout()
plt.show()

print('MSE : ',np.sqrt((residuals**2).mean())*1e6, 'ppm')
```

```python
# np.save(f'{output_dir}/pred_valid.npy', predictions_valid)
# np.save(f'{output_dir}/std_valid.npy', predictions_std_valid)
```



MSE： 105.53292849547191 ppm

Thanks for listening!