SUPER GAN

Super Resolution using GANs

Team Members: Sayema Mashhadi ,Leo Tronchon, Amardeep Singh, Yuri Shlyakhter

#challenge-super-resolution

Climate Crisis Hackathon 2021

Introduction

Why Resolve Weather Data at Higher Resolution?

- Super Resolution will allow for fast calculation of current weather data at high resolution - starting with just surface temperature in this challenge

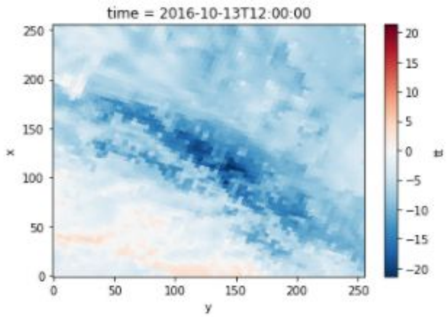
- Will allow to advance calculations further in time at higher resolution for weather and climate predictions

- Domains of weather and climate are very similar - same methods can be used in extending both precise and averaged climate scenarios to higher resolutions which will allow to do **local climate predictions**

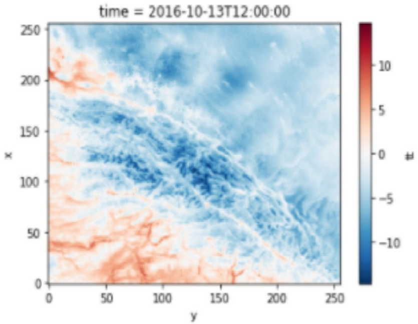
Challenge and Data

Task:
Statistical downscaling of climate data with inexpensive, fast and accurate methods making use of a wider region of climate data

Data:
Training data comprises of ~4300 data instances spanning 256 x 256 points with 15 different features



<u>Features are as follows:</u>
mg_delta: Difference between rdps and caldas for the variable MG (Water/land mask)
me_delta: Difference between rdps and caldas for the variable ME (Mean Elevation of Topography)
zp_delta: Difference between rdps and caldas for the variable ZP (Roughness length (CRESSMAN))
vg_delta: Difference between rdps and caldas for the variable VG (Dominant vegetation type)
td: Dew point temperature
tt: Air temperature
pn: Sea level pressure
nt: Total cloud cover
h: Height of boundary layer
rt: Total precipitation rate
i4: Water in the snow pack
5p: Fraction of grid covered by snow
i6: Albedo of snow
uu: U-component of the wind (along the X-axis of the grid
vv: V-component of the wind (along the Y-axis of the grid)



4 Approaches - Basic to Advanced

Feature Engineering:

Exploratory data analysis to evaluate features and generate new features

Physics and Earth Sciences:

Baseline model using temperature linear dependence on altitude wherever altitude changes

Traditional Machine Learning:

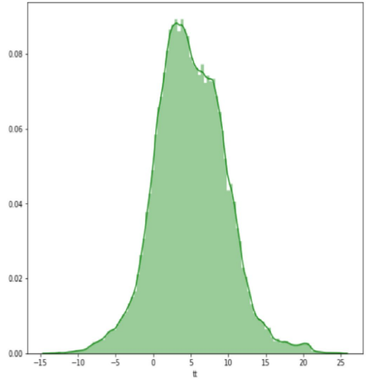
Linear Regression and Tree Based ML models with selected features

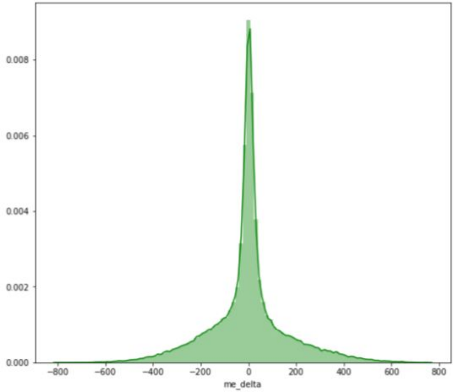
Deep Learning:

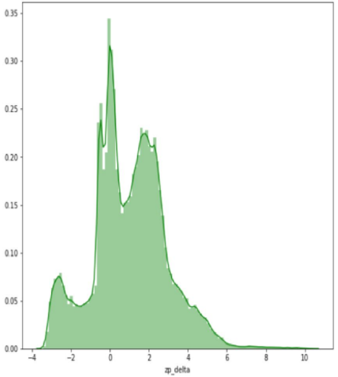
Pre-trained SRGAN, fine tuned with provided climate data

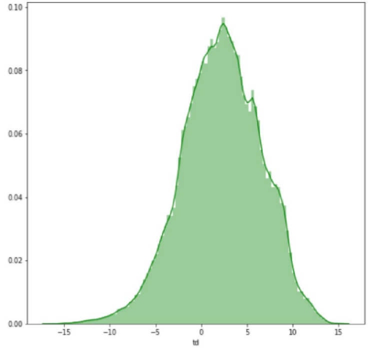
Data Analysis

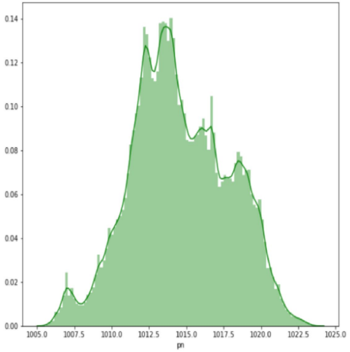
Distribution of data

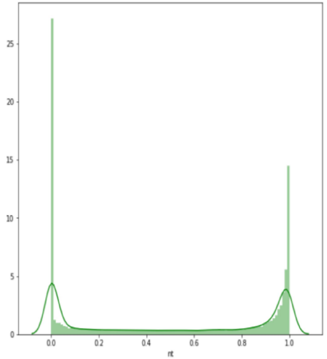


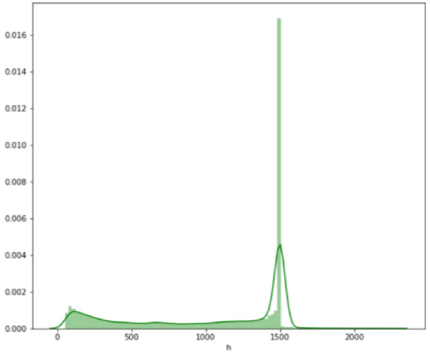


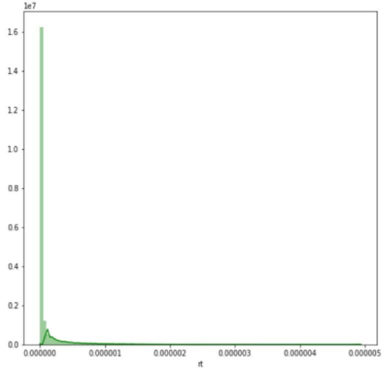




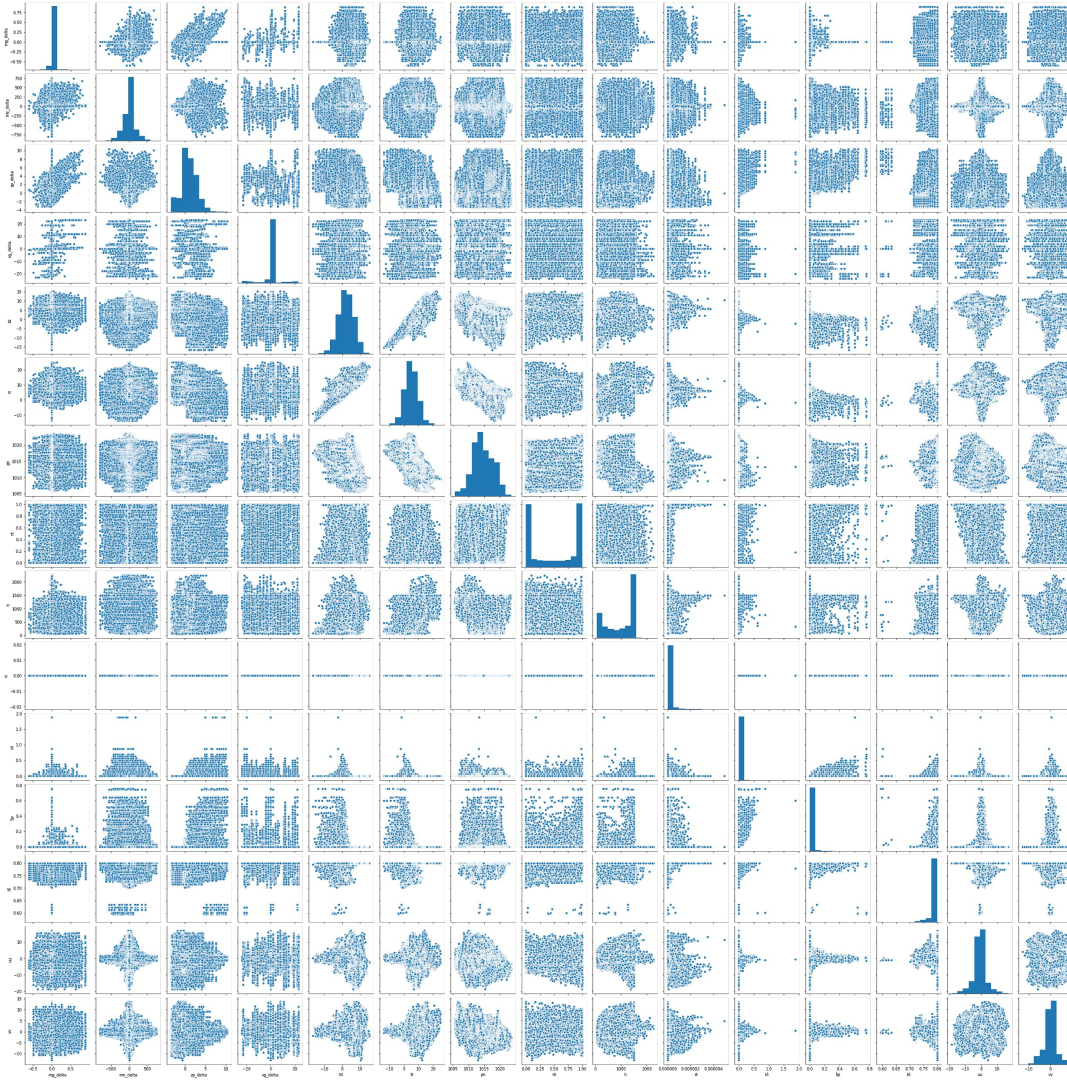




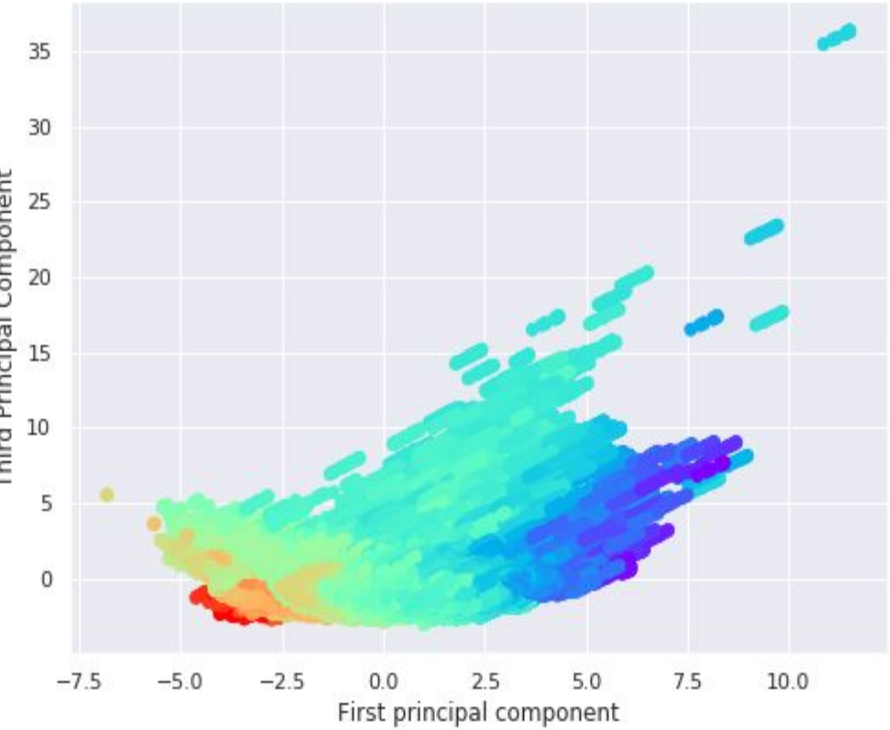


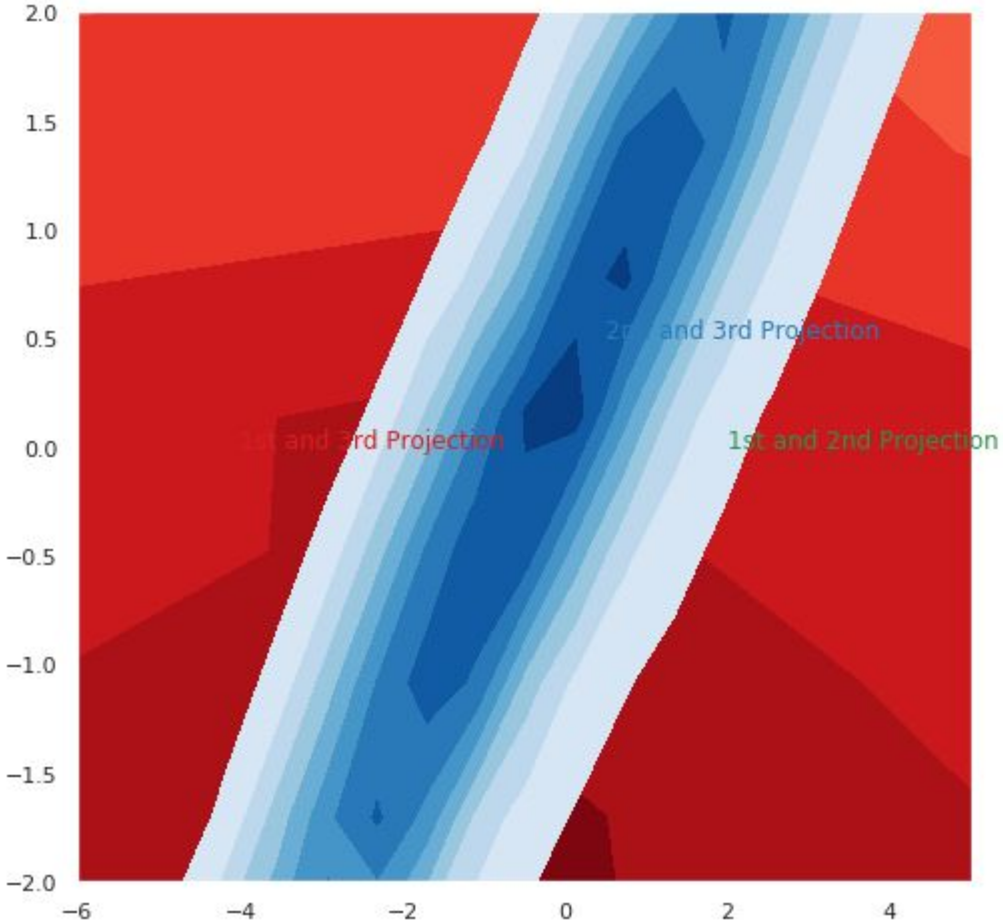


Feature correlation plots:



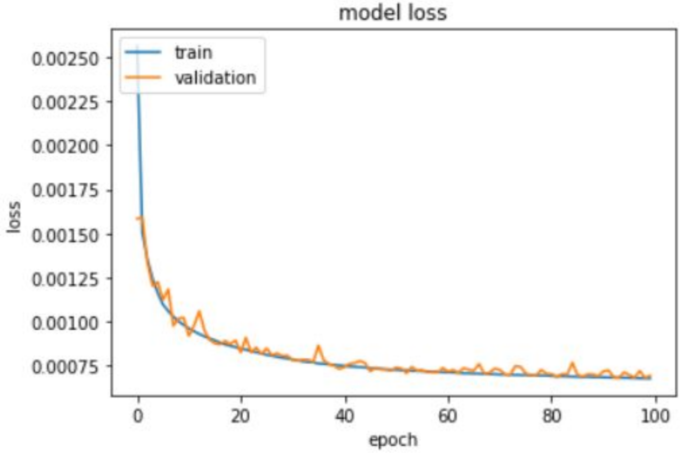
Principal Component Analysis

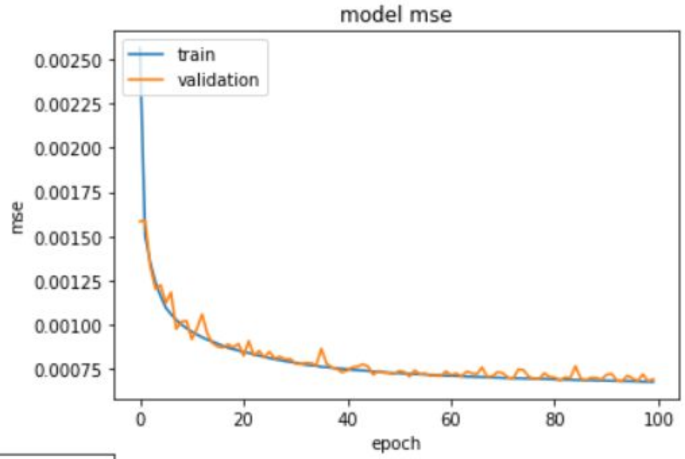


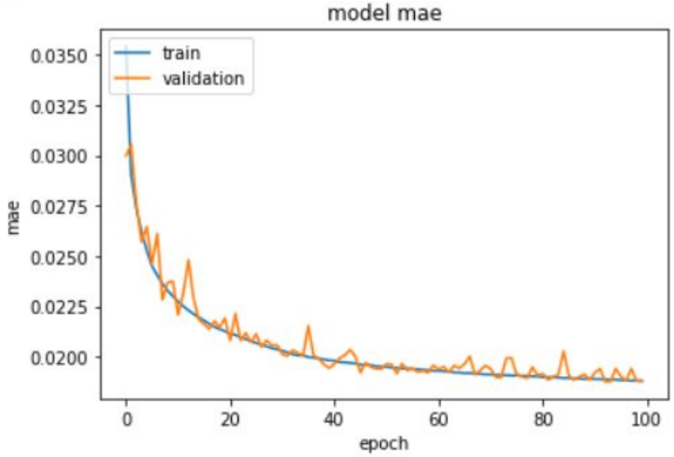


Model performance

LSTM Model Performance








```
model.evaluate(X_test, y_test)
```

```
10240/10240 [==============================] - 10s 958us/step - loss: 6.9614e-04 - mse: 6.9614e-04 - mae: 0.0189

[0.0006961408653296530, 0.0006961408653296530, 0.018852591514587402]
```

Random forest model performance:

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.35687567652141927
Mean Squared Error: 0.3186079601156035
Root Mean Squared Error: 0.564453682878944
```

```
for depth in range (1,10):
    tree_regressor=tree.DecisionTreeRegressor(max_depth=depth,random_state=1)
    if tree_regressor.fit(X,y).tree_.max_depth<depth:
        break
    score=np.mean(cross_val_score(tree_regressor,X,y,scoring='neg_mean_squared_error', cv=10,n_jobs=1))
    print(depth, score)
```

```
1 -9.780439412970836
2 -5.963767675701363
3 -4.774158928268536
4 -4.442155006197374
5 -4.094581830923138
6 -3.7695465469683826
7 -3.7735580122522108
8 -3.776314958861915
9 -3.7690939010429303
```

```
ElasticNetCV(alphas=None, copy_X=True, cv=10, eps=0.001, fit_intercept=True,
             l1_ratio=0.5, max_iter=1000, n_alphas=100, n_jobs=None,
             normalize=False, positive=False, precompute='auto', random_state=0,
             selection='cyclic', tol=0.0001, verbose=0)
```

```
regr16.fit(X_test, y_test)
regr16.score(X_test,y_test)
```

```
/home/ec2-user/anaconda3/envs/cha
versionWarning: A column-vector y
xample using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.8449874104691709
```

```
from sklearn.linear_model import RANSACRegressor
from sklearn.datasets import make_regression
X, y = make_regression(
    n_samples=200, n_features=2, noise=4.0, random_state=0)
reg20 = RANSACRegressor(random_state=0).fit(X, y)
reg20.score(X, y)

reg20.predict(X[:1,])
```

```
array([-31.94170869])
```

```
reg20.fit(X_test, y_test)
reg20.score(X_test,y_test)
```

```
0.8168837211953368
```

```
import numpy as np
from sklearn import linear_model
n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
clf = linear_model.SGDRegressor(max_iter=1000, tol=1e-3)
clf.fit(X_train, y_train)
```

```
/home/ec2-user/anaconda3/envs/chainer_p36/lib/python3.6/site-packages/sklearn/u
column-vector y was passed when a 1d array was expected. Please change the shap
().
  y = column_or_1d(y, warn=True)
```

```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

```
clf.fit(X_test, y_test)
clf.score(X_test,y_test)
```

```
/home/ec2-user/anaconda3/envs/chainer_p36/lib/python3.6/site-packages/sklearn/u
column-vector y was passed when a 1d array was expected. Please change the shap
().
  y = column_or_1d(y, warn=True)
```

```
0.839378145914544
```

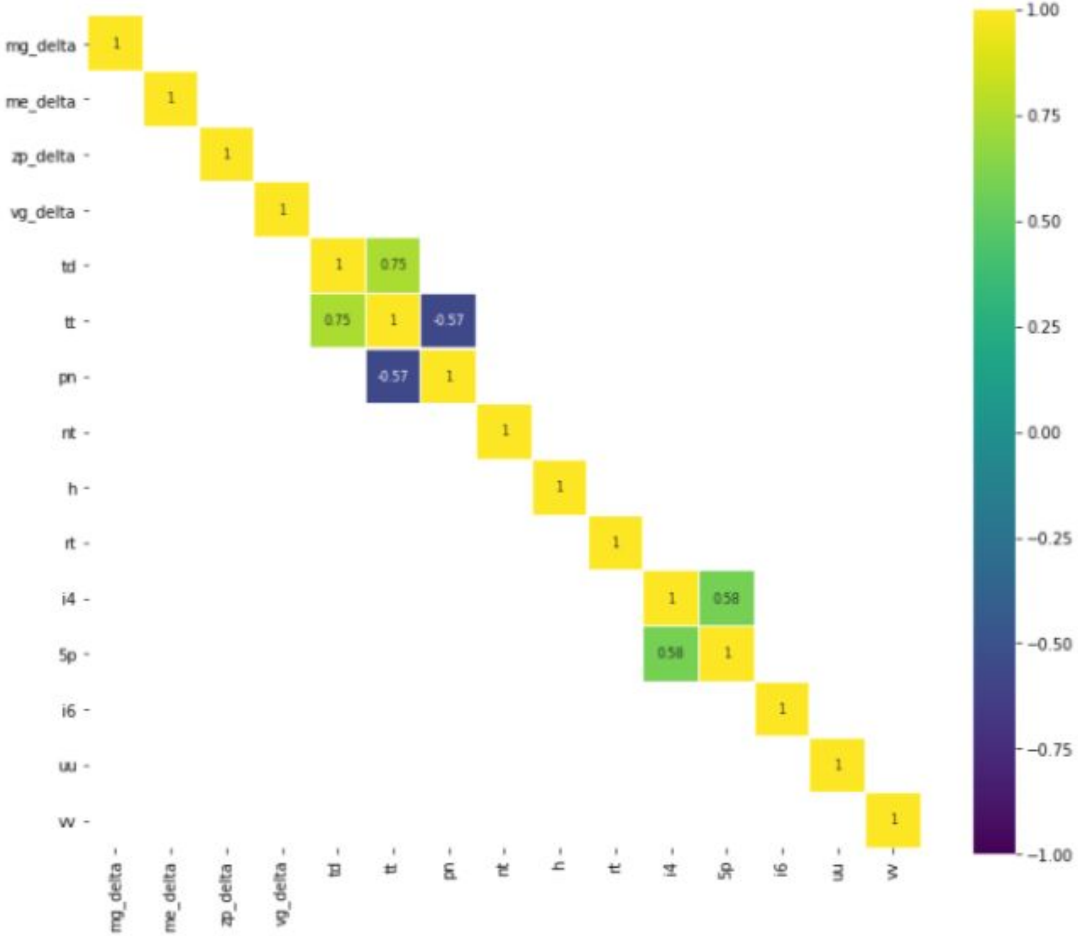
```
from sklearn.linear_model import OrthogonalMatchingPursuit
from sklearn.datasets import make_regression
X, y = make_regression(noise=10, random_state=5)
rego1 = OrthogonalMatchingPursuit().fit(X_train, y_train)
rego1.score(X_train, y_train)
```

```
0.8081635720267223
```

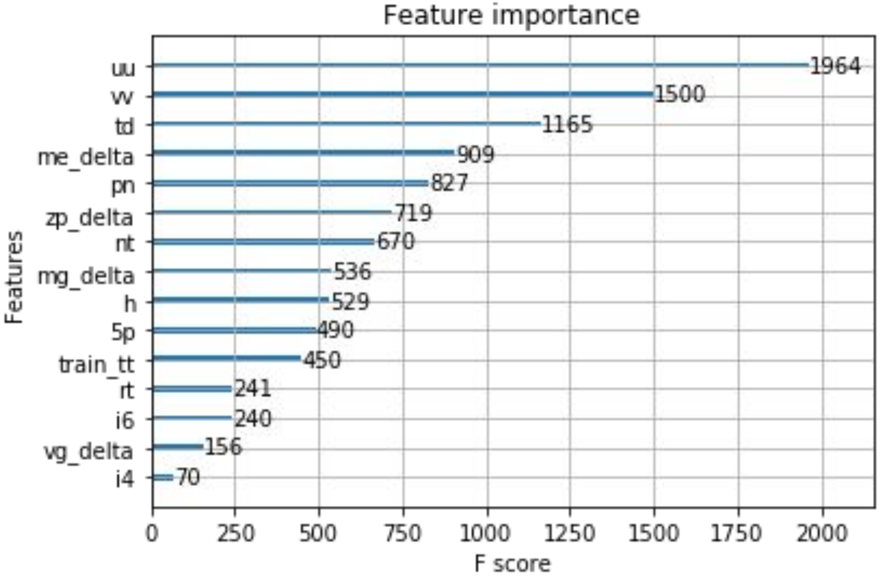
```
rego1.fit(X_test, y_test)
rego1.score(X_test,y_test)
```

```
0.8081147408891949
```

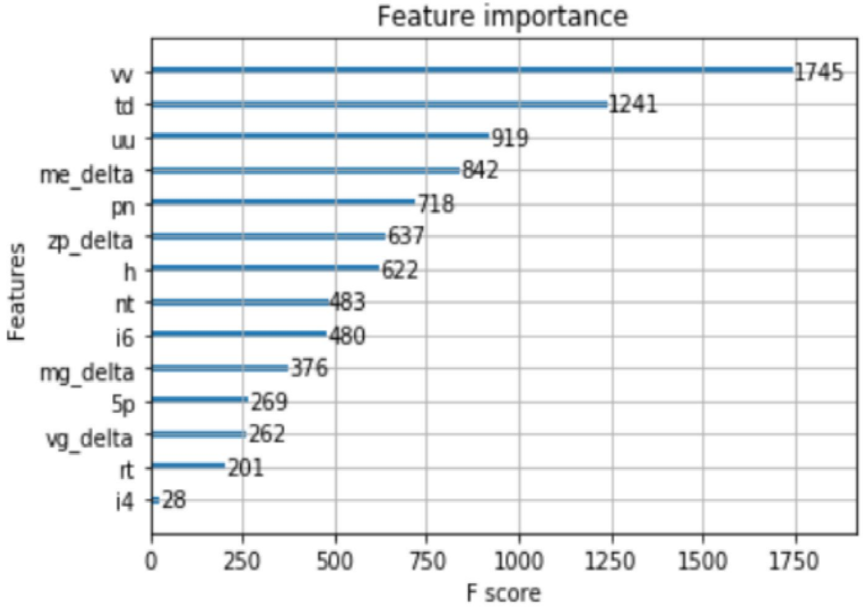
Pearson correlation matrix



Feature Importance

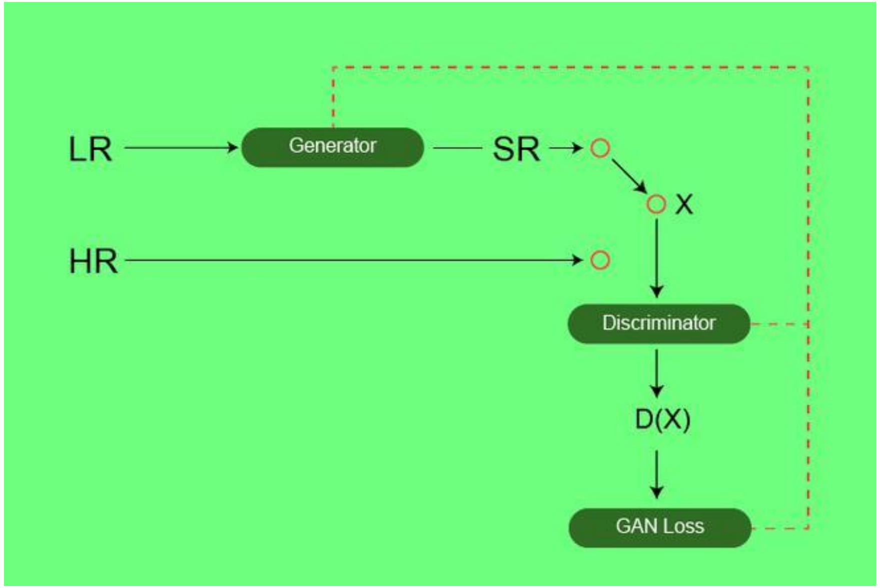


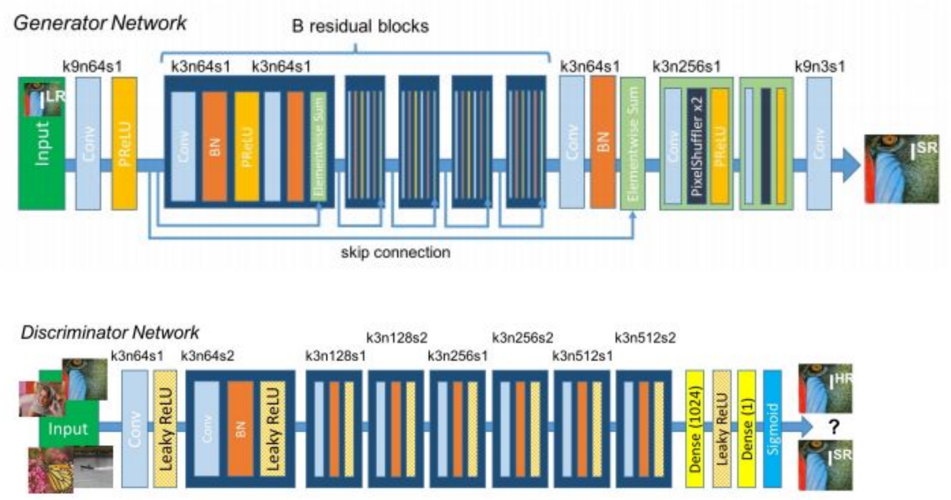
RMSE: 1.264600



RMSE: 0.268921

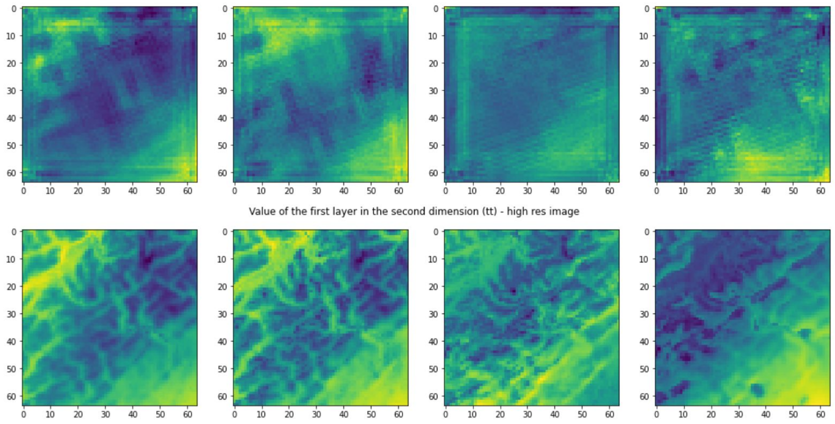
SRGAN


LR
Generator
SR
X
HR
Discriminator
D(X)
GAN Loss

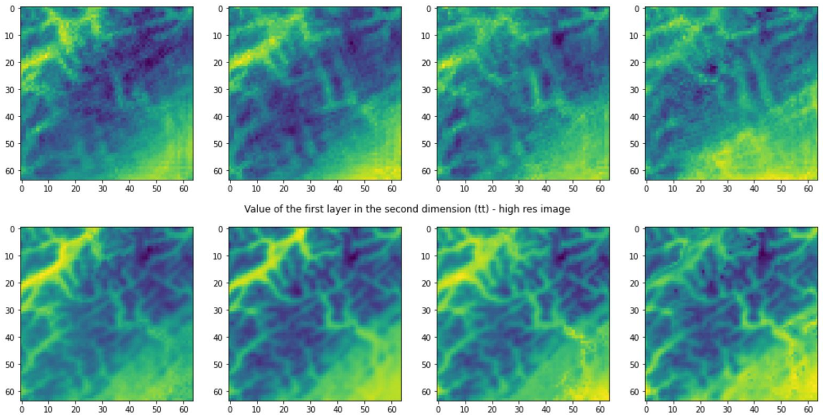

Generator Network
B residual blocks
k9n64s1
k3n64s1
k3n64s1
k3n64s1
k3n256s1
k9n3s1
Input
Conv
PReLU
BN
Elementwise Sum
PixelShuffler x2
skip connection
Discriminator Network
k3n64s1
k3n64s2
k3n128s1
k3n128s2
k3n256s1
k3n256s2
k3n512s1
k3n512s2
Input
Conv
Leaky ReLU
BN
Dense (1024)
Dense (1)
Sigmoid
?

SRGAN

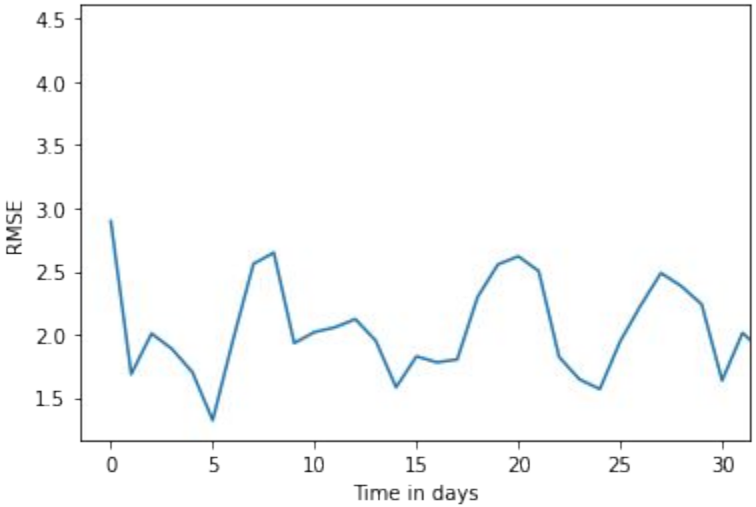
After 2 epochs:



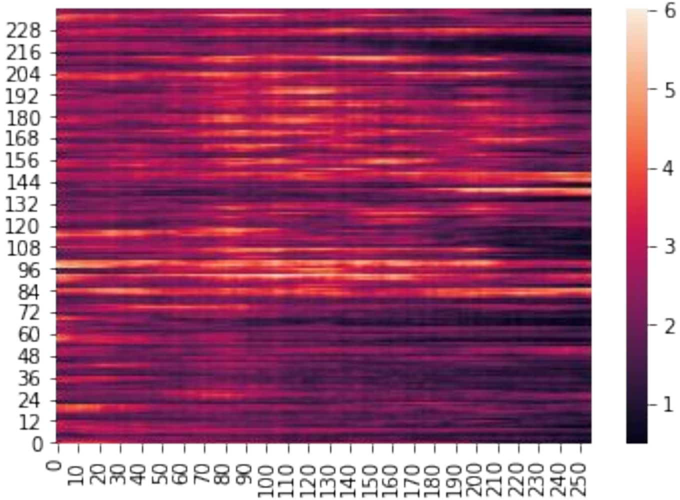
After 30 epochs:



Error distribution averaged in time and space (for baseline methods) - interesting finds



Seem to be too regular to be weather



Either error in method or bad data somewhere

Tools Used and Next Steps

Development Environment

- Colab notebooks for working with GPUs and TPUs
- GitHub for collaboration and sharing code from different platforms
- AWS
- Created smaller datasets to work on - both in xy (using 64 by 64 instead of 256) and in time (data increases linearly with time)

Model Selection and Next Steps

- Evaluated literature to choose current and simple-to-use methods
- Final model selection - SRGAN
- Other promising methods:
  - Next version - ESRGAN and variants
  - Time dependent solutions (LSTM, TegaGAN (from video domain))
  - GANs that do more that fill in data, that can take inputs can be more generalizable and can be applied to various location.
  - Transformers
- Directly compare to statistical downsampling
- Extend to other variables
- Apply to downscale climate scenarios