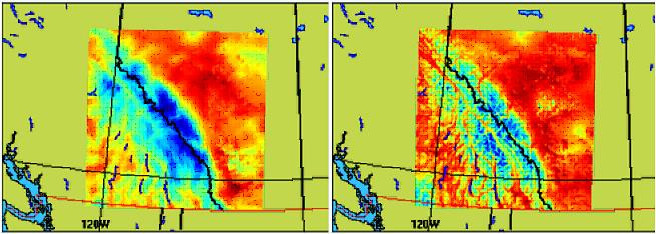
**Increasing Fine-Scale Temperature Details from Weather Model Forecasts**

**PROBLEM STATEMENT**

This challenge is to increase the resolution (the level of detail)of 2D surface temperature forecasts obtained from Environment and Climate Change Canada (ECCC) ’s weather forecast model, using as labelled images 2D temperature analysis at higher resolution. The scale factor between the model and the higher resolution analysis is 4 (from 10 km to 2.5 km). Numerous and relevant 2D low-resolution weather forecast fields are provided as predictors in the training set. In addition to temperature, these include fields like cloud coverage, wind, humidity, topography, etc. are also included in the training set files.



Here are examples of a temperature field over Western Canada at low resolution (10 km – left) and high resolution (2.5 km – right). The increase of resolution represents a factor of 4.

**SOLUTION:**

1. **Libraries Used:**
   1. I used Keras for Deep Learning and Theano as Backend
   2. Used DSSIMObjective from keras\_contrib.losses. Later converted DSSIM to SSIM for score calculation.

(<https://github.com/keras-team/keras-contrib>)

* 1. General Libraries: numpy, pandas, matplotlib, pickle, os, time

1. **Preparing to Load Data:**
   1. Generated a list of names of input\_training data for training my Model. It includes 51 input\_training files from ‘input\_training\_0000\_0099.npy’ to ‘input\_training\_5000\_5099.npy’.
   2. Generated a list of names of input\_training data for validation of my model. It includes 3 input\_training files from ‘input\_training\_5100\_5199.npy’ to ‘input\_training\_5300\_5342.npy’.
   3. Generated a list of 51 names of label\_training data for target training of my model from ‘label\_training\_0000\_0099.npy’ to ‘label\_training\_5000\_5099.npy’.
   4. Generated a list of 3 label\_training data for target validation of my model from ‘label\_training\_5100\_5199.npy’ to ‘label\_training\_5300\_5342.npy’.
   5. Now I created a dictionary named ‘partition’ which stores all the above lists as partition['train\_input'], partition['val\_input'], partition['train\_label'], partition['val\_label'].
   6. Created fuction named load\_data which performs following work:
      1. np.load(): Loads the data
      2. np.transpose(): transposes the data for training in keras model
      3. standardize() : normalizes the data for better performance.
2. **Creating Score Functions**:

It includes the following functions:

* 1. meanDifferenceCalculation()
  2. minDifferenceCalculation()
  3. maxDifferenceCalculation()
  4. scoreCalculation():
     1. Creates a DSSIMObjective object with kernel\_size=4
     2. Calculates DSSIM loss for images.
     3. Then calculates SSIM from DSSIM using formula:

ssim = 1 - ( 2 \* dssim ).

* + 1. Calculates Mean Squared Error
    2. Calculates meanDiff, minDiff and maxDiff (a. b. and c.)
    3. Adds up all the scores to gives final score.

1. **Convolution Neural Network**

Here is the Architecture for my ConvNet Model in keras.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) (None, 256, 256, 15) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1 (Conv2D) (None, 256, 256, 16) 2176

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bn\_conv1 (BatchNormalization (None, 256, 256, 16) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 256, 256, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2 (Conv2D) (None, 256, 256, 1) 145

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bn\_conv2 (BatchNormalization (None, 256, 256, 1) 4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_2 (Activation) (None, 256, 256, 1) 0

=================================================================

Total params: 2,389

Trainable params: 2,355

Non-trainable params: 34

**Optimization**: optimizer='adam' , loss='mean\_squared\_error', metrics=scoreCalculation (Function Built in Step 3. )

**Explanation for fitting the model:** As I was limited on computer resources I had to train each input file one by one. My system could not take the load of the memory of loading multiple files at once. Thus I was loading one file at a time, performing preprocessing and then fitting the file data in model.

Therefore I used a for loop for running 7 epochs and each epoch was training on 51 files and validating on 3 files.

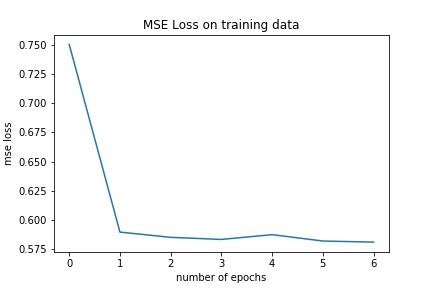
My validation split was of only 3 files because I wanted to mimic the train and test split size. As test data had around 248 entries thus I used 3 files for validation which has similar number of entries.

1. **Plotting train and validation (score and loss)**

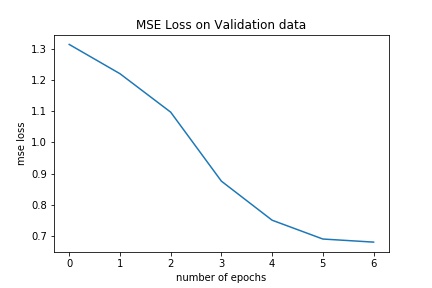
Before plotting the results I wrote some simple scripts to extract training Loss and Score from saved history of loss and score.

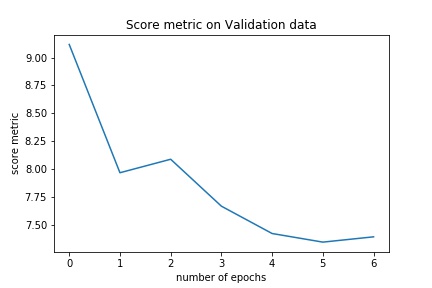
Then I Averaged up score/loss to give score/loss for 1 epoch as training score is for each file individually. So I averaged each batch of 51 scores to give score for 1 epoch.

Same was done for Validation also where number of files were 3.



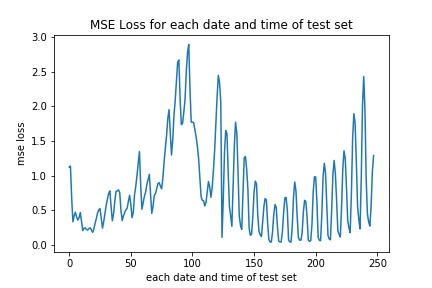


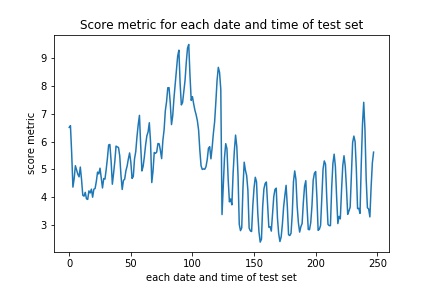




1. **Results with test data**

For evaluating each date and time entry individually, I used model.evaluate() for each record separately by reshaping each record to hold single entry/record ie. (1,256,256,15) and (1,256,256,1) and appended the loss and score to test\_history list.





Then I stored each date and time record’s result to pandas test\_results Dataframe having record for each date and time as follows.

|  | **DateTime** | **loss** | **score** |
| --- | --- | --- | --- |
| **0** | 2018011612 | 1.125401 | 6.509701 |
| **1** | 2018011615 | 1.139553 | 6.571133 |
| **2** | 2018011618 | 0.676640 | 5.577474 |
| **3** | 2018011621 | 0.336074 | 4.362339 |
| **4** | 2018011700 | 0.421846 | 4.661493 |

Using test\_resuts.describe() we can see that:

The average loss for test\_results is : **average\_loss=0.806500**

The average score for test\_results is: **average\_score=5.001534**

Results are stored in test\_results.csv files which can be reproduced from given code.

|  | **loss** | **score** |
| --- | --- | --- |
| **count** | 248.000000 | 248.000000 |
| **mean** | 0.806500 | 5.001534 |
| **std** | 0.626931 | 1.545484 |
| **min** | 0.043003 | 2.385495 |
| **25%** | 0.346850 | 3.940689 |
| **50%** | 0.648364 | 4.857776 |
| **75%** | 1.062439 | 5.815931 |
| **max** | 2.892654 | 9.489738 |

\*Improvements: As I had limited computer resources, therefore could not built a deep enough Network like ResNet which product state of the art results for Super Resolution Problems.