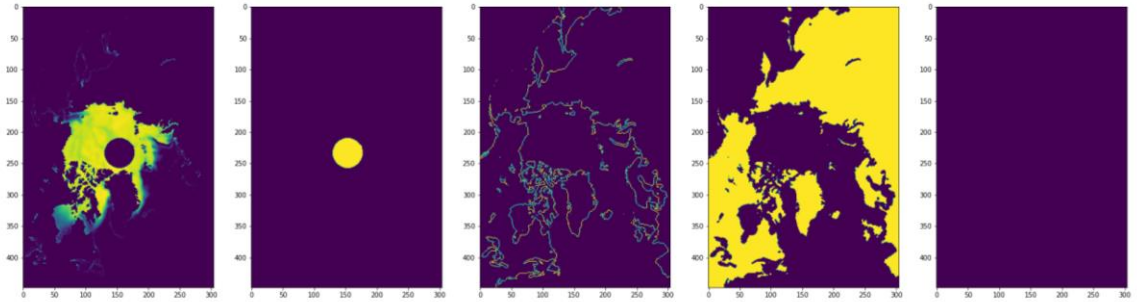
[INTRO.] As a result of global warming, various climate change signs had been detected including Earth's surface temperature change, irregular temperature, and rising sea levels. The ice concentration at the North Pole has also been decreasing.

[GOAL] Use the satellite images of the past North Pole's ice concentrations, and predict the ice concentration in the future

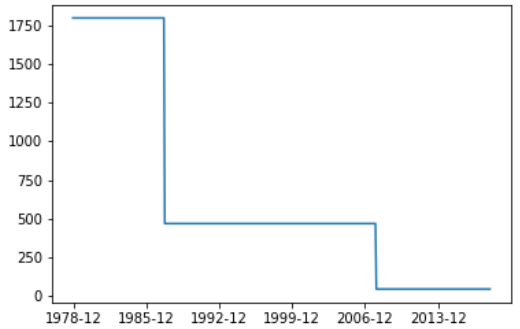
[DATA EDA]

Dataset Provided (image data of dim (448 x 304))

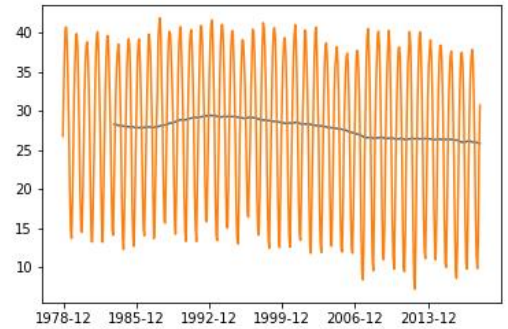


- 5 images(448 x 304) are provided for each month from Nov. 1978 to Dec. 2018
- Each data contains ice concentration(0~250), blind spot, shoreline, land border, and null value locations
- Since all the shoreline and land borderline images are the same, exclude from training

Area of the blind spot over time



Average Ice Concentration (Monthly vs 5 year moving average)



- The area of the blind spot has been decreasing thanks to technological improvements
- Monthly ice concentration values are cyclical, but 5 years moving average shows decreasing trend
- In order for the model to capture decreasing trend more easily, use only the past 15 years of data

[Approach]

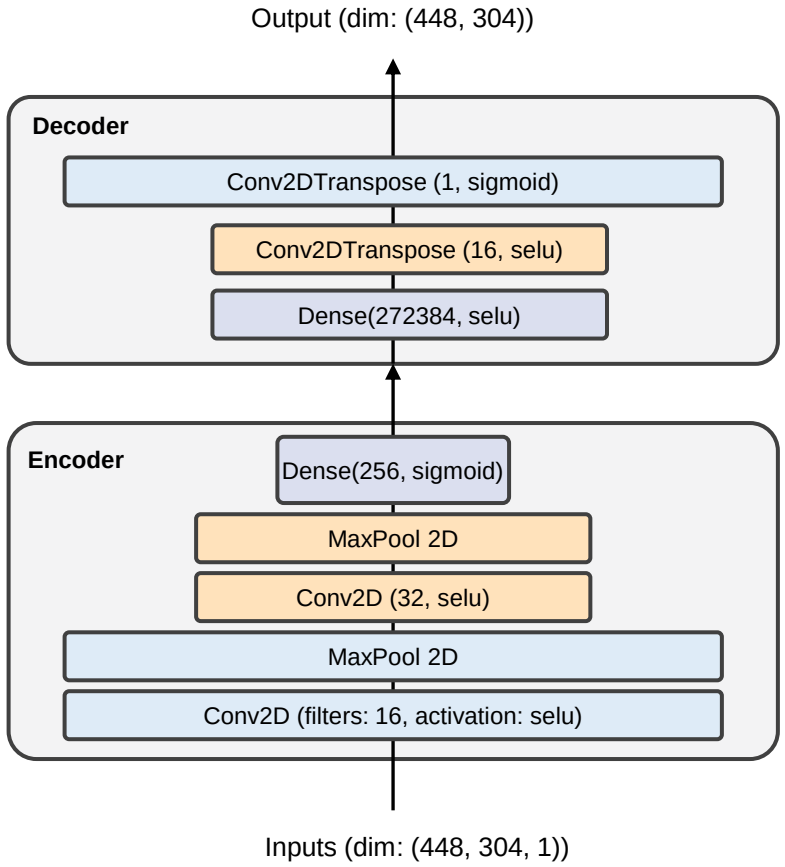
Since the image dimension is too big to be trained by itself, ways to train the model while decreasing the computing burden is needed

1. Perform dimension reduction using encoder layer of an Autoencoder, and use the encoded data to train and predict the sequential data
2. Crop the original image into smaller pieces and train/predict the model. Then re-combine the images to the original size after prediction

[Approach 1-1] Exploration of possible AutoEncoder models to perform dimension reduction

- First fill in the blind spot with the median value of that image's ice concentrations, and extract (256,) vector with an encoder
- Among multiple models utilizing Conv2D layers, two Conv2D+Maxpool layer with 16 & 32 filters worked the best
- Validation Loss = 0.0541 achieved with SGD optimizer (learning rate=0.1, momentum=0.9) after 700 epochs

[Best Model Architecture]



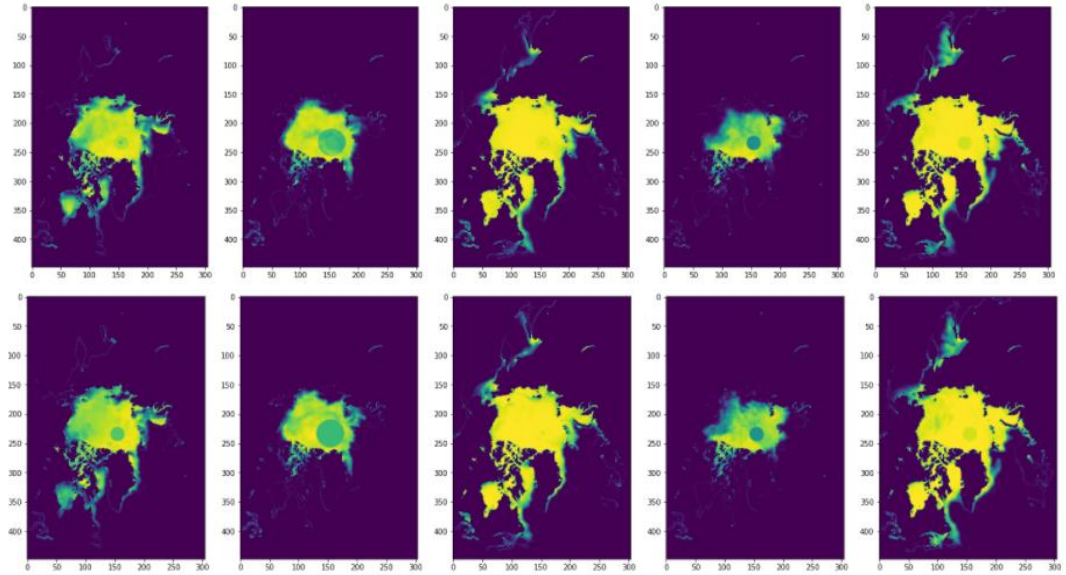
[Summary of Models]

Model (All Ran for 100 epochs)	Validation Loss
One Conv2D(filters: 16)+Maxpool layer	**0.0860**
Two Conv2D(filters: 16 & 32)+Maxpool layers	**0.0569**
Two Conv2D(filters: 16 & 32)+Maxpool layers with Dense layer weights synced in Encoder and Decoder	**0.0606**
Two Conv2D layers with smaller filters (filters: 8 & 16)	**0.0577**
Three Conv2D layers (filters: 16 & 32 & 64)	**0.0581**

[Final AutoEncoder images(top), Inputted Raw Images(Bottom)]

Output Image

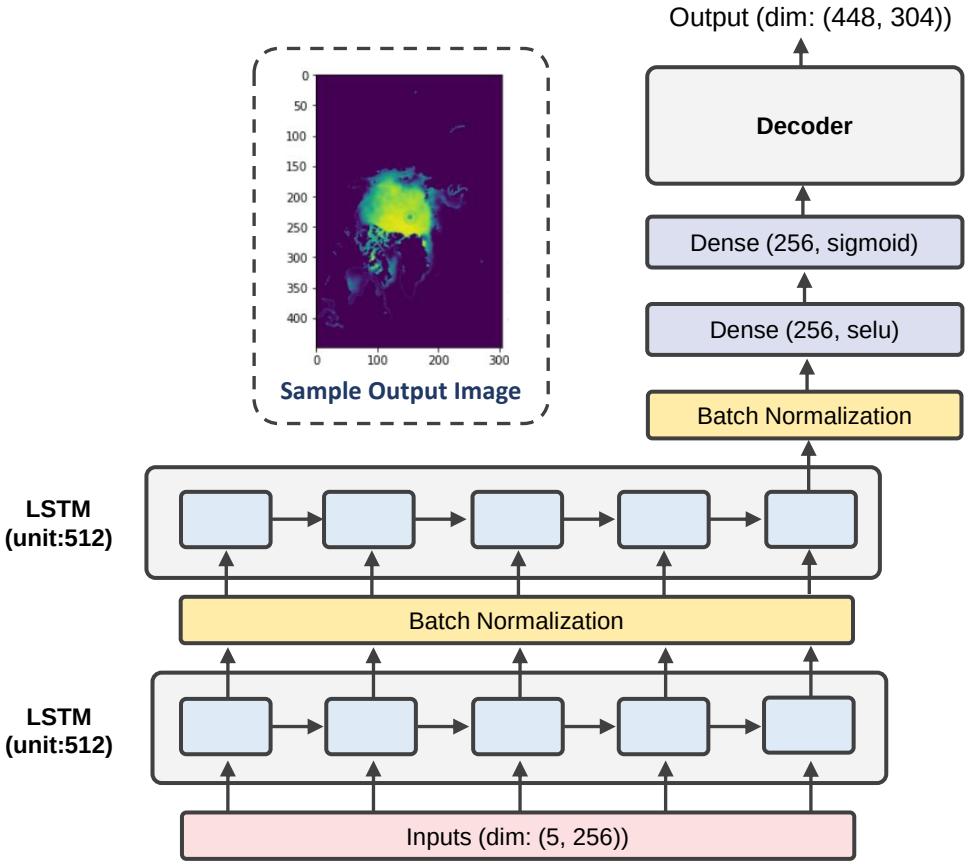
Input Image



[Approach 1-2] Prediction model for the encoded sequential data with LSTM

- **Input data** : 5 years of encoded data before the predict year; **Output Data** : 448 x 304 image after passing predicted encoded data through the decoder
- To track the loss more accurately, use decoder to output the predicted image of the original size to compare with the target
- Two LSTM layers with Batch Normalization works the best. Val Loss = 0.0600 with SGD(learning rate=0.001, momentum=0.999) after 400 epochs

[Best Model Architecture]



[Summary of Models]

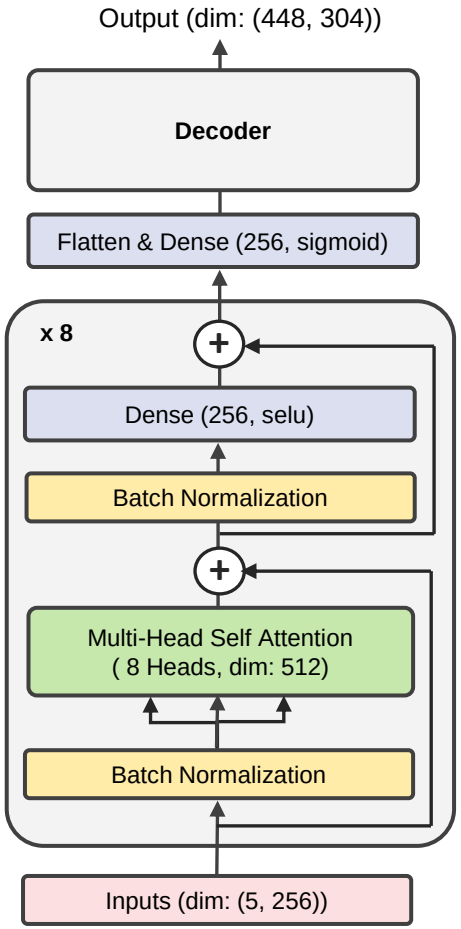
Model (All ran for 100 epochs)	Validation Loss
One LSTM layer (512 units)	**0.0611**
Two LSTM layers (512 units)	**0.0614**
Three LSTM layers (512 units)	**0.0626**
One LSTM layer with Batch Normalization (512 units)	**0.0620**
Two LSTM layer with Batch Normalization (512 units)	**0.0605**
Bigger LSTM unit with 1 layer (1024 units)	**0.0611**
Bigger LSTM unit with 2 layers (1024 units)	**0.0615**
Smaller LSTM unit with 1 layer (128 units)	**0.0616**
Conv2D before LSTM (2 filters, kernel size: 3, LSTM 512 units)	**0.0609**
Conv2D before 2 LSTM layers(2 filters, kernel size: 3, LSTM 512 units)	**0.0622**

Loss: Binary Crossentropy; optimizer : SGD

[Approach 1-2] Prediction of encoded sequential data with Multi-Head Self Attention

- **Input data** : 5 years of encoded data before the predict year; **Output Data** : 448 x 304 image after passing predicted encoded data through the decoder
- **Idea**: As Self-Attention model examines similarity and importance within K, Q, V, try out if the model works with encoded image data
- Referenced Transformer Model and Vision Transformer Model's encoder devised by Google Research Team
- Simple model utilizing BatchNorm, MultiHead, Dense layer with a few skip layers showed good performance (Val Loss=0.0611)

[Best Model Architecture]



[Summary of Models]

Model(Ran for 100 epochs)	Validation Loss
(BatchNorm + MultiHead + BatchNorm + Dense) x 4	**0.0635**
((BatchNorm + MultiHead + BatchNorm + Dense) + Skip Layer) x 8	**0.0611**

Loss: Binary Crossentropy; optimizer : Adam

[Referenced Papers]

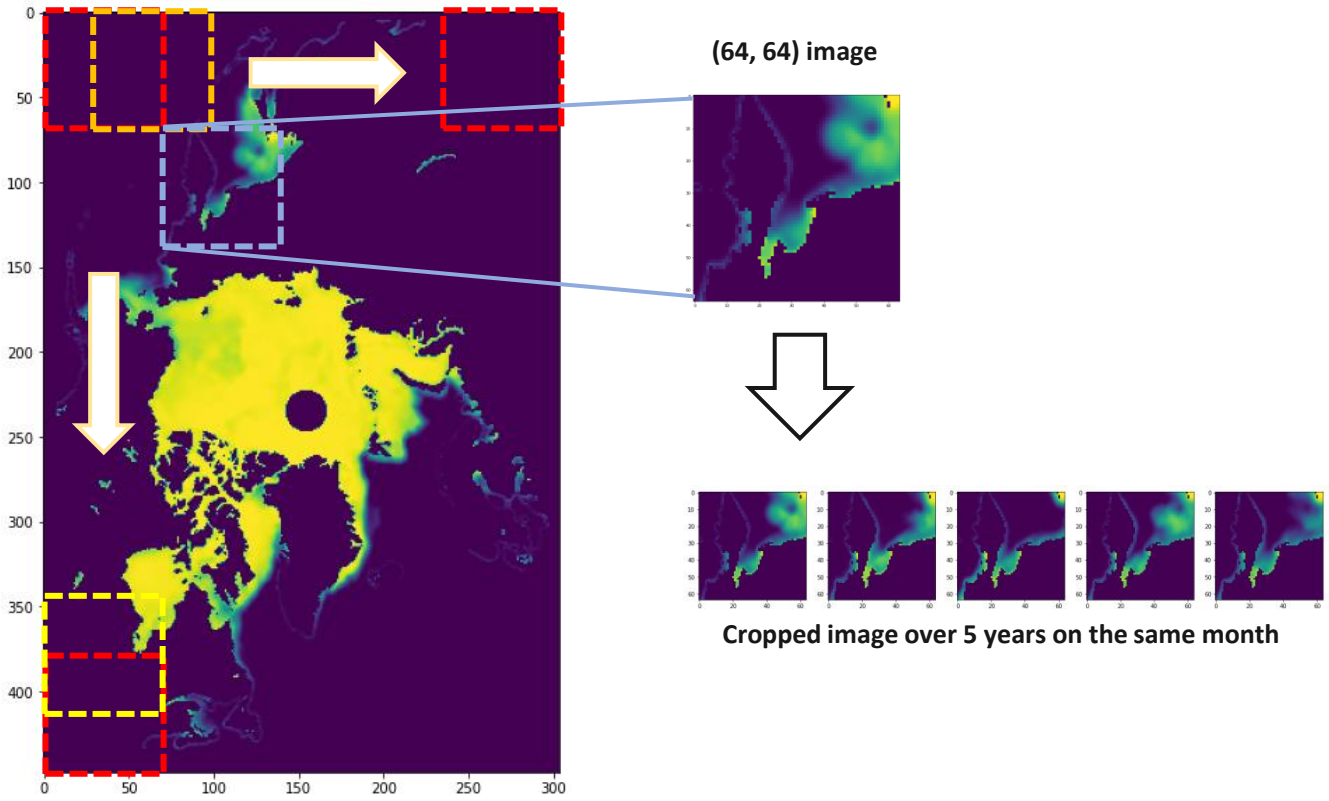
- An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020, Google Research Team; arXiv:2010.11929
- Attention Is All You Need, 2017, Google Research Team; arXiv:1706.03762

[Approach 2-1] Crop the original Image into smaller images and create image pad to use when re-combining the small images

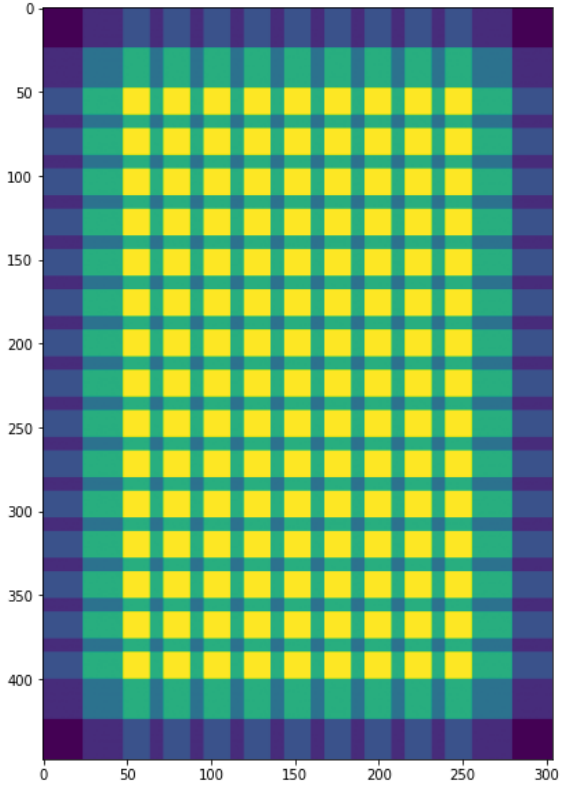
- Crop the original image into 64 x 64 image by moving 24 pixels to the right and below
- Re-group the predicted image by same month and use 5-year long data as input data
- Since there will be overlaps in the predicted images when combined, create image pad to average out the overlapped values
- Image pad moves by 24 pixels in the same way and add 1 to the corresponding pixels in the 64 x 64 image area. Use the pad to perform element-wise division

[Image Cropping Method]

Window size = 64, strides = 24



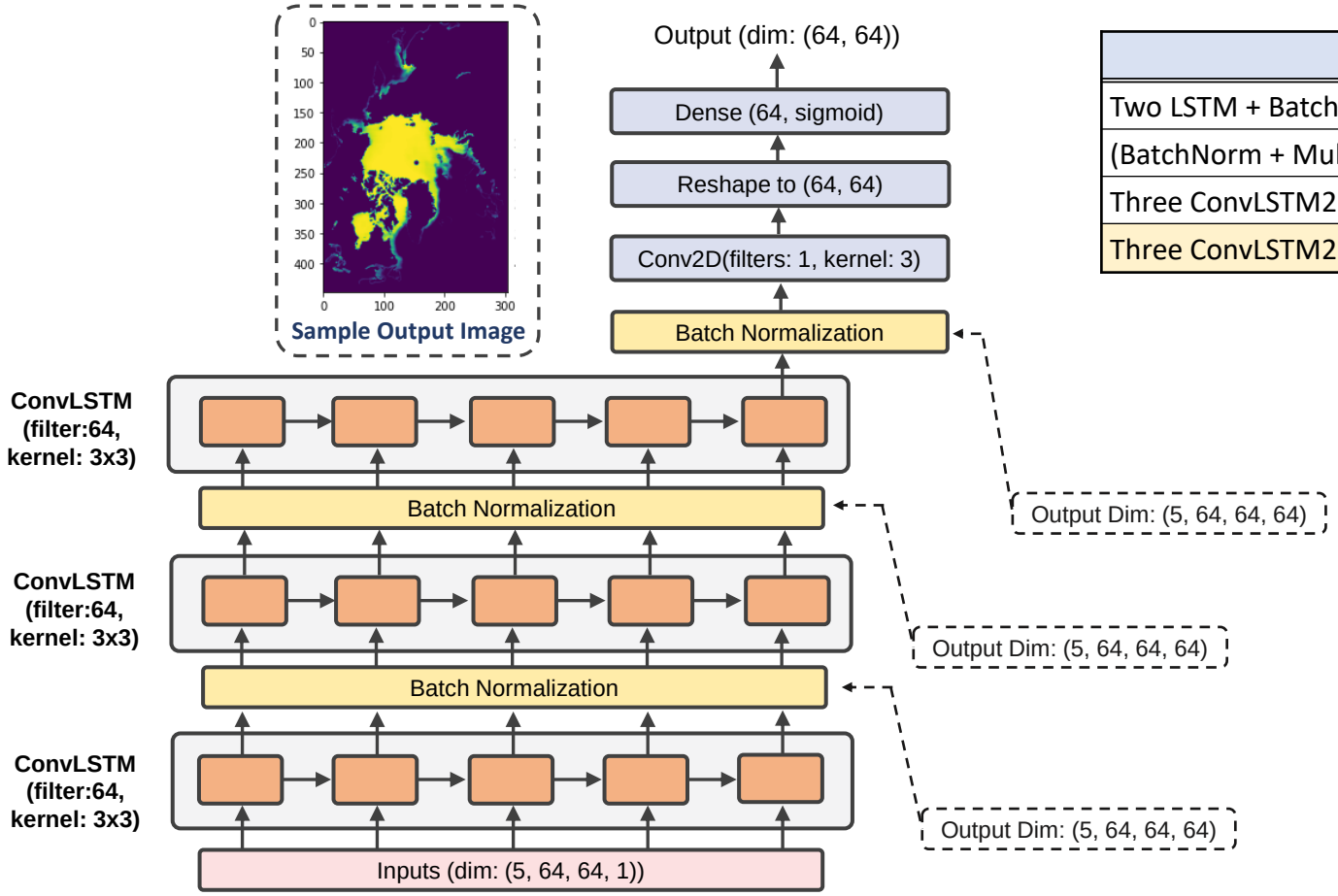
[Image pad to be used for re-combining smaller images]



[Approach 2-2] Prediction through the cropped Images and recombine

- **Input data** : 5 years of cropped images before the predict year; **Ouput Data** : cropped image of the predict year's image (64 x 64)
- Use two layers LSTM+BatchNorm model and Multi-Head Attention model which showed good performance in the previous experiments. Also use ConvLSTM model which handles sequential image data
- Three ConvLSTM+BatchNorm layers with sigmoid function at the last dense layer showed the best performance with MAE score of 2.95
- Since the model and training images (even after cropping) takes a lot of memory, so if model didn't not decrease loss, it was stopped during the training

[Best Model Architecture]



[Summary of Models]

Model	Validation Loss
Two LSTM + BatchNorm Layers	**18.3465 (6 epochs)**
(BatchNorm + MultiHead + BatchNorm + Dense) x 2 + Dense	**19.5847 (11 epochs)**
Three ConvLSTM2D + BatchNorm layers and Dense(64, relu)	**5.1563 (20 epochs)**
Three ConvLSTM2D + BatchNorm layers and Dense(64, sigmoid)	**≈2.9500 (20 epochs)***

Loss: Mean Absolute Error; optimizer : Adam
* Model input and output scaled to values between 0 and 1.
So, MAE * 250 is used to approximate. (original mae:0.0118)

[Final Evaluation] Predict with Test data using the trained models

- Use the trained models to predict test data and focus on the pixel value within 5~50 to calculate the final score
- For each model, the model with the least validation loss was selected as the best model
- Three models' finals scores were quite similar with 0.1~0.2 differences. Multi-Head Attention model had the best performance score

[Final Evaluation Metric]

```python
import numpy as np

def mae_score(true, pred):
    # calculate the regular MAE
    score = np.mean(np.abs(true-pred))

    return score

def f1_score(true, pred):
    # calculate using only the pixel value whtin 0~50 of the target image
    target = np.where((true>250*0.05)&(true<250*0.5))

    # divide pixel value with threshold value of 15
    true = true[target]
    pred = pred[target]
    true = np.where(true < 250*0.15, 0, 1)
    pred = np.where(pred < 250*0.15, 0, 1)

    # calculate precision and recall
    right = np.sum(true * pred == 1)
    precision = right / np.sum(true+1e-8)
    recall = right / np.sum(pred+1e-8)
    score = 2 * precision*recall/(precision+recall+1e-8)

    return score

def mae_over_f1(true, pred):
    # divide mae by f1
    mae = mae_score(true, pred)
    f1 = f1_score(true, pred)
    score = mae/(f1+1e-8)

    return score
```

[Summary of Models]

Model		Final Score
With Autoencoder	Two Conv2D(filters: 16)+Maxpool layers	**4.6961 (700 epochs)**
With Autoencoder	((BatchNorm + MultiHead + BatchNorm + Dense) + Skip Layer) x 8	**4.4870 (200 epochs)**
With Cropped images	Three ConvLSTM2D + BatchNorm layers and Dense(64, sigmoid)	**4.5689 (40 epochs)**

[Predicted images from Jan. to Dec.]