CLIMATE HACK.AI

2022

CONTENT

TASK

INPUT

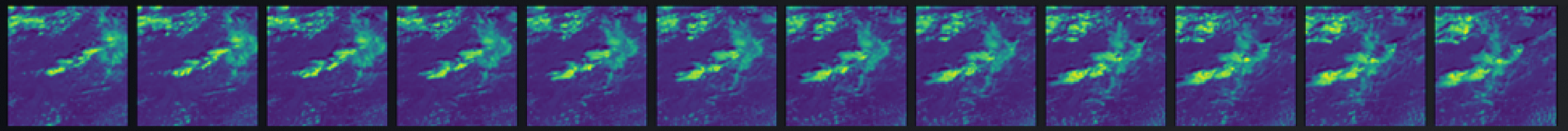
12x128x128
images

Preprocessing

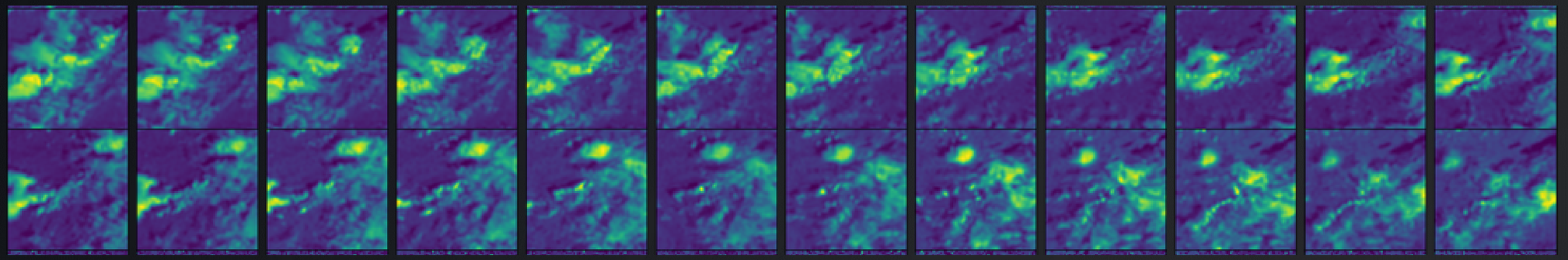
MODEL

OUTPUT

24x64x64
images



MODEL



MODELS

Optical Flow

0.50-0.55

Conv3D

0.55-0.60

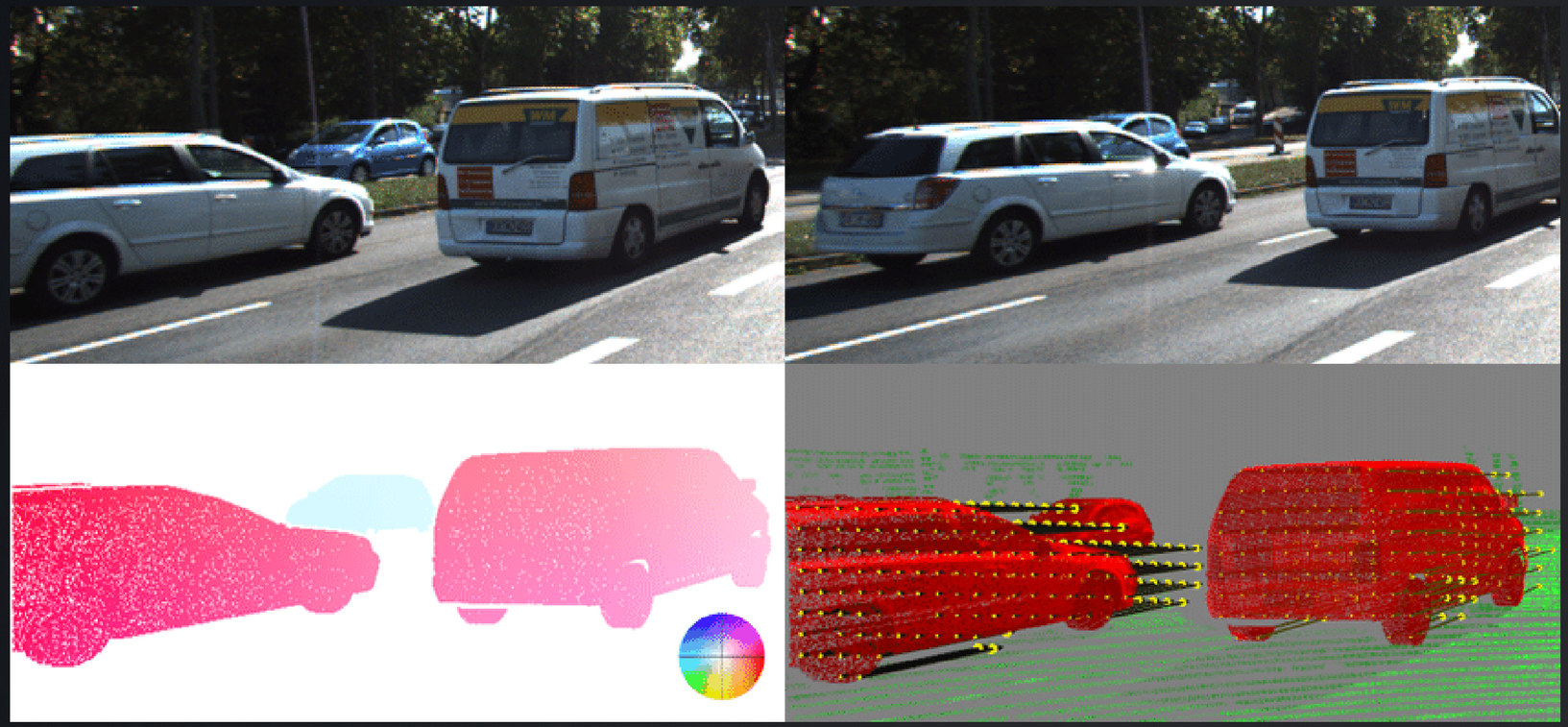
ConvLSTM

0.60-0.68

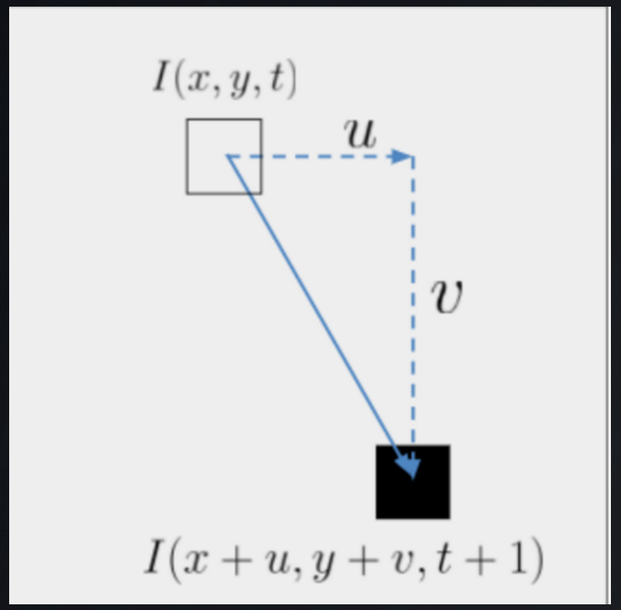
OPTICAL FLOW

OPTICAL FLOW

It is a pattern of motion between consecutive images



BASICS



Assumptions for determining (u, v):

- The motion is small
- The appearance does not change

Appearance does not change

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

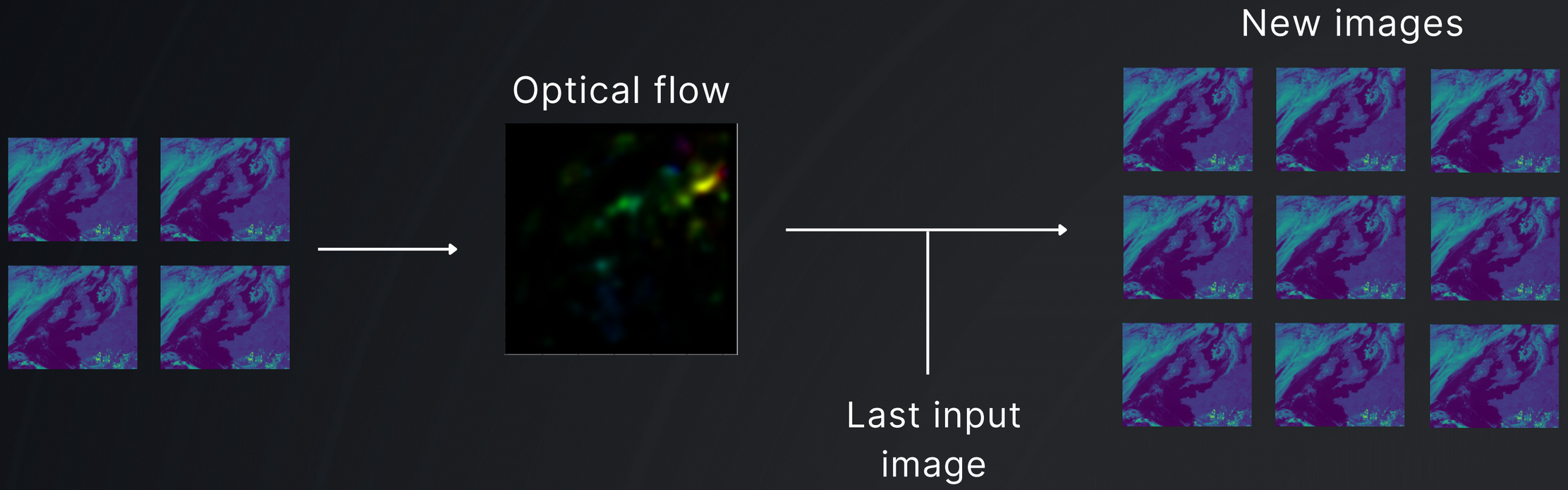
$$\frac{\partial I}{\partial t} + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v = 0$$

$$I_t + I_x u + I_y v = 0$$

Any change is caused my spatial motion

OPTICAL FLOW SYSTEM

Main Idea: warping images according to the optical flow

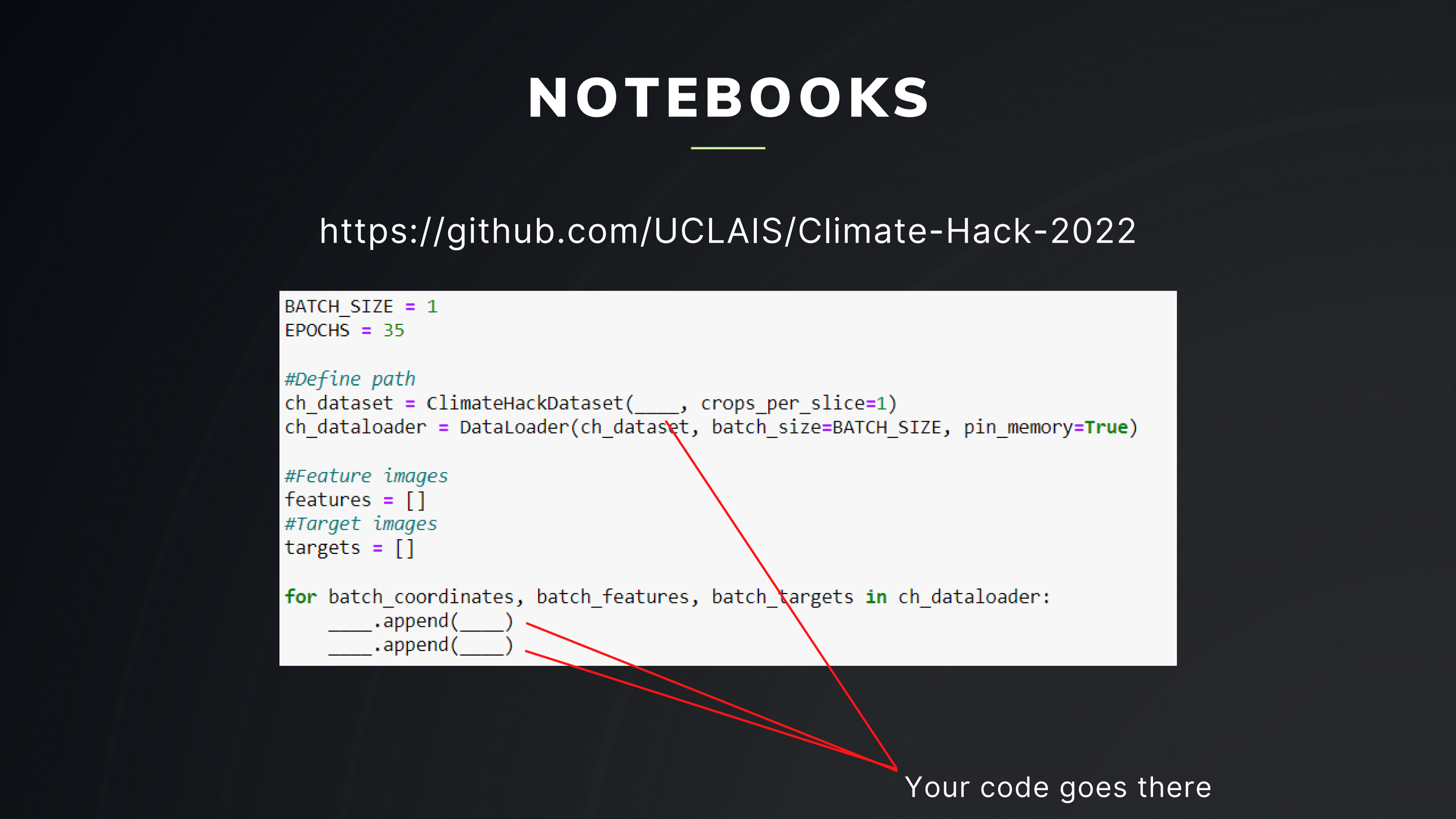


IMPLEMENTATION

NOTEBOOKS

https://github.com/UCLAIS/Climate-Hack-2022

```
BATCH_SIZE = 1
EPOCHS = 35

#Define path
ch_dataset = ClimateHackDataset(____, crops_per_slice=1)
ch_dataloader = DataLoader(ch_dataset, batch_size=BATCH_SIZE, pin_memory=True)

#Feature images
features = []
#Target images
targets = []

for batch_coordinates, batch_features, batch_targets in ch_dataloader:
    ____.append(____)
    ____.append(____)
```



IMPORTING DATA

```
ch_dataset = ClimateHackDataset("data.npz", crops_per_slice=1)
ch_dataloader = DataLoader(ch_dataset, batch_size=BATCH_SIZE, pin_memory=True)

#Feature images
features = []
#Target images
targets = []

for batch_coordinates, batch_features, batch_targets in ch_dataloader:
    features.append(batch_features)
    targets.append(batch_targets)
```

CONSTANTS

```
#Sample batch
img = batch_features[0].numpy() / 1023

#Defining the number of previous images
NUM_PREVIOUS_IMAGES = 12

#Number of images for prediction
NUM_PREDICTION_IMAGES = 24
```

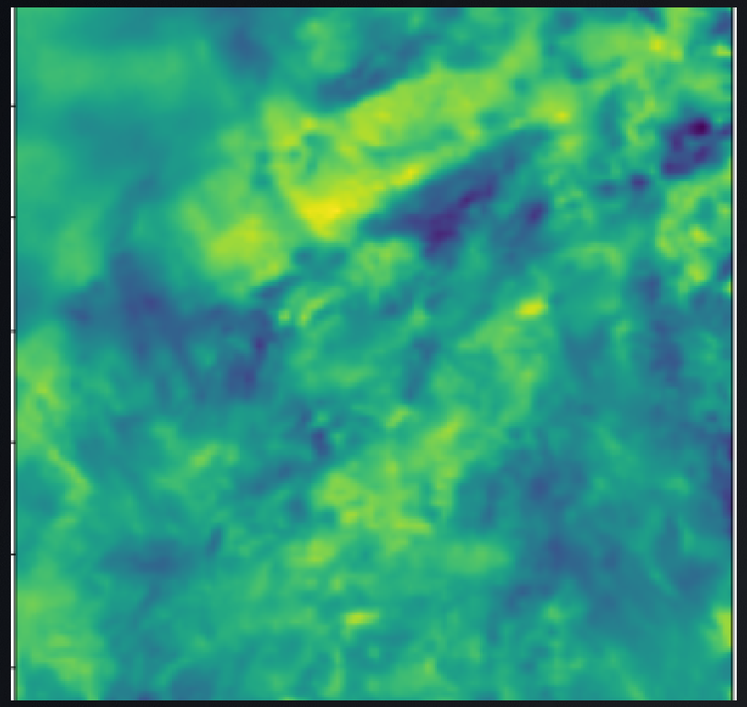
COMPUTING OPTICAL FLOW

```python
def compute_flows(**kwargs):
    flows = []
    for i in range(NUM_PREVIOUS_IMAGES):
        flow = cv2.calcOpticalFlowFarneback(prev=img[i-1], next=img[i], flow=None, **kwargs)
        flows.append(flow)
    return np.stack(flows).astype(np.float32)
```

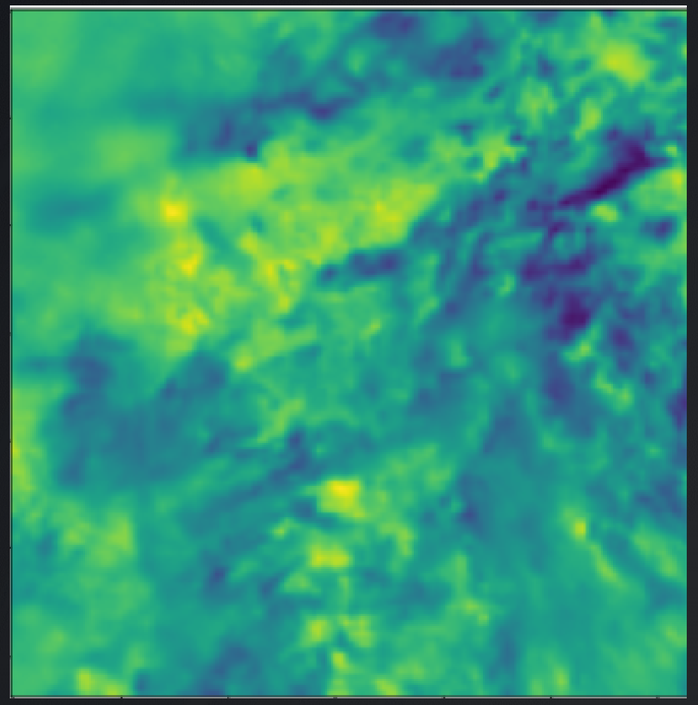

Averaging flows

```python
flows = compute_flows(
    pyr_scale=0.5, levels=3, winsize=15,
    iterations=10, poly_n=5, poly_sigma=1.2,
    flags=cv2.OPTFLOW_FARNEBACK_GAUSSIAN)

flows = weighted_average(flows)
```

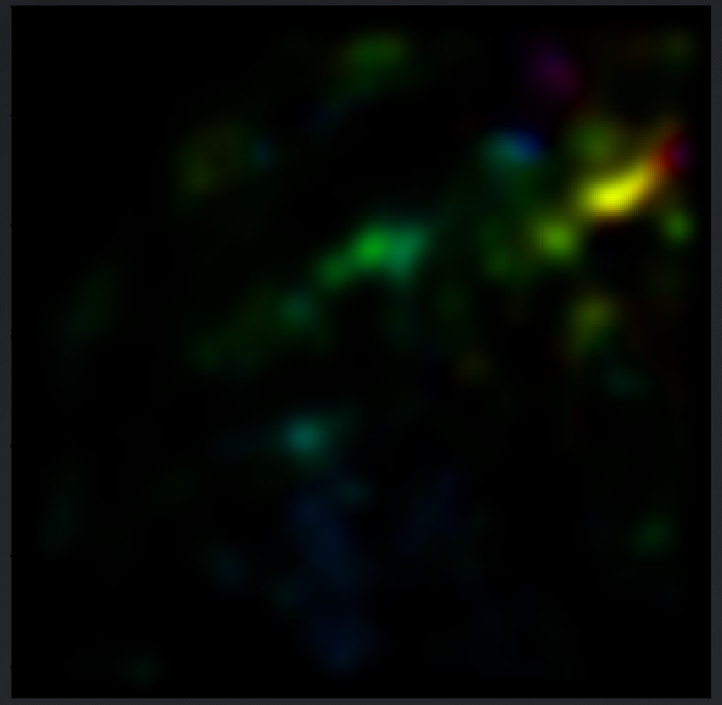
First image



Last image



Optical flow



PREDICTING FUTURE

Last image

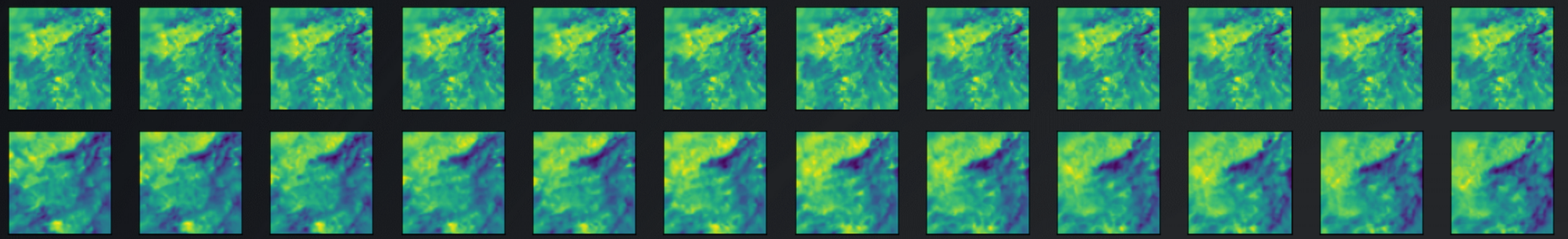
```python
def remap_image(image: np.ndarray, flow: np.ndarray) -> np.ndarray:

    height, width = flow.shape[:2]
    remap = -flow.copy()
    remap[..., 0] += np.arange(width)
    remap[..., 1] += np.arange(height)[:, np.newaxis]
    remapped_image = cv2.remap(src=image, map1=remap, map2=None, interpolation=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REPLI

    # Remember that the output has to be 64x64
    return cv2.resize(remapped_image, (64, 64))
```

24 warped
imaged

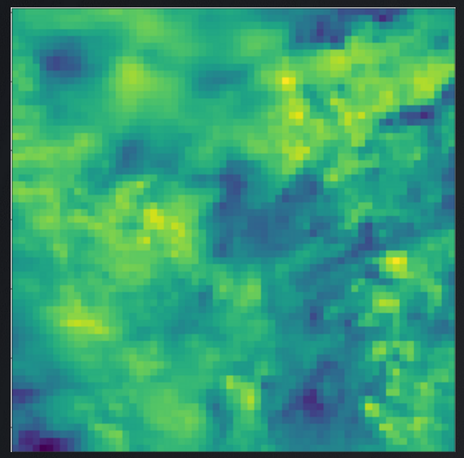
RESULTS

Generated

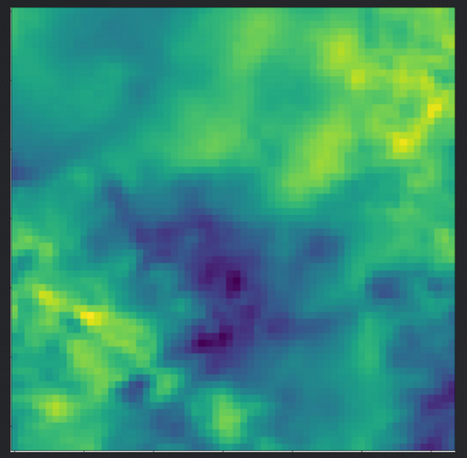
True



Generated



True

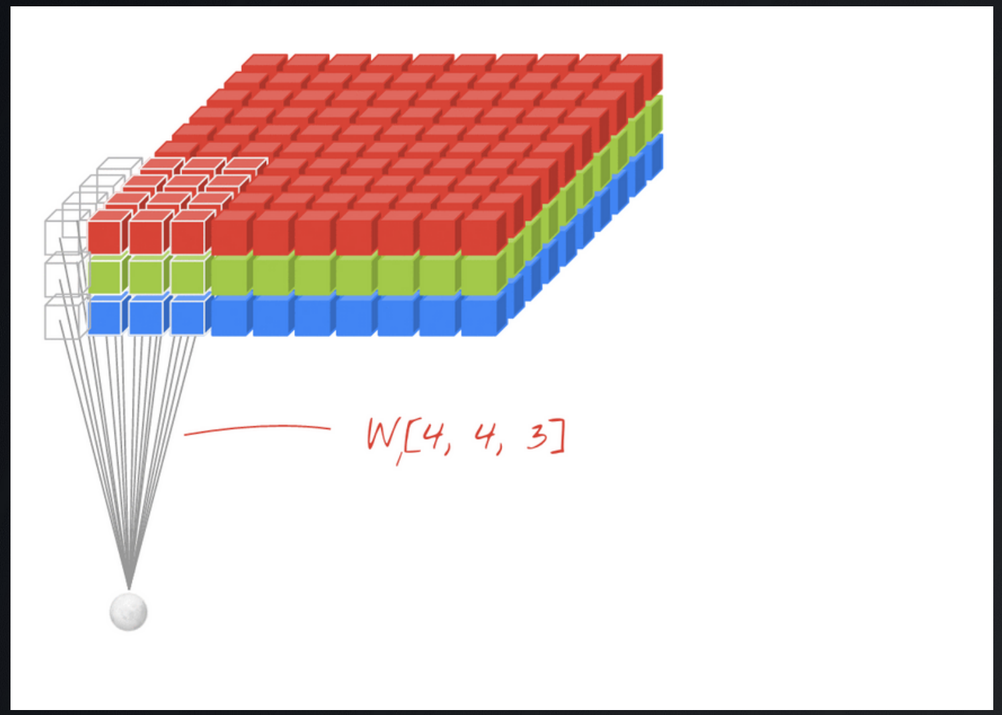


DEPLOYING OPTICAL FLOW

- Adding optical flow to *'model.py'*
- Changing *'evaluate.py'* script

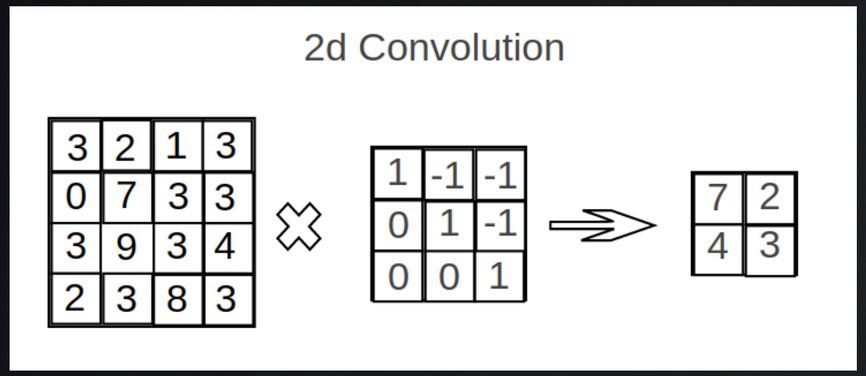
CONV3D

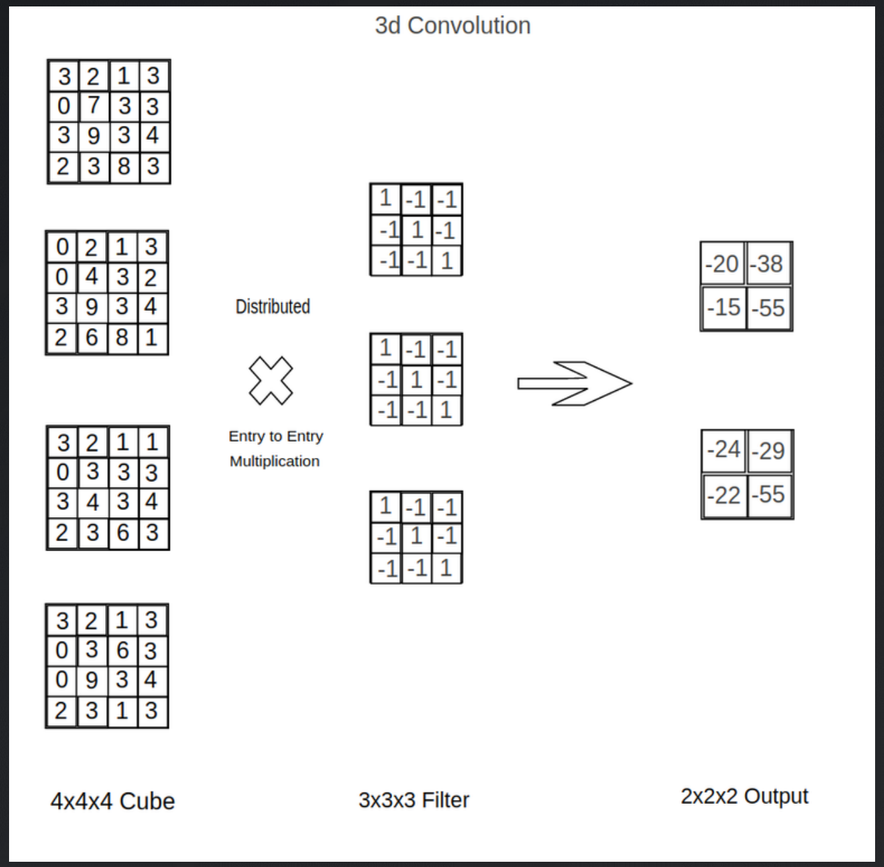
CONV3D



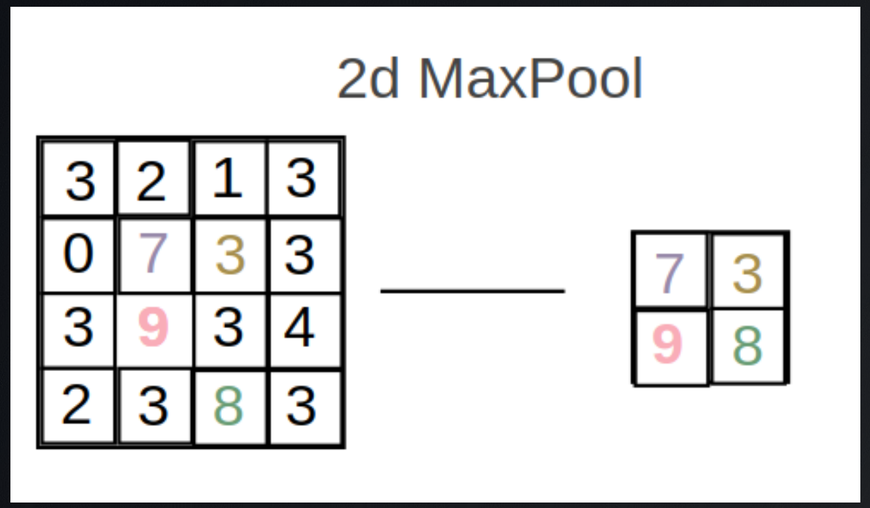
- Similar to 2D convolutional layer except a few differences in:
  - Convolution
  - MaxPooling

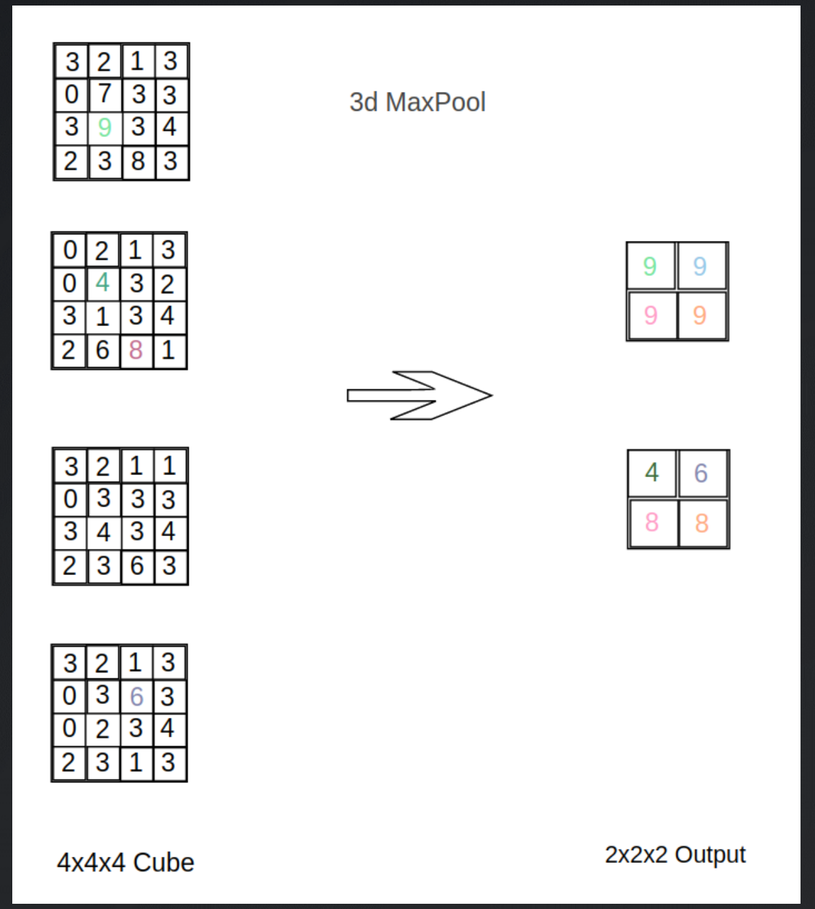
CONV2D VS CONV3D





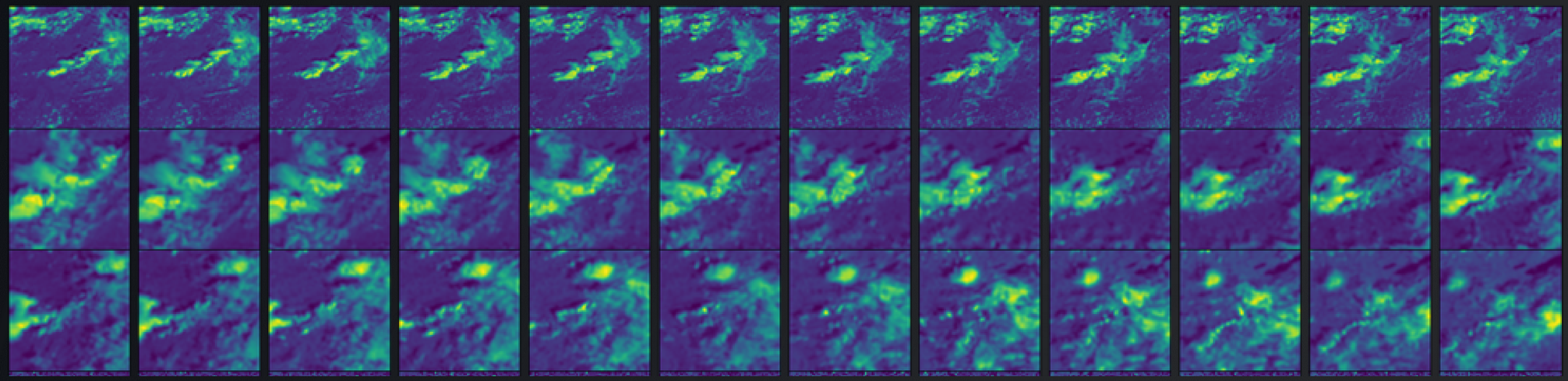
CONV2D VS CONV3D





WHY CONV3D?

Additional dimension helps to 'capture' change in time

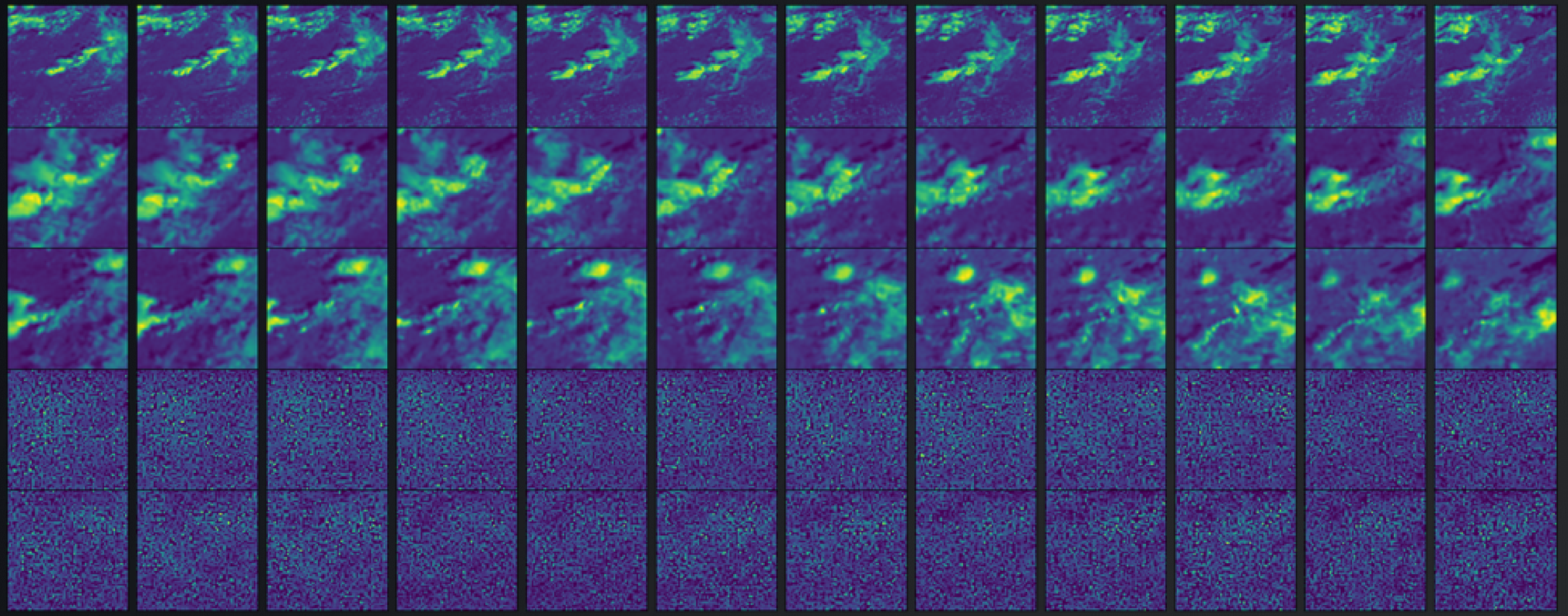


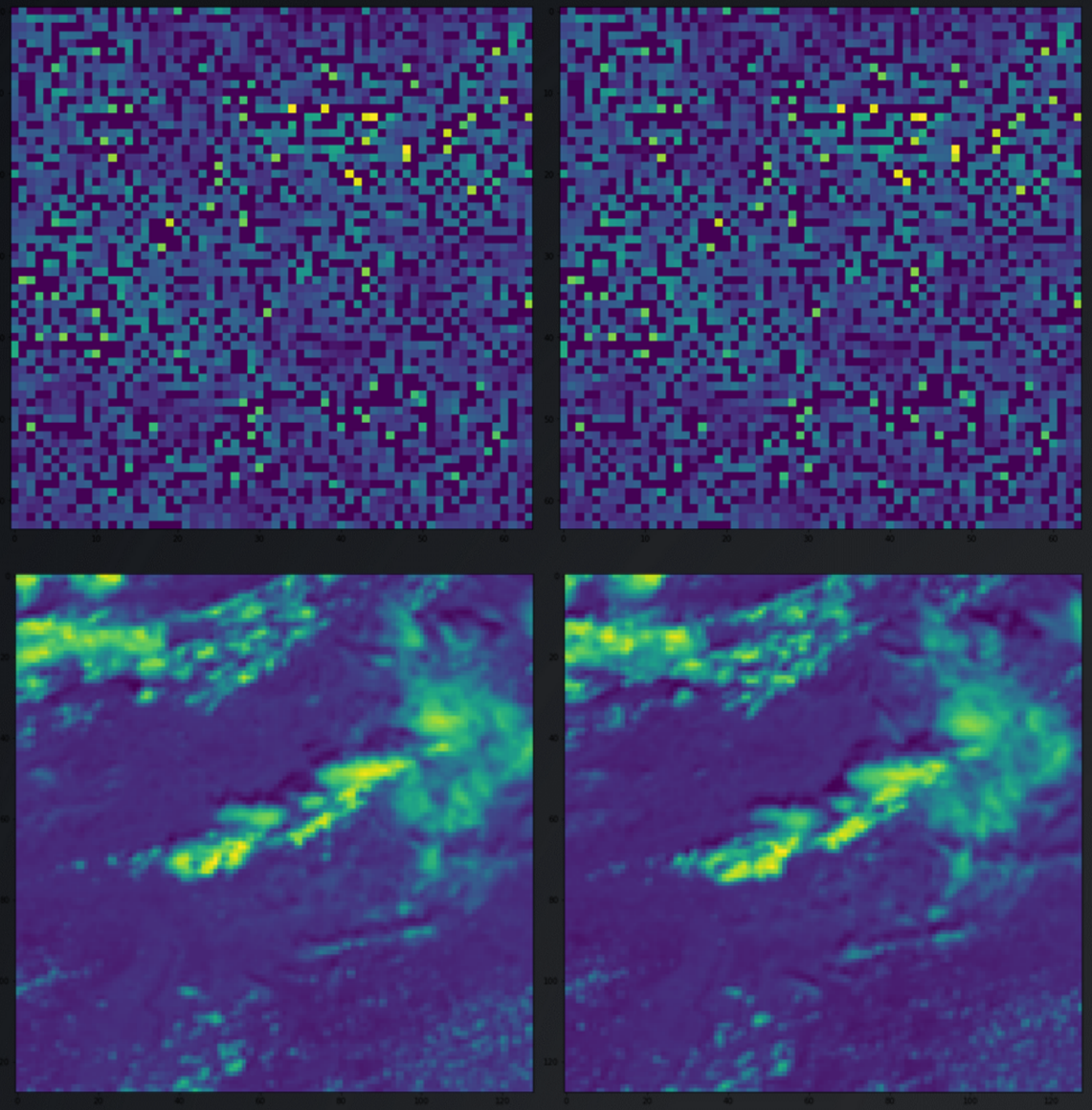
IMPLEMENTATION

MODEL

```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.down_pool = nn.MaxPool3d(kernel_size=(1, 2, 2))
        self.layer1 = nn.Conv3d(1, 4, stride=(1, 2, 2), kernel_size=5)
        self.pool = nn.MaxPool3d(kernel_size=(1, 2, 2))
        self.flatten = nn.Flatten()
        self.layer2 = nn.Linear(7200, 24 * 64 * 64)
        self.layer3 = nn.Linear(1152, 24 * 64 * 64)

    def forward(self, x):
        x = x.unsqueeze(dim=1) / 1023.0
        x = torch.relu(self.down_pool(x))
        x = self.layer1(x)
        x = torch.relu(self.pool(x))
        x = self.flatten(x)
        x = torch.relu(self.layer2(x))
        x = torch.relu(self.layer3(x))

        return x.view(-1, 24, 64, 64) * 1023.0
```

RESULTS

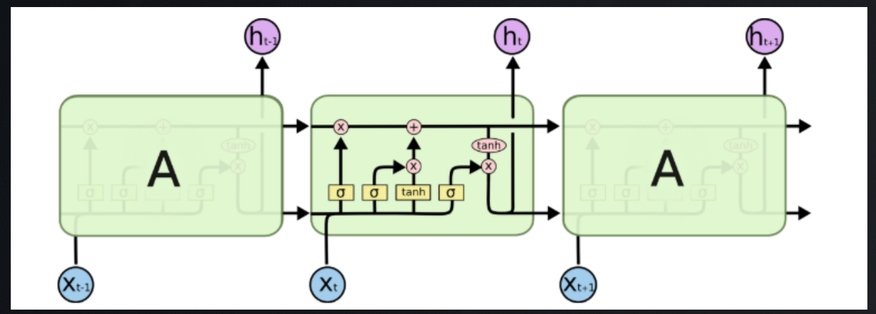


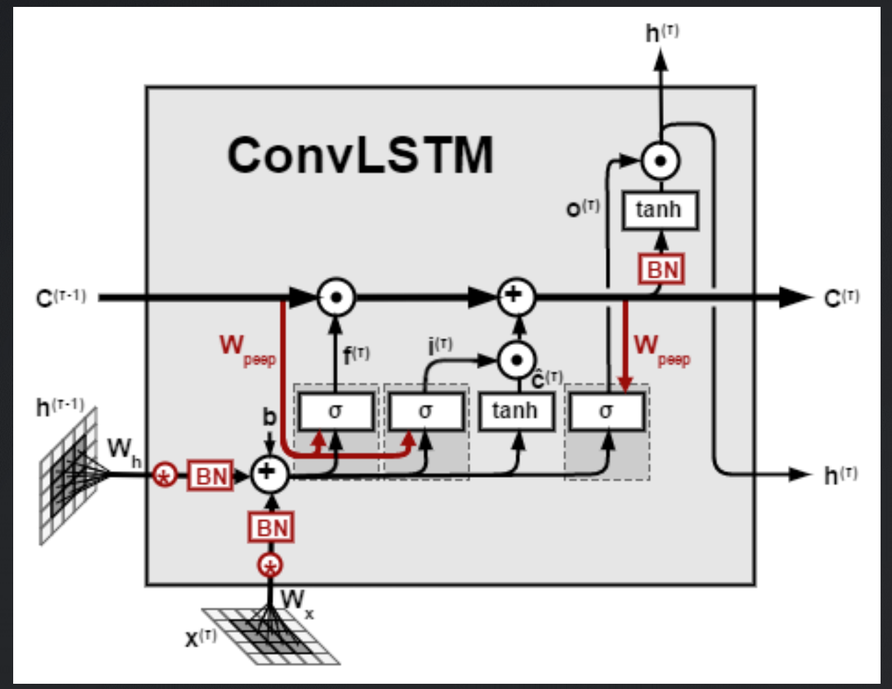


CONVLSTM

CONVLSTM

Combines CNN (image data) with LSTM (sequential data)





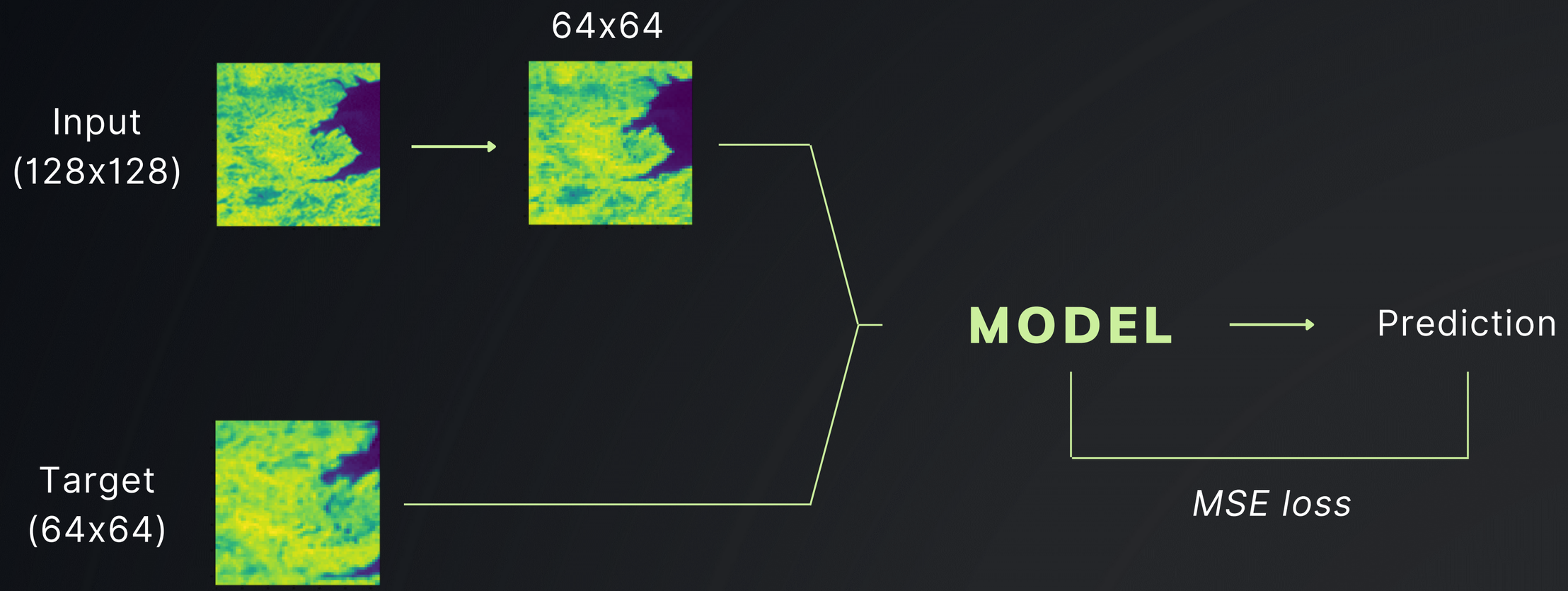
IMPLEMENTATION

TensorFlow
instead of Pytorch

Predicting one
frame

Using MSE
loss

TRAINING MODEL



DEPLOYING MODEL

STRUCTURE

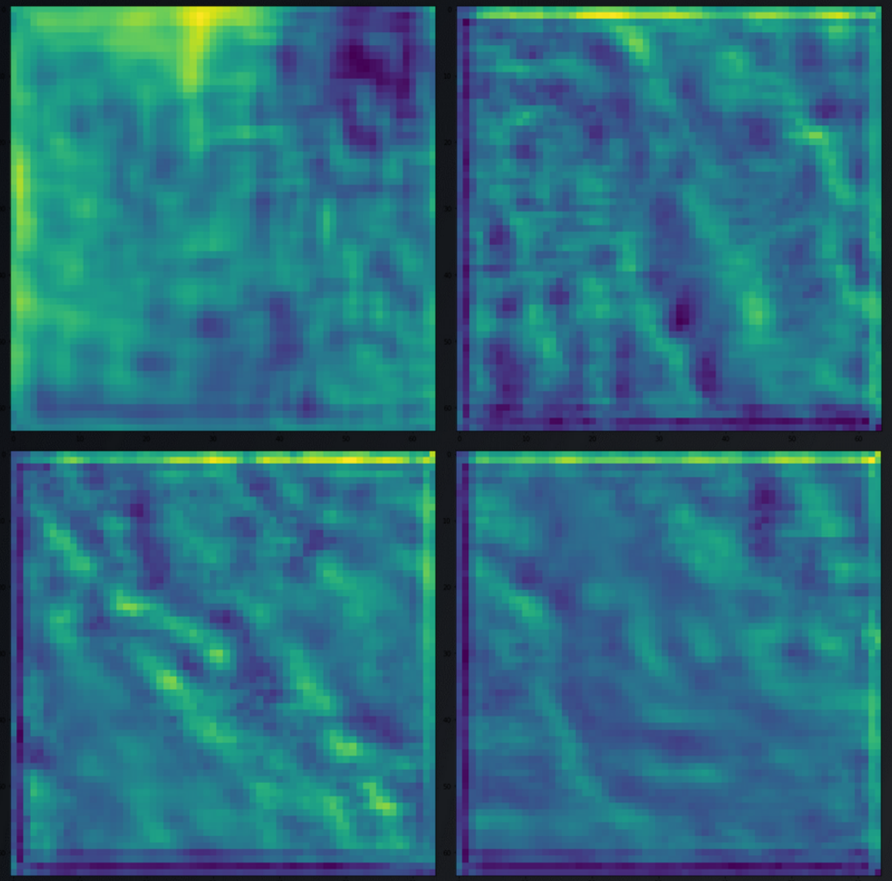
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, 64, 64, 1)  0
                             ]

 conv_lstm2d (ConvLSTM2D)    (None, None, 64, 64, 64)  416256

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

 conv_lstm2d_1 (ConvLSTM2D)  (None, None, 64, 64, 64)  295168

 batch_normalization_1 (Batc  (None, None, 64, 64, 64)  256
 hNormalization)

 conv_lstm2d_2 (ConvLSTM2D)  (None, None, 64, 64, 64)  33024

 conv3d (Conv3D)             (None, None, 64, 64, 1)   1729

 tf.math.multiply (TFOpLambd  (None, None, 64, 64, 1)  0
 a)

=================================================================
Total params: 746,689
Trainable params: 746,433
Non-trainable params: 256
_________________________________________________________________
```

MODEL

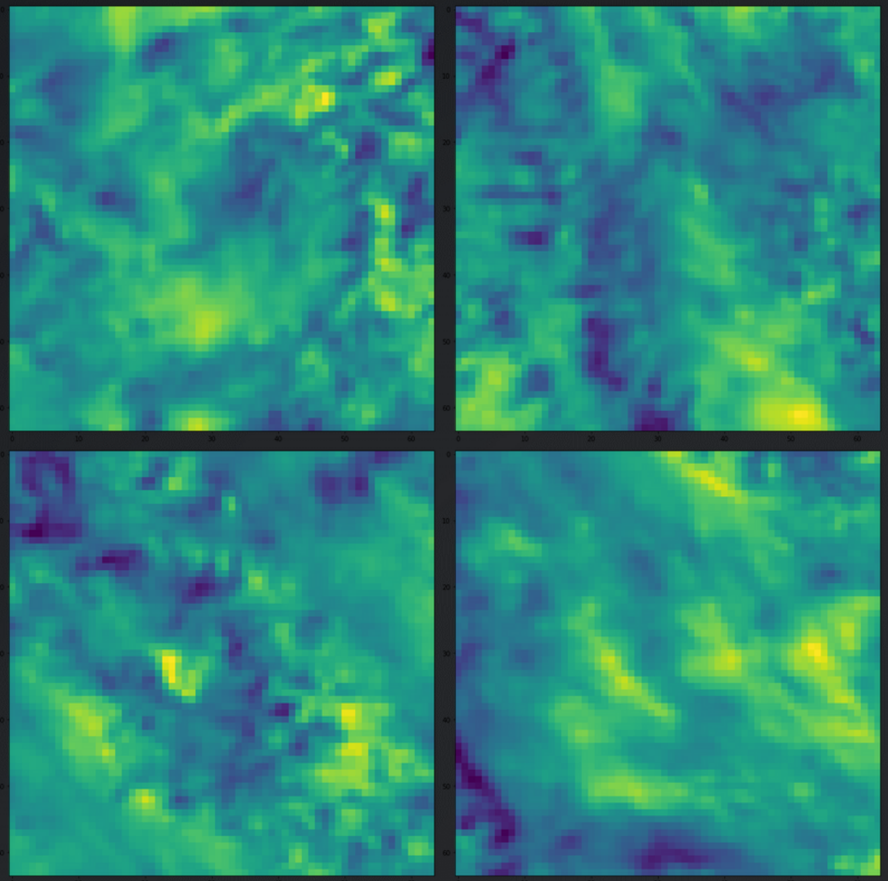
```
inp = layers.Input(shape=(None, *X.shape[2:]))

x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
x = layers.BatchNormalization()(x)

x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.BatchNormalization()(x)

x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)

x = layers.Conv3D(
    filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x) * 1023

model = keras.models.Model(inp, x)

model.compile(loss=tf.keras.losses.mse, optimizer='adam')
```

RESULTS

Generated

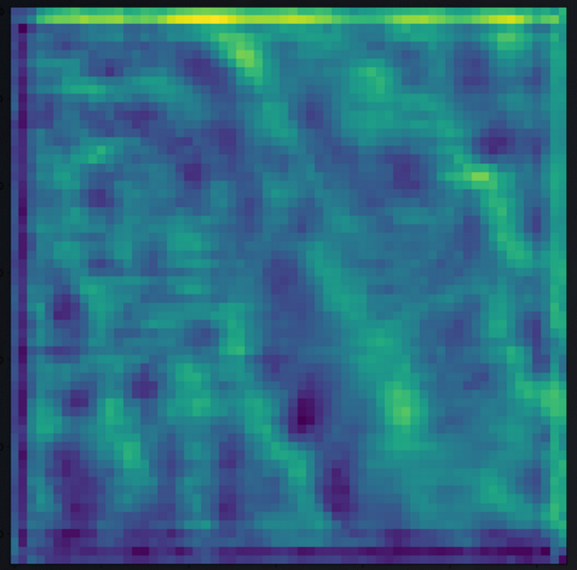


True

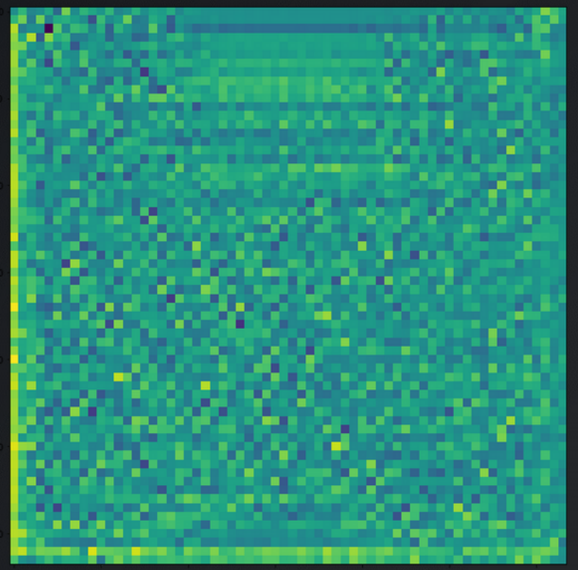


INTENSITY DECAY

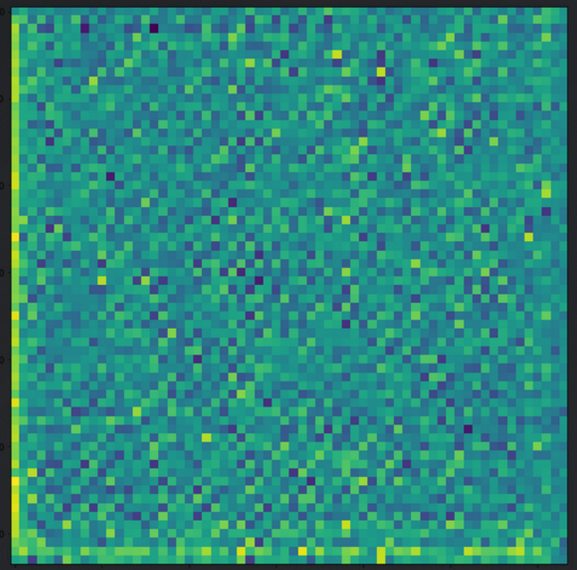
1st image



10th image



24th image



COMPARISON

	Optical Flow	Conv3D	ConvLSTM
Score	*0.55*	*0.58*	*0.62*

IMPROVEMENTS

IMPROVEMENTS

More data

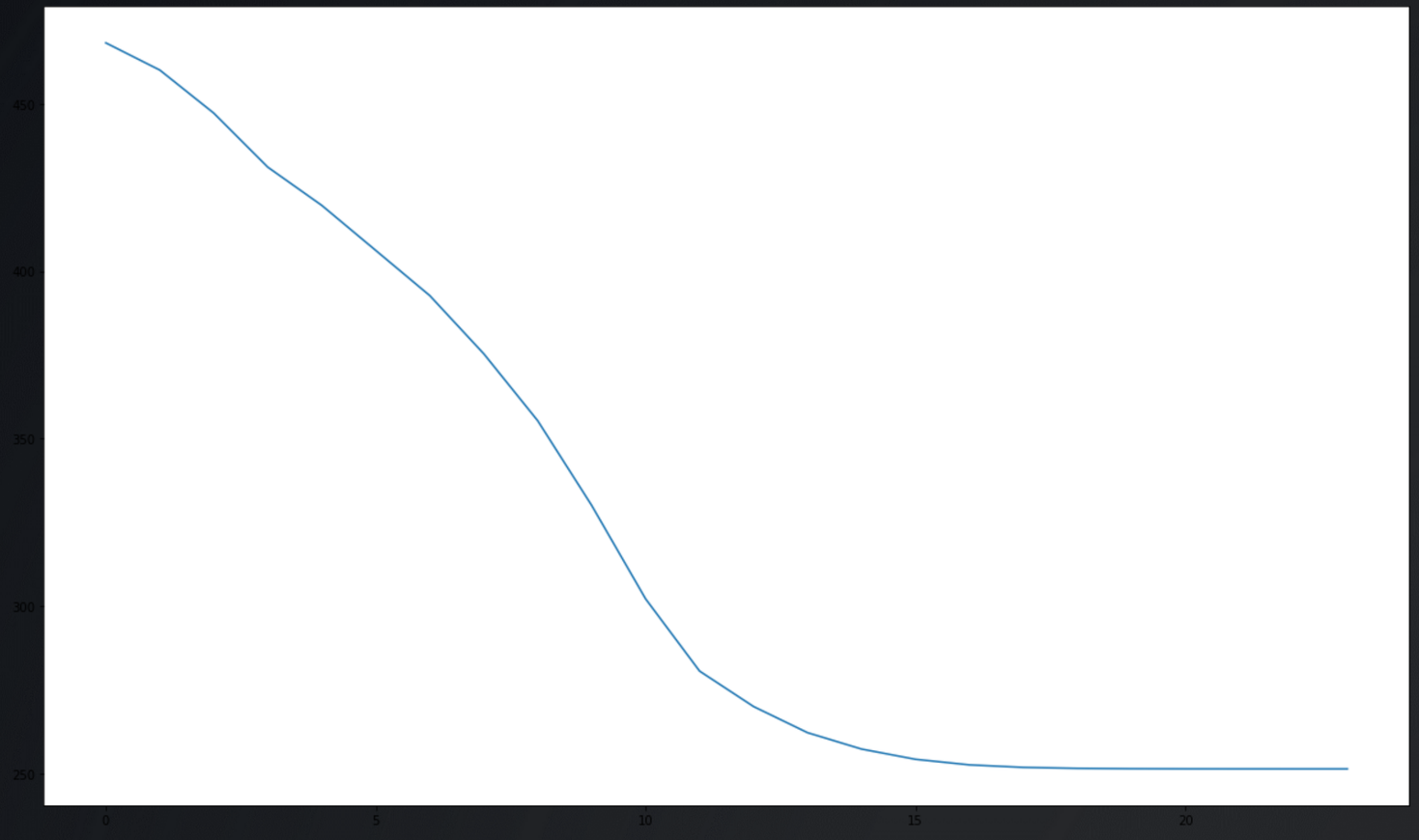
Longer epochs

MS SSIM loss function

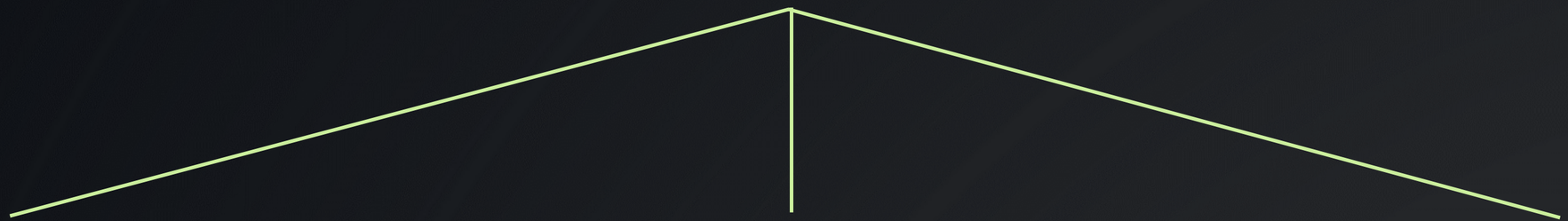
Smaller batches

Normalization

Max intensity variation graph



SOLUTIONS



Factorizing intensity

Increasing image intensity every step to counter decay

Weighted average

Adding weighted mean of previous images

Model structure

Training on 12-12 images pairs instead of 1-1

OTHER MODELS?

THANK YOU!