Hong Kong University of Science and Technology
COMP 4211: Machine Learning
Fall 2022

Programming Assignment 2
Due: 24 October 2022, Monday, 11:59pm

1 Objective

The objective of this assignment is to practise using the `TensorFlow` machine learning framework through implementing custom training modules and data reader modules for video prediction on the Moving MNIST dataset using a convolutional neural network (CNN) based architecture. Throughout the assignment, students will be guided to develop the CNN-based model step by step and study how to build custom modules on `TensorFlow` and the effects of different model configurations.

2 Major Tasks

The assignment consists of a coding part and a written report:

Coding:

> Build a video prediction model using `TensorFlow` and `Keras`. You need to submit a notebook containing all of your running results. Please remember to keep the result of every cell in the notebook for submission.

Written Report:

> Report the results and answer some questions.

The tasks will be elaborated in Sections 4 and 5 below. Note that [Qn] refers to a specific question (the nth question) that you need to answer in the written report.

3 Setup

- Make sure that the libraries `tensorflow-gpu`, `tensorflow-addons`, `numpy`, `matplotlib` and `ffmpeg` have been installed in your Colab environment.
- `Python` version 1.7.14 and `TensorFlow` version 2.8.2 have been verified to work well for this assignment. When `TensorFlow` 2.0+ is installed, `Keras` will also be installed automatically. You are allowed to use all the aforementioned packages, but other machine learning frameworks such as `PyTorch` should *not* be used.
- You should use GPU resources to complete this assignment, i.e., the GPU resources provided by Colab. Otherwise, you will likely get the error "Gradients for grouped convolutions are not supported on CPU".
- The dataset files to be used are provided as a ZIP file (`pa2_data.zip`), which includes two data files (`train-images-idx3-ubyte.gz`, `MNIST_test_seq.npy`), a file containing the code for moving digit generation, two folders recording the pretrained model weights, and a file named `readme.txt` describing the checkpoints that store the model weights.

- It is likely to be useful to run the following code to enable the `numpy` behavior in `tf.tensor` before you do the actual coding.

```
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior()
```

4 Video Prediction

Video prediction is one of the fundamental computer vision tasks and is widely used in many applications, such as human motion forecasting, weather prediction, and video surveillance. Video prediction is a task that predicts the sequence of realistic future frames given the past frames. This can be achieved by using a CNN-based encoder-translator-decoder architecture. We will load the data of the past video frames and the ground truth for the future frames, build a model based on the architecture, train the model using the data, and finally evaluate the video prediction performance of the trained model based on multiple metrics.

The overall structure of this assignment consists of five main parts plus an *optional* bonus section:

1. Build a data generator to generate the frame sequences from tensors of the given MNIST dataset (Section 4.1).
2. Build a CNN-based encoder-translator-decoder backbone network (Section 4.2).
3. Load the pretrained model weights before training (Section 4.3).
4. Define the evaluation metrics (Section 4.4).
5. Complete the whole rundown for training (Section 4.5).
6. (Bonus) Build and analyze the effects of different model configurations (Section 4.6).

4.1 Dataset and Data Generator

The Moving MNIST dataset can be found in the `./pa2_data` directory. You are recommended to navigate through the `numpy` tensors to visualize the content and generate an animation of the frame sequences using `matplotlib`. There are two files related to the dataset, a file containing the code below, two folders recording the checkpoints of the pretrained model, and a `readme.txt` file describing those checkpoints. You should load the checkpoints in the folder `pretrained_main` before training. It is optional to use the checkpoints in another folder `bonus`, with details provided in Section 4.6 and the `readme.txt` file. The file `train-images-idx3-ubyte.gz` contains 60,000 MNIST images with each image having the shape 28×28 and you will later process it to the shape of 64×64 along moving sequence generation. You need to construct a data generator to create moving sequences of two random digits. The file `MNIST_test_seq.npy` contains 10,000 samples, each of which contains 20 frames of moving MNIST images. In this assignment, you will use the first 10 frames as input and the last 10 as the prediction ground truth.

Regarding sequence generation, we provide a piece of code below for your reference. However, you need to define a custom dataset class `MMNISTDataset` using `tf.keras.utils.Sequence` and incorporate the given code inside it. For more details, you may study the example in the documentation of `tf.keras.utils.Sequence` (`https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence`).

The code for generating sequences of moving digits is as follows:

```
def gen_random_sequence():
    ‘‘‘ randomly generate a sequence of a digit’’’
    size = 64 - 28
    x, y, theta = random.random(), random.random(), random.random() * 2 * np.pi
    velocity_y, velocity_x = np.sin(theta), np.cos(theta)
    seq_x, seq_y = np.zeros(20), np.zeros(20)

    for i in range(20):
        y += 0.1*velocity_y
        x += 0.1*velocity_x

        if x <= 0:
            x = 0
            velocity_x = -velocity_x
        if x >= 1.0:
            x = 1.0
            velocity_x = -velocity_x
        if y <= 0:
            y = 0
            velocity_y = -velocity_y
        if y >= 1.0:
            y = 1.0
            velocity_y = -velocity_y
        seq_x[i], seq_y[i] = x, y

    # Scale to the size.
    seq_x = (size * seq_x).astype(np.int32)
    seq_y = (size * seq_y).astype(np.int32)
    return seq_y, seq_x



def random_mmnist():
    ‘‘‘generate frames of moving mnist. ’’’
    data = np.zeros((20, 64, 64), dtype=np.float32)
    # 10 input + 10 groundtruth = 20
    for n in range(2):
        seq_y, seq_x = gen_random_sequence()
        idx = random.randint(0, mnist_train_im.shape[0] - 1)
        mnist_image = mnist_train_im[idx]
        for i in range(20):
            # put the 2 moving digits into "data"
            data[i, seq_y[i]:seq_y[i]+28, seq_x[i]:seq_x[i]+28] =
                np.maximum(
                data[i, seq_y[i]:seq_y[i]+28, seq_x[i]:seq_x[i]+28],
                mnist_image
                )
    return data
```

[C1] To build a custom dataset, the first thing is to implement the function:

`MMNISTDataset.__init__(self, *arguments)`

You need to load the data from either of the two data files depending on the input arguments of `__init__` and define the following class attributes:

(1) `batchsize`

(2) `length` (i.e., length of the batch)

(3) `mnist_train_im` (i.e., data loaded from `train-images-idx3-ubyte.gz`)

(4) `mnist_test` (i.e., data loaded from `MNIST_test_seq.npy`)

Apart from the aforementioned class attributes, you can define more if necessary. Regarding the length attribute, its value for the training set can be the same as that for the test set. In addition, you also need to implement the `MMNISTDataset.__len__(self)` function, which returns the length attribute.

[C2] The second thing is to incorporate the two given functions `gen_random_sequence()` and `random_mmnist()` into the `MMNISTDataset` class. Slight modification of the given code is expected.

[C3] The final function you need to implement is `MMNISTDataset.__getitem__(self, idx)`. This function returns two `tf.tensor` objects (first 10 frames as input and last 10 frames as ground-truth), with both of shape (batchsize, frames, height, width, channels). Before returning the two tensors, you also need to perform **normalization** to ensure that all elements are in the range from 0 to 1. Depending on the class arguments, you will return data loaded from different files. Regarding the validation and test sets, you can split the data from `MNIST_test_seq.npy` into two sets, and return an item only on the corresponding splitted portion.

Then, you can obtain the training, validation and test sets by calling `MMNISTDataset`("train"), `MMNISTDataset`("valid"), and `MMNISTDataset`("test"), respectively. (Remark: The arguments "train", "valid" and "test" are just for illustration. You are expected to define your own `*argument` in the `__init__` function.)

4.2 Model Backbone

You have learned some popular CNN architectures in the course. Here, you are asked to implement a custom architecture built solely on CNN and use it as the backbone network of the video prediction model. You will further practise how to load pretrained weights instead of using randomly initialized weights. In the meantime, you can also use the pretrained weights to check the correctness of your implementation. This model is composed of three main modules, namely, *encoder*, *translator*, and *decoder*.

Some background knowledge you need to know in order to complete the assignment will be introduced first, followed by detailed description of the encoder architecture.

4.2.1 Background Information

A 2D convolutional layer can take the input shape (N, H, W, C_{in}), where N is the batchsize, H is height, W is width, and C_{in} is the number of input channels. It outputs $(N, H_{out}, W_{out}, C_{out})$,

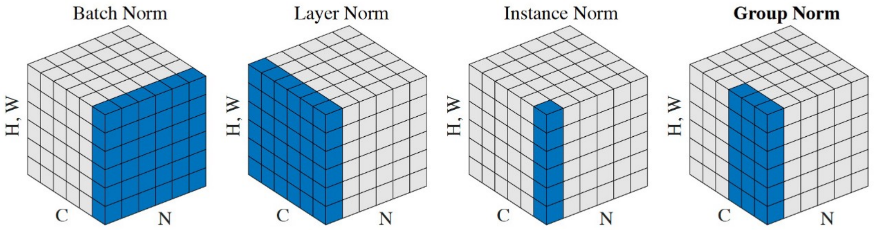

Figure 1: Different types of normalization

where the values of H_{out} and W_{out} are computed according to $\lfloor \frac{n+2p-k}{s} \rfloor$, with n denoting the length of one side of the feature map, p the padding, k the kernel size, and s the stride.

Group normalization is another important concept in this assignment. We partition C channels into n groups for a given value n and then perform normalization on each group. Thus, it is equivalent to layer normalization when the number of groups is equal to 1, as illustrated in Figure 1. (ref: `https://www.tensorflow.org/addons/api_docs/python/tfa/layers/GroupNormalization`)

Transposed convolution, or called **deconvolution**, is an important concept in the decoder module. It decodes the encoded hidden state back to a realistic sequence of video frames. In essence, it is a reverse process of the convolutional layer. Each element in the kernel is multiplied with the input feature map and the overlapping results will be summed together as illustrated in Figure 2. (ref: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose`)

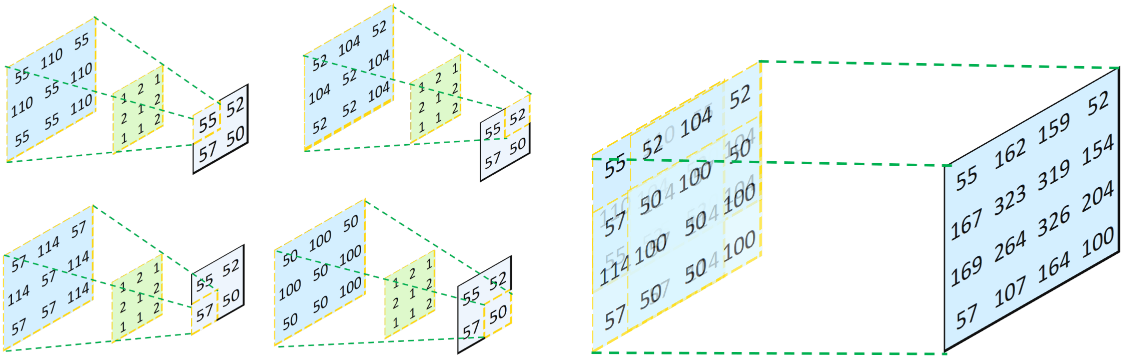

Figure 2: How transposed convolution works

4.2.2 Model Overview

As shown in Figure 3, the backbone architecture of the model we are trying to implement consists of three parts: **Encoder**, **Translator**, and **Decoder**, where each part is composed of some convolutional blocks.

A typical flow of the input through the network to the output is as follows:

1. The input consisting of a batch of video frames has shape (N, T, H, W, C), where N is the batchsize, T is the number of frames, H is the height of each frame, W is the width of each frame, and C is the number of channels of each frame.

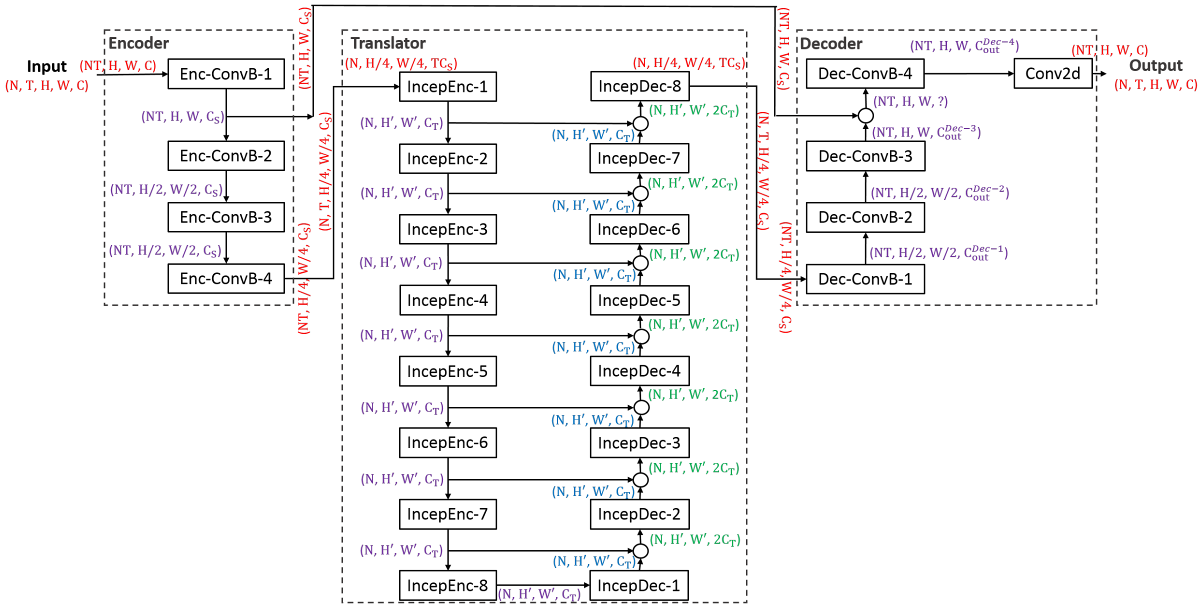

Figure 3: Model architecture overview

Table 1: Notation

Symbol	Value	Meaning
Input & output		
N	16	batch size
T	10	number of input video frames
T'	10	number of predicted video frames
H	64	height of an input image (one video frame)
W	64	width of an input image (one video frame)
C	1	number of channels of an input image
Intermediate tensor		
H'	16	height of embedded tensor passed to the Translator
W'	16	width of embedded tensor passed to the Translator
Model parameters		
C_S	64	hidden dimensionality of the Encoder or Decoder
C_T	256	hidden dimensionality of the Translator
N_S	4	number of layers in the Encoder or Decoder
N_T	8	number of encoding or decoding layers in the Translator

2. The Encoder first reshapes it to (NT, H, W, C), then allows this tensor to flow through several Enc-ConvB blocks, and outputs two tensors: one is the output of Enc-ConvB-1, $O_{Enc-ConvB-1}$, with shape (NT, H, W, C_S), which will be skip-connected to the Decoder later; and the other is the output of Enc-ConvB-4, $O_{Enc-ConvB-4}$, with shape $(NT, H/4, W/4, C_S)$, which is reshaped to $(N, T, H/4, W/4, C_S)$, and sent to the Translator.

3. The Translator reshapes its input to $(N, H/4, W/4, TC_S) = (N, H', W', TC_S)$, and lets

this tensor flow through several IncepEnc and IncepDec blocks inside the Translator, and finally outputs a tensor with the same shape (N, H', W', TC_S) as the input.

4. The tensor output from the Translator is reshaped to $(NT, H/4, W/4, C_S)$ and then sent to the Decoder to pass through several Dec-ConvB blocks inside the Decoder. In the Decoder, after the tensor passing through the Dec-ConvB-i ($i = 1, 2, 3$) blocks, and before going into the last Dec-ConvB-4 block, we concatenate it with the first Encoder output with shape (NT, H, W, C_S), and let the concatenated tensor pass through the last Dec-ConvB-4 block.

5. The last step of the Decoder is to pass the tensor through a Conv2d block to get an output tensor of shape (NT, H, W, C).

6. This output tensor is finally reshaped to (N, T, H, W, C), which is our output: a batch of predicted future frames.

Detailed descriptions of the symbols and parameters involved can be found in Table 1.

Next, we will introduce the implementation details of each part.

4.2.3 Encoder

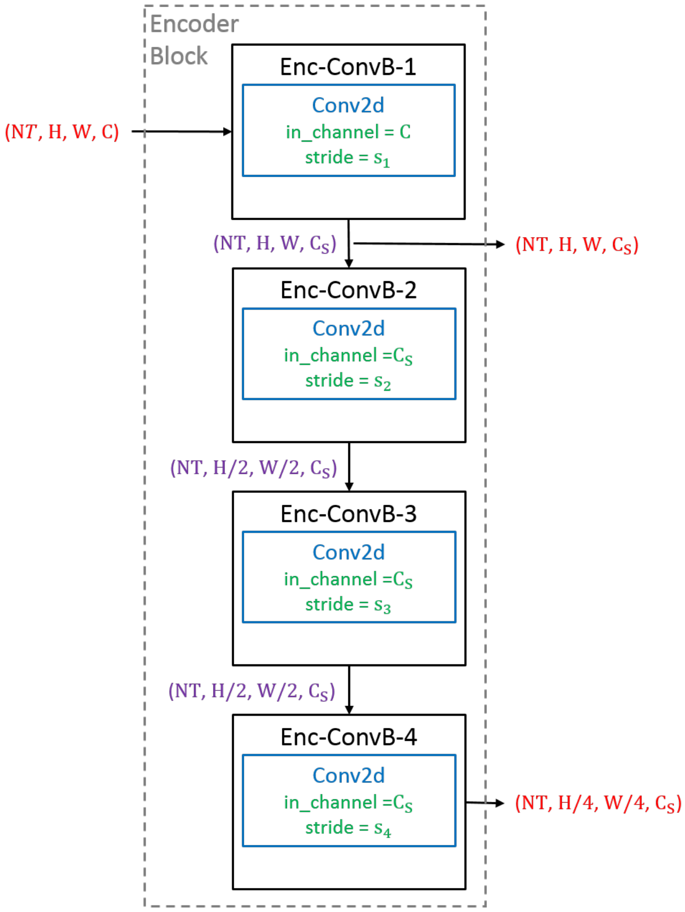

Figure 4: Encoder architecture

From Section 4.1, the input data is of shape (N, T, H, W, C) for batchsize N, frames T, height H, width W, and channels C. You need to reshape it to a 4D tensor with shape (NT, H, W, C) before feeding it into the Encoder. This allows the return of two 4D-tensors so you can later

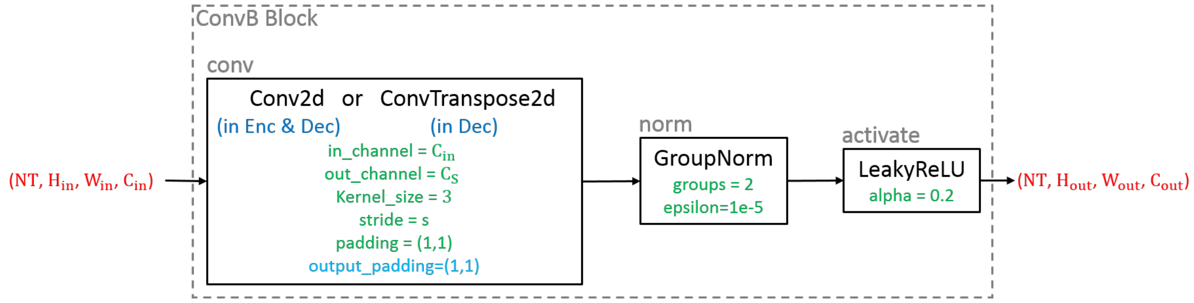

Figure 5: ConvB block

concatenate one of these to the Decoder.

[C4] **Encoder**
To construct the Encoder, the shape of the feature maps will shrink gradually to encode the input features. To achieve this goal, we stack $N_S = 4$ encoder layers together in which each encoder layer is composed of a convolution block Enc-ConvB-i ($i = 1, 2, 3, 4$) with different stride sizes. Details of the Enc-ConvB are described in the next paragraph. The Encoder consists of N_S encoder layers with output channel size C_{out}^{Enc-i} and returns two tensors, i.e., **(1) output after all encoder layers Enc-ConvB-i,** ($i = 1, 2, 3, 4$) and **(2) output after the first Enc-ConvB-1** which will be used as skip connection in the Decoder module. You need to construct this module according to the given information. You can refer to Figure 4 for more information.

[C5] **Enc-ConvB**
As shown in Figure 5, **Enc-ConvB** is composed of a **conv2d** layer with kernel size 3 and padding be "same", a **group normalization** layer and an **activation function** layer. Each channel is divided into two groups and the LeakyReLU activation function is used here. This module should have input arguments consisting of output channel size, stride size, and padding size which allow other modules to call it with different configurations.
Enc-ConvB.conv: a 2D convolution Conv2d with out_channels=C_{out}, stride=s, kernel_size=3, padding='same'. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D`)
Enc-ConvB.norm: a group normalization with all channels divided into two groups. (`https://www.tensorflow.org/addons/api_docs/python/tfa/layers/GroupNormalization`)
Enc-ConvB.activate: we use the LeakyReLU activation function with negative slope coefficient=0.2. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU`)

[Q1] What are the stride sizes s_i of Enc-ConvB-i for $i = 1, 2, 3, 4$? How do you compute these values?

[Q2] What is the number of trainable parameters in the encoder module?

4.2.4 Translator

[C6] **Translator overview**
After encoding, the shape of the Encoder's output is $(N, H/4, W/4, C_S)$. To pass it to the Translator, we reshape it to $(N, T, H/4, W/4, C_S)$. Since the shape $(H/4, W/4)$ will not change throughout the Translator, for simplicity, we use H' and W' where $(H', W') = (H/4, W/4)$. The output shape of the Translator is the same as the input shape $(N, T, H/4, W/4, C_S)$.

The translator will investigate the temporal information of different frames in one batch.

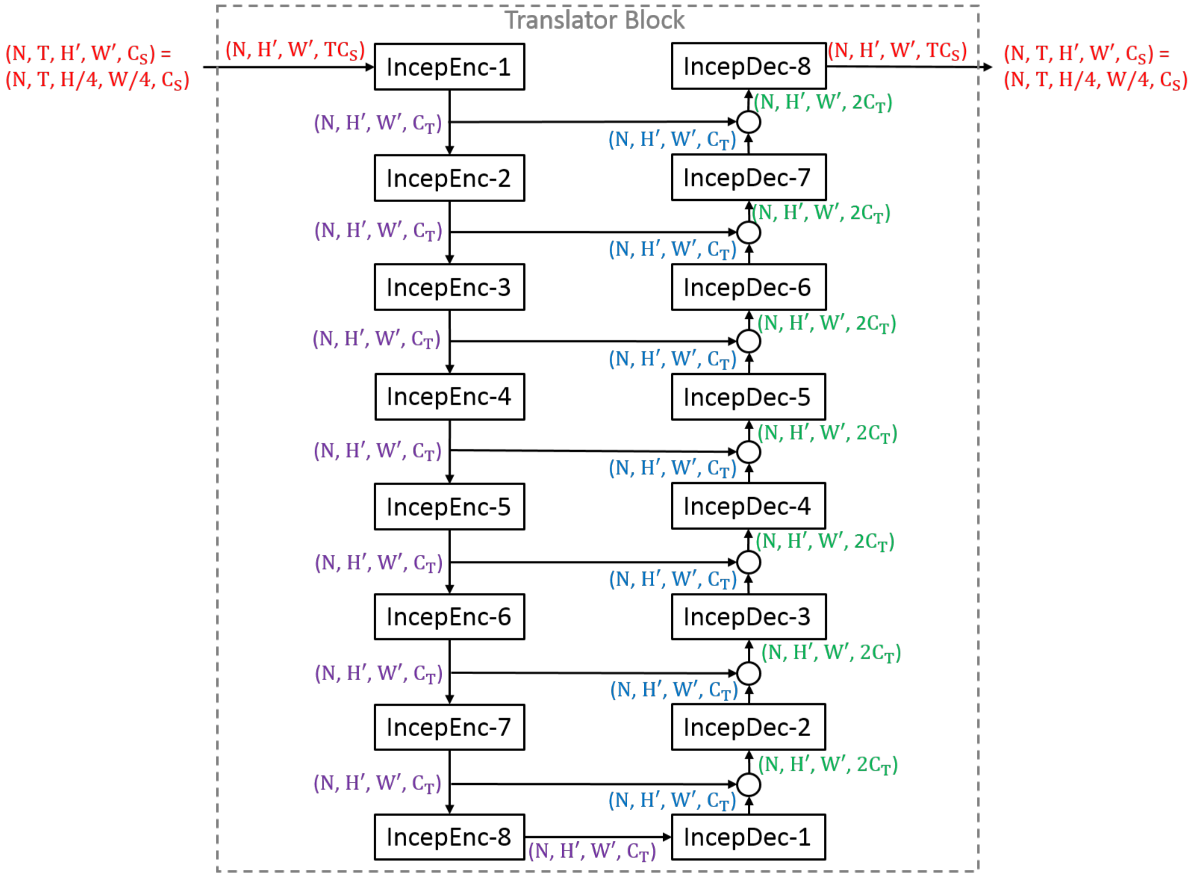

Figure 6: Translator architecture

The detailed architecture of the Translator is shown in Figure 6. It consists of an encoding part on the left and a decoding part on the right. Each encoding or decoding part is made of $N_T = 8$ Inception blocks.

Let us use the abbreviation IncepEnc$-i$ ($i = 1, 2, ..., N_T$) to denote the i^{th} Inception block for encoding, and IncepDec$-i$ to denote the i^{th} Inception block for decoding in the translator. The input and output sizes of each IncepEnc$-i$ and IncepDec$-i$ ($i = 1, 2, ..., N_T$) are as shown. Red characters are used for the input and output shapes of the whole translator block; purple characters are the output shapes of the encoding inception blocks; blue characters are the output shapes of the decoding inception blocks; and green characters are the input shapes of the decoding inception blocks. C_T denotes the hidden dimensionality of the translator, and we set $C_T = 256$.

- The input to the 1^{st} Inception Encoder IncepEnc-1, also the input to the whole translator block, is (N, H', W', TC_S).

- For the output of IncepEnc$-i$ ($i = 1, ..., N_T - 1$) with size (N, H', W', C_T), we pass it to IncepEnc$-(i + 1)$, and at the same time, keep a copy of it, to pass to the corresponding Inception decoder later.

- For the output of IncepEnc$-N_T$ with size (N, H', W', C_T), we directly pass it to IncepDec-1.

- For the remaining decoders in the translator, IncepDec$-i$ ($i = 2, ..., N_T$), their input is the concatenation of two parts, one from the previous decoder IncepDec$-(i-1)$ with size (N, H', W', C_T), and another is the previous copy of output of IncepEnc$-(N_T - i)$ with the same size (N, H', W', C_T). The concatenation operation is just to directly place the previous copy of inception encoder output after the current inception decoded component, aligned in **the channel axis**. So, the input size of IncepDec$-i$ $(i = 2, ..., N_T)$ is $(N, H', W', 2C_T)$.

∗ The output of IncepDec-8, also the final output of the Translator, is of size (N, H', W', TC_S), same as the input shape.

Details inside each Inception block will be introduced next.

[C7] **Inception block used in translator**

The above mentioned IncepEnc$-i$ and IncepDec$-i$ $(i = 1, ..., N_T)$ are all Inception blocks, with

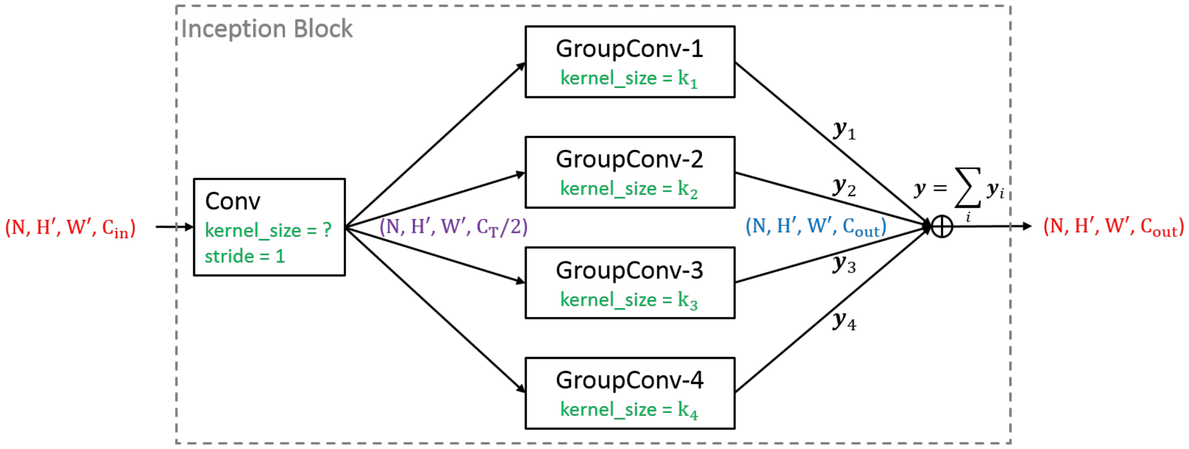

Figure 7: Inception block

the shared architecture, but changes in the number of input and output channels, and hidden dimensions.

Suppose in general the input size to the Inception block is (N, H', W', C_{in}) and the output size is (N, H', W', C_{out}). C_{in} and C_{out} are different for each inception block, e.g.,

$$(C_{in}, C_{out}) = \begin{cases} (TC_S, C_T) & \text{for IncepEnc}_1 \\ (C_T, C_T) & \text{for IncepEnc}_i, i = 2, .., N_T \\ (C_T, C_T) & \text{for IncepDec}_1 \\ (2C_T, C_T) & \text{for IncepDec}_i, i = 2, .., N_T - 1 \\ (2C_T, TC_S) & \text{for IncepDec}_8 \end{cases}$$

The detailed implementation of an Inception block is shown in Figure 7. Inside each Inception block, there are sequential and parallel convolutional parts. There are mainly two modules. The first one is just a 2D convolution, Inception.Conv. The other ones are parallel GroupConv blocks, Inception.GroupConv$-i$, $i = 1, 2, 3, 4$.

Inception.Conv: The input to the Inception block, and also to Inception.Conv, has shape (N, H', W', C_{in}). No matter what C_{in} is, Inception.Conv always has output shape $(N, H', W', C_T/2)$.

[Q3] What are the integer values of the input channel size of Inception.Conv every time it is called?

[Q4] For Inception.Conv, if we use stride=(1, 1) and do not use padding, what should be its kernel size?

Inception.GroupConv: The output of Inception.Conv is seperately sent to 4 parallel convolutional blocks, called Inception.GroupConv$-i$, $i = 1, 2, 3, 4$. Each of the Inception.GroupConv$-i$ is almost the same, except that they have different kernel sizes (k_i, k_i). In this implementation, the corresponding kernel sizes are set to $k_1 = 3$, $k_2 = 5$, $k_3 = 7$, $k_4 = 11$. The detailed implementation of Inception.GroupConv will be described below.
Suppose the output from each Inception.GroupConv$-i$ is $y_i, i = 1, 2, 3, 4$, where each of them has shape (N, H', W', C_{out}). The final output of the Inception Block is the sum of all these outputs, i.e., $y = \sum_{i=1}^{4} y_i$, of the same shape (N, H', W', C_{out}).

[C8] **GroupConv block used in Inception**
Each GroupConv consists of a 2D convolution (GroupConv.conv), a group normalization (GroupConv.norm) and an activation function (GroupConv.activate), see Figure 8.

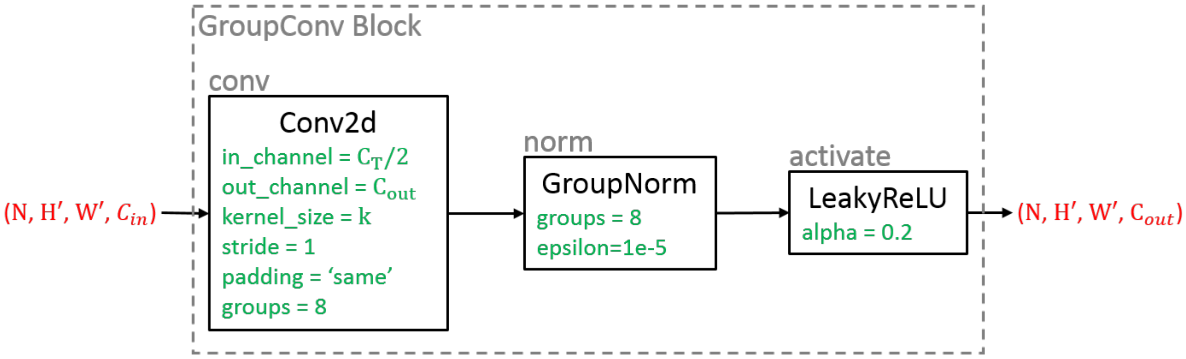

Figure 8: GroupConv block

GroupConv.conv: a 2D convolution with in_channels=$C_T/2$, out_channels=C_{out}, stride=(1,1), kernel_size=(k,k), padding='same', and group=8. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D`)
GroupConv.norm: a group normalization with all channels divided into 8 groups. (`https://www.tensorflow.org/addons/api_docs/python/tfa/layers/GroupNormalization`)
GroupConv.activate: we use the LeakyReLU activation function with negative slope coefficient=0.2. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU`)

[Q5] How many times in total is GroupConv.GroupNorm called in one forward pass through the whole network? (Please answer in integer value.) How did you get this number?

[Q6] For GroupConv.conv, what are its parameters C_{out} (output channels) and k (kernel size) each time it is called? (Please answer in integer values.)

4.2.5 Decoder

In contrast to the encoder, the feature maps gradually grow in the decoder module. From Section 4.2.4, the resultant tensor is of shape $(N, T, H/4, W/4, C_S)$. You need to reshape it to a 4D tensor with shape $(NT, H/4, W/4, C_S)$ before feeding it into the decoder as the deconvolution only accepts input of this shape.

[C9] To decode the features, we stack multiple decoder layers together where each encoder layer consists of 4 deconvolution blocks $Dec - ConvB - i$ $(i = 1, 2, 3, 4)$ with different stride sizes.

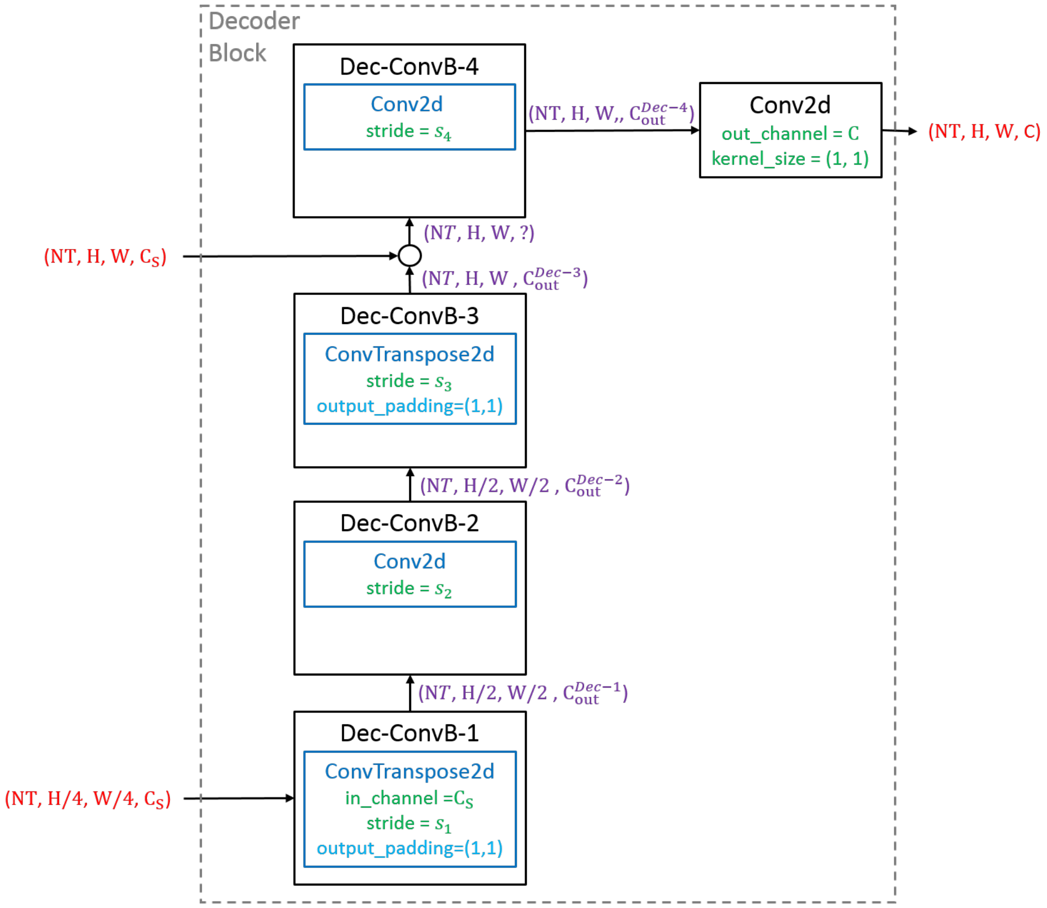

Figure 9: Decoder architecture

Details of a decoder layer are described in Figure 9. The decoder consists of $N_S = 4$ decoder layers with C_{out}^{Dec-i} output channels, followed by a Conv2d layer with C output channels and kernel size 1. Before feeding into the last decoder layer Dec-ConvB-4, **the output $O_{Dec-ConvB-3}$ from the previous Dec-ConvB-3 layer concatenates the output $O_{Enc-ConvB-1}$ of the encoder module along the channel axis**. You need to construct the decoder module according to the given information. You may refer to Figures 9 and 5 for more information.

[C10] **Dec-ConvB**

Based on the information above, you need to construct 4 deconvolution blocks **Dec-ConvB-i** modules. As shown in Figure 5, the Dec-ConvB architecture is basically the same as Enc-ConvB, except that some function or parameter choices are different. Each Dec-ConvB-i is composed of a `conv2d` or `conv2dtranpose` layer with kernel size 3 and padding be "same", a *group normalization* layer and an *activation function* layer. Each channel is divided into 2 groups and the LeakyReLU activation function with negative slope coefficient=0.2 is used here. This module should have input arguments consisting of the output channel size, stride size and padding size which allow other modules to call it with different configurations.

Dec-ConvB.conv: a 2D convolution or deconvolution, either `conv2d` or `conv2dtranpose`, with out_channels=C_{out}, stride=s, kernel_size=3, padding='same'. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D`, `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose`)

Dec-ConvB.norm: a group normalization with all channels divided into 2 groups. (`https://www.tensorflow.org/addons/api_docs/python/tfa/layers/GroupNormalization`)

Dec-ConvB.activate: we use the LeakyReLU activation function with negative slope coefficient=0.2. (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU`)

[Q7] What are the values of the number of output channels C_{out}^{Dec-i} for $i = 1, 2, 3, 4$ in the decoder module?

[Q8] What are the stride sizes s_i of Dec-ConvB-i for $i = 1, 2, 3, 4$?

[Q9] What is the number of trainable parameters in the decoder module?

4.3 Load the Pretrained Weights

[C11] In practice, we usually do not train a model from scratch but initialize it using pretrained weights. Although we are not using some common pretrained models from other domains in this assignment, you will try using the following code to load the model weights in the `pretrained_main` folder to shorten your training time. If you find that the model performance gets worse after loading the weights, it is likely an indication that your model is not correctly built in accordance with the specification. In order to earn the marks in this section, you have to call `model.evaluate()` on the test data to show that the evaluation result indeed improves after loading the weights. Marks will be given if the result after loading is better than that before loading, but there is no specific improvement percentage that needs to be attained.

```
model.load_weights()
```

4.4 Evaluation Metrics

To evaluate the quality of the predicted video frames and have a loss function for training, we need to specify some suitable evaluation metrics. For video prediction, we compare each image of the output predicted frames to the corresponding one in the ground-truth future frames.

Suppose the ground truth of the future frames are $\mathbf{Y}$ and our predicted frames are $\hat{\mathbf{Y}}$. Both $\mathbf{Y}$ and $\hat{\mathbf{Y}}$ are of the same size (N, T, H, W, C), where N is the batchsize, $T = 10$ is the number of frames, $C = 1$ is the number of image channels, and $H = W = 64$ is the image height and width. We can view $\mathbf{Y}$ as n images $\mathbf{y_i}$, $(i = 1, 2, ..., n)$ where $n = N \times T$; and $\hat{\mathbf{Y}}$ as n images $\hat{\mathbf{y}}_i$. Each image $\mathbf{y_i}$ or $\hat{\mathbf{y}}_i$ has size (H, W, C).

There are various evaluation metrics to measure the quality of images by comparing an image $\hat{\mathbf{y}}$ to the ground-truth image $\mathbf{y}$.

1. **Mean Squared Error (MSE)**
   measures the average of the squares of the errors.

$$\text{MSE}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{n} \sum_{i=1}^{n} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$$

   (`https://en.wikipedia.org/wiki/Mean_squared_error`)

2. **Mean Absolute Error (MAE)**
   measures the average of the absolute values of the errors.

$$\text{MAE}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{n} \sum_{i=1}^{n} |\hat{\mathbf{y}}_i - \mathbf{y}_i|$$

(`https://en.wikipedia.org/wiki/Mean_absolute_error`)

3. **Structural Similarity Index Measure (SSIM)**

$$\mathrm{SSIM}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{(2\mu_{\hat{\mathbf{y}}}\mu_{\mathbf{y}} + c_1)(2\sigma_{\hat{\mathbf{y}}\mathbf{y}} + c_2)}{\left(\mu^2_{\hat{\mathbf{y}}+\mu^2_{\mathbf{y}}+c_1}\right)\left(\sigma^2_{\hat{\mathbf{y}}} + \sigma^2_{\mathbf{y}} + c_2\right)}$$

with:
- $\mu_{\hat{\mathbf{y}}}$ the pixel sample mean of $\hat{\mathbf{y}}$
- $\mu_{\mathbf{y}}$ the pixel sample mean of $\mathbf{y}$
- $\sigma^2_{\hat{\mathbf{y}}}$ the variance of $\hat{\mathbf{y}}$
- $\sigma^2_{\mathbf{y}}$ the variance of $\mathbf{y}$
- $\sigma_{\hat{\mathbf{y}}\mathbf{y}}$ the covariance of $\hat{\mathbf{y}}$ and $\mathbf{y}$
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator
- L the dynamic range of the pixel-values
- $k_1 = 0.01$ and $k_2 = 0.03$ by default

There are many ways to compute SSIM. In your implementation, you are expected to use

```
from skimage.metrics import structural_similarity as SSIM
```

to compute $\mathrm{SSIM}(\hat{\mathbf{y}}, \mathbf{y})$. Then you compute the SSIM over all frames by

$$\mathrm{SSIM}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{n}\sum_{i=1}^{n}\mathrm{SSIM}(\hat{\mathbf{y}}, \mathbf{y})$$

(`https://en.wikipedia.org/wiki/Structural_similarity`)

4. **Peak Signal to Noise Ratio (PSNR)**

$$\mathrm{PSNR}(\hat{\mathbf{y}}, \mathbf{y}) = 10\log_{10}\left(\frac{(\max\hat{\mathbf{y}})^2}{MSE(\hat{\mathbf{y}}, \mathbf{y})}\right) = 20\log_{10}(\max\hat{\mathbf{y}}) - 10\log_{10} MSE(\hat{\mathbf{y}}, \mathbf{y})$$

where

$$\mathrm{MSE}'(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{HW}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1}[\hat{\mathbf{y}}(h, w) - \mathbf{y}(h, w)]^2$$

and $\max\hat{\mathbf{y}}$ is the maximum possible pixel value of the image, which is 255 in 8-bit representation (`np.uint8`). Therefore, for a simple implementation, we can compute it as

$$\mathrm{MSE}'(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{HW}\left(\mathrm{np.uint8}(255\hat{\mathbf{y}}) - \mathrm{np.uint8}(255\mathbf{y})\right)^2$$

$$\mathrm{PSNR}(\hat{\mathbf{y}}, \mathbf{y}) = 20\log_{10}(255) - 10\log_{10}(MSE'(\hat{\mathbf{y}}, \mathbf{y}))$$

$$\mathrm{PSNR}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{n}\sum_{i=1}^{n}\mathrm{PSNR}(\hat{\mathbf{y}}, \mathbf{y})$$

(`https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio`)

All the aforementioned metrics have been implemented by `Tensorflow` and `NumPy`, and directly using different implementations may result in different values. For MSE, MAE and PSNR, you are expected to implement them with `NumPy` according to the given equations. As for SSIM, you are expected to use the metric from `skimage`.

4.5 Rundown

To complete the whole rundown, you need to:

- Build the dataset. Details are in Section 4.1.
- Build the model. Details are in Section 4.2.
- [C12] You need to include the computation of MSE and MAE metrics on the validation set during training and keep recording their values for each epoch on the notebook. Your model should be trained for 10 epochs with batchsize 16, MSE loss function, and the Adam optimizer. You are free to set and explore different values for all the unmentioned hyperparameters of the model and also functions of the model.
- [C13] Report the evaluation results of (1) MSE, (2) MAE, (3) SSIM, and (4) PSNR in the test set and randomly select a test sample to produce a video of your prediction and a video of the ground truth using the notebook. You need to be aware of whether the range of the ground-truth tensor is the same as that of your predicted tensor. For SSIM and PSNR, please be reminded to keep the value range of your prediction from 0 to 1 before computing, and the value computed is expected to be the average value per frame. For MSE and MAE, they are also expected to be computed per frame while the mean squared/absolute difference along the height and width axes should be summed together afterwards. You need to obtain MSE $\leq$ 68, MAE $\leq$ 170, SSIM $\geq$ 0.78 and PSNR $\geq$ 32 to earn the 4 marks for prediction.
- [C14+Q10] Using the selected sample above, draw `matplotlib subplots` in the shape of $(3, 5)$ with the first row showing the ground truth of the future frames of the test sample, the second row showing your corresponding predicted result, and the third row showing their differences, in 5 frames with time equal to [0, 2, 4, 6, 8]. Describe the outcome along the 5 frames.

4.6 Bonus

The questions in this part are optional and they will not be counted towards your grade for this assignment. As mentioned in class, students who do reasonably well for the bonus questions will be entitled for one day late in the submission of problem set or project later.

[C15+Q11] **Study Kernel Size**
In the Inception block, we use multiple kernel sizes $k_1 = 3, k_2 = 5, k_3 = 7, k_4 = 11$ for the 4 parallel Inception.GroupConv blocks. How will the performance change if we use single Inception.GroupConv blocks with $k = 3, 5, 7$ and 11 respectively? The weights of the 4 models are provided in the `bonus` folder in `pa2_data.zip`. You need to load the model weights, explore it and use the experimental results to explain your answer.

[C16+Q12] **Study Group Normalization**
What if we replace it with batch normalization, layer normalization or instance normalization? How will the performance change? Please write and describe how to implement the above normalization (remark: you do not need to run the code). Apart from batch normalization, you should not import other modules to build the normalization. The weights of the 3 models are provided in the `bonus` folder in `pa2_data.zip`. You need to load the model weights, explore it and use the experimental results to explain your answer.

Apart from the pretrained weights trained in the above configuration, we also provide the model trained in the original setting for performance comparison. All these pretrained models provided were trained for 100 epochs with loss update less than 0.0001. Be reminded that you need to modify the code first before loading the model weights.

5 Written Report

Answer [Q1] to [Q10] ([Q1] to [Q12] if you do the bonus part as well) in the report.

6 Some Programming Tips

As is always the case, good programming practices should be applied when coding your program. Below are some common ones but they are by no means complete:

- Using functions to structure your code clearly
- Using meaningful variable and function names to improve readability
- Using consistent styles
- Including concise but informative comments
- Using a small subset of data to test the code
- Using checkpoints to save partially trained models

7 Assignment Submission

Assignment submission should only be done electronically in the Canvas course site.

There should be two files in your submission with the following naming convention required:

1. **Report** (with filename `report.pdf`): in PDF format.

2. **Source code and prediction** (with filename `code.zip`): all necessary code should be compressed into a single ZIP file. The ZIP file should include at least one notebook recording all the training and evaluation results. The data should not be submitted to keep the file size small.

When multiple versions with the same filename are submitted, only the latest version according to the timestamp will be used for grading. Files not adhering to the naming convention above will be ignored.

8 Grading Scheme

This programming assignment will be counted towards 15% of your final course grade. The maximum scores for different tasks are shown below:

Table 2: [C]: Code, [Q]: Written report, [P]: Prediction

Grading Scheme	Code (70)	Report (22)	Prediction (8)
Dataset and Data Generator (12)			
- [C1] Build init and len functions	4		
- [C2] Incorporate the given code	2		
- [C3] Build getitem function	6		
Model (69)			
- [C4] Build encoder	8		
- [C5] Build ConvB	4		
- [Q1] Stride size		3	
- [Q2] Enc # params		2	
- [C6] Build translator	12		
- [C7] Build inception	4		
- [Q3] IncepConv C_in		3	
- [Q4] IncepConv kernel size		2	
- [C8] Build GroupConv	4		
- [Q5] GroupConv.conv #called		3	
- [Q6] GroupConv.conv C_out, kernel size		2	
- [C9] Build decoder	10		
- [C10] Build Dec-ConvB	4		
- [Q7] C_{out}^{Dec-i} value		2	
- [Q8] Stride size		2	
- [Q9] Dec # params		2	
- [C11] Load the pretrained weights	2		
Rundown (19)			
- [C12] Model training and log reporting	2		4
- [C13] Test set evaluation and video visualization	6		4
- [C14+Q10] Per-frame visualization	2	1	
Bonus			
- [C15+Q11] Study kernel size			
- [C16+Q12] Study group normalization			

Late submission will be accepted but with penalty.

The late penalty is deduction of one point (out of a maximum of 100 points) for every minute late after 11:59pm. Being late for a fraction of a minute is considered a full minute. For example, two points will be deducted if the submission time is 00:00:34.

9 Academic Integrity

Please refer to the regulations for student conduct and academic integrity on this webpage: `https://registry.hkust.edu.hk/resource-library/academic-standards`.

While you may discuss with your classmates on general ideas about the assignment, your submission should be based on your own independent effort. In case you seek help from any person or reference source, you should state it clearly in your submission. Failure to do so is considered plagiarism which will lead to appropriate disciplinary actions.