Hong Kong University of Science and Technology
COMP 4211: Machine Learning
Spring 2023

Programming Assignment 2
Due: 3 April 2023, 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 image generation on the Chinese Calligraphy 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 `numpy`, and `matplotlib` have been installed in your Colab environment. As opposed to the previous version of the assignment, there is no need to install `tensorflow-gpu`.

- `Python` version 3.9 and `TensorFlow` version 2.11.0, which is also the default setting of the Colab environment, 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".

- This assignment provides a compressed ZIP file named `pa2.zip` that includes several necessary files. It consists of a `train` subfolder that contains training images, a `test` subfolder that contains testing images, a Jupyter notebook file named `autoregressive_model.ipynb` that has skeleton code to build the autoregressive image generation model, and a weights file `pixel_cnn_e5.h5` that is utilized for loading the pretrained 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 Image Generation

Image generation is one of the fundamental computer vision tasks, referring to the process of generating new images that are visually realistic and similar to real-world images. It is widely used in many applications, such as super-resolution, photo editing and 3D modeling.

One approach to image generation is to use models that learn to predict the probability distribution of pixel values, given the values of all the previous pixels. These models generate images one pixel at a time, using the previously generated pixels to condition the generation of the next pixel.

We will load the images from the dataset, 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 criteria.

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 Chinese Calligraphy dataset (Section 4.1).
2. Build a CNN-based autoregressive backbone network (Section 4.2).
3. Load the pretrained model weights before training (Section 4.3).
4. Complete the whole rundown for training (Section 4.5).
5. Generate predictions and evaluate the performance (Section 4.4).
6. (Bonus) Build and analyze the effects of different model configurations (Section 4.6).

4.1 Dataset and Data Generator

The Chinese Calligraphy dataset can be found in two subfolders:

1. `./pa2/train` contains 42,000 JPG images for training
2. `./pa2/test` contains 10,500 JPG images for testing

You are recommended to navigate through the `numpy` tensors to visualize the content using `matplotlib`. You should be able to obtain the calligraphy examples as shown in Figure 1.

You need to define a custom dataset class `CalligraphyDataset` using `tf.keras.utils.Sequence`,

Figure 1: Examples from the Calligraphy dataset

so that you can customize your input images before feeding them into the network. 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`).

[C1] To build a custom dataset, the first thing is to initialize the dataset class `CalligraphyDataset`. You need to load the data from either of the two sub-directories. The `__init__` function of `CalligraphyDataset` should define the following class attributes:

1. Batch size of the data to be fed into the network.
2. Directory from which data are loaded.

Apart from the aforementioned class attributes, you may define more if necessary.

The second thing is to implement the `CalligraphyDataset.__len__(self)` function, which returns the total number of batches.

[C2] Another function you need to implement is `CalligraphyDataset.__getitem__(self, idx)`. This function returns two `tf.tensor` objects with shape (batch_size, height, width, channels). Before returning the tensors, you also need to perform the following data transformation operations:

1. Convert the images to gray scale.
2. Resize the images to the size of 32×32 pixels.
3. Normalize the images to ensure that all elements are in the range from 0 to 1.
4. Binarize the image. Assign a value 0 to a pixel if its intensity is less than 0.33. Otherwise, assign a value 1.

Please note that for some of the tasks in the later part of this assignment, you are expected to perform holdout validation. Make sure that you can generate all training, validation and testing datasets. It is suggested to split the data from `./pa2_data/train` into training set and a validation set in a 80:20 ratio.

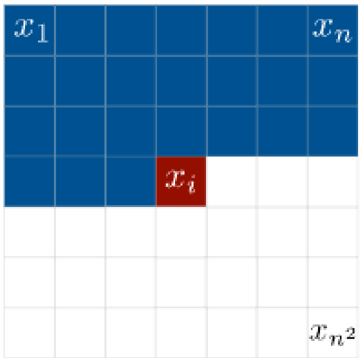

Figure 2: Example of autoregressive image modeling

4.2 Model Backbone

This assignment involves creating a convolution-based model to generate images, where pre-trained model weights will be loaded and utilized to verify the correctness of your implementation. The model consists of two main components: *causal convolutions* and *gated residual blocks*. Before proceeding with the architecture and its building blocks, it is recommended that you gain familiarity with the background knowledge that is essential for completing the assignment successfully.

4.2.1 Background Information

Autoregressive image modeling. Natural images are typically represented as a 3-dimensional variable, with dimensions of $H \times W \times C$, where the final dimension denotes the color channel. Autoregressive models process images by first imposing an ordering and then modeling the likelihood of a pixel given all previous ones. Thus, the autoregressive model for high-dimensional data $\mathbf{x}$ factors the joint distribution into the product of conditionals, as shown in Equation 1. Figure 2 illustrates the modeling of pixel x_i as a conditional probability distribution based on all previous (blue) pixels.

$$p(\mathbf{x}) = p(x_1, \ldots, x_{n^2}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1}). \tag{1}$$

While autoregressive models are strong likelihood-based image generation models, their sampling process can be slow because generating one image requires $H \times W$ forward passes. Therefore, in this assignment, we will introduce causal convolution to accelerate the feature extraction process instead of relying on pixel sampling.

Causal convolution. To efficiently model images, a convolution-based model is preferred over RNN or LSTM models. However, regular 2D convolutions cannot be directly applied as they violate the causal constraint, which dictates that the prediction for a given pixel should only be influenced by its previous pixels, not future ones.

To achieve the "raster scan" order as shown in Figure 3, where the left pixels come before the right pixels and the top rows before the bottom rows, padding is added to the top left of the input tensor along the height and width dimensions. Specifically, if the size of the filter is k, then $k-1$ zeros are added to the beginning of the input tensor along the axis. This ensures that the model cannot use information from pixels that will be generated in the future to predict the current pixel. For example, as shown in Figure 4, when generating the first pixel

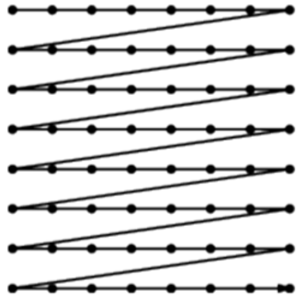

Figure 3: Raster scan order

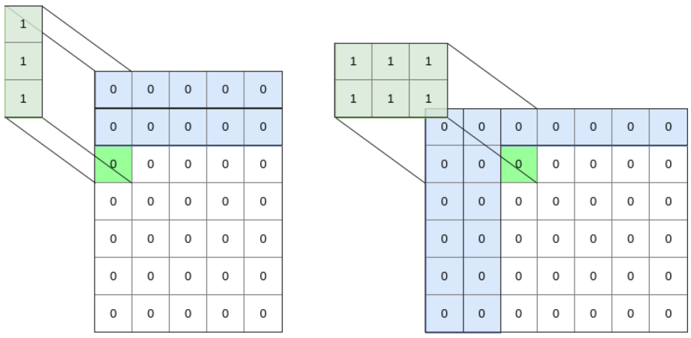

Figure 4: Causal convolution

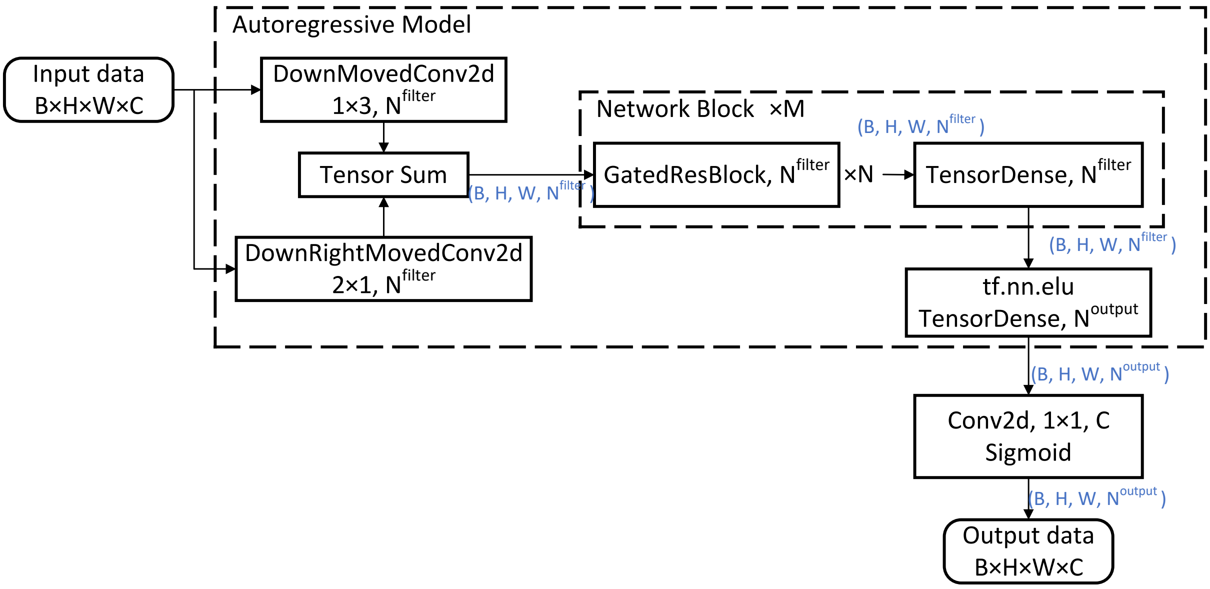

Figure 5: Model architecture overview

in the image, the model will only have access to the top left pixel of the input image and zeros due to the padding, limiting the information available to the model. **You can find more information in the "Shift-and-Crop with Regular Convolution" section of** `https://thomasjubb.blog/autoregressive-generative-models-in-depth-part-3/`.

Table 1: Notation

Symbol	Value	Meaning
Input & output		
B	32	batch size
N^{filter}	64	number of filters
N^{out}	10	number of output feature channels
H	32	height of an input image
W	32	width of an input image
C	1	number of channels of an input image
Model parameters		
N	6	number of repeated GatedResnets in one network block
M	6	number of repeated network blocks
Layer definitions		
Tensor Sum		sum two tensors
Tensor Concat		concatenate two tensors channel-wise
Slice		slice a tensor into two tensors
DownMovedConv2d		conv2d layer with causal padding, described in Section 4.2.3.5
DownRightMovedConv2d		conv2d layer with causal padding, described in Section 4.2.3.5
TensorDense		densely-connected NN layer
GatedResnet		core block which will be described in Section 4.2.3.5

4.2.2 Model Overview

As shown in Figure 5, the backbone architecture of the model we are trying to implement mainly consists of **causal convolutions** (`DownMovedConv2d` and `DownRightMovedConv2d`), **MLP layers** (`TensorDense`) and **gated residual blocks** (`GatedResnet`). Detailed descriptions of the symbols and parameters involved can be found in Table 1.

The typical flow of input through the network is as follows:

1. The network takes in input of (B, H, W, C).
2. The input is first processed by two blocks that are designed to maintain causality and avoid information leakage. The specifics of these blocks will be explained in Section 4.2.3.
   (a) `DownMovedConv2d` layer followed by a `down_move` function
   (b) `DownRightMovedConv2d` layer followed by a `right_move` function
   (c) The output tensors of the two layers are summed up
3. The output from the previous step passes through a `NetworkBlock`, which consists of N `GatedResnet`s and one layer of `TensorDense`.
4. Repeat step 3 for M times.
5. The output passes through one `tf.nn.elu` activation function then one `TensorDense` layer.

6. Finally, it passes through one `Conv2d` layer to reshape it to (B, H, W, C). After that, a `sigmoid` activation function is applied to pixels to predict a value between 0 and 1. You do not have to do anything for this step, as its definition is already provided in the code.

4.2.3 Components of the Model

For easier development of our autoregressive image generation model, we can start by completing its individual sub-modules before assembling them into the final model.

4.2.3.1 Provided Functions This sub-section introduces the provided functions in the skeleton code. You do not have to do anything for it.

- `down_move`: shifts the features down in the height dimension by padding zeros to the top and dropping the bottom pixel values. It is used to avoid information leakage in a causal network.

- `right_move`: shifts the feature right in the width dimension. It is used to avoid information leakage in a causal network.

- `concat_elu`: an activation function. It duplicates the input tensor, with one copy of it passing through the positive part of the ELU activation function, while the other copy of the input passing through the negative part of the ELU activation function. Note that this non-linearity function doubles the depth of the input tensor. You can find more information in `http://arxiv.org/abs/1603.05201`.

[Q1] How does the ELU function differ from the ReLU function?

4.2.3.2 Down-Right Moved 2D Convolution. This class inherits the `keras.layers.Layer` object and aims to build the causal convolution block. Its `__init__` method should take in the arguments specified in Table 2.

Parameter	Data Type	Default Value	Description
`num_filters`	integer	N/A	the number of output filters of the 2D convolution
`filter_size`	integer or a list of 2 integers	[2,2]	the size of the filter of the 2D convolution
`strides`	integer or a list of 2 integers	[1,1]	the stride of the 2D convolution
`padding`	string	'valid'	the type of padding to be applied to the input tensor ('valid' or 'same')
`activation`	string	None	the activation function to be applied after the convolution operation (e.g., 'relu', 'tanh', 'sigmoid')

Table 2: Arguments of the `__init__` method of `DownRightMovedConv2d`. Default value with N/A means it is not set.

Prior to performing the 2D convolution, it conducts a special padding operation. Specifically, the **zero** padding is added to the height and width dimensions of the input tensor. The height dimension is padded at the top with [height of its `filter_size` -1] rows and not padded at the bottom. The width dimension is padded on the left with [width of its `filter_size` -1] columns and not padded on the right. The `TensorFlow` built-in function `tf.pad` may be helpful.

You should also perform kernel initialization for the convolution using `tf.keras.initializers.RandomNormal`, where the mean is set to 0 and standard deviation is set to 0.05.

[C3] Implement the `__init__` and `__call__` methods of `DownRightMovedConv2d`.

[Q2] Can you explain the difference between using `'same'` and `'valid'` for the `padding` parameter in `tf.keras.layers.Conv2D`? Given that `padding = 'same'`, `strides = 1`, `kernel_size = 5`, `filters =` $N^{filters}$, and all other parameters use their default values, can you explain how the padding is applied to an input tensor with dimensions (B, H, W, C)?

[Q3] How does the `padding` parameter in `DownRightMovedConv2d` ensure that the convolution is causal? In other words, how is the convolution operation designed to ensure that each pixel in the output tensor only depends on the previous pixels along the raster scan order, i.e., processed from left to right and from top to bottom?

4.2.3.3 Down Moved 2D Convolution.

This class is very similar to `DownRightMovedConv2d`:

- They share exactly the same arguments in the `__init__` method.
- They conduct padding before the 2D convolution operation.
- Same intialization for the kernel of the 2D convolution operation.

The only **difference** is in the padding. The height dimension is padded at the top with [height of its `filter_size` -1] rows and not padded at the bottom. The width dimension is padded equally on both sides with [(width of its `filter_size` -1)/2] columns.

[C4] Implement the `__init__` and `__call__` methods of `DownMovedConv2d`.

4.2.3.4 Customised Dense Operation.

The `TensorDense` class, inheriting the `keras.layers.Layer` object, performs a dense tensor operation with the reshaping of its input. Its `__init__` method accepts the following arguments:

- `num_units` (integer): defining the number of output units of the dense layer
- `activation` (string): the activation function to be applied after the dense layer

The `TensorDense` layer is designed to operate on a 4-dimensional tensor of shape (B, H, W, C). When the `call` method is called, it reshapes the input tensor to a 2-dimensional tensor of shape $(B \cdot H \cdot W, C)$, then applies a `keras.layers.Dense` layer to the reshaped tensor. Finally, it reshapes the output tensor back to the original shape of (B, H, W, C).

Remember to initialize the weights of the dense layer using `tf.keras.initializers.RandomNormal` with mean 0 and standard deviation 0.05.

[C5] Implement the `__init__` and `__call__` method of `TensorDense`.

4.2.3.5 Gated Residual Block. The `GatedResnet` class applies gated residual connections to the input tensors for feature extraction. The `__init__` method takes in a number of arguments:

- `num_filters` (integer): specifies the number of filters to be used in the neural layers
- `activation` (function): the activation function to be applied after the dense layer. The default value is `concat_elu`, which is defined in the skeleton code.

In the `__init__` method, two individual instances of the `DownRightMovedConv2d` object are created. Here they are referred to as `nnLayer_1` and `nnLayer_2`, and are initialized with `num_filters` and `num_filters *2` as their number of filters respectively.

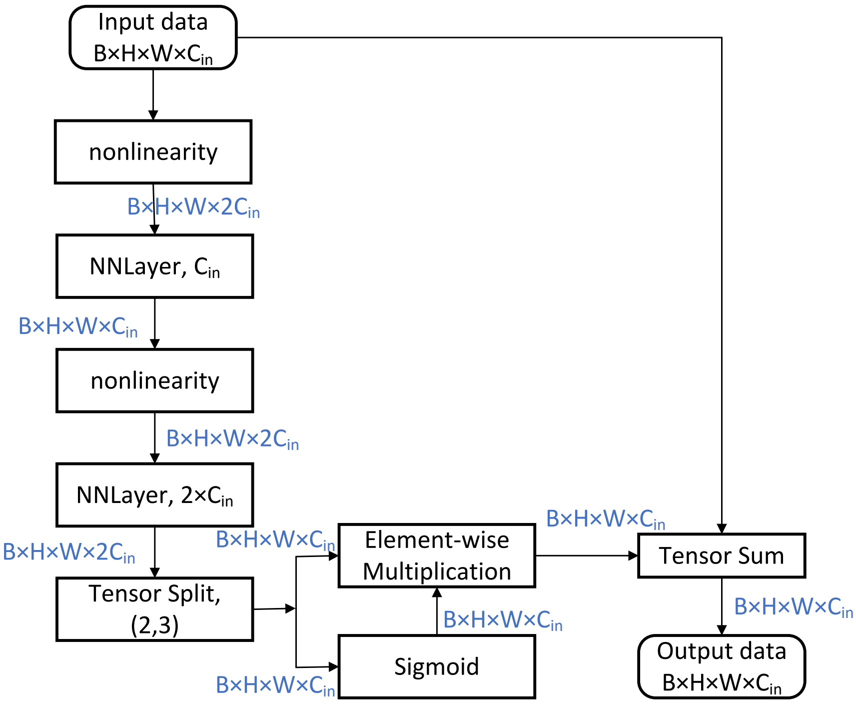

Figure 6: Gated residual block architecture

The `call` method accepts an input tensor of size (B, H, W, C). The `GatedResnet` consists of three parts:

1. **Feature extraction with NN layers**: The input first passes through the `activation` function defined in `__init__`, then through `nnLayer_1` for feature extraction. Then, it passes through the `activation` function again and `nnLayer_2`. The output tensor shape is $(B, H, W, 2C)$.

2. **Feature Gated Strategy**: We split the resulting tensor of the `nnLayer_2`along the channel dimension into two equal parts: the feature tensor F and the gate weight G. That is, both tensors have identical dimensions of (B, H, W, C). Finally, the output is given by the element-wise multiplication of F and Sigmoid(G).

3. **Residual Connection**: The final output tensor is given by the sum of input of `GatedResnet` and output of **feature gated strategy**, as illustrated in Figure 6.

[C6] Implement the `__init__` and `__call__` methods of `GatedResnet`.

[Q4] What is the total number of trainable parameters in `GatedResnet` when `num_filters` is set to N^{filter}?
[Q5] What is a residual connection? How can it be beneficial for training deep neural networks?

4.2.4 Putting it all together

Now, let us put it all together to form the superior model layer `AutoregressiveModel`. This class inherits the `keras.layers.Layer` object and aims to combine all basic components and form the model architecture shown in Figure 5. Firstly, the arguments of the `__init__` method are specified in Table 3.

Parameter	Data Type	Description
`n_filters`	integer	the number of main filters in our network, i.e., N^{filter} in Table 1
`n_resnet`	integer	the number of `GatedResnet` blocks in each `NetworkBlock`, i.e., N in Table 1
`n_block`	integer	the number of attention blocks, i.e., M in Table 1
`n_output`	integer	the number of channels of the output tensor for `AutoregressiveModel`

Table 3: Arguments of the `__init__` method of `AutoregressiveModel`

Secondly, the `__init__` method should define the variables specified in Table 4.

Parameter	Data Type	Initialization Values
`self.down_moved_conv2d`	`DownMovedConv2d`	`num_filters = self.n_filters` `filter_size =` $[1, 3]$
`self.down_right_moved_conv2d`	`DownRightMovedConv2d`	`num_filters = self.n_filters` `filter_size =` $[2, 1]$
`self.out_dense`	`TensorDense`	`num_units = self.n_output`
`self.ul_list_gated_resnet`	a list of `GatedResnet`	`num_filters = self.n_filters`
`self.ul_list_dense_layer`	a list of `TensorDense`	`num_units = self.n_filters`

Table 4: Variables to be initialized in the `__init__` method of `AutoregressiveModel`

Finally, you should implement the `call` function with reference to Section 4.2.2.

[C7] Implement the `__init__` and `__call__` methods of `AutoregressiveModel`.

4.2.5 Attribute Naming in Each Class

In total, you need to implement 5 classes inherited from `keras.layers.Layer`. In order to load the pretrained weights in Section 4.3 successfully, you should strictly follow the attribute naming listed below:

1. `AutoregressiveModel`
    - refer to Table 4 for attribute naming

- the order of variable initialization will affect the loading of pre-trained weights, so try to initialize the data members in this way:

```
        self.out_dense = TensorDense(self.n_output)
        self.ul_list_gated_resnet = []
        self.ul_list_dense_layer = []
        .... < loops for self.ul_list_gated_resnet
        and self.ul_list_dense_layer > ...
```

2. `GatedResnet`

   - `self.nnLayer_1`: the first `DownRightMovedConv2d` with number of filters `num_filters`.
   - `self.nnLayer_2`: the second `DownRightMovedConv2d` with number of filters `num_filters` $\times$ 2.
   - `self.nonlinearity`: the nonlinearity layer, with `concat_elu` as default.

3. `DownMovedConv2d`

   - `self.filter_size`: number of filters in the convolution layer.
   - `self.conv`: An initialized `keras.layers.Conv2D` with parameters initialized by `tf.keras.initializers.RandomNormal` with `mean` 0.0 and `stddev`=0.05.

4. `DownRightMovedConv2d`

   - `self.filter_size`: number of filters in the convolution layer.
   - `self.conv`: An initialized `keras.layers.Conv2D` with parameters initialized by `tf.keras.initializers.RandomNormal` with `mean` 0.0 and `stddev`=0.05.

5. `TensorDense`

   - `self.num_units`: number of filters in `keras.layers.Dense` layer.
   - `self.dense`: An initialized `keras.layers.Dense` with parameters initialized by `tf.keras.initializers.RandomNormal` with `mean` 0.0 and `stddev`=0.05.

4.3 Load the Pretrained Weights

[C8] 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 to load the model weights (`pixel_cnn_e5.h5`) in the `pa2` 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

The performance of an image generation model can be evaluated using either qualitative methods, which involve evaluating the generated images based on personal judgement, or quantitative methods, which involve evaluating the images using scoring rules or metrics. Both approaches can provide valuable insights into the performance of the model and help to identify areas for improvement.

[C9 + Q6] To perform qualitative evaluation, you need to generate 10 new images using the trained model. The input images should be selected from the test set. Recall that since the model generates each pixel based on its conditional probability, the image generation is done sequentially, pixel by pixel. To do this, first initialize an empty frame with the same size as the input image. Then, iterate over each pixel in the image, starting from the top-left corner, and generate the model's prediction for the next pixel value. You should also apply the following post-processing procedures on the newly generated pixel:

1. Add noise to the newly generated pixel using a function like `tf.random.uniform` in order to introduce some randomness in the generation process. This can help prevent the model from always generating the same pattern.

2. Apply thresholding to the image, such that all predicted values are mapped to either 0 or 1.

[C10 + Q7] For quantitative evaluation, report the **binary cross entropy loss** against the test set. It is not necessary to run the test multiple times - reporting the mean and standard deviation of the testing loss is not required.

[Q8] Consider the function `tf.keras.losses.BinaryCrossentropy`. When should we set the parameter `from_logits` as `True`? When should we set it to `False`?

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.

- Load the pretrained weights. Details are in Section 4.3

- [C11] Train your model. Your model should be trained for 10 epochs after loading the pretrained model. The suggested batch size is 32. You should use the Adam optimizer with the following parameters: `learning_rate = 0.0001, beta_1=0.95, beta_2=0.9995, epsilon=1e-6, use_ema=True, ema_momentum=0.9995`. During the training, it can be beneficial to print the validation loss to monitor for overfitting. It is recommended to use the model with the lowest validation error for evaluation.

- Conduct evaluation on the test set. Details are in Section 4.4.

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.*

The following two bonus questions require you to modify the network, which means that the pre-trained model cannot be used. Try to set the number of epochs larger to make sure that your network converges.

[C12+Q9] **Study Model Parameters**
How does varying the number of filters `n_filters` affect the performance of the `GatedResnet` blocks? Can we expect any improvements in performance if we use different numbers such as 32, 128, or 256 instead of the default value of 64? Please explain the impact of these changes on performance.

Additionally, you can also experiment with other model parameters such as `n_resnet`, `n_block`, and `n_output`. Please select two parameter settings. Report the experimental results and provide analysis to your findings. Visualization such as line charts are preferred.

[C13+Q10] **Study Loss Function**
To generate images in this project, we treat the task as a binary classification task where each pixel in the image is classified as either 0 or 1. Can you use other classification losses or even regression losses to predict values between 0 and 1, instead of using the default `binary_crossentropy` loss function? You may want to try out the following loss functions:

1. `tf.keras.losses.BinaryFocalCrossentropy` for classification task

2. `tf.keras.losses.MeanAbsoluteError` for regression task

Note that in the regression task, if necessary, you can customize the activation function used in the output layer to be something other than the default `sigmoid` function. You are free to change the optimizer or adjust your learning rate. Besides, the format of the input data should also be changed to `float` value between 0 and 1 instead of `int` value 0 or 1 in the classification task.

Select the above two losses for the classification task and regression task respectively. Please provide analysis of your experimental results and compare the generated images from different settings in your report.

5 Written Report

Answer [Q1] to [Q8] ([Q1] to [Q10] 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 and running processes should be recorded 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 5: [C]: Code, [Q]: Written report, [P]: Prediction

Grading Scheme	Code (70)	Report (22)	Prediction (8)
Dataset and Data Generator (9)			
- [C1] Build `init` and `len` functions	4		
- [C2] Build `getitem` function	5		
Model (63)			
- [Q1] ELU vs. ReLU		2	
- [C3] Build `DownRightMovedConv2d`	12		
- [Q2] `padding` parameter		3	
- [Q3] Causal convolution		4	
- [C4] Build `DownMovedConv2d`	8		
- [C5] Build `TensorDense`	8		
- [C6] Build `GatedResnet`	12		
- [Q4] Trainable parameters in `GatedResnet`		3	
- [Q5] Residual connection		2	
- [C7] Build `AutoregressiveModel`	7		
- [C8] Load the pretrained weights	2		
Evaluation (20)			
- [C9+Q6] Qualitative evaluation	5	3	4
- [C10+Q7] Quantitative evaluation	3	3	
- [Q8] Binary cross-entropy		2	
Rundown (8)			
- [C11] Model training and log reporting	4		4
Bonus			
- [C12+Q9] Study model parameters			
- [C13+Q10] Study loss function			

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.