Hong Kong University of Science and Technology
CSIT 6000M: Recent Advances in Deep Learning
Spring 2023

Project 1
Due: 27 March 2023, Monday, 11:59pm

1 Preamble

1.1 Introduction

This is an individual project. Through using and developing software to solve certain tasks, you will gain both practical skills and a deeper understanding of several models covered in this course.

The project is composed of two main parts:

Image classification on the number of items: Given a set of images in the form of 3-dimensional tensors and the software implementation of a classification model based on a convolutional neural network (CNN), follow the instructions provided to solve several tasks and present the results obtained.

Video prediction on a spatio-temporal sequence: For this part, the dataset is extended from a set of images to a set of image sequences. Using a modified U-Net which is another CNN-based model, we will predict the upcoming t frames based on the preceding t frames.

These two parts will be elaborated separately in Section 3 and Section 4.

1.2 Implementation

In this project, no starting code is provided. You are recommended to start with an empty `Jupyter notebook` (or `Colab notebook`) and follow the tasks in the following sections.

Please implement your models using either `PyTorch` or the `Keras` API of `Tensorflow 2`, which are the most popular choices in the deep learning community with numerous online learning resources. You may also find the following Python libraries useful:

- `NumPy`
- `Matplotlib` (or `Seaborn`)
- `Scikit-Learn`

You can also use other common packages in machine learning and data science (i.e., those available in Google Colab).

1.3 Computing Facilities

You are recommended to use Google Colab (`https://colab.research.google.com`), Kaggle Notebooks (`https://www.kaggle.com/code`), or Gradient Community Notebooks (`https://gradient.run/free-gpu`) to work on this project, for they generously offer free GPU resources to the public. Since the performance and limitations of these cloud computing services may vary, you need to explore and see which one performs best for you. You can also use other computing facilities available to you.

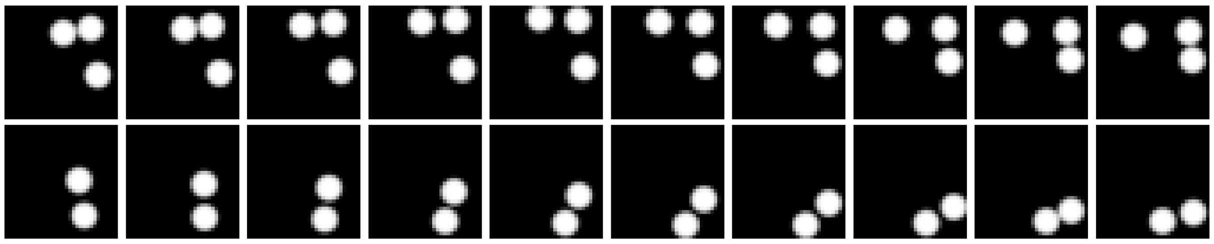
Figure 1: Visualization of sample sequences of the dataset for $T = 10$.

2 Dataset

We will use a modified subset of the **Bouncing Ball Dataset.**[1] The resulting dataset is a set of single-channel (greyscale) image sequences with size N. There are totally T frames in each sequence. Each frame consists of n balls moving on a 2D plane. In this project, we can always assume $n \in [1, 3]$. Overall, the image data is stored in 3D arrays with shape $(N, T, H \times W)$, where H and W are the height and width, respectively, of the image.

We provide three files for the data as described in Table 1 below. The format and usage of each file will be further explained in their corresponding sections.

Table 1: A brief overview of the three files provided.

File Name	Content & Shape	Remarks
`part1_bballs_training.npz`	`data`: (500, 1, 1024) `labels`: (500,)	Training and validation data for Part 1
`part1_bballs_test.npz`	`data`: (200, 1, 1024) `labels`: (200,)	Test data for Part 1
`part2_bballs_sequence.npz`	`data`: (5000, 10, 1024) `labels`: (5000,)	Training and validation data for Part 2

Pure CNN models do not support a temporal dimension for the input. At the moment, you can simply treat the channel dimension as the temporal dimension. Hence, for a 5-frame sequence of greyscale images, we will treat it as a single image with five channels. Besides, for consistency, we will adopt the channel-first convention (T, H, W) when describing the dimensions of the images. However, in your actual implementation, you can still use the channel-last convention (H, W, T) if you wish.

3 Part 1: Image Classification on the Number of Balls

In this part, we will use a simple CNN model for image classification. Given an image with shape $(1, H, W)$, the model should output the number of balls $n \in [1, 3]$. In other words, the model outputs the probabilities for three possible classes, which correspond to the prediction of 1 ball, 2 balls, and 3 balls.

3.1 Data Loading and Preprocessing

The `.npz` files are zipped archives of NumPy arrays. They can be loaded using the function `numpy.load()` with `allow_pickle=True`. All the provided files, after loading, are Python dictionaries with two key-value pairs, `data` and `labels`, where `data` stores the flattened image as an array in shape $(N, T, H \times W)$ and `labels` stores an N-vector with the i-th item indicating

[1]Ilya Sutskever, Geoffrey E. Hinton, and Graham W. Taylor. The Recurrent Temporal Restricted Boltzmann Machine. *NeurIPS*, 2008.

the number of balls of the i-th array in `data`. In this part, since classification is performed per image, each sequence consists of only a single frame ($T = 1$).

We need to first perform a training-validation split on `part1_bballs_training.npz` such that the first 80% of the data go into the training set and the remaining 20% go into the validation set. As the data has an overall shape of $(500, 1, 1024)$, the training set will have shape $(400, 1, 1024)$ and the validation set will have shape $(100, 1, 1024)$. It is not necessary but optional to shuffle the data. Furthermore, you need to manually reshape the flattened vector $(T, 1024)$ back to the proper image shape $(T, 32, 32)$.

Hint: For PyTorch, you will need a dataset and a dataloader to perform model training. You can use `torch.utils.data.TensorDataset` to convert the data to a PyTorch dataset. On the other hand, Keras natively supports array-like data as input.

3.2 Model

You need to implement the CNN model specified in Table 2 below. Note that zero padding may be required in some of the layers. You need to input the correct value such that the output shape is consistent with that in the table.

Table 2: Model architecture for Part 1 where B indicates the batch dimension.

Layer	**Kernel size**	**No. of filters / hidden units**	**Stride**	**Output shape**
Model Input	-	-	-	(B, 1, 32, 32)
2D Convolution + ReLU	(3, 3)	16	1	(B, 16, 32, 32)
2D Convolution + ReLU	(3, 3)	32	1	(B, 32, 32, 32)
2D Max Pooling	(2, 2)	-	2	(B, 32, 16, 16)
2D Convolution + ReLU	(3, 3)	32	1	(B, 32, 16, 16)
2D Convolution + ReLU	(3, 3)	16	1	(B, 16, 16, 16)
2D Max Pooling	(2, 2)	-	2	(B, 16, 8, 8)
Flatten	-	-	-	(B, 1024)
Fully Connected + ReLU	-	64	-	(B, 64)
Fully Connected (Model Output)	-	3	-	(B, 3)

[TASK 1] Implement the model shown in Table 2. In the report, show the model summary (For PyTorch, please use the `torchsummary` module).

Hint: This is a sequential model. You may want to use `torch.nn.Sequential` or `keras.Sequential` for simplicity.

3.3 Training and Validation

After completing the dataloader and the model, implement the training loop. For classification tasks, we will adopt the **cross-entropy loss**. The model in Section 3.2 outputs three logits from its last layer, which have an unbounded range of values. To map the logits back to probability scores, you can either apply the softmax function or use `torch.nn.CrossEntropyLoss` in PyTorch or `keras.losses.SparseCategoricalCrossentropy` with `from_logits=True` in Keras.

In addition, the following hyperparameters should be used:

- Optimizer: Adam (with default learning rate)
- Batch size: 4
- Number of epochs: 5

Perform validation at the end of every epoch and record the loss and accuracy of your model.

[TASK 2] Follow the configuration and hyperparameters shown above and start the training. Report (1) the training and validation losses over epochs and (2) the training and validation accuracies over epochs. Rather than simply listing the numbers, you need to visualize the trend by plotting a graph using `Matplotlib` or other libraries available.

3.4 Evaluation and Discussion

In this section, we will use `part1_bballs_test.npz` to evaluate the model trained in the previous section. Perform the loading and preprocessing in Section 3.1 to `part1_bballs_test.npz` except that there is no need to further split the test set. After loading, test the model on the loaded data.

Since the test set provided has a different data distribution compared to the training set (which is also common in other real-life datasets), the performance is expected to have a certain drop.

[TASK 3] Report the model performance on the test set. In the report, you need to show the test accuracy and a 3-by-3 confusion matrix. Besides, describe the results and discuss any noteworthy findings.

[TASK 4] Try tuning the hyperparameters or changing the model architecture. After that, start another model training such that it outperforms that in [TASK 3]. Report the new test accuracy and confusion matrix. Please clearly state your modifications and discuss why they work.

4 Part 2: Video Prediction on the Movement of Balls

Unlike the last part where only a single frame is used in the data for classification, in this part we will also consider the temporal dimension of the data sequences. Given a set of t consecutive images of the bouncing balls $\mathbf{y}_{[0:t]}$, our goal is to predict the t frames ahead. The model f can be written as:

$$\hat{\mathbf{y}} = f(\mathbf{y}_{[0:t]})$$

Here we adopt the Python-style slicing notation, where $\mathbf{y}_{[0:t]}$ denotes $[\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{t-1}]$. Different from the classification tasks, the output of the model should be an image sequence in which each image has the same dimension as the input images. For simplicity, we set $T = 10$ and $t = 5$. Thus, both the input and output shapes will be $(t, H, W) = (5, 32, 32)$, neglecting the mini-batch dimension.

4.1 Data Loading and Preprocessing

You will need to load `part2_bballs_sequence.npz` in this section. The format of the data is similar to the two used in Part 1, except that (1) $T = 10$ instead of 1, and (2) the `labels` field is no longer required, although it is still provided in the data.

Again, we first perform a training-validation split on the dataset such that the first 80% of the data go into the training set and the remaining 20% go into the validation set. This time, the data has an overall shape of $(5000, 10, 1024)$. The splitting will yield a training set of shape $(4000, 10, 1024)$ and a validation set of shape $(1000, 10, 1024)$. It is not necessary but optional to shuffle the data. Similar to the previous part, you need to manually reshape the flattened vector $(T, 1024)$ back to the proper image shape $(T, 32, 32)$. Remember to also split the data in the temporal dimension, such that the first $t = 5$ frames are sampled to be the training input while the last $T - t = 5$ frames are sampled to be the ground-truth samples.

4.2 Model

To perform video prediction on the bouncing balls, you will implement a modified 4-layer **U-Net**, a generic image-to-image model that is widely used in different applications. U-Net comprises two main components, the encoder, where the images are downsampled and converted into high-level features, and the decoder, where the high-level features are gradually upsampled and converted back to images. Between the two components, the intermediate outputs of the encoder blocks are concatenated to the inputs of the decoder blocks. A detailed illustration of our target U-Net is shown in Table 3 and Figure 2.

Table 3: The target U-Net architecture, decomposed into different components, where B indicates the batch dimension.

(a) Convolution Block (k filters)

Layer	Output Shape	Remarks
Module Input	(B, c, h, w)	-
2D Convolution + ReLU	(B, k, h, w)	kernel size = 3, stride = 1
2D Convolution + ReLU	(B, k, h, w)	kernel size = 3, stride = 1

(b) Transposed Convolution Block (k filters)

Layer	Output Shape	Remarks
Module Input	(B, c, h, w)	-
2D Convolution + ReLU	(B, k, h, w)	kernel size = 3, stride = 1
2D Transposed Convolution + ReLU	$(B, k, 2h, 2w)$	kernel size = 2, stride = 2

(c) Encoder

Module	Output Shape	Remarks
Module Input	$(B, 5, 32, 32)$	-
Convolution Block (32 filters)	$(B, 32, 32, 32)$	output: skip 1
2D Max Pooling	$(B, 32, 16, 16)$	kernel size = 2, strides = 2
Convolution Block (64 filters)	$(B, 64, 16, 16)$	output: skip 2
2D Max Pooling	$(B, 64, 8, 8)$	kernel size = 2, strides = 2
Convolution Block (128 filters)	$(B, 128, 8, 8)$	output: skip 3
2D Max Pooling	$(B, 128, 4, 4)$	kernel size = 2, strides = 2

(d) Decoder

Module	Output Shape	Remarks
Module Input	$(B, 128, 4, 4)$	-
Transposed Convolution Block (128 filters)	$(B, 128, 8, 8)$	-
Element-wise Addition	$(B, 128, 8, 8)$	with skip 3
Transposed Convolution Block (64 filters)	$(B, 64, 16, 16)$	-
Element-wise Addition	$(B, 64, 16, 16)$	with skip 2
Transposed Convolution Block (32 filters)	$(B, 32, 32, 32)$	-
Element-wise Addition	$(B, 32, 32, 32)$	with skip 1
Convolution Block (5 filters)	$(B, 5, 32, 32)$	-

(e) Modified U-Net

Module	Output Shape	Remarks
Model Input	$(B, 5, 32, 32)$	-
Encoder	$(B, 128, 4, 4)$	return output together with 3 intermediate features
Decoder	$(B, 5, 32, 32)$	accept input together with 3 intermediate features

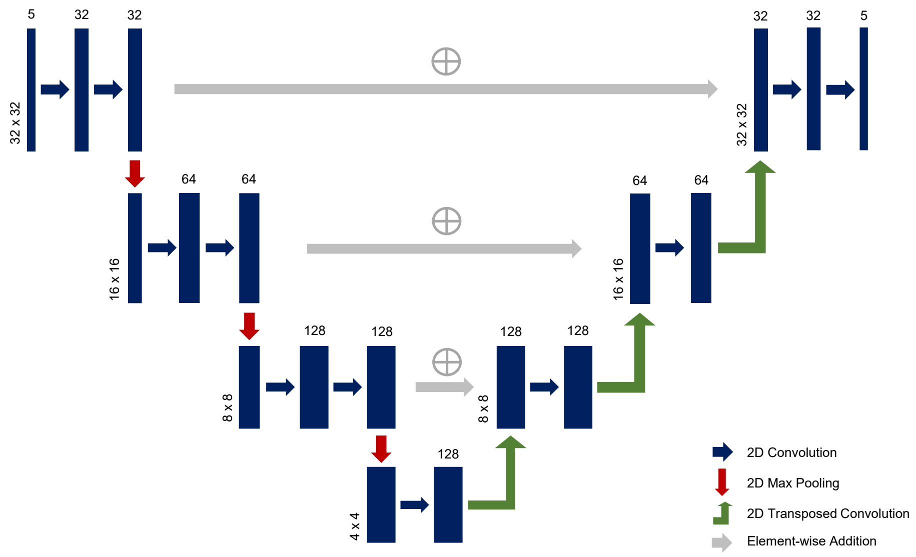

Figure 2: An overview of the target U-Net architecture.

Unlike the traditional U-Net where the channels are concatenated during skip connections, here we perform **element-wise addition**. Compared with concatenation, applying addition in the skip connections promotes learning the residual (or difference) between the input and output, which can help when the input and output are very similar. (Note: the specification differs from most online resources. Simply adopting implementations from the public repositories might get you wrong in this task.)

[**TASK 5**] Implement the U-Net based on the specification above. In particular, you need to define a Python class inheriting `torch.nn.Module` or `keras.Model`. You may also want to define sub-classes for convolution blocks and transposed convolution blocks. In the report, discuss the advantages of skip connections.

4.3 Training and Validation

We will use the Mean Squared Error (MSE) as the loss function in training the U-Net, which is defined as follows:

$$\mathrm{MSE}(\hat{\mathbf{y}}_{[0:T]}, \mathbf{y}_{[0:T]}) = \frac{1}{T}\sum_{t=0}^{T-1} (\mathbf{y}_{[t]} - \hat{\mathbf{y}}_{[t]})^2$$

where T here denotes the sequence length and the inner operation is an element-wise subtraction between two image arrays. In addition, the following hyperparameters should be used:

- Optimizer: Adam (with default learning rate)
- Batch size: 4
- Number of epochs: 20

To measure the similarity between the model prediction and the ground-truth frames, pixel-wise errors such as MSE could only give the absolute error between images, without considering the high-level details and human perception. Therefore, we will use two more metrics that align better with human vision to assess image quality.

First, we will evaluate the average Structural Similarity (SSIM) index of the two sequences, which is defined as:

$$\text{SSIM}(\hat{\mathbf{y}}_{[0:T]}, \mathbf{y}_{[0:T]}) = \frac{1}{T}\sum_{t=0}^{T-1} \frac{(2\mu_{\hat{\mathbf{y}}_{[t]}}\mu_{\mathbf{y}_{[t]}} + c_1)(2\sigma_{\hat{\mathbf{y}}_{[t]}\mathbf{y}_{[t]}} + c_2)}{(\mu^2_{\hat{\mathbf{y}}_{[t]}} + \mu^2_{\mathbf{y}_{[t]}} + c_1)(\sigma^2_{\hat{\mathbf{y}}_{[t]}} + \sigma^2_{\mathbf{y}_{[t]}} + c_2)}$$

where $\mu_{\mathbf{y}}$ is the pixel mean of $\mathbf{y}$, $\sigma_{\mathbf{y}}$ is the pixel variance of $\mathbf{y}$, $\sigma_{\mathbf{y}_1\mathbf{y}_2}$ is the covariance of $\mathbf{y}_1$ and $\mathbf{y}_2$, c_1 and c_2 are constants derived based on the value range of the pixels.

Moreover, we will also inspect the Peak Signal-to-Noise Ratio (PSNR) of the two sequences:

$$\text{PSNR}(\hat{\mathbf{y}}_{[0:T]}, \mathbf{y}_{[0:T]}) = 10\ \log_{10}(\frac{c^2_{\max}}{\text{MSE}(\hat{\mathbf{y}}_{[0:T]}, \mathbf{y}_{[0:T]})})$$

where $c_{\max}$ is the maximum available value of the image (peak signal). For example, $c_{\max} = 2^8 - 1$ for pixel values with range $[0, 255]$ and $c_{\max} = 1$ for normalized pixel values with range $[0, 1]$.

[TASK 6] Follow the configuration and hyperparameters shown above and start the training. Report the model performance quantitatively and qualitatively. Specifically, you need to report the following items:

1. a plot of the training loss over epochs
2. a plot of the validation loss over epochs
3. the validation SSIM (Structural Similarity) index
4. the validation PSNR (Peak Signal-to-Noise Ratio)
5. visualizations of the model output and the ground truth, in a format similar to Figure 1

Hint: You do not need to manually implement SSIM and PSNR. For example, `skimage` provides APIs that you can invoke directly. However, you still need to check the data value range and constants for a correct output score.

4.4 Customization and Discussion

In this section, you can perform any customization to the model and the hyperparameters. We seek to improve the model in any aspect, such as performance, output quality, efficiency, convergence rate, robustness, etc. Alternatively, you can also adopt another existing model or propose your own model. The only restriction is that you have to work on the same video prediction task using the same training subset of `part2_bballs_sequence.npz`.

Your new model/setup should be compared against two baselines. One is the setup in [TASK 6], in which you should already have the data. The other is the 'last frame' of the input. For example, with an input of $\mathbf{y}_{[0:t]}$ and the ground truth of $\mathbf{y}_{[t:T]}$, we can obtain the sequence by stacking the last input frame t times, forming a sequence $[\mathbf{y}_{[t-1]}, \mathbf{y}_{[t-1]}, ..., \mathbf{y}_{[t-1]}]$. Comparing against the last frame can reflect the capability of the model of deriving new information from the input sequence.

[TASK 7] After your customization, compare the new results with the aforementioned baselines. Clearly state your modifications, discuss why they work, and whether the actual result is within expectation. This task will be graded based on the workload, novelty, your justification and presentation in the report.

5 Assessment Components and Submission

There are two assessment components:

- Project report
- Source code

Note that this project should not be used for earning credits in a different course to avoid double-dipping.

5.1 Project Report

The report should cover the following aspects of the project:

- Student's full name, student ID, and HKUST email address
- Results, visualizations, and discussions of the tasks
- Description of any additional operations performed

5.2 Source Code

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 exhaustive:

- Using functions to structure program 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

All the source codes that you have written for this project should be submitted for grading. In case your code is modified from another source, you are expected to acknowledge it clearly in your report. Failure to do so is considered plagiarism.

Data files should not be submitted to keep the submission file size small.

5.3 Submission

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

Your submission should contain two files: *report* (`report.pdf`) and *compressed source code* (`code.zip`). 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.

6 Grading Guidelines

This project will be counted towards 25% of your course grade. The breakdown is as follows:

- Seven tasks [70 points in total]
- Clarity of results and discussions [10 points]
- Good use of visualization techniques [10 points]
- Good programming practices in source code [10 points]

Grading will be based on rubrics with five levels of achievement (excellent, good, satisfactory, unsatisfactory, poor) for each of the items above.

An important general criterion is clarity, to the extent that others can replicate your experiments based on the information provided in the report.

Late submissions will be accepted but with a penalty. The late penalty is a 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.

7 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 project, 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.