Dr. Tolga Soyata *© George Mason University, Department of ECE* *tsoyata@gmu.edu*

ECE655 (Advanced GPU Programming and Deep Learning) Project 4 (10 points)

Deadline: Oct 26, 2025 11:59PM. 2 point penalty per each late day

- In this project, you will be using your knowledge from Lectures 5-7 to apply binary image classification to a simple object recognition application. You are required to generate results using PyTorch.
- Document your results in a report, written in LaTeX (Overleaf). The report is expected to be 7-10 pages, which is packed with plots and tables.
- Name your code **`P4A.py`**, **`P4B.py`**, … as in the previous projects.
- Submit your report and code files under **Project 4 Submission Area**, *as a single ZIP file*, which must include:
  - Your Python source files and other necessary files for me to generate your results.
  - Your Report PDF and source files (the entire Overleaf directory)

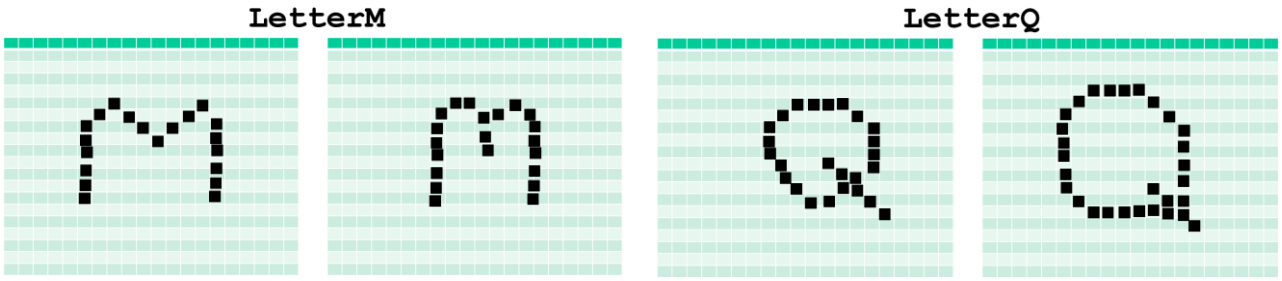
PROJECT 4 DESCRIPTION:
In this project, you are required to generate synthetic dataset containing 200 test images of two letters **Q** , **M**. Each image is specified as a 20x20 binary image, where each pixel is **`float`**.

PART A (1.5 points) SYNTHETIC DATA GENERATION:

- Create two tensors named **`LetterQ`**, **`LetterM`**.
- Both tensor sizes are (10, 1, 20, 20) in NCHW format:
  - N=10 since there are 10 images in each tensor.
  - C=1 since each image has a single channel (gray scale)
  - H=20, W=20, since each image has 20x20 pixels.
- Each tensor represents a 20x20 binary image, where each pixel is **`float`** and has either value 0.000 (black) or 255.000 (white).
- Synthetically generate 10 images of letter **Q**, and put them in the **`LetterQ`** tensor.
  - N dimension is the ImageID
  - C is a "ghost" dimension and will always have the index 0. We are including it to be compatible with the way PyTorch stores images in NCHW format.
  - H, W are the pixel row, column index, respectively.
  - The 10 images you create will have <u>different size</u> letter **Q**, but they do not have to be symmetrical and good-looking letters. See the examples below.

In this step, your **Q** letters must be centered in the 20x20 image; each must have exactly the same orientation (i.e., no rotation, no shifting). Later, you will create many different shifts and orientations, flips, etc. to augment your data.

- Generate 10 images of letter **M** in the same way and put them in the **`LetterM`** tensor.



PART B (1.5 points) DATA AUGMENTATION:

- In this step, you will augment your 20 letters (10+10) by a factor of 10, i.e., end up with 200 letters that are still **Q**s and **M**s and others.
- To augment, you can use rotation, shift, and horizontal and vertical flip. They are all operations you learned in the previous lectures.
- Put the resulting 200 images in a single tensor named **`Letters`**, which has the dimension (200, 1, 20, 20) according to the NCHW convention.
  - First 100 images in this tensor are **Q**s (indexes 0-99).
  - Next 100 images are **M**s (indexes 100-199).

PART C (2 points) DATA VISUALIZATION:

- Write a tensor visualization program that shows all 10 images of **`LetterQ`**, **`LetterM`**, and others in PART A. To make it more visually-pleasing, have a super-title that is "**`LetterQ`**, and **`LetterM`**." A very similar version of this is already in class projects. Recycle as much of my code as you can.
- Use the same program to visualize all augmented letters in PART B.
- Put these visualizations in your report.

PART D (5 points) BINARY IMAGE CLASSIFICATION (BIC):

- In PART D, use the augmented dataset.
- Use the two classes (**`Q`**, **`M`**) and create a model that distinguishes between these two characters (i.e., binary classification).
- In PART D, you are not allowed to use CNNs. Input your images as flat pixels. Don't worry about the dimension explosion.
- Split your **`LetterQ`** and **`LetterM`** tensors into two pieces: training (80%) and validation (20%).
- Treat your **`LetterM`** as the "positive" class and **`LetterQ`** as the "negative" class.
- Create a BIC model that produces the probability for two classes, i.e., p=1.000 means **`LetterM`** (positive) and p=0.000 is LetterQ (negative class). Clearly document the architecture of your model.
- It is up to you to choose the hyper-parameters, i.e., **`lr`** and number of epochs as well as the loss function and the optimizer. Document your choices.
- Train your model and plot the training/validation loss progression.
- Calculate TPR, FPR, and accuracy for a given threshold, $\boldsymbol{p = 0.5}$.
- Plot the ROC curve by using different thresholds.