**1/2023 Final Exam  20:00 1/12/2023 – 19:59 2/12/2023**
**- 2110431 Introduction to Digital Imaging**
**- 2147329 Digital Imaging and Vision Systems**

Name Punyaphat Surakiatkamjorn ID 6432106821

## Instructions:

1) There are 73 points in total for the examination for Problem 1-4.
2) This exam is "open book," which means you are permitted to use any materials handed out in class, your own notes from the course, the text book, internet searches, google, ChatGPT and anything
3) The exam must be taken completely alone. You cannot share your answers or code with anyone.
4) You can ask questions on discord in **#finalexam**, further announcement will be posted on **#announcement**
5) You must complete the exam within this **exam paper** and the **ipynb** files will be randomly inspected. Make sure you write "clear" answer to all questions.
6) You must submit
   **(1) This exam paper (in pdf)**
   **(2) .ipynb file (problem 2, 3) (without (2), received 50% of the score)**
   In MyCourseVille before 17:59, December 2, 2022. Late submission is not accepted.

THE EXAM starts on 20:00, December 1, 2023 and is **DUE** ON 19.59, December 2, 2023.

**Rules of Conduct for Students during Examinations**

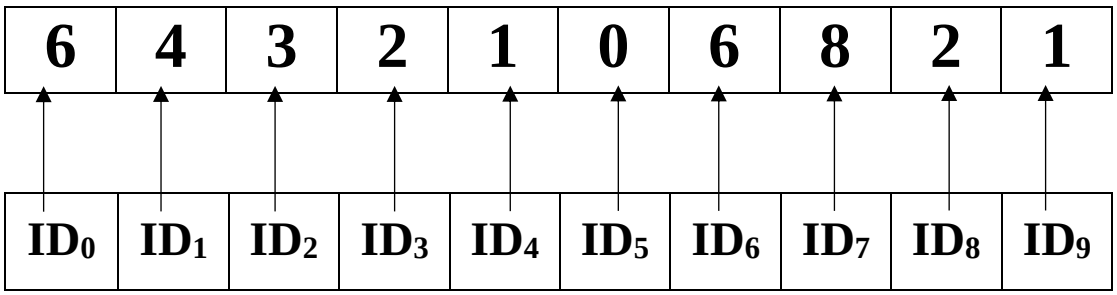
Unacceptable examination conduct includes

Share your answers or code with your friends or looking at exam materials of other students or anyone or allowing others to look at their exam materials.

When a student cheats or is in suspicion, the supervisory officer has the power to investigate the matter. There are serious consequences to academic misconduct, including receiving an “F” on the work or examination, an “F” in the course.

Name Punyaphat Surakiatkamjorn ID 6432106821

**Preliminaries:**
Fill in your ID here




**Problem 1 Thresholding & Morphology**
1.1 (3 Points) Generate a 4 × 10 image which is a combination of your id below

`image(x,y) =`

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $ID_0$ | $ID_1$ | $ID_2$ | $ID_3$ | $ID_4$ | $ID_5$ | $ID_6$ | $ID_7$ | $ID_8$ | $ID_9$ |
| $ID_9$ | $ID_8$ | $ID_7$ | $ID_6$ | $ID_5$ | $ID_4$ | $ID_3$ | $ID_2$ | $ID_1$ | $ID_0$ |
| $ID_0$ | $ID_1$ | $ID_2$ | $ID_3$ | $ID_4$ | $ID_5$ | $ID_6$ | $ID_7$ | $ID_8$ | $ID_9$ |
| $ID_9$ | $ID_8$ | $ID_7$ | $ID_6$ | $ID_5$ | $ID_4$ | $ID_3$ | $ID_2$ | $ID_1$ | $ID_0$ |

Then apply binary thresholding on your image using the condition below

$$\texttt{dst(x,y)} = \begin{cases} 0, & if\ image(x,y) > 4 \\ 1, & otherwise \end{cases}$$

**Answer**

`dst(x,y) =`

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

1.2 (3 Points) Apply **dilation** on the image resulted from 1.1 using a 3 × 3 structural elements with all values are ONE, zero padding and provide the result below

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

1.3 (3 Points) Apply **erosion** on the image resulted from 1.1 using a 3 × 3 structural elements with all values are ONE, zero padding and provide the result below

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

1.4 (7 Points) Calculate number of the **connected components** on the image resulted from 1.1 using a 3 × 3 structural elements ~~with all values are ONE~~, zero padding, connectivity = 4 and provide the **second** and the **last** iterative step below. The starting point starts from left to right and then top to bottom of the image.

1.4.1 Provide the result of the **second** iterative step:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

1.4.2 Provide the result of the **last** iterative step:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | **1** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **0** |
| **0** | **1** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **0** |

1.4.3 Answer the total number of the components = _____1_____

**Problem 2 Image Classification**

From the lecture "Object Recognition II" using CNN model, see https://colab.research.google.com/drive/19Cap6l8AdTmELvInC6pj4JkwWS68z9u9?usp=sharing

Apply CNN model in the code above with 7 bird species classification in the code prepared in https://colab.research.google.com/drive/1DyuQ4rjUYV8llf817zvMPoac6Gx6QiJZ?usp=sharing

copy and save the colab file into your drive and then run the code.

2.1 (5 points) Apply the model from the lecture "Object Recognition II" with `Birds20Tiny` dataset which contains 7-class of birds including Abbotts Babbler, Abyssinian Ground Hornbill, Albatross, Alexandrine Parakeet, Alpine Chough, American Bittern, and American Goldfinch. Provide the changes you have made below. You may change the variables and parameters and provide the selected values below here.

**Hint**: if the code doesn't work, (1) modify channels (MNIST contains grayscale images, Birds20Tiny uses RGB images) and the (2) Linear layer (or fully connected) for input image size of (3, 224, 224) -> (depth, width, height) (3) number of class

Your code changes:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=3,
                out_channels=48,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(48, 96, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 7 classes
```

```
        self.out = nn.Linear(96 * 56 * 56, 7)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)

        # flatten the output of conv2 to (batch_size, 96 * 56 * 56)
        x = x.view(x.size(0), -1)
        output = F.log_softmax(self.out(x), dim=1)
        return output


model = Net().to(device)
print(model)

summary(model, input_size=(3, 224, 224))

# Optimizing the Model Parameters
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
```

| Parameter | Value |
|---|---|
| Optimizer | SGD |
| Learning rate / Momentum (if applicable) | 1e-4 |
| Epochs | 75 |
| Kernel size | 5 |
| Number of layers | 7 |
| Batch size | 16 |
| Total parameters | 2,226,343 |
| Environment (Colab with X-XXX GPU, etc.) | Colab with T4 GPU |
| Others | input_size=(3, 224, 224) |

2.2 (5 points) Report the actual performance of your model, identify which kind of data you have used to measure the performance here and why you choose that kind of data.

| |
|---|
| By training at 75$^{th}$ epoch<br>Val Accuracy: 42.9%,  Avg loss: 1.029163 |

By testing accuracy
Test Accuracy: 42.9%,  Avg loss: 1.347635

Use test_dataloader(test_data) for measure the performance because the model never sees this kind of data before so it's a good dataset for testing the model

2.3 (5 points) Add one more convolutional layer with 64 feature output, kernel size = 5, stride = 1, padding = 2 followed by ReLU function and Max pooling (2 x 2 kernel), print out the model and paste your code and highlight all the changes below

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=3,
                out_channels=48,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(48, 96, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(96, 64, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 7 classes
        self.out = nn.Linear(64 * 28 * 28, 7)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
```

```
        # flatten the output of conv2 to (batch_size, 144 * 28 * 28)
        x = x.view(x.size(0), -1)
        output = F.log_softmax(self.out(x), dim=1)
        return output
```

and capture the model architecture using

```
summary(model, input_size=(3, 224, 224))
```

and paste the result in the blank box below

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 48, 224, 224]           3,648
              ReLU-2         [-1, 48, 224, 224]               0
         MaxPool2d-3         [-1, 48, 112, 112]               0
            Conv2d-4         [-1, 96, 112, 112]         115,296
              ReLU-5         [-1, 96, 112, 112]               0
         MaxPool2d-6           [-1, 96, 56, 56]               0
            Conv2d-7           [-1, 64, 56, 56]         153,664
              ReLU-8           [-1, 64, 56, 56]               0
         MaxPool2d-9           [-1, 64, 28, 28]               0
           Linear-10                    [-1, 7]         351,239
================================================================
Total params: 623,847
Trainable params: 623,847
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 65.46
Params size (MB): 2.38
Estimated Total Size (MB): 68.41
----------------------------------------------------------------
```

2.4 (3 points) Explain how to calculate learnable parameters in the 3rd conv layer and the final layer (linear / fully connected layer)

For the 3rd conv layer, the learnable parameters can calculate by using this formula: Number of Parameters=(Input Channels × Kernel Size + 1) ×Output Channels

for my 3rd conv layer is

```
self.conv3 = nn.Sequential(
            nn.Conv2d(96, 64, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
```

Number of Parameters = (96 × (5 × 5) + 1) × 64 = 153664

For the final layer, Use the formula

Number of Parameters = (Output Channels + 1) × Output Class

```
 self.out = nn.Linear(64 * 28 * 28, 7)
```

For my final layer, Number of Parameters = (64 × 28 × 28 + 1) × 10 = 351239

2.5 (10 points) In the colab link, I have added the example of data augmentation, select **three** more kind of transforms and provide the reason below

See more https://pytorch.org/vision/stable/transforms.html

| Selected transform + parameters | Reason |
|---|---|
| `transforms.RandomHorizontalFlip(0.5)` | Birds can appear in various orientations in images, and training the model with flipped images can improve its ability to recognize birds regardless of their left or right orientation. To avoiding over-reliance on flipped images, a 50% probability is balance between providing diverse training |
| `transforms.RandomRotation((-30,30))` | Birds may appear in various angles in images, and rotating the images during training ensures that the model learns features from birds in different orientations. A rotation range of -30 to 30 degrees allows for a diverse set of |

| | |
|---|---|
| | rotated images, helping the model become robust to different bird poses and orientations. |
| `transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0)` | Bird images in the wild can have diverse lighting conditions and color variations. By applying color jitter, the model learns to recognize birds under different lighting and color conditions, improving its robustness. Small adjustments in brightness, contrast, and saturation help the model become invariant to variations in lighting conditions and color. But random hue changes might lead to unrealistic color variations that are not representative of the natural environment. Setting hue=0 helps in avoiding such unrealistic transformations. |

Provide the code for your transform here and report the result.

```
data_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(0.5),
        transforms.RandomRotation((-30,30)),
        transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2, hue=0),
        transforms.ToTensor(),
])
```

Accuracy test result:
Test Accuracy: 57.1%,  Avg loss: 0.910942

If the result has increased or decreased from previous experiment? Analyze why?

It’s increased because applying the transformation increases the dataset's number and makes the dataset variant include enhanced robustness to variations in orientation, pose, lighting conditions, and color satuation, leading to a more versatile model and ready for more variant real-world data.

**Problem 3 U-Net**

See the example of using U-Net library in the class for human segmentation: https://colab.research.google.com/drive/1rWxBlRnjGW_7a0m7ykLiPYFZo3En_2Rf?usp=sharing

copy and save the colab file into your drive and then run the code.

3.1 (5 points) Provide the performance of the model on ~~testing data~~ validation data for human segmentation and identify the values of the parameters used in the model.

| |
|---|
| Best valid loss: 0.49149464443325996<br><br>Parameter:<br>ENCODER = 'vgg16'<br>WEIGHTS = 'imagenet'<br>in_channels = 3<br>classes = 1 |

3.2 (5 points) Try to change one thing (parameters/configurations) to improve the performance, identify which thing you have modified and what is the performance of the modified model.

| |
|---|
| Change encoder from ‘vgg16’ to ‘resnet18’<br>New best valid loss of the modified model<br>Best valid loss: 0.3666181340813637 |

3.3 (6 points) Apply U-Net model that you have trained with these 3 images (in the downloaded zip file). See the code in Colab Task 9.

| Original image | Segmented image (mask) |
| --- | --- |
|  “1.jpg” |  |
| Analyze the result | The model can accurately segment the first person, second person, and third person, but its performance is less robust for the last person. The presence of the clothing he are holding may be influencing the segmentation results. |

| Original image | Segmented image (mask) |
| --- | --- |
|  “2.jpg” |  |
| Analyze the result | It's challenging to confidently determine whether the subject is human due to intricate details in the picture. The model struggles with image quality and subtle color blending between the clothing and the road, leading to confusion and mis segmentation where the road is erroneously identified as part of the clothing. |

| Original image | Segmented image (mask) |
| --- | --- |
|  “3.jpg” |  |
| Analyze the result | This result indicates relatively accurate segmentation for the second walking person, but the performance declines for other individuals. The model struggles to segment people in the background, and for the first, third, and fourth persons, the model faces difficulty distinguishing between the background and the individuals. The inability to segment people in the background affects the model's overall categorization. |

3.4 (3 points) What do you think to use U-Net followed by Connected component for counting the number of people the image? Any advantages or disadvantages?

| |
| --- |
| Using a U-Net architecture followed by connected components analysis for counting people in images offers notable advantages. U-Net excels in semantic segmentation, capturing intricate details for accurate individual segmentation and providing an end-to-end solution with hierarchical feature utilization. This framework proves valuable for tasks demanding precise people counting, as demonstrated in the accurate segmentation of 1.png. If segmentation is accurate, connected components analysis ensures an accurate count. |

However, challenges include potential false positives and negatives in complex scenes, necessitating careful post-processing. The computational intensity of U-Net, especially with high-resolution images, poses limitations for real-time and resource-constrained applications. Balancing accurate segmentation with computational demands is crucial. Examples include false positives in 2.png, where connected components may overcount, and false negatives in 3.png, reflecting the model's inability to accurately segment people and background, leading to undercounting.

**Problem 4 U-Net + GAN (10 points)**

When deal with medical image, many cases, the image is ambiguous and unclear. A medical project needs to use image segmentation to segment particular areas from the image but the researcher plans to use U-Net for image segmentation and also apply super-resolution techniques using GAN model. Write a paragraph between 150 to 250 words to describe the steps to use these two methods, and also select a GAN model, if you can explain one of these: input/output/architecture for super-resolution, particularly, the key features for higher resolution. Only accept the writing with references.

In addressing the challenge of ambiguous medical images, the proposed project employs a two-phase strategy by integrating U-Net for precise image segmentation and harnessing a Generative Adversarial Network (GAN) for subsequent super-resolution. The initial phase entails training the U-Net architecture on annotated medical images to accurately identify and delineate specific regions of interest, facilitating the precise isolation of anatomical structures or abnormalities. Subsequently, the focus transitions to super-resolution using the SRGAN (Super-Resolution Generative Adversarial Network) model. SRGAN is characterized by a dual-network architecture comprising a generator, responsible for enhancing image resolution, and a discriminator, assessing the realism of the generated high-resolution images. Key features underpinning superior resolution in SRGAN include the incorporation of perceptual loss functions, integration of high-level feature representations, and the implementation of residual blocks and skip connections. These elements collectively contribute to the preservation of crucial image details during the upscaling process, ensuring the generation of high-resolution medical images with enhanced clarity and precision.

Reference:

https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28
https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5
https://towardsdatascience.com/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a
https://paperswithcode.com/method/srgan
https://ieeexplore.ieee.org/document/8099502
https://arxiv.org/abs/1902.06068
https://arxiv.org/abs/1603.08155

**Bonus point (1 point)**

- How long does it take for you to taking this exam: 7 hours 50 mins
- How do you think about this exam (write X in front of the answer):

____Easy ____ Slightly Easy ____Intermediate _X_ Slightly Hard _____Hard