COMP4423 – Computer Vision

Assignment 2

If you have any questions, please feel free to contact TA with email:
sheldon.shen@connect.polyu.hk

Pre-Notes:

Every task will be highlighted with an arrow and red words.

Every tip (including idea tips and code tips) will be underlined.

We'll give you the task breakdown and mark distribution to boost your performance with purple words.

We'll give you relevant files with orange words.

You are highly recommended to read the story before the task which is the background of the task.

The final grade for this assignment will be the minimum of 100 and the total score achieved. The maximum possible score is 150 (two parts: 80 + 70).

About submission:

Follow the steps below:

1. Name the .py file as Assignment2_<your_ID>_<your_name>.py.
   e.g., `Assignment2_12345678d_CHAN_Dawen.py`
2. Name the report as Assignment2_<your_ID>_<your_name>.pdf.
   e.g., Assignment2_12345678d_CHAN_Dawen.pdf
3. Compress the two files into a .zip file and rename the .zip file.
   e.g., Assignment2_12345678d_CHAN_Dawen.zip
4. Upload the .zip file to the blackboard system.

Warning:
If you are unable to complete the whole program, try to accomplish part of the tasks and make sure it can run successfully.

Any wrong file naming and submission will be given a ZERO mark in this assignment.

The deadline for this assignment is 00:00:00 30th March 2023.

Late submission penalty

10% is deducted for each day that the work is late. The penalty will be applied up to a maximum number of three days after and including the submission deadline day. After three days the work will be marked at zero.

This assignment is individual work. All work must be done on your own. Plagiarism is a serious offence. Copying code from web resources is prohibited as well. Any plagiarism case (for both the copier and the copiee) will be given a ZERO mark in this assignment.

Ok. Now let's begin our assignment 2 journey.

On a quiet Sunday afternoon, Sheldon, Alice, and Bob are sitting in the lab, staring at their computer screens.

Part 1: CNN Architecture Analysis

Their first task is to analyze CNN architecture and understand why certain improvements have led to significant performance gains. Sheldon initiates an academic discussion.

Task 1: ResNet vs VGG

Sheldon: "Alice, Bob, have you ever wondered why ResNet significantly outperforms traditional CNNs like VGG?"

Alice: "I trained both networks, but my VGG model struggles with deep layers. The loss fluctuates a lot."

Bob: "Maybe it's because of the gradient vanishment according to the lecture? "

Alice: "Let me derive the mathematical form of the gradient."

Alice grabs a marker and writes:

$$\mathbf{x}^{l+1} = \mathbf{x}^{l} + \mathbf{F}(\mathbf{W}^{l}, \mathbf{x}^{l})$$

where:

- x^l is the activation at layer l,
- $F(W^l, x^l)$ is the residual function $f(W^l x^l + b^l)$,
- The identity mapping x^l ensures information flow across layers.

Alice: "The key difference between ResNet and traditional CNNs is this additive identity mapping. Now let's differentiate both sides to analyze the gradient flow."

$$\frac{\partial L}{\partial x^l} = \frac{\partial L}{\partial x^{l+1}} \cdot \frac{\partial x^{l+1}}{\partial x^l} = \frac{\partial L}{\partial x^{l+1}}\left(I + \frac{\partial F}{\partial x^l}\right) = \left(\prod_{k=l}^{L}\left(I + \frac{\partial F}{\partial x^k}\right)\right)\frac{\partial L}{\partial x^L}$$

Alice: "Now, compare this to a traditional CNN like VGG, which lacks the identity mapping. In VGG, we would be multiplying only weight matrices and activation derivatives:"

$$\frac{\partial L}{\partial x^l} = \left(\prod_{k=l}^{L}\left(\frac{\partial F}{\partial x^k}\right)\right)\frac{\partial L}{\partial x^L}$$

where:

- $\frac{\partial F}{\partial x^k}$ corresponds to $W^k f'(x^k)$

Alice: "So in VGG, the gradient is a product of weight matrices and activation derivatives. If the singular values of W^k are small, gradients shrink exponentially!"

Bob: "And if the singular values are large, gradients explode, making training unstable."

Alice: "Right! But in ResNet, we have an identity term at every step, ensuring that even if $\frac{\partial F}{\partial x^k}$ is small, the gradient retains its flow."

Sheldon: "Alice, that was a fantastic derivation!"

Bob: "I can visualize the loss landscape and gradient flow plots to confirm our findings."

- Help Bob visualize the gradient flow plots and loss landscape of VGG16 and resnet18. Compare them and get the results. (50 marks)

Tips: Bob is ready to visualize the gradient flow and loss landscape of VGG16 and ResNet18. Instead of training models from scratch, he will use pretrained models and conduct visualizations using CIFAR-10, a simple dataset.

Code Tips:

You can get average gradients of each parameter with the code below:

```
for name, param in model.named_parameters():
    if param.requires_grad and "weight" in name:
        layers.append(name)
        ave_grads.append(param.grad.abs().mean().cpu().numpy())
```

There is no need for you to visualize loss landscape with your own code. Use the package: loss-landscapes

Task Breakdown & Marks Distribution:

Successfully input data into the pretrained model and get appropriate output (15 marks)

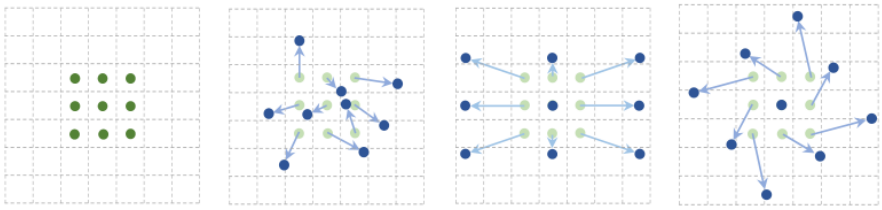
Get the visualization result of loss landscape (15 marks)

Get the visualization result of gradient flow (15 marks)

Compare visualization results, identify differences, and explain the importance of the loss landscape (5 marks)

Task 2: Deformable CNN

Bob: "In traditional CNNs, the receptive field is fixed, determined by the kernel size, stride, and pooling layers. However, Deformable CNNs introduce learnable offsets, allowing the receptive field to dynamically adjust. Let me show a visualization of how Deformable CNNs



work."

Alice: "Exactly! While conventional CNNs have a structured and fixed receptive field, Deformable CNNs allow each position to learn an offset, making the receptive field more flexible and adaptive to the target's shape."

Sheldon: "Alice, could you implement a Deformable CNN, focusing on the key components of Deformable Convolution?"

Alice: "Sure! I'll implement a simple Deformable CNN model using PyTorch."

Alice starts writing her code in dCNN.py.

Sheldon: "It looks like you haven't completed the class definition yet."

Bob: "I'm not sure I fully understand how your offset operation works."

Alice: "I want to compute the offset for each element in the feature map, but I'm unsure about the exact operation I should use."

Sheldon: "Think of the offset operation as a spatially invariant transformation. Since we need to determine the offsets for all kernel elements based on the same mapping, we need a consistent approach."

Bob: "That means we just need to use a convolution layer to generate the offsets!"

Alice: "That makes sense! Let me implement it. But what should the number of output channels be for the offset convolution?"

- **Help Alice complete the deformable convolution implementation and explain why the offset convolution has that specific number of output channels. (30 marks)**

Task Breakdown & Marks Distribution:

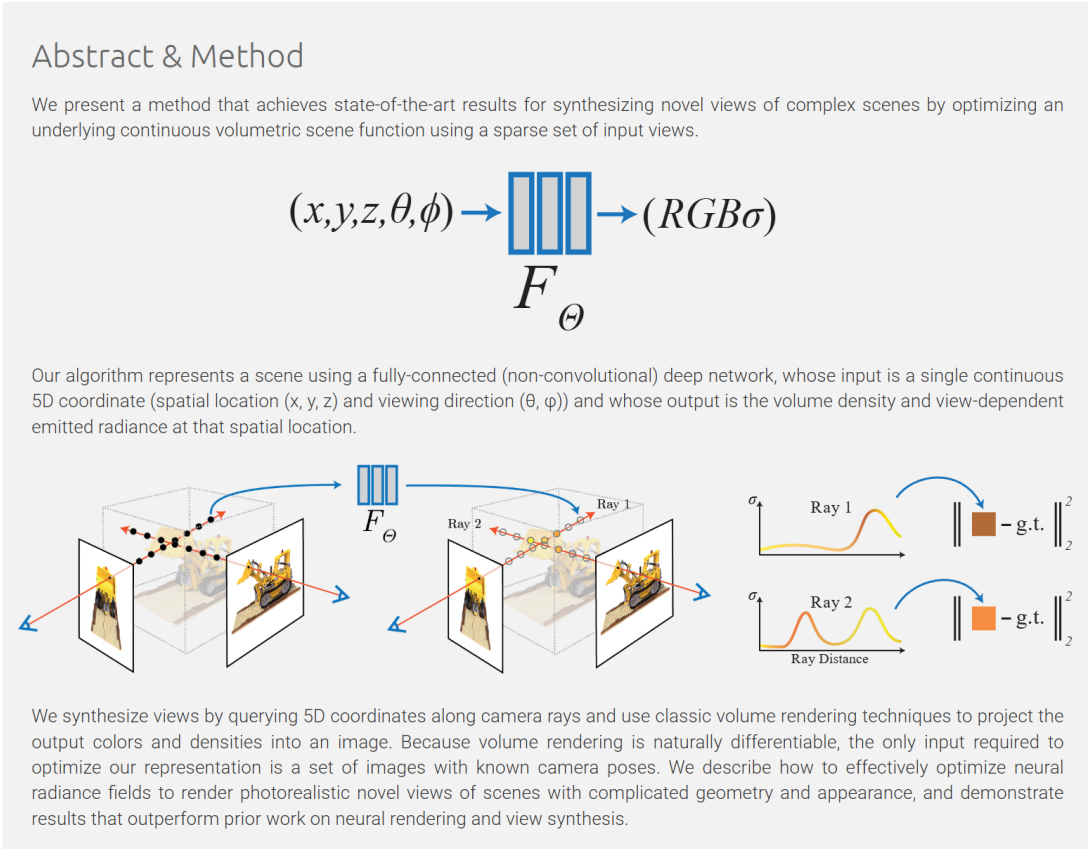
Complete the class definition for deformable convolution. (12 marks)

Explain why the offset convolution requires that specific number of output channels. (18 marks)

Sheldon: "I believe you now have a solid understanding of CNN implementations and their theoretical foundations. Now, let's discuss an application of deep learning!"

Part 2: 3d reconstruction with tiny NeRF

Sheldon, Alice, and Bob continued their deep learning discussion, moving from CNN-based analysis to 3D reconstruction. This time, Sheldon introduces Neural Radiance Fields (NeRF),



and they decide to implement a Tiny NeRF for 3D scene reconstruction.

Sheldon: "Have you heard about NeRF? It's a neural rendering technique that can synthesize high-quality novel views of a 3D scene using just a set of 2D images."

(details: https://www.matthewtancik.com/nerf

Relevant video introduction:

https://www.youtube.com/watch?v=JuH79E8rdKc

https://youtu.be/CRlN-cYFxTk?si=ruc_Nc4AeB-kz0CL)

Bob: "That sounds amazing! But isn't NeRF computationally expensive?"

Sheldon: "Yes, standard NeRF models require a lot of computation and memory, but we can implement a simplified version—Tiny NeRF—to get a basic understanding of how it works."

Alice: "Sounds like a great idea! Let's implement Tiny NeRF step by step and explore how we can use it for 3D scene reconstruction."

-----Understanding NeRF's Input and Output----

Sheldon: "We need to understand the core concepts of NeRF. Can anyone explain its input and output?"

Alice: "From my research, NeRF takes multiple 2D images from different viewpoints as input, along with camera position and direction information. The output is a synthesized 2D image from an arbitrary viewpoint."

Bob: "So, NeRF essentially trains a neural network to model a 3D scene, allowing it to generate views from angles we haven't seen before?"

Sheldon: "Exactly! NeRF is an implicit 3D representation that uses an MLP (Multi-Layer Perceptron) to map 3D spatial points to color and density values."

-----NeRF's Loss Function----

Alice: "How do we train this network? What does the loss function look like?"

Bob: "From what I understand, the loss function is related to the rendering process. We use volumetric rendering to compute the color of a pixel and compare it to the actual pixel color using Mean Squared Error (MSE)."

Sheldon: "Right! NeRF learns how to generate 2D images from a 3D scene by accumulating color contributions along rays using a volumetric rendering formula."

Bob: "Wait a minute, Sheldon. You said, NeRF learns how to generate 2D images from a 3D scene by accumulating color contributions along rays using a volumetric rendering formula. But isn't the rendering process fixed? What exactly is NeRF learning?"

Sheldon: "Great question, Bob! Let me clarify. NeRF does not learn how to render images. The volumetric rendering formula is a fixed process that we use to compute 2D images.

What NeRF actually learns is how to represent the 3D scene—specifically, it learns a function that predicts:

1. The color at any 3D point.
2. The density at that point, which tells us how much it contributes to the final image."

Alice: "Oh, so NeRF is learning the 3D properties of the scene, and then we use a separate fixed rendering step to generate 2D views?"

Sheldon: "Exactly! Think of it like this:

- NeRF is like an artist trying to understand what's in the 3D scene.
- The rendering formula is like a camera that simply takes the artist's knowledge and produces a picture."

Bob: "That makes sense! So during training, NeRF keeps adjusting its understanding of the 3D scene until the rendered 2D images match the real ones?"

Sheldon: "Exactly! The loss function compares the generated 2D images with the real images, and based on this difference, we adjust NeRF's internal 3D representation. Once trained, NeRF can generate novel views from any perspective!"

Bob: "Wait, Sheldon, I have another question. If NeRF is learning the 3D properties of a scene and adjusting its internal representation, does that mean once we train a NeRF model on one scene, we can use it to render completely different 3D scenes? Why or why not?"

➢ Help Sheldon explain this question. (10 marks)

Task Breakdown & Marks Distribution:

Yes/No. (4 marks) Explain why. (6 marks)

Sheldon: "Good question! The answer is no, a single trained NeRF model cannot generalize to a new, unseen 3D scene. Here's why:

NeRF learns a specific function that maps 3D coordinates to color and density for one particular scene. It doesn't store discrete 3D objects like a mesh or a dataset of shapes—it directly encodes the entire scene into the neural network's weights.

----- Ray sampling -----

Alice: "Let's start with ray sampling, the first step in NeRF."

Sheldon: "The first step is ray sampling. Does anyone know what it does?"

Alice: "From what I read, NeRF uses ray marching, which means we shoot rays from the camera into the 3D scene and sample points along each ray."

Bob: "It's like simulating a beam of light! We sample multiple points along the ray at different depths and feed them into the neural network."

Sheldon: "Exactly! We need the camera's intrinsic and extrinsic parameters to compute the ray directions. Then, we sample points along each ray and pass them into the NeRF model."

Bob: "Sheldon, I understand that we sample points along each ray, but why do we need sampling in the first place? Can't we just directly compute the color at a single point on the ray?"

Sheldon: "Great question, Bob! The reason we need sampling is because NeRF represents the scene as a continuous function rather than discrete surfaces. Let me explain why that matters.

The Scene is Not a Traditional 3D Mesh

- In traditional 3D rendering, objects are typically represented as meshes (a collection of vertices, edges, and faces).
- This allows us to compute an exact surface intersection when a ray hits an object.
- However, NeRF does not use meshes—instead, it models the scene as a volume where every point in 3D space has some color and transparency.

Alice: "Oh, so instead of checking where the ray intersects with a solid object, NeRF assumes that every point in space could contribute some color to the final image?"

Sheldon: "Exactly! Because the scene is represented as a continuous field of color and density, we can't just compute a single intersection. Instead, we need to sample multiple points along the ray and use those samples to determine the final color of the pixel."

Bob: "Okay, I get that we need to sample multiple points, but how do we decide where to sample?"

Sheldon: "Good question! The way we sample points along each ray affects both the quality and efficiency of NeRF. The two main approaches are:

1. Uniform Sampling: We evenly space out a fixed number of points along each ray.

2. Hierarchical Sampling: We first do coarse sampling and then refine it with fine sampling, focusing on important areas.

Sheldon: "Alright! Now that we roughly get the approaches of sampling. let's introduce an important technique called Stratified Sampling. Does anyone know what it does?"

Alice: "From what I read, stratified sampling ensures that our sampled points are more evenly distributed within each interval, rather than being strictly uniform. But I'm not sure how that works in practice."

Bob: "Wait, so does this mean instead of sampling at fixed points along the ray, we introduce some randomness?"

Sheldon: "Exactly! Instead of placing the samples at exact uniform intervals, we introduce a bit of randomness within each sampling bin. This helps avoid aliasing and makes the model more robust."

How Stratified Sampling Works

1. We first divide the sampling range $[t_{near}, t_{far}]$ into equal intervals.
2. Instead of always sampling at the center of each interval, we sample randomly within each interval.
3. This prevents regular patterns or bias in our samples.

Alice: "I have finished the uniform sampling in Raysampling.py. I think we can develop stratified sampling based on uniform sampling."

Bob: "I have finished the steps of stratified sampling in Raysampling.py"

➢ Help Bob complete this function and relevant visualization. (20 marks)

Task Breakdown & Marks Distribution:

Finish sampling codes (12 marks)

Get the visualization results (8 marks)

----- Volume Rendering -----

Alice: "After sampling, we need to compute the final pixel color using volume rendering. How does this work?"

Sheldon: "NeRF's volume rendering is based on ray accumulation. We compute each point's color c_i and density σ_i, then sum them up using weighted integration."

Bob: "That sounds complex! Can we break it down?"

Sheldon: "Sure! The process follows these steps:

1. Compute opacity: The density σ_i controls each point's transparency.
2. Compute weighted contributions: Points with higher transparency contribute more.
3. Integrate along the ray: Accumulate the colors based on their contributions."

Alice: "Can we express this mathematically?"

Sheldon: "Yes, the pixel color is computed as:

$$\mathbf{C} = \sum_{\mathbf{i}} \mathbf{T_i}\big(\mathbf{1} - \mathbf{exp}(-\boldsymbol{\sigma_i}\boldsymbol{\delta_i})\big)\mathbf{c_i}$$

where:

- $T_i = exp(-\sum_{\{j<i\}} \sigma_j \delta_j)$ is the transmittance (accumulated transparency).
- σ_i is the density.
- δ_i is the distance between adjacent points.
- c_i is the point color."

Alice: "Can we write a function for this?"

➢ Help Alice complete this function and get final picture with corresponding visualization function. (20 marks)

Task Breakdown & Marks Distribution:

Finish sampling codes (12 marks)

Get the visualization results (8 marks)

------ Training NeRF -----

Sheldon: "Now that we know how to sample and render, let's train NeRF. What kind of neural network do we need?"

Alice: "NeRF's MLP takes 3D coordinates $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and viewing direction $(\boldsymbol{\theta}, \boldsymbol{\phi})$ as input and outputs color and density."

Bob: "To capture fine details, NeRF applies positional encoding to the input coordinates."

Sheldon: "Right! Positional encoding maps the input into a higher frequency space using sine and cosine functions:

$$\boldsymbol{\gamma}(\mathbf{p}) = [\mathbf{sin}\,(2^{\mathbf{0}}\boldsymbol{\pi}\mathbf{p}), \mathbf{cos}\,(2^{\mathbf{0}}\boldsymbol{\pi}\mathbf{p}), \ldots, \mathbf{sin}\,(2^{\mathbf{L}}\boldsymbol{\pi}\mathbf{p}), \mathbf{cos}\,(2^{\mathbf{L}}\boldsymbol{\pi}\mathbf{p})]$$

This helps the MLP learn high-frequency details."

```
def positional_encoding(x, L=10):
    """Apply positional encoding to a 3D coordinate."""
```

```
    frequencies = 2.0 ** np.arange(L)
    encoding = np.concatenate([np.sin(frequencies * x), np.cos(frequencies * x)], axis=-1)
    return encoding
```

Bob: "Now we're ready to train Tiny NeRF!"

Sheldon: "Yes! You're right! Now I'll give you a basic tiny NeRF implemented by pytorch tiny_nerf.ipynb. Maybe you can first run the code to get the result."

Alice: "I think we can try different methods to further improve the tiny NeRF."

- Improve this architecture with one method, get the result of your improved method. You need to prove that your method is better than the older version with visualization. (20 marks)

Tips:

There is no need for you to implement too complex improvements. Just one tiny modification is OK. But you need to compare it with the original tiny NeRF. And explain you idea. (Higher accuracy? Convergence faster?)

Task Breakdown & Marks Distribution:

Your improvement about this architecture. (4 marks)

Get the results of both versions. (12 marks)

Explain the advantages of your new version (4 marks)