Answer all problems following the instructions. Please submit a PDF report with pictures of the output for each step followed by relevant code snapshots/screenshots that executed the function.

1 Assignment Instructions

Assignments in EEE 598 are due by the date assigned. You must use Latex to typeset your reports. **We will not accept Jupyter notebooks as reports.** Your report should have a section for each problem, and present your approach to solving the problem, any relevant code and figures to explain your method, and your results. Think about this like a job, you need to explain to your supervisor/boss what you did without dumping a whole bunch of code, answering questions in comments (big NO), and using labeled figures with captions. All figures must have axes labeled and an appropriate title.

2 Problems

Problem 1. *Creating Perceptual Losses for Images*

In this problem, we are going to explore the capabilities of using pre-trained CNNs to extract visual features and realize perceptual losses which are more correlated with human perception and can potentially work better than standard ℓ_p losses for various image/computer vision tasks.

1. Load in an image of your choosing that is at least 1080p resolution (e.g. 1080×1920). Display this image. Now take this image and translate it by 1 pixel to the left. Calculate the ℓ_1 and ℓ_2 losses between this translated image and the original image, and report the values.

2. Download the pre-trained VGG-16 architecture. Pick 4 different places to extract visual features from the architecture and label it in a diagram for us. Then compute the change in feature space between the original and translated image for the 4 features in question. What do you observe, and how does it compare to the $\ell_{1,2}$ losses earlier? What feature would you implement for a "perceptual" loss and why?

3. Test your perceptual losses' robustness to additive noise, highlighting the results both qualitatively and quantitatively.

4. Sometimes when imaging at long ranges, cameras can suffer from distortion known as atmospheric turbulence. This causes warping and blurring in the images as light rays are bent in the atmosphere due to a change in refractive index. In this subproblem, we want you to design a perceptual loss to be robust to atmospheric turbulence.

   Using the simulator (created by your own TA Ripon Saha!) `https://github.com/Riponcs/QuickTurbSim`, create a small dataset of 500-1000 images (does not need to be high resolution) of images with and without simulated atmospheric turbulence. Then train a network to extract visual features from the image, and show that your visual feature can be robust

to atmospheric turbulence. You can try various techniques including autoencoding, multi-scale visual feature extraction, use of pre-trained networks. This problem is open-ended in that there can be many solutions, we want to see that you test against standard visual features (i.e. your solutions for the parts above) and come up with something new/novel. Explain to us what your solution is and why you think it works well (note: the best solutions may have an opportunity to be expanded into a research paper if its sufficiently novel!) It's also ok if you don't find anything that works for this problem, but don't just tell us it doesn't work, show us what experiments you did and why it lends insight into the difficulty of the problem. Make sure you acknowledge any references or sources you use for this problem.

Problem 2. *Train and evaluate an EfficientNetV2-S model on the Oxford Flowers 102 dataset, perform architectural modifications, and visualize learned features using Grad-CAM.*

1. Load the Oxford Flowers 102 dataset. Summarize the dataset in a table (how many images, resolution of the images, how many classes, etc). Show 5 sample images along with their corresponding labels.

2. Use the EfficientNetV2-S model and train it on the 'train' set of Oxford Flowers 102 dataset for 2 epochs. Compute and report the accuracy on the 'test' set. Record the training time using the Python `time.time()` function. Inspect the model summary using the `torchsummary` package. Provide a snapshot of the last 7 lines showing the **Total parameters** and **Estimated total size**. Calculate and report the number of parameters in the model. [Tips: Use the implementation available at: `https://github.com/d-li14/efficientnetv2.pytorch`.]

3. Modify the architecture of EfficientNetV2-S to widen the network while ensuring the total parameter count remains within 10% of the original model. Show the model summary (last 7 lines) using `torchsummary`. Write what are the modification you performed. Train the modified model for 2 epochs on the 'train' set and report the accuracy on the 'test' set. Record and report the training time.

4. Modify the code to train the modified model using 2 GPUs (any available GPUs can be used. SOL is preferred but not required). Train the model for 2 epochs and report the time taken for training. What was the speedup in this case? Did you see any change in accuracy?

5. Create your own custom data augmentation strategy (not any built-in one from PyTorch). Explain your strategy, and evaluate whether it improves performance on this dataset for the EfficientNetV2-S model.

6. Go through Grad-CAM implementation from https://github.com/jacobgil/pytorch-grad-cam and explain how this method works. Select 4 different images of flowers from the dataset, pick two where the algorithm correctly identified the class, and two where it misidentified the class. Visualize the saliency maps of these 4 images using your trained and customized model, and reason on potential explanations for why the model failed.

Problem 3. *ImageNet Dataset*
In this problem, you will have to design your own custom Resnet training model and implement it on one of the largest and most popular image classification datasets, ImageNet. This dataset is available on Sol and you will have to use batch scripts to run training for only a few epochs to report accuracies. On Sol, ImageNet is available in the following directory: `/data/datasets/community/deeplearning/imagenet/train`. Note that we are only using the train set for this problem due to availability of labels, so create your train-val-test split out of the given train set. We have provided a mapping file for the labels for ImageNet in the Canvas page for your reference.

1. Create a prototype dataset from ImageNet that will serve as a test bench for your model. You will later be expected to run on the full dataset but it will be time consuming, so this will aid you in iterating your designs before your complete training attempt. You may create your training subset in any way you wish (fewer classes, fewer samples per class), but ensure there are enough samples to give reliable data for prototyping. You must also provide a summarized visualization of your sub-sampled dataset in the report with statistics about the dataset (You might be asked to run it any time).

2. Design a custom 36-layer Resnet model that will serve as your base model for training on the dataset. You may configure the layers any way you wish, but write a brief in your report as to what benefit you hope to see with your additional layers over the more popular Resnet-34 architecture. This can be supported with evidence by training both models on a prototype dataset.

3. Create your own custom activation function to be used in your Resnet-36 design. Write any unique function that is not among the ones made available to you through PyTorch and include it as your activation function in the custom Resnet model. Once again, write a brief with your expectations for this chosen function, visualize/graph the activation function, and compare with the more popular ReLU function used in typical Resnet architectures on your prototype dataset in terms of performance.

4. Commit to training on the entire ImageNet dataset now and train for only a few epochs. Note: for the right batch size and asking for at least 8 cpu cores in Sol when you make the request for GPU access, you can expect training to take roughly one hour per epoch on an A100 GPU. You can either train for a fixed number of epochs or for as many as you can in a fixed amount of time. Report the training and validation loss curve for your training, as well as the final test scores and comment on the results you have achieved. Compare your test performance of your model to a pre-trained Resnet on the same test set (note: you do not need to train/fine tune this pre-trained network as you should find one already trained on ImageNet).

3 Grading and Report

The assignment report is very important for the grading of this assignment. We will return reports that we are not satisfied with the presentation, this is to help you practice improving your written communication skills. Grading breakdown is as follows:

- Problem 1: 25 points
- Problem 2: 25 points
- Problem 3: 30 points
- Presentation (are answers clearly defined and easy to find, are code snippets and figures utilized well for the report, overall readability): 20 points
- **Total:** 100 points

Please only upload the PDF of the final report (we do not want any code). Also acknowledge any web sources, classmate help that you used in this assignment in an acknowledgements section.