ECE 685D HW3

Submission Instructions:

1. Upload your Jupyter Notebook (.ipynb file).
2. Export all outputs and necessary derivations as one or multiple PDF files and upload them.

LLM policy:
The use of large language models (LLMs) is not allowed for this assignment.

1 Problem 1: Object Detection with Convolutional Neural Networks (CNNs)

In this task, you will train a CNN model for a simplified human-face detection problem. The dataset contains 2204 imaegs and a .csv file for bounding box coordinate. It can be downloaded from `https://www.kaggle.com/datasets/sbaghbidi/human-faces-object-detection/data`.
Please divide your dataset into training and testing sets with a ratio of 4:1. After training, evaluate the model's performance on the test set.

1.1 Data Preprocessing (15 points)

Typically, bounding box annotations for object detection tasks are provided in formats like .xml, .json, or .csv. Your task is to implement a custom dataset class for object detection by inheriting from the 'torch.utils.data.Dataset' class. The dataset should return an image along with its corresponding bounding box when the '__getitem__' function is called. Additionally, a spatial transformations (e.g., random affine transformations, random elastic deformations) that should be applied to both the image and its bounding box as part of the function call. You may use any library to accomplish this task. After implementing the dataset, sample from the dataset you implemented and visualize the bounding boxes overlaid on the images. Verify that the bounding boxes are correctly positioned and that any spatial transformations are consistently applied to both the images and their respective bounding boxes. Sample and plot twice to make sure that your augmentations are indeed random and consistent.

1.2 Object Detection with Pre-trained Feature Extractor(25 pts)

Once your dataset is ready, you can download a pre-trained classifier (e.g., one trained on ImageNet) from any source to use as your feature extraction backbone. This backbone will be used to extract latent representations from your images. These latent representations will then serve as the input to a module that you will train to predict the coordinates (or transformed coordinates) of the bounding boxes. Note that the weights of the feature extraction backbone should remain frozen and must not be modified during this process. Evaluate the performance of your bounding box regression network on the test set using IOU (see Fig. 1) as the metric and plot a few examples of your output.

1.3 Object Detection with End-to-End Fine-Tuning(15 pts)

Next, unfreeze the weights of your feature extraction network and ensure that the feature extraction backbone and the bounding box regression module are connected with gradients. Train both the backbone and the bounding box prediction module together and observe if there is any improvement in model performance. Finally, report the Intersection over Union (IoU) for the model and visualize a few examples of your bounding box predictions.

Note: For simplicity, most images in this dataset are selected to contain just one object

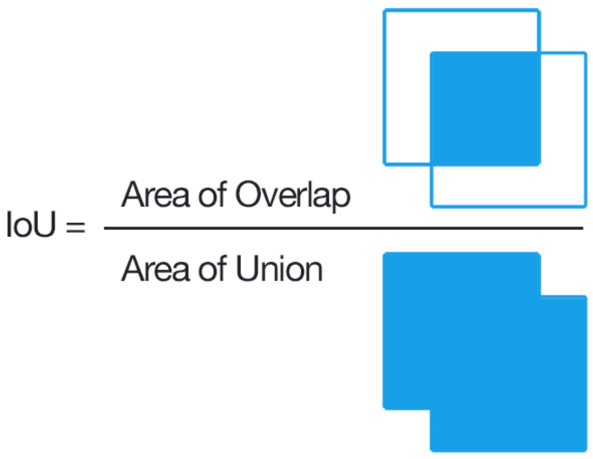

Figure 1: IOU for bounding box

of interest. It's ok to have the bounding box regressor to predict only one bounding box. However, think carefully how do you want to represent your bounding box. If the image sizes are too larger to fit into the GPU memory, consider using "torch.cuda.amp.autocast()" or reducing your image size and batch size

1.4 What if there are multiple objects? (10pts)

In the previous dataset, we assumed that each image contains exactly one object. Would your current model still work if there were zero or multiple objects from different categories in the images? Propose a modification that would enable your model to handle such cases and explain how this adjustment addresses or mitigates the issue. List relevant literature when applicable.

2 Problem 2: Optimizers from Scratch

2.1 Optimizer Implementaiton(20pts)

In this problem, you will implement the commonly-used optimizers from scratch. Train a CNN for MNIST classification on cross-entropy loss with *L1-regularization* using the following optimizers. Showing the correctness of your implementation by reporting the classification accuracy of your model after training.

1. Momentum method with parameter $\beta = 0.9$ (5 pts)
2. Nesterov's Accelerated Gradient (NAG) with parameter $\beta = 0.95$ (5 pts)
3. RMSprop with parameters $\beta = 0.95, \gamma = 1$ *and* $\epsilon = 10^{-8}$ (10 pts)
4. Adam with parameters $\beta_1 = 0.9, \beta_2 = 0.999,$ *and* $\epsilon = 10^{-8}$ (10 pts)

Note: You can use the Autograd package from Pytorch to compute the gradient when building your optimizer. However, you are NOT allowed to use any built-in optimizers.
Hint: Kingma et al. stated in [1] an alternative implementation of Adam, which has lower clarity but higher computation efficiency. (read the last paragraph before Section 2.1 for that paper)

2.2 Comparing your optimizer efficiency (15pts)

After implementing your optimizers, set your batch size to {4, 8, 16, 32}. Then, compare the changes in training and validation loss for each optimizer at various chosen learning rates (choose at your discretion) throughout the training process by creating plots for each batch size.

References

[1] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).