Computer Vision - Spring 25

Assignment 2

Deadline : 8 February 2025 11:55 P.M.

Instructors: Prof Ravi Kiran Sarvadevabhatla
and Prof Makarand Tapaswi

January 25, 2025

General Instructions

- Your assignment must be implemented in Python.
- While you're allowed to use LLM services for assistance, you must explicitly declare in comments the prompts you used and indicate which parts of the code were generated with the help of LLM services.
- Plagiarism will only be taken into consideration for code that is not generated by LLM services. Any code generated with the assistance of LLM services should be considered as a resource, similar to using a textbook or online tutorial.
- The difficulty of your viva or assessment will be determined by the percentage of code in your assignment that is not attributed to LLM services. If during the viva you are unable to explain any part of the code, that code will be considered as plagiarized.
- Clearly label and organize your code, including comments that explain the purpose of each section and key steps in your implementation.
- Properly document your code and include explanations for any non-trivial algorithms or techniques you employ.
- Ensure that your files are well-structured, with headings, subheadings, and explanations as necessary. Your assignment will be evaluated not only based on correctness but also on the quality of code, the clarity of explanations, and the extent to which you've understood and applied the concepts covered in the course.
- Make sure to test your code thoroughly before submission to avoid any runtime errors or unexpected behavior.

- Make separate notebooks (or folders, if using python files) for each question. The answers to all the questions should be written in markdown cells inside the notebook.
- **Report all the analysis, comparison and any metrics in the notebook or a separate report that is part of the submission itself. No external links to cloud storage files, wandb logs or any other alternate will be accepted as part of your submission. Only the values and visualizations as part of your commits will be graded.**

1 Convolutional Blocks of ResNet18 [30 marks]

This question will require you to train ResNet models on the provided custom dataset (36 x 36 size), and adapt the models to the peculiarities of this dataset. You are required to log every experiment on Wandb, and display the appropriate graphs after each subtask.

1.1 Baseline - Training ResNet [5 marks]

For the task below, you will be using 36 x 36 sized images from the custom dataset. Do **not** resize them.

1. Load the standard ResNet18 architecture from `torchvision.models`.
2. Import the dataset, and write a method to train this model for classification, with an appropriate loss function. Report the losses and accuracy during and after training. [**2**]
3. To compare, use a ResNet model that has been pretrained on ImageNet and use the same training method to retrain it on dataset. You will need to modify the last layer to account for the difference in the number of classes. Report the losses and accuracy during and after training. [**2**]
4. What are the spatial dimensions of image after each layer/block? What are these dimensions, in the layer just before average pooling? [**1**]

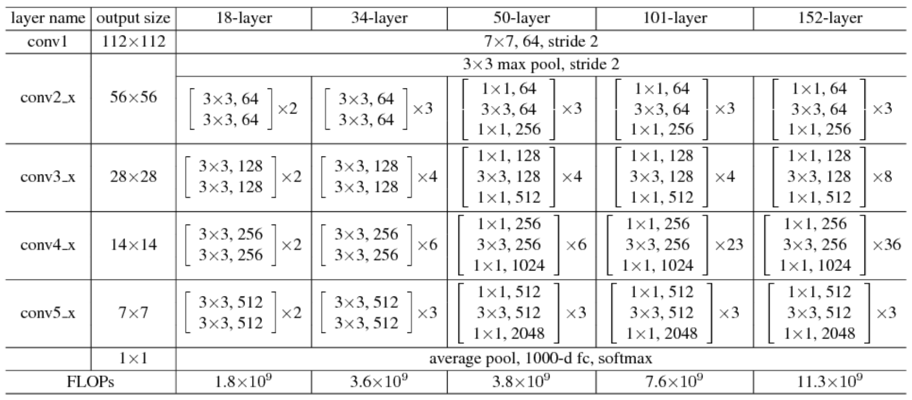
1.2 Training ResNet on resized images [5 marks]

The original ResNet architecture was devised to be trained on 224 x 224 images of ImageNet. So, the performance on smaller images may be sub-optimal. Your task will be to get better performance but without changing the architecture.

1. Resize the images from the dataset to the standard 224 x 224. Retrain both the above models with this modified dataset (ResNet18 from scratch, and ResNet18 pretrained on ImageNet). Compare the final accuracy of these models and report the losses. [**4**]
2. Better accuracy may come at cost. What changed/degraded from the previous set up? [**1**]

1.3 Modifying the architecture of ResNet18 to suit the given dataset [15 marks]

The first convolutional layer combined with Max Pooling reduces the 36 x 36 sized image to 9 x 9. The next convolutional blocks will again halve the spatial dimensions. So, the spatial height and width are not enough to extract all useful information from the image.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 1: ResNet18 architecture

This problem can be solved by resizing to 224 x 224 sized images, but it comes with a computational cost. Also, interpolating up does not increase the content of information in the image, and so is not required.

1. Modify the initial layers of the architecture (`model.conv1` and `model.maxpool` in standard ResNet PyTorch models) to try and improve the performance on the dataset. Train it from scratch and report atleast 3 different modifications. [**6**]

2. This time, modify (atleast 3 modifications) pre-trained models and compare the accuracy. Report loss graphs during training. [**6**]

3. In the case of the pretrained model, the first layer needs to be initialized from scratch, while the other layers have weights of the pretrained model. Would such an initialization (with different distributions in different layers) be a problem and make the model learn worse, or does it not affect the training significantly? [**3**]

1.4 Comparison [5 marks]

1. In the report, compare all the different aspects of the trained models. Draw comparisions between pretrained versus non-pretrained, effects of the size of the image, the kernel size, etc. [**3**]

2. Additionally, look at the F1 score and confusion matrices as accuracy is not always the perfect measure. [**1**]

3. Explain why you think those differences arise. [**1**]

2 Network Visualisation [30 marks]

The dataset linked here contains 10 classes, with 5 images for each.

The classes (in order) are:

1. *arctic fox*
2. *corgi*
3. *electric ray*
4. *goldfish*
5. *hammerhead shark*
6. *horse*
7. *hummingbird*
8. *indigo finch*
9. *puma*
10. *red panda*

Use these values as the mean and standard deviation while preprocessing.

```
MEAN = np.array([0.485, 0.456, 0.406])
STD = np.array([0.229, 0.224, 0.225])
```

The model to be used is **ResNet18**, but the last fully connected layer has to be modified. The original model which can be loaded from `torchvision.models` accounts for a 1000 classes.
Load the standard ResNet18 model from `torchvision.models`, change the last fc to 512x10 from 512x1000. Load the pretrained model weights from the folder linked (network_visualization.pth).

2.1 Saliency Maps [15 marks]

Saliency maps are images that highlight the relevant regions for machine learning models. The goal of this map is to portray the degree of importance of a pixel in an image for an ML model.
You will be using the model mentioned above to try and visualize which parts of the image it focuses on during classification.

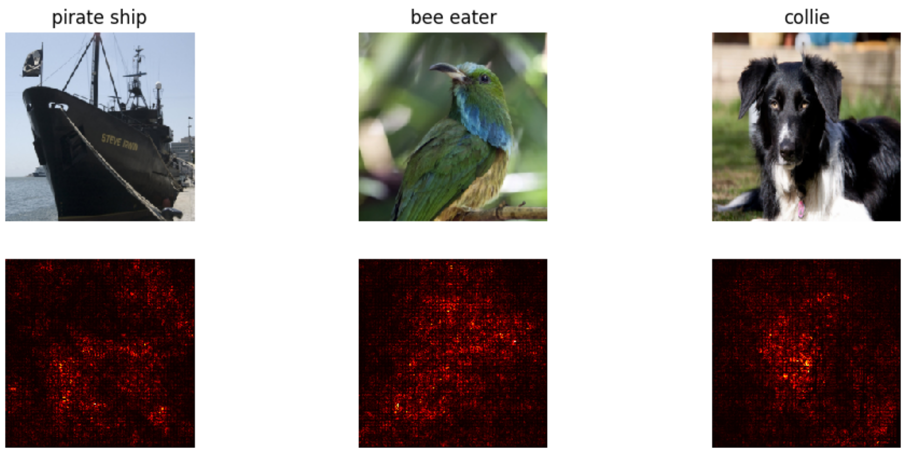


Figure 2: Example of saliency maps

1. Pick a subset of the data, with one image from each class. You will be using this subset for further tasks. Visualize these images, along with both the labels, ground truth and predicted. Create a separate function to preprocess this data (resizing and normalising) that will be reused in the second part. **[1]**

2. To compute saliency maps, we need to compute the gradient of the unnormalized score (also called logits) corresponding to the correct class (ground truth) with respect to the image pixels. In short, we need to perform a backward pass over the image with logit (instead of cross entropy loss). The absolute value of gradient over the image, is the required saliency. Write a function to compute this saliency map for any given image. **[7]**

3. Visualize these maps for the subset of the data selected. **[1]**

4. For these images, pick an appropriate threshold and mask out the pixels that do not change significantly during the backward pass. For the mask, replace those pixels with:

   (a) A constant value (0, 0, 0) ideally

   (b) Gaussian noise, by generating a random normal distribution.

   Try classification with these modified images and report any changes. Does the model misclassify these images? Why or why not? **[6]**

2.2 Fooling the network [15 marks]

Adversarial attacks are techniques that deceive the network into behaving unexpectedly using a defective input. These inputs are specifically designed to trick the system while being usually imperceptible to humans. This task involves creating such an input that can deceive the model into misclassifying the image provided.

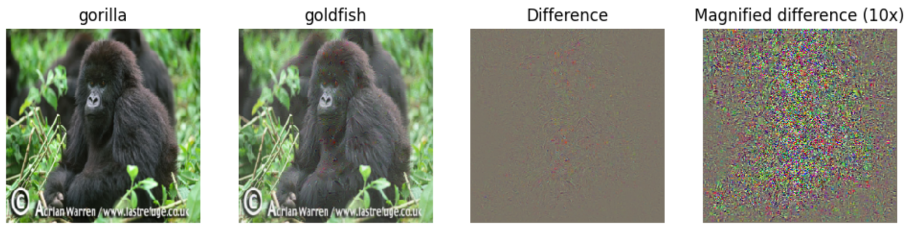


Figure 3: Adding noise to image, causes mispredictions.

Your task for this part is to find the least amount of noise that you need to add to the images for the model to misclassify them.

1. Make another set of 10 images, again, one from each class, that have not been used in the previous part of this task.

2. Try adding small amounts of Gaussian noise to the images and test the accuracy of the network. Does the image still visually appear the same? [**6**]

3. Similar to the previous question, make the image vector a learnable parameter, while freezing the rest of the network. Over a couple of epochs, maximize the logits for a class other than the groundtruth, forcing the network to optimize the image for the custom class, causing misclassifications in the network.

   (a) For an image, maximize the logits for the class with the next highest probability. [**4**]

   (b) Maximize the logits for the class with the lowest probability. [**4**]

   Compare the final image for both of these cases and report any differences observed.

4. Visualize both the original and modified images, and a representation of the noise added by the network (difference of intensities). [**1**]

3 Style Transfer [40 marks]

In this section, you have to use a simple optimisation to transfer style from one image to other.
Reference: https://arxiv.org/abs/1508.06576
The content and style images to be used have been provided to you here.
Utilize a standard **vgg19** model along with its weights from `torchvision.models`.

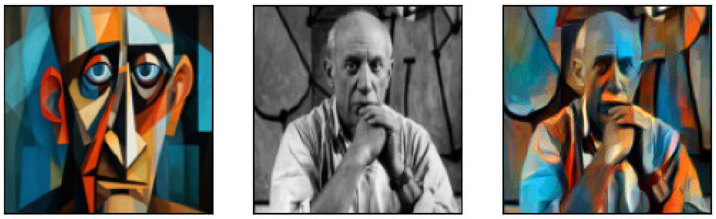


Figure 4: Style image, Content image, Final Output (in order)

3.1 Loss Function [10]

This tasks uses a custom loss function. The point of this is to preserve the features of the content image, as well as apply the characteristics of the style image.
Content loss calculation is required for features of 4th convolutional layer, style loss is a sum of the style losses for the first 5 convolutional layers. You can change these to find optimal hyperparameters.

1. The content loss function takes in the feature maps of a layer in a network and returns the weighted content distance between the image X and the content image C.

   For every layer, we define this loss as the squared difference between the feature representations of the content image (F^C) and the generated image (F^G) .

   $\mathcal{L}_{\text{content}} = \| F^C - F^G \|^2$

   Make a function that calculates this content loss given an input and a target image. [**5**]

2. In order to compute the style loss, we need to calculate something called the gram matrices of feature representations for both the style image and the generated image. The Gram matrix G_{XL} is computed from the feature maps F_{XL} of a specific layer L.

   A 4D tensor of shape $1 \times C \times H \times W$ is reshaped into a $C \times N$ matrix $(\hat{F}_{XL})$ where C is the number of channels and $N = H \times W$.

   The Gram matrix G_{XL} is calculated as:

$G_{XL} = \hat{F}_{XL} \cdot \hat{F}_{XL}^{\intercal}$

This gram matrix is normalized by dividing it by $K \times N$ to prevent the influence of layers with large spatial dimensions (N).

Using these gram matrices, the style loss is calculates as the MSE between the Gram matrix of the style image and the Gram matrix of the generated image at that layer.

Create a function that returns this style loss. [**5**]

3.2 Gradient Descent [30]

1. As mentioned in the paper, the optimization it uses is LBFGS, available in torch as optim.LBFGS. So instead of using optim.Adam or optim.SGD, you have to use appropriate optimizer, and also provide it parameters (image pixels here) and learning rate.

2. L-BFGS, unlike simpler optimizers like gradient descent, needs to evaluate the function and its gradient multiple times within a single optimization step. This is because it uses a line search algorithm to determine the best step size along the search direction. The closure function provides a way for L-BFGS to re-evaluate the loss and gradients again, without having to explicitly write the forward and backward pass code multiple times.

   ```
   def closure():
      # Function which calculates and returns loss
   for _ in range(optimization_steps):
      optimizer.step(closure)
   ```

3. Import a pretrained vgg19 from `torchvision.models` . Create a method to correctly get the losses and use them in a gradient descent loop to modify the content image such that a weighted sum of both the losses is minimized. Make sure the weights used ensure that the losses are of similar magnitudes. [**15**]

4. Try changing the weights to vary the importance of both the components (style and content) (5 configurations). [**5**]

5. Image on which updates are performed has constraints (0-1 range), so make sure to ensure this constraint after each backward pass.

6. Compare the results from using Adam and L-BFGS on given images. [**5**]

7. Record observations about quality of output, and in what domains does transfer work/doesn't work. [**5**]

8. Take a photograph of yourself, and transfer some style on it.

Good luck with the assignment!