Computer Vision - Spring 26

Assignment 2

Instructors: Prof. Makarand Tapaswi and Prof. Charu Sharma

February 5, 2026

The datasets for both the tasks can be found here.

1 Convolutional Blocks of ResNet18

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

1.1 Baseline Training ResNet

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

1. Load the standard ResNet 18 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.
3. To compare, use a ResNet model that has been pretrained on ImageNet and use the same training method to retrain it on the 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.
4. What are the spatial dimensions of image after each layer/block? What are these dimensions, in the layer just before average pooling?

1.2 Training ResNet on resized images

The original ResNet architecture was devised to be trained on 224×224 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 224×224. 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.
2. Better accuracy may come at cost. What changed/degraded from the previous set up?

Table 1: ResNet architecture

Layer Name	Output Size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	$7 \times 7, 64$, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	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

1.3 Modifying the architecture of ResNet18 to suit the given dataset

The first convolutional layer combined with Max Pooling reduces the 36×36 sized image to 9×9. 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.

This problem can be solved by resizing to 224×224 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.

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

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?

1.4 Comparison

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

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

3. Explain why you think those differences arise.

2 Network Visualisation

The dataset for this task 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 1000 classes. Load the standard ResNet18 model from `torchvision.models`, change the last fc to 512×10 from 512×1000. Load the pretrained model weights from the folder linked (`network_visualization.pth`).

2.1 Saliency Maps

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.

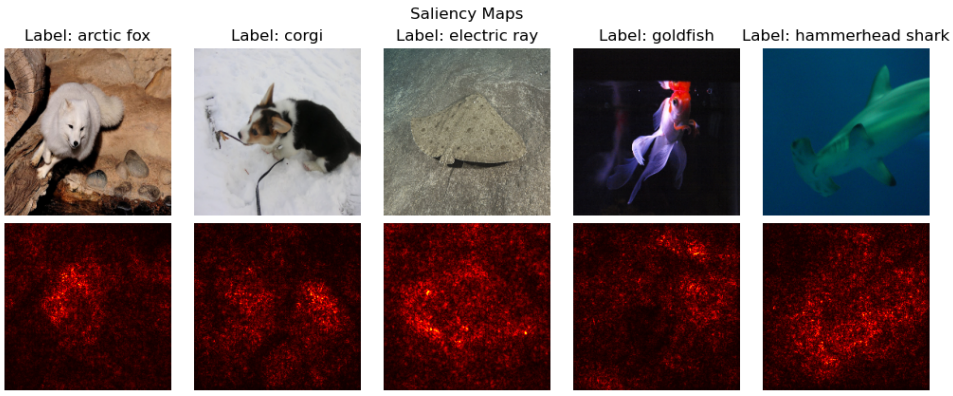

Figure 1: Example of saliency maps

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.

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.

3. Visualize these maps for the subset of the data selected.

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?

2.2 Fooling the network

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.

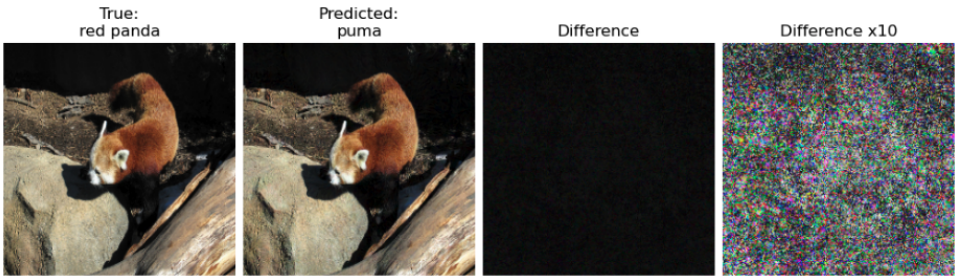

Figure 2: Adding noise to image, causes mispredictions.

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?

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.

   (b) Maximize the logits for the class with the lowest probability.

   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).