

# EECS 442 Computer Vision: Homework 5

## Instructions

- This homework is **due at 11:59:59 p.m. on Friday, April 9, 2021.**
- We are going to use [Google Colab](#) to run our code. For more information on using Colab, please see our [Colab tutorial](#).
- The submission includes two parts:

1. **To Canvas:** submit a `zip` file of all of your code.

**We have indicated questions where you have to do something in code in red.**

**We have indicated questions where we will definitely use an autograder in purple.**

Please be especially careful on the autograded assignments to follow the instructions. Don't swap the order of arguments and do not return extra values.

If we're talking about autograding a filename, we will be pulling out these files with a script. Please be careful about the name.

Your zip file should contain a single directory which has the same name as your unickname. If I (David, unickname `fouhey`) were submitting my code, the zip file should contain a single folder `fouhey/` containing all required files.

**What should I submit? At the end of the homework, there is a canvas submission checklist provided.** We provide a script that validates the submission format [here](#). If we don't ask you for it, you don't need to submit it; while you should clean up the directory, don't panic about having an extra file or two.

2. **To Gradescope:** submit a `pdf` file as your write-up, including your answers to all the questions and key choices you made.

**We have indicated questions where you have to do something in the report in blue.**

You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: <https://combinepdf.com/>.

The write-up must be an electronic version. **No handwriting, including plotting questions.**  $\LaTeX$  is recommended but not mandatory.

# 1 Fashion-MNIST Classification [35 pts]

In this part, you will implement and train Convolutional Neural Networks (ConvNets) in **PyTorch** to classify images. Unlike HW4, backpropagation is automatically inferred by PyTorch, so you only need to write code for the forward pass.

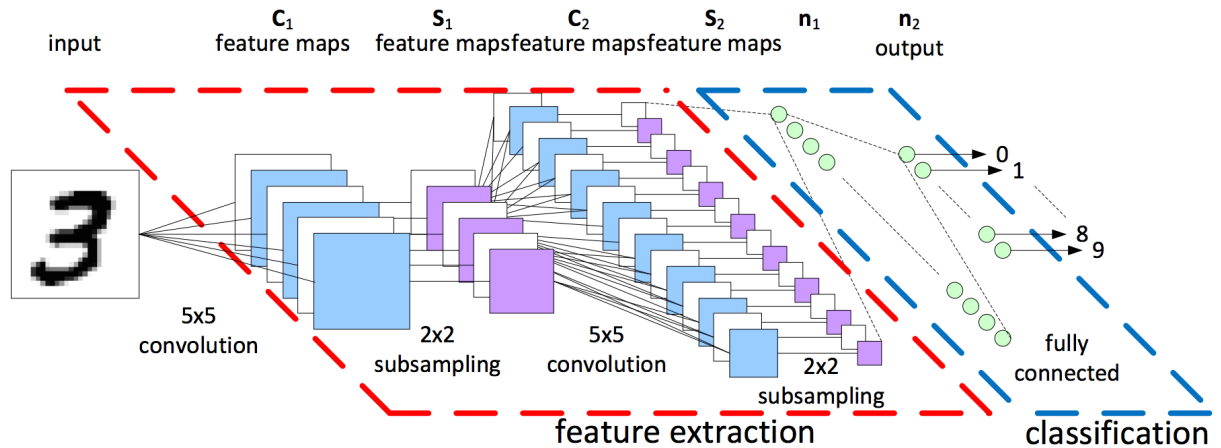


Figure 1: Convolutional Neural Networks for Image Classification<sup>1</sup>



Figure 2: Example images from Fashion MNIST dataset [1]

The dataset we use is Fashion-MNIST dataset, which is available at <https://github.com/zalando-research/fashion-mnist> and in `torchvision.datasets`. Fashion-MNIST has 10 classes, 60000 training+validation images (we have splitted it to have 50000 training images and 10000 validation images, but you can change the numbers), and 10000 test images.

<sup>1</sup>Image comes from <https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117>

Open the `part1.ipynb` notebook in Google Colab and implement the following:

- The architecture of the network (define layers and implement forward pass)
- The optimizer (SGD, RMSProp, Adam, etc.) and its parameters. (`weight_decay` is the L2 regularization strength)
- Training parameters (batch size and number of epochs)

You should train your network on training set and change those listed above based on evaluation on the validation set. You should run evaluation on the test set **only once** at the end.

Complete the following:

1. **Submit** the notebook (**with outputs**) that trains with your best combination of model architecture, optimizer and training parameters, and evaluates on the test set to report an accuracy at the end. (15 pts)
2. **Report** the detailed architecture of your best model. Include information on hyperparameters chosen for training and a plot showing both training and validation accuracy across iterations. (10 pts)
3. **Report** the accuracy of your best model on the test set. We expect you to achieve over **85%**. (10 pts)

**Hints:** Read PyTorch documentation for `torch.nn` at <https://pytorch.org/docs/stable/nn.html> and pick layers for your network. Some common choices are

- `nn.Linear`
- `nn.Conv2d`, try different number of filters (`out_channels`) and size of filters (`kernel_size`)
- `nn.ReLU`, which provides non-linearity between layers
- `nn.MaxPool2d` and `nn.AvgPool2d`, two kinds of pooling layer
- `nn.Dropout`, which helps reduce overfitting

Your network does not need to be complicated. We achieved over **85%** test accuracy with two convolutional layers, and it took less than 5 mins to train on Colab. You will get partial credits for any accuracy over **70%**, so do not worry too much and spend your time wisely.

## 2 Pre-trained NN, Saliency Map and Adversarial Attack [35 pts]

### 2.1 Task 1: Pre-trained NN (8 pts)

In order to get a better sense of the classification decisions made by convolutional networks, your job is now to experiment by running whatever images you want through a model pretrained on ImageNet. These can be images from your own photo collection, from the internet, or somewhere else but they should belong to one of the ImageNet classes. **Look at the `idx2label` dictionary in `part2.ipynb` for all the ImageNet classes.**

### For this task, you have to find:

- **One image (img1)** where the pretrained model gives reasonable predictions, and produces a category label that seems to correctly describe the content of the image
- **One image (img2)** where the pretrained model gives unreasonable predictions, and produces a category label that does not correctly describe the content of the image.

You can upload images in Colab by using the upload button on the top left. For more details on how to upload files on Colab, please see our [Colab tutorial](#). **Submit the two images with their predicted classes in your report.**

## 2.2 Task 2: Saliency Map (12 pts)

When training a model, we define a loss function which measures our current unhappiness with the model's performance; we then use backpropagation to compute the gradient of the loss with respect to the model parameters, and perform gradient descent on the model parameters to minimize the loss.

In the next two tasks will do something slightly different. We will start from a convolutional neural network model which has been pretrained to perform image classification on the ImageNet dataset. We will use this model to define a loss function which quantifies our current unhappiness with our image, then use backpropagation to compute the gradient of this loss with respect to the pixels of the image. We will then keep the model fixed, and **perform gradient descent on the image** to synthesize a new image which minimizes the loss. As a part of this task we will first explore saliency maps and then perform an adversarial attack in the next task.

A **saliency map** tells us the degree to which each pixel in the image affects the classification score for that image. To compute it, we calculate the gradient of the unnormalized score corresponding to the correct class (which is a scalar) with respect to the pixels of the image. If the image has shape  $(3, H, W)$  then this gradient will also have shape  $(3, H, W)$ ; for each pixel in the image, this gradient tells us the amount by which the classification score will change if the pixel changes by a small amount. To compute the saliency map, we take the absolute value of this gradient, then take the maximum value over the 3 input channels; the final saliency map thus has shape  $(H, W)$  and all entries are nonnegative. More specifically, we will compute the saliency map as described in the section 3.1 of [2].

**Implement the `compute_saliency_maps` function in `part2.ipynb`** to build the saliency maps for the two images from for the previous task (sec 2.1) and **submit the saliency maps in the pdf.**

## 2.3 Task 3: Adversarial Attack (15 pts)

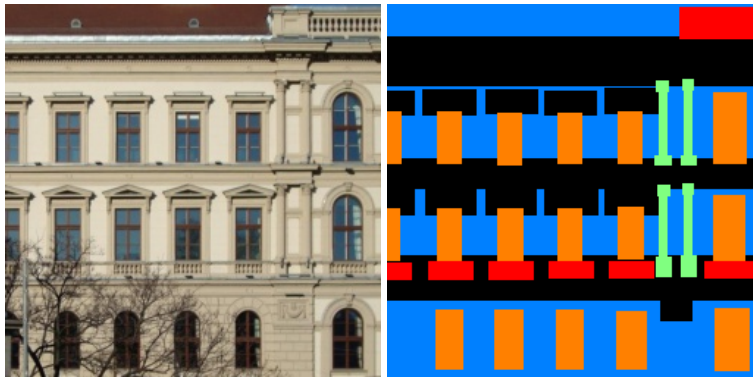
We can also use image gradients to generate "adversarial attacks" as discussed in [3]. Given an image and a target class (not the correct class), you have to perform **gradient descent over the image** as we did in the previous task to maximize the target class, stopping when the network classifies the image as the target class.

**Implement the `make_adversarial_attack` function in `part2.ipynb`** to fool the network for the **correctly predicted** image from the first task (sec 2.1). **Submit the original image, the attacked image, as well as the difference between them** which you can get by running the last cell of the notebook.

### 3 Semantic Segmentation [30 pts]

Besides image classification, Convolutional Neural Networks can also generate dense predictions. A popular application is semantic segmentation. In this part, you will design and implement your Convolutional Neural Networks to perform semantic segmentation on the Mini Facade dataset.

Mini Facade dataset consists of images of different cities around the world and diverse architectural styles (in .jpg format), shown as the image on the left. It also contains semantic segmentation labels (in .png format) in 5 different classes: balcony, window, pillar, facade and others. Your task is to train a network to convert image on the left to the labels on the right.



Note: The label images are plotted using 8-bit indexed color with pixel value listed in Table 1. We have provided you a visualization function using a “jet” color map.

Table 1: Label image consists of 5 classes, represented in [0, 4].

class	color	pixel value
others	black	0
facade	blue	1
pillar	green	2
window	orange	3
balcony	red	4

**Open up the `part3.ipynb` notebook in google colab and implement the following:**

- A convolution network which takes an RGB image of size  $H \times W \times 3$  and returns a segmentation map of size  $H \times W \times 5$ . There is a dummy network already present which uses a 1x1 convolution to convert 3 channels (RGB) to 5 channels, i.e. 5 heatmaps for each class, with cross-entropy loss.
- The optimizer (SGD, RMSProp, Adam, etc.) and its parameters. (`weight_decay` is the L2 regularization strength)
- Training parameters (batch size and number of epochs)

We will evaluate your model on its average precision (AP) on the test set (higher the better). We have provided you the code to evaluate AP and you can directly use it. An introduction to AP is available at [https://scikit-learn.org/stable/modules/model\\_evaluation.html#precision-recall-f-measure-metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics).

### Complete the following:

1. **Report** the detailed architecture of your model. Include information on hyperparameters chosen for training and a plot showing both training and validation loss across iterations. (10 pts)
2. **Report** the average precision on the test set. The function `cal_AP` will calculate the average precision on the test set. All hyperparameter tuning should be done on the validation set. We expect you to achieve over **0.45 AP** on the test set. (10 pts)
3. **Submit** the notebook which contains your best combination of model, optimizer and training parameters. (5 pts)
4. Find a satisfactory result in the **output\_test folder generated by the `get_result` function** and **paste it in your pdf**. There should be three images for each input - `gt<num>.png` (ground truth label), `x<num>.png` (input RGB image) and `y<num>.png` (predicted label). Below is an example of a satisfactory result. (5 pts)

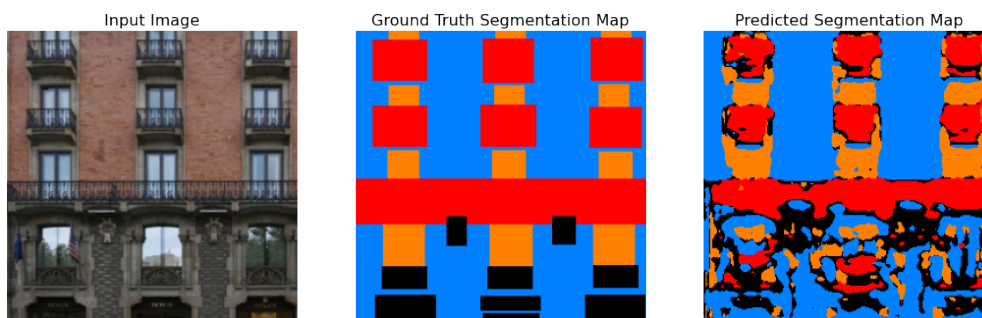


Figure 3: Semantic Segmentation Result

**Hints:** You should still choose layers from classes under `torch.nn`. In addition to layers mentioned in the previous part, you might want to try `nn.Upsample` that upsamples the activation map by a simple interpolation, and `nn.ConvTranspose2d` that upsamples by performing transpose convolution. Any test AP over **0.25** will receive partial credits. You will easily get a high test AP if you use the U-net [4] based architecture outlined in the notebook or you can try to implement a network from [5] [6], but they can take a little longer to train.

### Canvas Submission Checklist

In the `zip` file you submit to Canvas, the directory named after your username should include the following files:

- Code files: `part1.ipynb`, `part2.ipynb` and `part3.ipynb`

All plots and answer to questions should be included in your **pdf report** submitted to Gradescope. Run all the cells of your **Colab notebooks**, and do not clear out the outputs before submitting. **You will only get credit for code that has been run.**

## References

- [1] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” 2017.
- [2] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” *CoRR*, vol. abs/1312.6034, 2014.
- [3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *CoRR*, vol. abs/1312.6199, 2014.
- [4] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *ArXiv*, vol. abs/1505.04597, 2015.
- [5] E. Shelhamer, J. Long, and T. Darrell, “Fully convolutional networks for semantic segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 640–651, 2017.
- [6] A. Newell, K. Yang, and J. Deng, “Stacked hourglass networks for human pose estimation,” in *ECCV*, 2016.

## Acknowledgement

The Mini Facade dataset are modified from CMP Facade Database by Radim Tyleček and Radim Šára. Please feel free to similarly re-use our problems while similarly crediting us.