RBE549 Computer Vision : Homework 0

Thabsheer Jafer Machingal
Robotics Engineering (M.S))
tmachingal@wpi.edu
Subbmitted on: September 1 2022

***Abstract*—Homework0: Alohomora, has two phases. Phase1: Shake My Boundary is to develop a simplified version of pb(probability of boundary), which finds boundaries by examining brightness, color and texture information across multiple scales. The output of the algorithm is per pixel probability of boundary and that can be visualized as edges. The algorithm significantly outperformed the well regarded Canny and Sobel edge detection algorithms. Second phase of this assignment is tasked to build a simple ConvNet architecture and trained and tested using CIFAR-10 image dataset.**

***Index Terms*—Edge detection, CNN, Filter banks**

I. Introduction

This homework is a prerequisite assignment for RBE :549, computer vision, at Worcester Polytechnic Institute. The home work has two phases. The first phase covers classical approach to edge detection and second phase is a deep dive into deep learning. For phase one, we are implementing the mode explained in paper titled "Contour Detection and Hierarchical Image Segmentation"[1]. Phase 2 is implementation of a CNN architecture, train the model and test it. For the later part of the assignment the accuracy of the model is improved by normalizing the dataset. Convoluational neural networks(cnn) IS the best way for computer to look at an image and learn features of the image without explicitly directing it do so [2].

II. PHASE 1: Shake My Boundary

A. Designing Filter banks

Filter bank is an arrangement of band-pass filters that split the input signal into a set of analysis signals [3]. Filter banks are used for signal processing and image processing. For this assignment we are required to build three different filter banks and integrate them to filter through an image. in total

1) Oriented DoG filters: A collection of Oriented DoG filters can be created by convolving a simple Sobel filter and a Gaussian kernel and then rotating the result multiple times. In total 32 DoG filters were created. I used two different scale, the standard deviations of the 2D Gaussian kernels are 1 and 2 and 16 different orientation between 0° and 360°.

2) Leung-Malik filters: Leung-Malik filters (LM filters) are a collection of multi-scale, multi-orientation filter bank with 48 filters. For this assignment, we are considering LM Small (LMS) and LM Large (LML) filter bank, they differ in the scale of standard deviations. I specifically chose, 48 filters of kernel size 57x57 for LML and LMS. Out of 48 filters, 36 made from derivative of Gaussian kernels, namely first and second derivatives of Gaussian kernel for three different sigma values and six orientations. For LMS, the values are; σ = 1, $\sqrt{2}$, 2 and for LML, σ = $\sqrt{2}$, 2, $2\sqrt{2}$. 12 other filters include eight LOG (Laplacian of Gaussian) filters and four Gaussian filters. For the 8 LOG filters, the filtering occur at σ and 3σ. Where the scales for LMS are σ = 1, $\sqrt{2}$, 2, $2\sqrt{2}$ and for LML, σ = $\sqrt{2}$, 2, $2\sqrt{2}$,4.

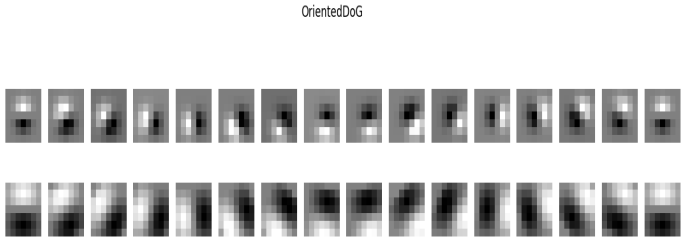

Fig. 1. oriented DoG filters

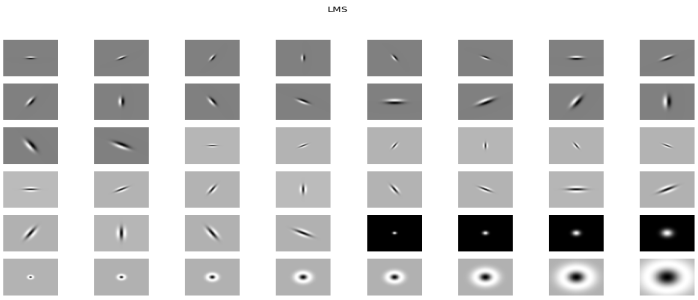

Fig. 2. Leung-Malik Small filters

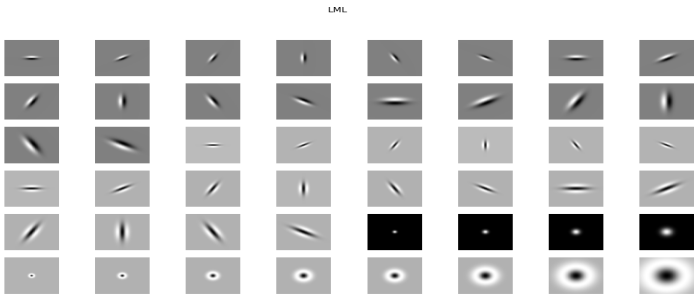

Fig. 3. Leung-Malik Large filters

3) Gabor filters: A 2D gabor filter is gaussian kernel function modulated by a sinusoidal plane wave.

$$g(x,y) = s(x,y)w_r(x,y) \tag{1}$$

Here the $s(x,y)$ is a complex sinusoid, known as the **carrier** and $w_r(x,y)$ is a 2-D Gaussian-shaped function known as the **envelope**[4]. The complex sinusoidal function is defined as:

$$g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = \exp\left(\frac{-x'^2+\gamma^2y'^2}{2\sigma^2}\right)\exp\left(i(2\pi\frac{x'}{\lambda}+\psi)\right) \tag{2}$$

where $x' = x\cos(\theta)+y\sin(\theta)$ and $y' = -x\sin(\theta)+y\cos(\theta)$. For the purpose of this assignment, the real part of the equation (2) is sufficient and is used to design a Gabor filter. The standard deviation in 2D is split between x and y as $\sigma_y = \frac{\sigma_x}{\gamma}$.The constant parameters in equation (2) are wave length of the sinusoidal factor(λ), orientation of the normal to the parallel stripes of a Gabor function (θ), phase offset (ψ), standard deviation of the Gaussian envelope(σ) and the spatial aspect ratio(γ). γ describes the ellipticity of the support of the Gabor function[5].

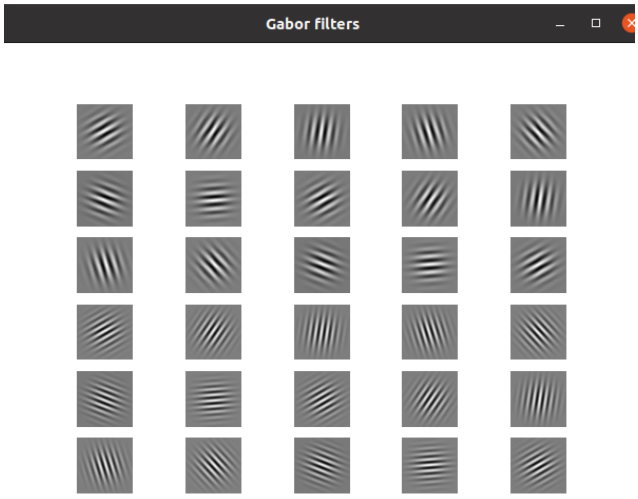

Fig. 4. Gabor filters

4) Texton Map,T: Filtering an input image with each element of your filter bank results in a vector of filter responses centered on each pixel. A distribution of these N-dimensional filter responses could be thought of as encoding texture properties. And this representation is simplified by replacing the N-dimensional vector with a texton ID, this achieved by clustering the filter responses at all pixels in the image in to K textons using kmeans from Scikit learn[6]

5) Brightness, B: Its a map of changes in brightness with in an image.

6) Color Map, C: Color Map captures the changes in color or chrominance of an image.

B. Texture, Brightness and Color Gradients T_g,B_g,C_g

To obtain T_g,B_g,C_g, we need to compute differences of values across different shapes and sizes. This can be achieved very efficiently by the use of Half-disc masks.

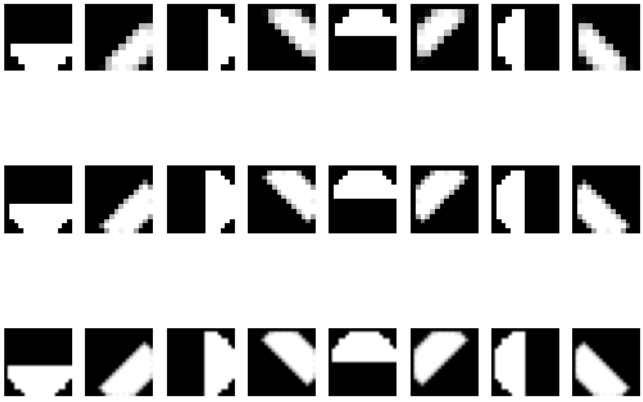

Fig. 5. half disc masks

C. Sobel and Canny baselines

Sobel and Canny baselines are given as the output images from the sobel and canny operations. Out Pb-lite algorithm is compared against these baselines.

D. Pb-lite Output

The pb-lite algorithm evidently performed well in detecting edges. In a semantic pov, Pb-lite is capable of detecting edges that are useful and good at omitting not so useful edges.

E. Results

1) Sample image: The results of pb-lite is tested on a sample image. Along with the pb-lite results of every step is given below

Fig. 6. image 1

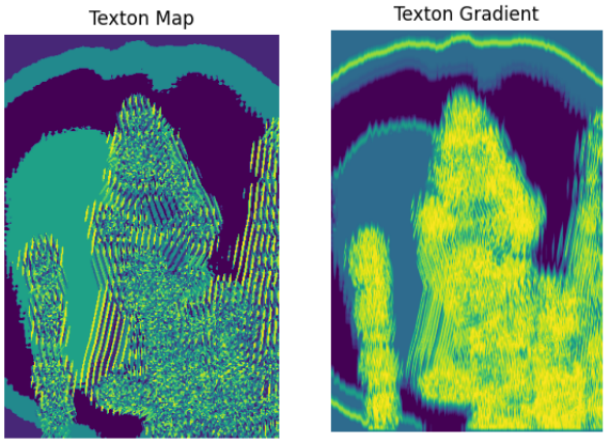

Fig. 7. Texton map and texton gradient

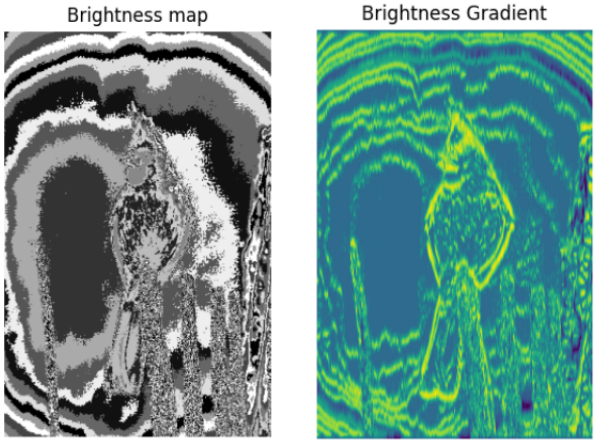

Fig. 8. brightness map and brightness gradient

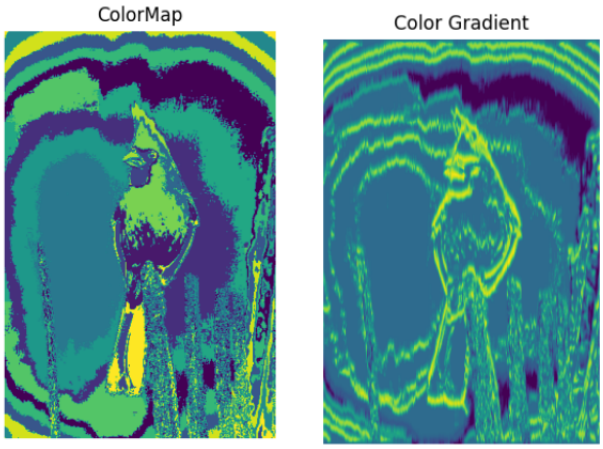

Fig. 9. color map and color gradient

Fig. 10. sobel(Left) and canny(Right) baseline

Fig. 11. Pb-lite output

III. PHASE2: Deep Dive on Deep Learning

A. Problem Statement

For this problem, A simple convoluational neural network(convnet) is built with pytorch, trained the model and tested using images from CIFAR-10 dataest. For the later parts of this problem, I'm implememnting Three other neural network models namely, DenseNet, ResNet, ResNeXt. The code is implemented on google colab notebook and the model is trained on GPU.

B. Dataset

CIFAR-10 dataset is a collection of 60000 32x32 images of 10 different classes with 6000 images per class. In which 50000 is training images and test images are 10000. The dataset is fairly balanced and spread across between the classes.

C. Model the first neural network

For this part of the assignment I built a simple cnn architecture. It has six convolutional and three linear layers. A model summer of the architecture is printed out and given below.

```
CIFAR10Model(
  (network): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU()
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=4096, out_features=1024, bias=True)
    (17): ReLU()
    (18): Linear(in_features=1024, out_features=512, bias=True)
    (19): ReLU()
    (20): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Fig. 12. Model Summary of the CIFAR10Model

D. Train your first Neural Network

Some of the functions given in the starter code was edited to better understand the code and adapt to the dataset. Also, my local machine was not performance-wise competitive so the model is trained on GPU on google colab notebook using cuda toolkit from NVIDIA. The data is trained in batches of 100 and for 14 epochs. The training training time is around 205 seconds on GPU. It is evident from the above plot that accuracy of

```
network.0.weight 32
network.0.bias 32
network.2.weight 64
network.2.bias 64
network.5.weight 128
network.5.bias 128
network.7.weight 128
network.7.bias 128
network.10.weight 256
network.10.bias 256
network.12.weight 256
network.12.bias 256
network.16.weight 1024
network.16.bias 1024
network.18.weight 512
network.18.bias 512
network.20.weight 10
network.20.bias 10
```

Fig. 13. Number of parameters and weights of the network

the training dataset increase over number of epochs and further improvement after certain number of epochs.

E. Test your first Neural Network

The trained model is tested with 10000 images from the CIFAR10 dataset. Our model has classification accuracy of 76.79% test dataset. A confusion matrix of the test dataset is given below.

F. Improving Accuracy of your Neural Network

1) Normalizing the dataset: The dataset is normalized to improve the accuracy. By normalizing the dataset I was able to improve accuracy of training and test set by a few points. The reported test accuracy for the normalized dataset is 77.61 %. The improved model's confusion matrix is given below.

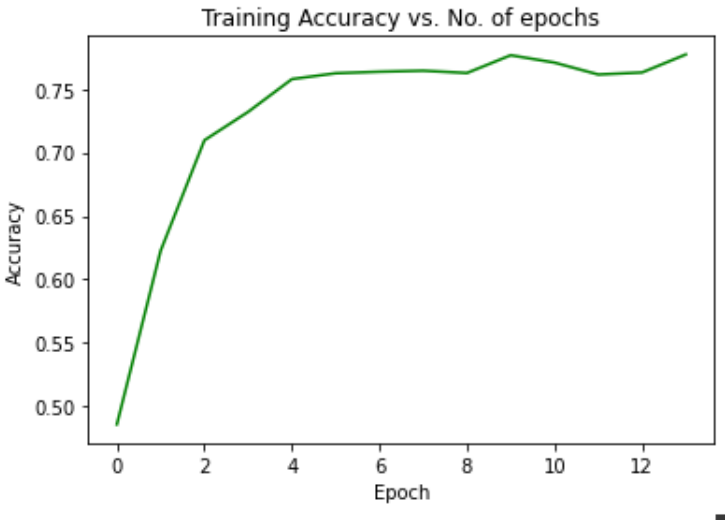

Fig. 14. Training accuracy vs Number of epochs

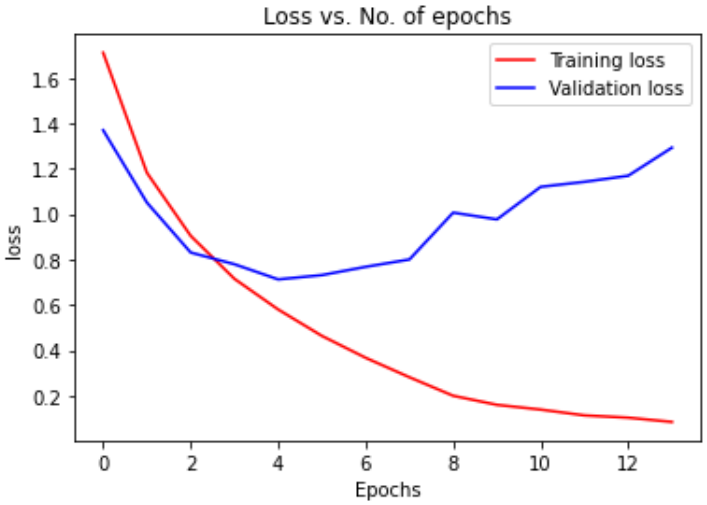

Fig. 15. Training and validation loss vs Number of epochs

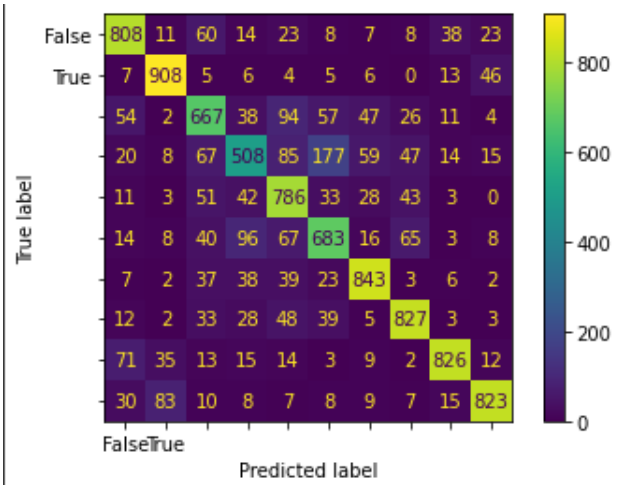

Fig. 16. Confusion matrix of the test dataset

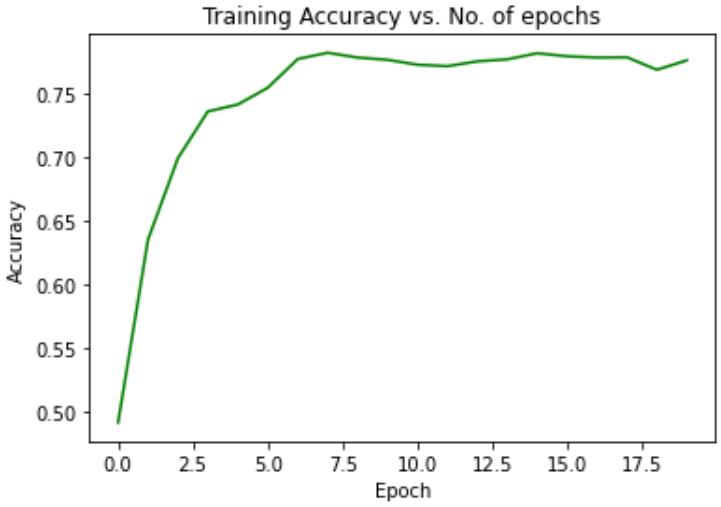

Fig. 17. Training accuracy vs Number of epochs

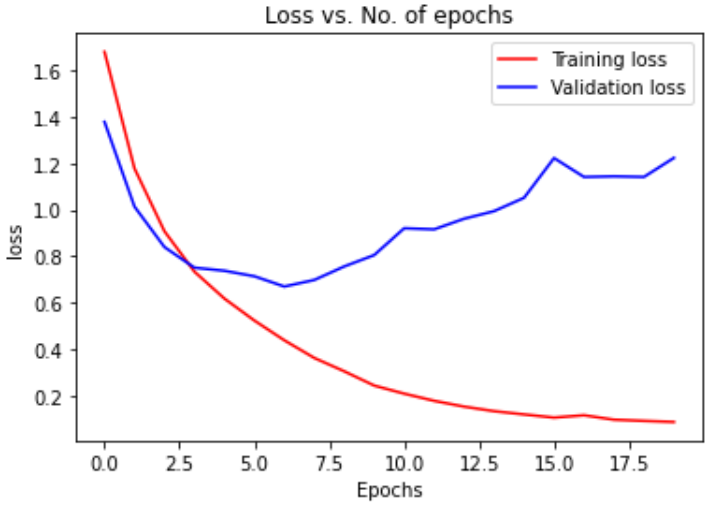

Fig. 18. Loss vs Number of epochs

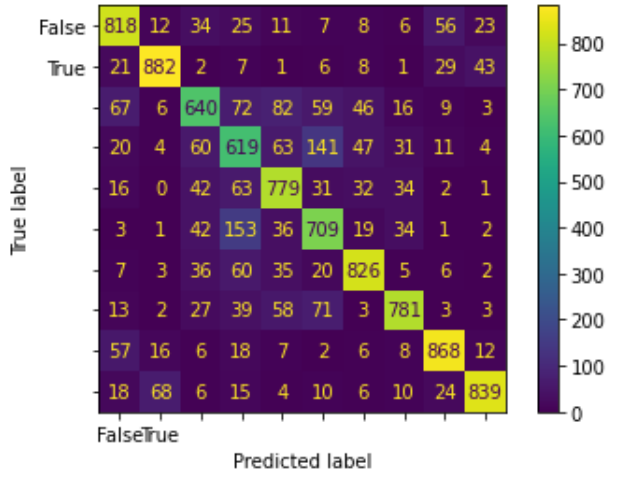

Fig. 19. Confusion matrix of the normalized dataset

G. Phase2: Results

1) Number of epochs: six and above is a good number of epochs, by optimising number of epochs we might be able to reduce training time and computation complexity. For this given problem, 14 epoch took around 205 seconds and 15 epochs was 233 seconds. But for more complex problem, optimizing number of epochs might save a lot of time.

2) Normalization: Normalizing the dataset improved the accuracy by a small scale.

REFERENCES

[1] P Arbeláez et al., "Contour Detection and Hierarchical Image Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence 33, no. 5 (May 2011): 898–916, accessed September 1, 2022, http://ieeexplore.ieee.org/document/5557884/.

[2] "What Is a Convolutional Neural Network?," Western Governors University, accessed September 1, 2022, https://www.wgu.edu/blog/what-convolutional-neural-network2008.html.

[3] "Filter Bank: What Is It? (DCT, Polyphase, Gabor, Mel And FBMC)," Blog, April 19, 2021, https://www.electrical4u.com/filter-bank/.

[4] Javier Movellan, "Tutorial on Gabor Filters," n.d., 4, https://inc.ucsd.edu/mplab/75/media//gabor.pdf.

[5] "Filter Bank: What Is It? (DCT, Polyphase, Gabor, Mel And FBMC)," Blog, April 19, 2021, https://www.electrical4u.com/filter-bank/.

[6] "Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.1.2 Documentation," accessed September 1, 2022, https://scikit-learn.org/stable/.