Project 4 description

Training a Convolutional Neural Network

Due: Tuesday, March 31st at 11.59PM

You will be training a CNN on the wallpaper images we previously classified in the last project. You will be using the Matlab inbuilt CNN training functions (because of their ease). You will learn how to augment the data, build a CNN, and train on the data. Do not wait until the deadline for this project since some steps can take a while to run. Start this project early.

Data: Wallpaper Pattern Images

The dataset consists of 17,000 images, 1,000 images per group, and each image containing a wallpaper pattern. The images are 256x256 and 1 channel (grayscale). You will be training networks to discover these patterns. The datasets are located in the "data/wallpapers/<train,test>/<group>/" folders. The train and test images are in separate folders and within them, there are folders for each wallpaper group's images. To know more about Wallpaper groups, check out this link: https://en.wikipedia.org/wiki/Wallpaper_group (链接到外部网站。)

Step 1: Training and Testing the CNN

This step is to get you familiar with the CNN interface and to train your first network. The starter code comes complete with an example of a convolutional neural network. The data is not augmented so this data is very separable (as seen by your last project). You can adjust the setting if it doesn't run on your computer (such as decrease the batch size if your GPU doesn't have enough memory for the images). This network should train quickly (getting above 10% accuracy after the second epoch). You should train this for at

least 10 epochs (though you are free to train it more if you like). This should get around ~70%+-5% accuracy. (Try modifying learning rate and batch size to figure out which combination increases the accuracy). Is there anything you notice about what happens after you reduce the training rate?

Step 2: Augmenting the Training and Testing data

Now that you have quickly trained on the original patterns, the next step is to train and test on augmented patterns. The purpose of this experiment will be to see if we can train the network to understand the patterns after the images have been rotated, scaled, translated, and reflected. To do this, you will need to create new images from on the original dataset. These images should also be cropped to 128x128 (so that you can scale the patterns more). Because of the smaller image sizes, you will need to adjust the input to the network to take a [128, 128, 1] image. You may use Matlab tools like imrotate and imresize for this project. You should allow for 360-degrees of rotation, any valid translation (don't translate off the image), and a scale range between 50% - 100% from the original 256x256 image. Make sure to use a uniform scale (the same scale) for both x and y). Make sure to describe any choices you make while applying these augmentations such as what you do at the border of the image. In your report, show at least 5 examples per augmentation and then combining all augmentations. Also, show the distribution of the augmentations with a histogram of the scales, rotations, and translations you have used.

Original	Rotation	Scale	Translation
The original image	Rotated by 40 degrees	Scaled up by 120%	Translated by ~5 pixels in the x and y direction

You should create folders called 'train_aug' and 'test_aug' in the 'data/wallpapers' folder. Within each of these folders, have a folder for each group (just like the train and test folders). The

labels are automatically assigned based on the folder name (so all the images in the P1 folder are considered to be in the group P1). Within each of these folders, you must augment each pattern from the original dataset with a combination of all of these augmentations at least 5 times per training image (so you should have 85,000 training images in the subfolders of 'train_aug') and once per testing image. A larger dataset will make training easier for the next step so more examples here are better so if your computer can handle more examples.

Since each dataset is unique so it is impossible to say how long it will take to converge but you can expect to not get a high accuracy here. It's ok if this doesn't really learn, just show that you tried.

Step 3: Building Your Own Network

Now that you have a harder dataset which the initial network should have a hard time on classifying, now we will need to build a larger network.

Some common network types are:

INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC
INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC

Where CONV is a convolutional layer, FC is a fully connected layer, *2 means all the layers in the brackets are done twice in serial. You can play around with the number of filters and the size though usually the later layers are smaller and have more filters. It's a tradeoff between more filters and more depth so you should explore this tradeoff. You may add other kinds of layers if you so wish (not required).

Larger networks take a while to train so resuming from a previous checkpoint the next day or running it overnight on your own machine will help if you can't do it in one sitting. You should monitor the training accuracy since if it does not start going up after ~3 epochs, it probably won't be going up. So you don't need to train for an hour to see if it is going to work.

Once you have a large network trained on your harder data created in Part 2, go back and train it on the data from Part 1. Do you do better or worse than the original network?

You must train 2 networks, one skinny and one with lots of filters:

- A skinny network has more layers and but not many filters at each layer. This should at least be the length of the example at the bottom of the starter code.

- A wide network has fewer layers but many filters. You should at least double the number of filters in the starter code network. You can also make it longer as well, just make the skinny network even longer if you do.

In addition to the two networks, you should implement:

- Transfer Learning - Fine-tune a large pretrained network (>= to AlexNet in size) for this task. Describe exactly how you fine-tuned the network and evaluate it on our test sets.

Then, Visualizations:

- Visualization 1: Visualize the first layer of filters for **all** of your networks.

- Visualization 2: Do a t-SNE multidimensional reduction on the fully connected layer activations of your network trained on the augmentations. (Check out the documentation on the "activations" and "tsne" commands in Matlab on how to do it.)

Since this dataset is harder, you will need to train these networks for longer than the first problem. You may also have to change the learning rate. If you never get a decent accuracy, you should show that the networks are doing better than random chance and how you have attempted to improve the accuracy.

Here is the link to the starter code: p4_startercode.zip

Report:

- Explain your augmentation process. (methods and parameters used)

- For each step, you must report the classification matrix/confusion matrix for the training, validation, and testing

datasets. (Use confusionchart or heatmap function from matlab to visualize the matrices).

- Record the amount of time you took training each network and the training accuracy and loss.

- Make sure to show a comparison of the original and augmented data in your report.

- What did you learn about creating convolutional neural networks?

- Describe each network with the total number of parameters and activations for the entire network as well as for each layer.

- Does a decrease in loss/error directly translate into an increase in accuracy? Why or why not?

Notes:

- The network configuration in the stater code may or may not give a good accuracy. Try to modify the training parameters like batch size, learning rate and number of epochs.

- Matlab saves a checkpoint for the network after each iteration. These can take up a lot of data very quickly so make sure to delete the old networks frequently while training (you should leave at least the last checkpoint just in case something happens though).

- If you are having issues running the starter code or with the project please contact the TA early on, do not wait for the deadline.

- The starter code has been tested on Matlab R2016a and later. It won't work if you use any version of Matlab <R2016a.

- If you have access to a computer with an NVIDIA GPU and Matlab, it is **highly** recommended you use it. It can take 13-14 minutes per epoch for the network in the startercode network using only the CPU in the Westgate computer lab. With a GPU, it takes <1 minute per epoch (unfortunately the Windows lab in Westgate does not have NVIDIA GPUs). Matlab will automatically use the NVIDIA GPU if the computer has one.

- If your network accuracy seems to just drop, try going back to your last good checkpoint and lowering the learning rate.

Extra Credit (Upto 30 Points if you do one of the options.

Upto 50 points if you do both):

Option 1 (MoCap Data): https://forms.gle/HcFPp18uGCpwCtX18 （链接到外部网站。）

Option 2 (Footpressure Data): https://forms.gle/jgCkvhwriSuXuggf7

Kindly fill one of the the above forms. On completion, you will see a link on your screen which has the data folder and a README file that contains details on what to do for the extra credit part.

If you have any questions, feel free to email me.

IMPORTANT NOTE:

This project is the last chance to use any of your late days. You CANNOT use late days for term project.

Python:

It is **NOT** recommended but you are allowed to use Python for this project. Only do this is you are experienced with using deep learning libraries in Python. Not being able to finish because you are using Python or problems with using the other deep learning libraries are **NOT** valid excuses since the Matlab works natively and is automatically configured for working with the GPU. If you insist on using python, you **MUST** use python 3.6.5 (conda 4.3.30) The libraries you may use are Keras 2.2.4 with tensorflow 1.10. If you want to use another deep learning library, you will need explicit permission from the TA.

You must replicate the network in for Parts 1 and 2 from the starter code. This should be relatively simple if you are familiar with Keras since the Matlab network creation is very similar to the sequential model. For Part 2, you must write your own data augmentation algorithm and create the separate training set of data. You should use other layers such as batch normalization, dilated

convolution, separable convolution and/or residual layers as well since they are available.

Kindly fill this google sheet with the requested info.
https://docs.google.com/spreadsheets/d/1NO_qPoYv4ev1T25HiiOW6bK5VBmkK5RzApvMvdIHXQ8/edit?usp=sharing

Also, Email the TA with a brief description of your background in deep learning with Python (what libraries you have used, what problems have you have worked on, and what networks you have trained). We want to see how many people are doing this and to make sure you are qualified.