Project #4 Description
Deep Learning

CSE583/EE552 Pattern Recognition and Machine Learning, Spring 2021

Release Date: Monday, March 15, 2021
Submission Due Date: Friday, April 2, 2021

Contents

1 Introduction

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.

2 Dataset: 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.

3 Implementation

- **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. Is there anything you notice about what happens after you reduce the training rate?

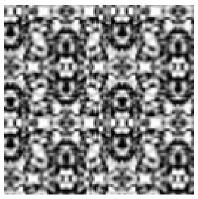
- **Step 2: Augmenting the Training and Testing data**
  Now that you have quickly trained on the original patterns, the next step is to train and test on augmented patterns. The purpose of this experiment will be to see if we can train the network to understand the patterns after the images have been rotated, scaled, and translated. 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 100% - 200% from the original 256×256 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

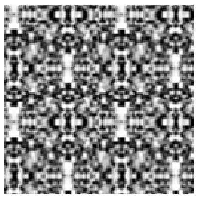
augmentations such as what you do at the border of the image. In your report, show at least 5 examples per augmentation and then combining all augmentations. Also, show the distribution of the augmentations with a histogram of the scales, rotations, and translations you have used. https://www.mathworks.com/help/deeplearning/ug/preprocess-images-for-deep-learning.html

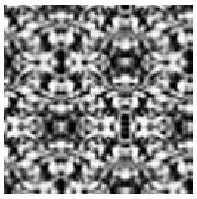
You should create folders called 'train_aug' and 'test_aug' in the 'data/wallpapers' folder. Within each of these folders, have a folder for each group (just like the train and test folders). The labels are automatically assigned based on the folder name (so all the images in the P1 folder are considered 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 **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.

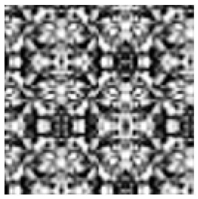
Remark:Since each dataset is unique so it is impossible to say how long it will take to converge but you can expect to not get a high accuracy here. It's ok if this doesn't really learn, just show that you tried.

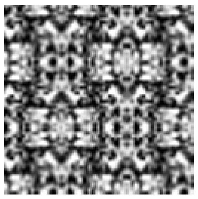
Figure 1 shows 5 examples of each transformation methods: scale, rotate and translate.

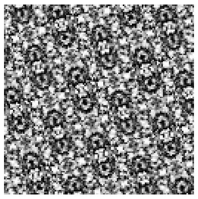

(a) Scale 1

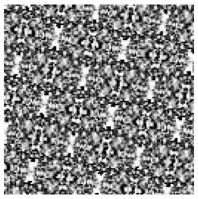

(b) Scale 2

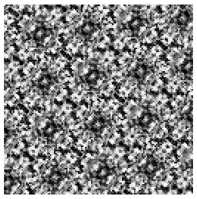

(c) Scale 3

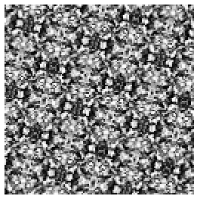

(d) Scale 4

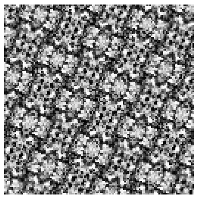

(e) Scale 5

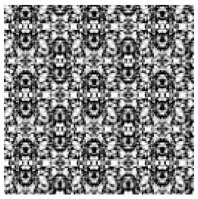

(f) Rotate 1

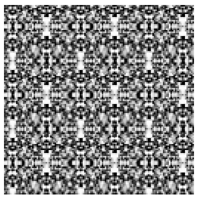

(g) Rotate 2

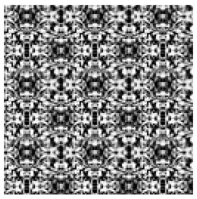

(h) Rotate 3

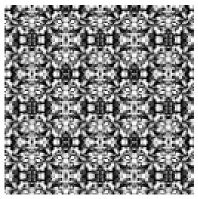

(i) Rotate 4


(j) Rotate 5


(k) Translate 1


(l) Translate 2


(m) Translate 3


(n) Translate 4

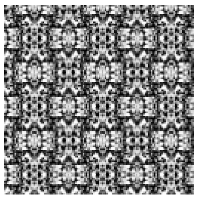

(o) Translate 5

Figure 1: Examples of different transform methods on CMM wallpaper group images.

- **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:

  $$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \rightarrow RELU \rightarrow FC$$
  $$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow [FC \rightarrow RELU] * 2 \rightarrow FC$$

  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 trade-off between more filters and more depth so you should explore this trade-off. 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.

  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.

  Remark: 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?

4 Requirements

1. You need to firstly train and test network in the starter code on the original dataset.
2. Then you need to do data augmentation, with at least 5 times per training image, and once per testing image with a **combination** of following augmentations:
   - 360-degrees of rotation,
   - Uniform scaling between 100% - 200% from the original image,
   - Valid translation in any directions without going out off the image,
   - Finally crop the image to 128×128.

   https://www.mathworks.com/help/deeplearning/ug/preprocess-images-for-deep-learning.html

3. You MUST train and test 2 networks on your own augmented dataset:
   - A **skinny** network: which 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: which 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.*
4. In addition to the two networks, you should implement transfer Learning, by fine-tuning a large pretrained network ($\geq$ to AlexNet in size) for this task. Describe exactly how you fine-tuned the network and evaluate it on your test sets. See an example of transfer learning using AlexNet in Matlab: `https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html`
5. Finally you need to do visualization on **ALL** of the networks:
   - Visualize the first layer of filters for **ALL** of your networks. You can use Matlab build-in function *deepDreamImage* to achieve this: `https://www.mathworks.com/help/deeplearning/ref/deepdreamimage.html`
   - 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:
     - `https://www.mathworks.com/help/deeplearning/ref/seriesnetwork.activations.html`
     - `https://www.mathworks.com/help/stats/tsne.html`

5 Report

Your report should include:

- Explain your augmentation process methods and parameters, and plot a histogram of the scales, rotations, and translations to show the distribution of the augmentations.
- Make sure to show a comparison of the original and augmented data in your report.
- For each required implementation, you must report the confusion matrix for the training, validation, and testing datasets. (Use *confusionchart* or *heatmap* function from Matlab to visualize the matrices)
- Show visualization results of each required implementation as "requirement" section mentioned.

- Record the amount of time you took training each network and the training accuracy and loss.
- Describe each network with the total number of parameters and activations for the entire network as well as for each layer. You can use Matlab function *analyzeNetwork* to achieve it. https://www.mathworks.com/help/deeplearning/ref/analyzenetwork.html
- What did you learn about creating convolutional neural networks?
- Does a decrease in loss/error directly translate into an increase in accuracy? Why or why not?

6 Extra Credits

From Taiji foot pressure to key poses:

1. By using foot pressure data only, train a CNN to classify different Taiji key-poses. (Up to 20 pts)
   Remark: for the data format of foot pressure and the Taiji key-poses to classify, kindly refer to project 3 description
2. Then do experiment on different labelling strategy. Show different training and testing results based on different pairs of (M, N), which indicates M frames before the key-frame and N frames after the key-frame to be labelled as the key-pose. (Up to 20 pts)

For extra part, you need to report the confusion matrix for training, validation and testing datasets. And show the visualization results together with a brief description of the whole network as required for regular project part.

7 CyberLAMP Cluster

Here are some reference link for you to get familiar with CyberLAMP clusters:

- https://ics.psu.edu/computing-services/icds-aci-user-guide/
- https://wikispaces.psu.edu/display/CyberLAMP/CyberLAMP+Documentation+and+User+Manual

8 Tips:

- **Important note:** This project is the last chance to use any of your late days. You CANNOT use late days for term project.
- 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.
- It is highly recommended you to use a computer with an NVIDIA GPU and Matlab to do this project. It can take 13-14 minutes per epoch for the network in the startercode network using only the CPU. But with a GPU, it takes $<$ 1 minute per epoch. If you already installed Parallel Computing Toolbox (which is a part of license by PSU software for students), then 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.

9 Common Issues

Notice that this section will be updated if more common issues are asked by the students.

- For every image in the training data set, do we have to augment the image as a combination of different values for rotation, translation, and rescale?

  Answer: Yes. Each augmentation should contain **all types of** transformation, as a combination of rotation, translation and scaling.

- When I test the starter code, error comes, it says can not find function: *imageInputLayer*

  Answer: Kindly check if you already installed Deep Learning Toolbox in your Matlab. It is free and availiable for PSU students with Matlab Student License.

- Can I use other programming languages for project 4?

  Answer: Yes, other programming languages are allowed for project 4 because it is a project about deep learning. So if you feel more comfortable with Python or some else language to work on it, that is acceptable. The only thing you need to know is that we do not provide Python or else starter code. Hence, you need to re-implement the network structure provided by starter code in other languages. And lateness (besides regular late days you have) because of using other languages **will not be accepted**, since we already provide you with the Matlab starter code.

- I cannot been assigned to CyberLAMP GPU nodes with the command in slides.

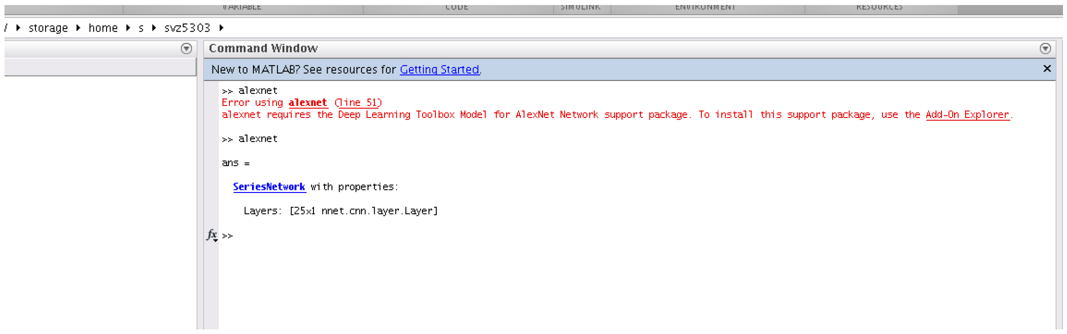
  Answer: You shall try the following command:
  qsub -I -A cyberlamp -l qos=cl_gpu -l nodes=1:ppn=1:gpus=1:shared -l pmem=4gb -l walltime=20:00

  Instead of specifying -l nodes=1:ppn=1 -l gpus=1:shared as two directives, please combine them to -l nodes=1:ppn=1:gpus=1:shared.

- AlexNet is not installed on CyberLAMP clusters.

  Answer: If you need to use AlexNet on the CyberLAMP clusters, you shall follow these steps to install the "AlexNet Network support package" by yourself:

  1. Make sure you have the X11 application running on your local machine, and X11-forwarding is enabled with your SSH session. Here is a reference link if you are not familiar with X11 forwarding: `https://netsarang.atlassian.net/wiki/spaces/ENSUP/pages/31654141/Using+X11+forwarding`
  2. Run the following commands to open Matlab:
     - module load matlab
     - matlab
  3. In Matlab interface, run "alexnet" and you shall see an error message as following figure shows:



```
New to MATLAB? See resources for Getting Started.

>> alexnet
Error using alexnet (line 51)
alexnet requires the Deep Learning Toolbox Model for AlexNet Network support package. To install this support package, use the Add-On Explorer.

>> alexnet

ans =

  SeriesNetwork with properties:

    Layers: [25x1 nnet.cnn.layer.Layer]

fx >>
```

Figure 2: Screenshot of Installing AlexNet

4. You need to click on ‘Add-On Explorer’, which will open a new window with ‘Install’ dropdown button, then click on ’Install’, this will ask you to login to your Matlab account and once authenticated, the package will be installed in **/Documents/MATLAB** directory automatically.
5. Now when you run ’alexnet‘ in command window, it should load the Network.

Remark: If you cannot setup X11 forward properly, you could directly connected to ”RHEL7 Interactive Desktop” app on Portal website and installed the toolbox in Matlab.

1. Go to https://portal.aci.ics.psu.edu/pun/sys/dashboard
2. Click on ‘Interactive Apps’ and then submit the form to get “RHEL7 Interactive Desktop” session. When the session starts, click on ‘Launch NoVNC in New Tab’ button
3. Once the remote session starts, go to ‘Applications’ ¿ ‘System Tools’ ¿ ‘Terminal’
4. Run the following commands to open Matlab:
   - module load matlab
   - matlab

- I’m trying to build my own network but I can’t make the accuracy more than 10%

  Answer: This is an experimental task, and it takes you some time to find a good parameter combination. Here are my suggestions:

  1. Always train your model with 2-3 epochs under different learning rate and batch size. If a model shows accuracy increasing/loss decreasing, then you do a refined search of parameters around the current baseline model. It could help you save time, and try to make the searching more targeted.
  2. When you observe any type of over fitting (e.g. high accuracy for training set, but low accuracy for testing), you may consider to add a dropout layer which is designed to prevent over-fitting.
  3. We **highly recommend you to do training on GPU machine** . Running 2-3 epochs will not take you too much time ($<$ 5 mins) for each attempt.
  4. Generally, a good design of network architecture should start with larger filter size and gradually decrease it in deeper layers. Meanwhile, you shall **at least double the number of filters** from the starter network for augmented dataset since it is more complicated and requires more computation capacity.
  5. To speed up training and reduce the sensitivity to network initialization, you may use **batch normalization layers** between convolutional layers and nonlinearities, such as ReLU layers. Here is a reference about batch normalization: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.batchnormalizationlaye html

6. **Always use checkpoints to save your model during training**. This will be extremely helpful if you observe over-fitting at the end of training. You can use checkpoints to resume training from an earlier stage.

- How much accuracy for testing and validation should we get for the augmented dataset for full credit?

  Answer: $\sim 50\%$ on testing and validation augmented dataset will be sufficient. But you are welcome to train it for several more epochs to get better performance! (just be careful of overfitting problem)

- Alexnet expects 3 dimensions of input which is $227 \times 227 \times 3$, but in our case it is only 1 dimension, which is $128 \times 128 \times 1$. How to proceed with this?

  Answer: We recommend you to do an image pre-processing from $128 \times 128 \times 1$ back to $227 \times 227 \times 3$. You can use *augmentedImageDatastore* with 'ColorPreprocessing' option to help you achieve this.