10701 Final Project

Jieli Zhou

May 12, 2017

Problem 1 Left/Right Footprint Detection

Data Pre-processing

The data, 10000 footprint images all have different sizes, so the first thing to do is to resize them to the same size. For the purpose of computation time, I resized all images to 28x28 for the CNN, 227x227x3 for alexnet, and 28x28 for autoencoder. Since all images are grayscale where each pixel has integer values from 0 to 255, we can consider converting them to double, or real values between 0 and 1. The second thing to consider is to augment data by rotating and adding noise. In the autoencoder model, I tried augment the data to 20000 images, but the train and test accuracy both dropped, so I stopped proceeding in this direction.

Convolutional Neural Network

This is a well-defined image classification problem, so we can use a classic image classification algorithm convolutional neural network. The first successful CNN was developed by Yann LeCun, and successfully classifed a small set of well-processed handwritten digits (10 classes) for above 90% accuracy. Since then researchers kept developing deeper and deeper networks to successfuly classify more and more classes of objects. For example, Alexnet won the ImageNet Competition by succesfully classifying a million images of 1000 classes. Because of these stunning results, I decided to construct a basic CNN to see whether CNN can achieve good results on our data set.

The fundamental building blocks for a CNN are

- a Input layer which takes in pixel values of each image;
- a Convolutional layer which acts like a local filter that transforms the input image to stacks of neurons by computing the dot products between the local pixel values and their corresponding weights and adding a bias
- a differentiable activation layer that thresholds each neuron value, this can be max(0, x), or relu.
- a pool layer which performs a downsampling along some spatial dimensions, this can be taking the maximum of a block or the average across a block;
- a fully-connected layer that computes scores for each class, in our case two scores.

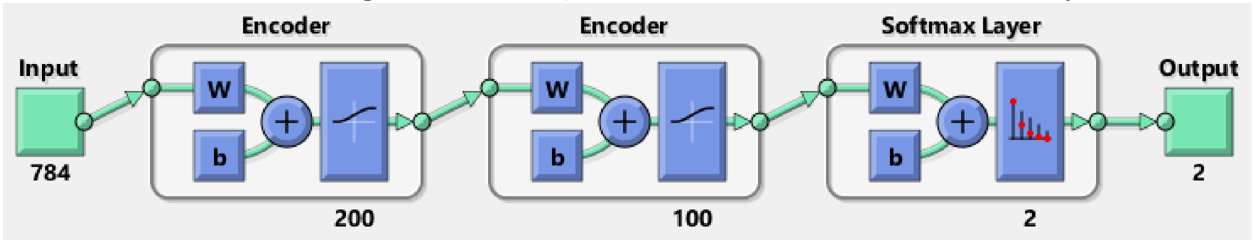
Then each image would go through this network first feedforwardly, then backward by backpropagating. This process use stochastic gradient descent.

We can then stack different combinations of Conv, relu and pool layers to make more complicated network models as long as the first layer is input and the last layer is output. Because of the limited computing power I have, I only tried Conv-Relu-MaxPool, and Conv-Relu-AvgPool-Conv-Relu-MaxPool. I trained each model 4-5 hours for about 10-20 epochs and achieve about 90% training accuracy and about 80% validation nd test accuracy.

Since the training accuracy is not very high, it is reasonable to increase model complexity, i.e. increasing number of layers. In addition, the large discrepency between train and test accuracy is also worth noting, so we need to worry about overfitting.

Transfer Learning Alexnet

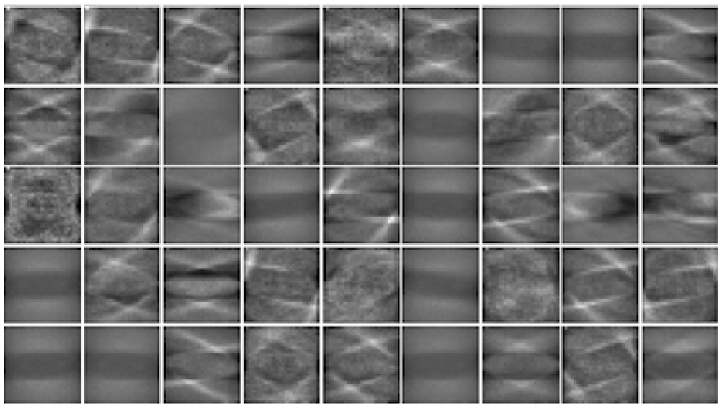
A complex CNN would need a large amount of time to train but now we want to increase both our train accuracy and test accuracy, so we can adapt a pre-trained CNN, by trucating the input and output layers, and ‘training’ on our dataset. The 25-layer Alexnet was trained on one million images of 10 classes. After running about 10 epochs on our 10000 images for about 3 hours, there is a clear issue of overfitting. While the test and validation accuracy both rise above 90%, the training accuracy approaches 100%. This makes sense since the network in a way memorizes all the information of the training data. However, it is not that horrible since test accuracy is above 90%.



Autoencoder

In a CNN, each layer can learn features at different levels of abstraction. However, as we see, if we stack more hidden layers, the computation time would skyrocket. One effective way is to train one layer at a time. We can use an autoencoder to achieve this purpose. An autoencoder is a feedforward neural network that aims to reproduce its input at the output layer. It does so by feeding forward the input pixels to a reduced-sized hidden layer, which is called an encoding process, then decode the hidden layers to output layer which is of the same size of the input layer. The degree of sucessfulness of the compression is measured by a loss function which compares the difference between the original image and the ‘decoded’ image. In our case, since we have 0-255 pixel values, we use Mean Squared Error with L2 Regulizer as our loss function. We can repeat this process by feeding in the first reduced-sized hidden layer to the second reduced-sized hidden layer, then feed the second hidden layer to a simple softmax layer to compute class scores. The constructed model is shown above.

If the compression is successful, we can even discover some meaningful features of abstraction like below. It is stunning we can achive 88% validation and test accuracy with such a simple neural network.



Conclusion

At last, I achived about 92% test accuracy using transfer learning of Alexnet. Further research should be done by more careful image pre-processing and feature engineering.

Problem 2 Footprint Matching

Data Pre-processing

Since there is only one training data for each class, directly running models would be distarstrous (although I achieved my best validation accuracy 0.7% by feeding the original data set into antoencoder). I augment the data 10 times by randomly rotating and adding Gaussian noise. One issue with random rotating is after each rotation, Matlab fills the gap with black pixels, but the validation set contains white pixels in these gaps. So we need to change these regions to white. The image sizes are resized accordingly as before.

CNN, Alexnet Transfer Learning, Autoencoder

I spent a long time training on the augmented data using neural network based models and tweaking parameters like pooling window sizes, learning rate, epochs, etc, and finally break the chance probability, though none of them produced validation/test accuracy $> 1\%$. However, most of them could achieve 100% training accuracy since there are only 10 images in each class. The key reason of my failure is I rotated the images randomly instead of using the rotation angles in the validation images. If one rotated all images to each angle present in the validation set, validation accuracy would be much better. Though CNN is location-insensitive, the rotated images can produce blank areas which can greatly affect feature extractions.

SVM on Features from Fully Connected Layers

Since the second last layer in CNN/Alexnet contains features of our images. We can train a simple classifer like logistic regression or SVM on them. However, this is not doing very well both on test/validation set. The reason might be the training set is not well-preprocessed like MNIST or CIFAR, and the noise masked up useful features. If we use layer from autoencoder, it does even worse.

Filters, Handcrafted features as in Computer Vision

After blindly trying different classfication methods for 3 days without breaking 1% test accuracy, I read a paper *Automatic shoeprint matching system for crime scene investigation.* The paper proposes to use Gabor transform to extract features from validation/test dataset. Basically, gabor transform segment an image to smaller parts and different directions in order for better match. This seems like what each layer in CNN is trying to do if we have enough training data. This could be a further research direction.

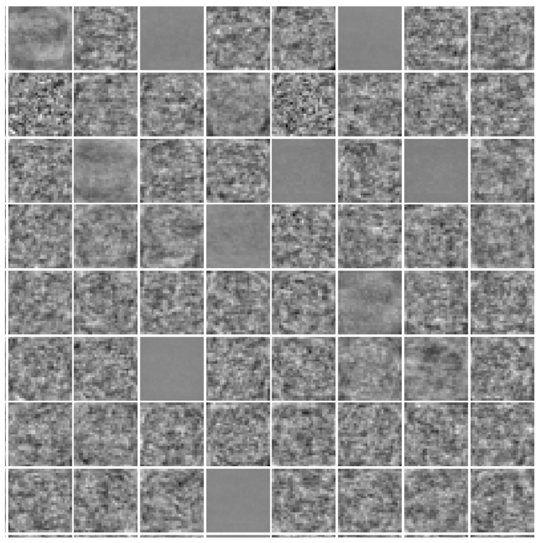
Lessons and Further Research

I achieved about 0.3% test accuracy and 0.7% validation accuracy by autoencoder and CNNs. This task lets me know that feature engineering is much more important than the algorithms. With bad and noisy features like in our case, even a very deep neural network cannot do anything meaningful. To augment footprint data, we can use Generative Adversarial Network, with one network to generate artificial images and another network to discriminate the true versus the artificial images. We can train the GAN until the discriminator cannot discriminate between true and artificial images. Another way, perhaps what people with high accuracy did, is to calculate the degree of rotation of footprint images in the validation set, and augment the images in the training set correspondingly. This way, the test images are more similar to the train images, and feature extraction in each hidden layer of the CNN can be more effective.

Problem 3 Pattern Discovery

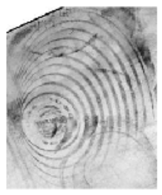
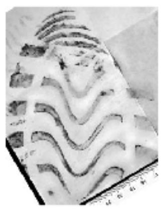
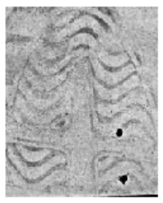
Autoencoder

Since autoencoder produces a reduced-sized hidden layer which can be used as features, naturally we approach this problem by resizing all the images and feeding them into an autoencoder. After some trials, by resizing all images into 56x56, and set the number of hidden nodes to be 200, we get a weight plot which shows some suspicious features like edges, loops, curves, etc. Here is an interesting snippet from the weight plot.

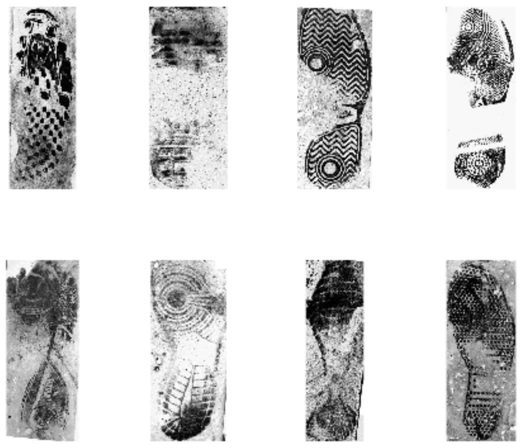


K-means clustering on Autoencoder features

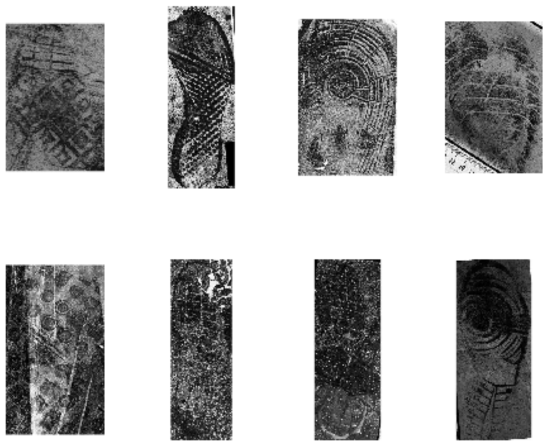
Since we can encode the features of the hidden layer in the autoencoder, we can do some clustering on top of them as well. So we use K-means to cluster all images to 10 clusters. The number of clusters are arbitrary since it is very hard to manually detect a good cluster number. However, when we plot some images of the same cluster alongside each other, we can see some distinct features. For example, these three images are in the same cluster. Apparently we can see they all have some curves.

These 8 footprints are also in the same cluster, as we can clearly see they have reasonably clear egdes.



These 8 footprints also belong to the same cluster. Though there are no high-level shared features, we can still feel they are very similar visually. For example, they all have a significant level of shadow in them.



Conclusion

Using the features extracted from an autoencoder, we can see there are indeed some features in these footprints like curvatures, shadows, edges, etc. However, none of them are very impressive compared to the clear features from MNIST/CIFAR found in other papers. The main reason is that our images are very noisy because of a lot of random noise, loss of information due to cropping, etc. For this purpose, we can use denoising autoencoder which delibrately introduces noise into training images. However, the better alternative would be to make more clean data. If we have very clean data, then the feature would be more distinct. This problem and problem 2 all remind us that data preprocessing is very important if not the most important in Signal Processing problems like footprint recognition problem.