Image Classification Using Phase Stretch Transform Neural Networks

Team #2: Aditya Gorla, Griffin Romanek, and Mai Saleh

1 Introduction

IMAGE classification is an important task for many fields, such as facial recognition in security, consumer targeting in marketing, and disease diagnosis in medicine. While it is possible to perform manually, automated methods for classification are valuable for tasks that require both speed and scale beyond human capability.

Convolutional neural networks (CNNs) have traditionally been used for automated image classification tasks. The convolutional layers of the network perform various types of feature extraction on images, such as edge detection, by applying filters across small sections of the image [1]. After convolution, often performed multiple times with pooling layers in between, the results are passed to fully connected layers of the network, which can learn correlations between the extracted image features and the classification labels. There are many different CNN architectures that perform well on different aspects of classification, such as LeNet-5 for handwriting classification [2] and AlexNet for recognizing features regardless of position or orientation [3].

CNNs are widely used because they generally produce accurate results [1]. However, the convolutional layers must learn many parameters and apply small filters many times across the same image, which can be computationally intensive and result in slow runtimes. For example, the famous AlexNet architecture uses approximately 60 million parameters [3]. While this is large even for CNNs, even simple CNNs can have thousands of parameters to learn [2].

A phase stretch transform (PST) provides an alternate approach to edge detection, imitating how an image would be affected when passed through a diffractive medium, as seen in Figure 1 [4]. One of its main benefits is that it preserves phase information about the image in the Fourier space, which was shown by Ashgari et al. to be useful data regarding the features of the image that is lost when phase information is not preserved [4].

This paper describes a new neural network architecture, the Phase Convolutional Neural Network (PCNN), which utilizes PST filters in the place of, and potentially in tandem with, convolutional layers in a CNN to perform image classification. This is because both PSTs and convolution perform edge detection, but a PST has the capability to be both faster and more informative than convolution. While any 2D image is data that can be used to train and test a CNN, we will use the MNIST fashion dataset of ten types of clothing and accessories as the dataset for the PCNN, as it is a standard test set and its images are 28x28, which will negate the need for long running times as the networks train and learn [5]. However, if successful, the PCNN could be applied to a plethora of different data types, such as MRI images, CT scans, X-rays, etc.

This paper posits that a PCNN theoretically can be both faster and more accurate than current CNNs. In terms of speed, the PCNN has promise because PSTs can be applied to an entire image at once, rather than applying a small filter many times across an image as is done in convolution [4]. Additionally, a PST filter only has three parameters that need to be learned (LPF, Phase strength, and Warp strength), rather than the hundreds or thousands that need to be learned in convolution. In terms of accuracy, the phase information preserved by the PST could allow for better classification, as this information is not present in convolution, leaving the network with less information to work with [4].

Constructing a PCNN is not without challenges. The phase information in the PST is captured in the complex number space, and traditional neural network training mechanisms such as gradient descent operate only in the realm of real numbers. Additionally, it is uncertain if filtering the whole image with a PST rather than the small windows of convolutional filters will result in the loss of key benefits of CNNs, such as being able to detect the same feature at various sizes and orientations.

A PCNN that is faster than current CNNs would be an incredibly beneficial architecture. Faster and more accurate image classification could be used to perform better and speedier medical diagnoses on MRIs and CT scans, or to improve any other possible image classification task. It is the goal of this paper to determine both an architecture and training mechanism for PCNNs that will allow it to perform at optimal speed and accuracy.

2 METHODS

2.1 Framework and Dataset

We chose to use Tensorflow 2.0 as the framework we would build the PCNN in. We chose Tensorflow because it has one of the best toolkits in place for creating custom layers and neural networks, which is perfect for constructing the custom PCNN architecture. Once the model is constructed, it also does the heavy lifting training-wise, which is another useful feature. We chose to use the Fashion MNIST dataset of 28x28 grayscale images, as it is the standard choice for testing image classification networks. The dataset consists of 70,000 images from 10 different classes. We split them into 60,000 training images, and 10,000 testing images, a typical train to test ratio [5].

2.2 PCNN

The purpose of this paper is to determine if the PCNN is better than traditional CNNs, either in speed or accuracy. Thus, we created a very simple architecture for this purpose. There is one PST layer which has a single PST filter applied to the entire image, the output of this is connected to a 128 node densely connected layer with a sigmoid activation function, which is in turn densely connected to a 10 node output layer. Even setting up this simple architecture took a great deal of time as it had to be custom built, and the entire PST algorithm had to be rewritten from its original form using NumPy functions to a Tensorflow compatible form (see the file PST_func.py in the github repository). The network trains the PST filters LPF, phase strength, and warp strength parameters. The construction of the layer required us to appropriately bound these trainable values between 0 and 1. We had to carefully pick an appropriate initializer to ensure that the initial values stayed within the bounds. To achieve this we used a truncated normal distribution with a mean of 0.5 and a standard deviation of 0.25. Since the initializers will only pick from values within two standard deviations of the mean, the layer will always initialize these parameters with values between 0 and 1. Although the PST algorithm itself works within the complex space, the LPF, phase strength, and warp strength parameters do not, and so we decided to try and train them with gradient descent, as this is the learning method that Tensorflow was meant to be used for. We used sparse categorical cross entropy as the loss function, as this is the typical choice for multi-class classification. It is important to note that the other parameters to the PST algorithm - Threshold_min, Threshold_max, and Morph_flag - are vestigial for our purposes, as the PCNN did not use the morphological variation of the PST function.

2.3 CNN Comparisons

We next built several extremely basic CNNs to serve as a comparison for the PCNN. The first included a layer of 32 convolutional filters with a 3x3 kernel, connected to a 128 node densely connected layer, which was densely connected to a 10 node output layer. This network would be similar to what someone implementing a single layer CNN might reasonably construct, however our baseline PCNN includes only one PST filter, not 32. Thus, we created a second CNN that was the exact same as the first, but uses just 1 filter and a 3x3 kernel. In the rest of the paper, we will refer to these baseline CNNs as CNN1 and CNN2, respectively.

2.4 Testing

The code is in the private repository found at this link (https://github.com/Adigorla/PCNN.git). To run the CNN networks, download the appropriate packages and run "python3 ./tf_prac1.py" from within the github repository. Line 23 in this file can be changed to adjust the number of filters and the kernel size of those filters within the CNN. To run the PCNN, run the command "python3 ./pst_m1.py" from within the repository. Both of these commands will train and test the respective networks for 10 epochs, and report the training and testing times, losses, and accuracies to standard output.

3 Results

After CNN1, CNN2, and the PCNN were trained and tested for 10 epochs each, the training loss, training accuracy, testing loss, testing accuracy, training time, and testing time were all recorded. In Figures 2, 3, and 4, plots of the train and test accuracies for each type of network are shown on an example run of the program, with a value recorded at the end of each epoch. Figure 5 shows a table for comparison with all of the above statistics as recorded at the end of the tenth epoch for each type of network.

CNN1 achieved the lowest loss and the highest accuracy, although the model began overfitting the data around the sixth epoch. It achieved a 95% training accuracy and an 89.5% testing accuracy after 10 epochs. It had a testing loss of 0.41, and typical training and testing times per epoch were 48.72 seconds and 1.42 seconds respectively.

CNN2 achieved the second highest accuracy, and did not begin overfitting during the 10 epoch training/testing period. It achieved a 88.83% training accuracy and an 86.06% testing accuracy after 10 epochs. It had a testing loss of 0.38, and typical training and testing times per epoch were 9.07 seconds and 0.37 seconds respectively.

The PCNN achieved the lowest accuracy, and did not begin overfitting during the 10 epoch training/testing period. It achieved a 80.68% training accuracy and a 75.3% testing accuracy after 10 epochs. It had a testing loss of 0.76, and typical training and testing times per epoch were 129.81 seconds and 11.92 seconds respectively.

4 Discussion

The results show that our basic, single PST layer PCNN implementation is neither as accurate, nor as fast, as both a 32 filter convolution layer CNN or a single filter layer CNN. Both CNN1 and CNN2 had 10-15% greater testing accuracy than the PCNN, and trained at a minute or more faster than the PCNN per epoch. Thus, this paper failed to show that the PCNN is a better alternative to traditional CNNs, however the case is certainly not closed. The PCNN, even with just a single PST layer, still had a 75.3% testing accuracy after 10 epochs, which while not fantastic, is still decent and leads us to believe there is promise in future experiments building upon this basic PCNN.

4.1 Strengths and Weaknesses

While we failed to show that the PCNN is better than traditional CNNs, the results still showed many strengths in the idea. Perhaps most importantly, the PCNN improved its test accuracy with each epoch, which demonstrates that the model was indeed learning, and that gradient descent is a viable option to train the PST parameters (LPF, Phase Strength, Warp Strength). Because the values of those parameters are not necessarily better or worse at a given higher or lower numerical value, we were not sure if the PCNN would even be able to learn with gradient descent. Our results have shown that it can. Additionally, 75.3% testing accuracy is not so bad that it indicates there is no hope that some form a PCNN can be better than a CNN, as random chance guessing would have produced approximately a testing accuracy of approximately 10%. The code written for this paper also provides an easy to build upon foundation for those looking to use the PST in Tensorflow applications or further experiment with larger and more complex PCNNs.

One weakness in the experiment was the simplicity of the architecture; a single layer PST layer with just one filter is not a realistic comparison to the complexity of a network one might use in the industry. Furthermore, it is hard to say if a single convolution layer is an apples to apples comparison to a single PST filter, as the convolutional layer uses a sliding kernel. The PCNN also has much longer training and testing times. This is probably because the PST is more computationally expensive than convolution, requiring both a regular Fourier transform and inverse Fourier transform on the image [4]. In order for a PCNN to be practical, this process would need to become more efficient.

4.2 Future Experiments

There are several possibilities as to why the CNNs outperformed the PCNN, each of which are opportunities for future experiments to explore. One reason could be that the CNN filter is applied via a 3x3 kernel shifted across the whole image, which allows the filter to pick up on small features in the image. On the other hand, the PST filter is applied to the entire image at once, which might cause the network to lose sight of smaller features. A future experiment could explore using a PCNN with PST filters that are applied in a similar fashion to convolutional filters using smaller kernels that are slid across each pixel.

Future experiments also could explore having a PST layer with multiple PST filters instead of just one. Each filter could potentially learn different LPF, warp strength, and phase strength parameters, which would allow the network to notice features that one PST filter might have missed. The reason this could be a promising path to explore is that in the Jalali Labs paper about PST, you can notice that the PST captures certain edges really well, such as the strands of hair in the Lena image, but completely misses other features such as her nose [4]. With different input parameters, another PST might be able to capture her nose but miss out on her hair. If we combine both of those PSTs there is the potential to capture both her hair and her nose, which would provide more image information to help the PCNN classify. However, it is also possible that in a PST layer with multiple PST filters, all of the filters might learn the same parameters or different parameters that produce the same results, adding no new information to the network but increasing the training and testing time even more.

Another possibility for future experiments could be creating a PCNN with a mix of both PST and convolution layers. The results could be compared with a multi-layered CNN. This has promise because while PST misses some edges, the edges it does detect it detects very well. Having convolution to help identify specific features could help the PST catch all features in the image, while the PST might do a better job detecting edges within those features than just convolution alone. Lastly, future experiments could use larger images, as they will likely contain more edges and information than the fashion MNIST images, which might lead to better performance by the PCNN.

5 Conclusion

The purpose of this paper was to explore the use of the Phase Stretch Transform in neural networks used for image classification as a possible alternative or supplement to convolution. We found that while not necessarily poor in performance, the PCNN did not perform as fast or as accurately as traditional CNNs used for comparison. However, it performed well enough, with a testing accuracy of 75.3%, to warrant future exploration, detailed in the section above. This paper used only a simple architecture with one PST layer composed of a single PST filter. Future experiments can use this architecture as a baseline to add onto and explore more complex architectures of PCNNs, with the hope of developing a better paradigm for classification tasks. If successful, this would allow for advances in a range of applications, such as security, medical diagnosis, species identification, and many more.

References

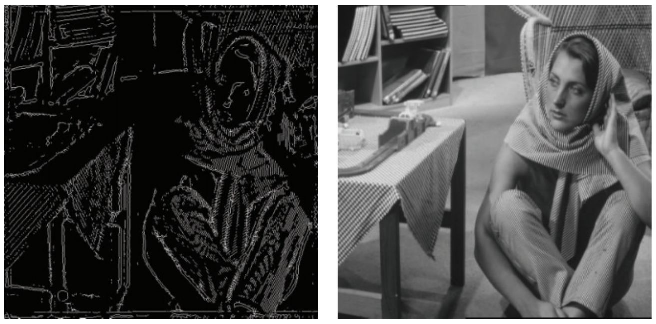
[1] N. Sharma, V. Jain, and A. Mishra, "An analysis of convolutional neural networks for image classification," *Procedia Computer Science*, vol. 132, pp. 377–384, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.05.198

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] M. H. Asghari and B. Jalali, "Edge detection in digital images using dispersive phase stretch transform," *International journal of biomedical imaging*, vol. 2015, 2015. [Online]. Available: https://doi.org/10.1155/2015/687819

[5] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.



(h) $W = 14$, $S = 50$ (i) Original image

Fig. 1. A black and white image (right) and the resulting image after a Phase Stretch Transform with warp strength (W) of 14 and phase strength (S) of 50 pictured on the left. As is easily observed, the PST detects the edges of the original image [4].

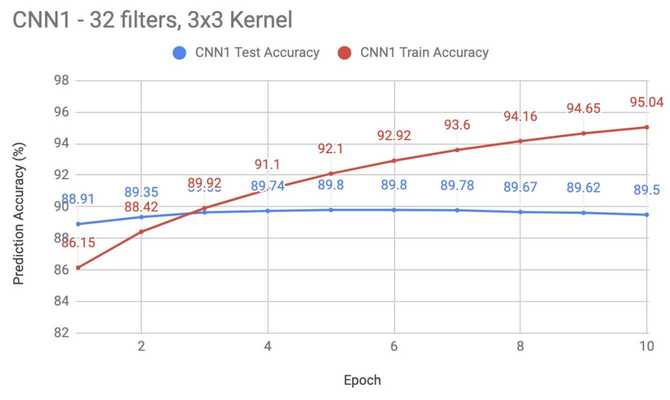


Fig. 2. Training accuracy (blue) and testing accuracy (red) of CNN1 over ten epochs.

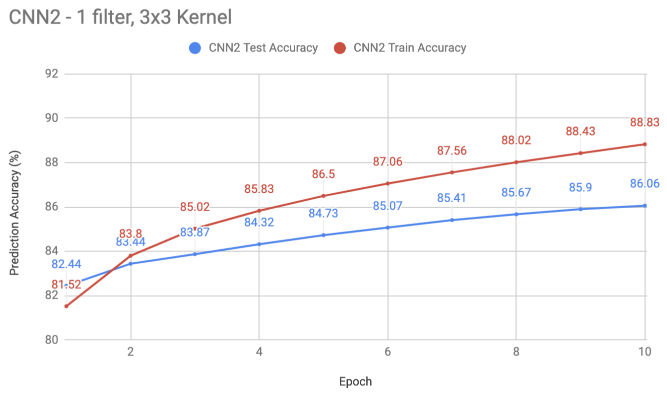


Fig. 3. Training accuracy (blue) and testing accuracy (red) of CNN2 over ten epochs.

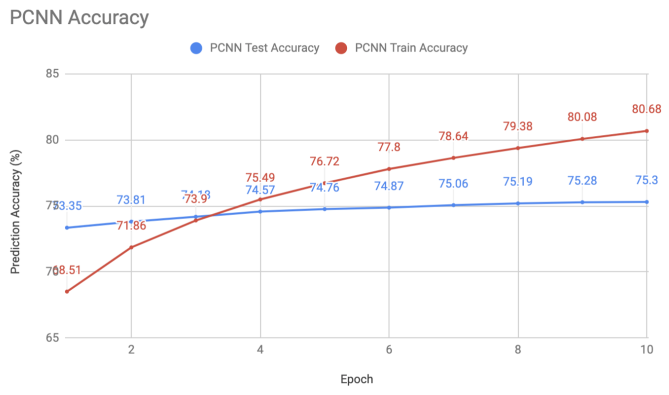


Fig. 4. Training accuracy (blue) and testing accuracy (red) of PCNN over ten epochs.

	CNN1	CNN2	PCNN
Training Loss	0.14	0.31	0.55
Training Accuracy	95.04%	88.83%	80.68%
Testing Loss	0.37	0.38	0.76
Testing Accuracy	89.50%	86.06%	75.30%
Training Time (per epoch)	48.72	9.07	129.81
Testing Time (per epoch)	1.42	0.37	11.92

Fig. 5. Loss, accuracy, and speed statistics for training and testing of PCNN, CNN1, and CNN2 for comparison. The PCNN is both the slowest and least accurate, while the CNN1 is the best predictor and CNN2 is the fastest.