Eager Training of Autoencoders in Parallel

Mudit Bhargava
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
mbhargava@cs.umass.edu

Mohit Surana
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
msurana@cs.umass.edu

Abstract

We present a novel combination and training strategy for autoencoders for unsupervised feature extraction and dimensionality reduction. Multiple autoencoders are trained in parallel and are eagerly stopped before reaching the minimum possible reconstruction error on the validation dataset. We show how, in low data environments, the diversity in the parallel autoencoders leads to more robust feature learning in a faster time, capturing salient and disparate features in the data while preventing memorization and overfitting when compared to a single monolithic autoencoder. Convolutional autoencoders (CAE) are used to study and demonstrate the performance on the MNIST (digit recognition) and CIFAR-10 (object recognition) datasets.

1 Introduction

Autoencoders [1] have been in use for unsupervised feature extraction and dimensionality reduction. Their unsupervised training procedure makes them a very powerful tool. The autoencoder takes input features and transforms them onto a new latent space where examples with similar properties tend to cluster near each other.

Although autoencoders were able to transform high dimensional data into low dimensions, it came at the cost of high training time and often would be prone to weak generalization performance. Especially in the case of image data, where the input dimensions are very high, this problem is magnified. In the recent years, new architectures have built upon autoencoders to help them generalize better. The convolutional autoencoder has convolution and convolution transpose layer stacked. Due to parameter sharing, these models tend to have high generalization performance. Dropout [2] has also been used to prevent co-adaptation of feature weights and has improved the performance, at the cost of increased training time.

Autoencoders have been very effective in environments where there is abundance of unlabelled data or in semi supervised use cases. Many attempts have been made to tackle the problem of low data, eg. while dealing with image data, rotations and other minor distortions were made to create more examples to train on. Denoising autoencoders [3] were able to learn much more robust representations discarding the unnecessary noise and retaining salient features present in the input. But in some cases, such changes to the data may have unintended consequences. Medical data, for example, requires carefully processing of data where minor perturbations to the data may lead to incorrect learning by the model. When lives are at stake, the confidence in such methods tend to drop.

Another crucial issue with low data environments is that we cannot make use of additional compute resources even if we have them at our disposal. Model parallelism involving splitting of weights across nodes will not be helpful as data will not allow us to train larger models without the risk of overfitting. In these cases, it would be better to just train a small classifier that fits into a single node. Data parallelism involving splitting the entire dataset across the nodes and synchronization between weights. Again, the low amount of data prohibits the use of data parallelism as the overheads

outweigh the benefits. The convolutional layers may benefit through parallelism on a GPU but the speed up is limited beyond a point.

We wish to tackle these two problems of low training data and long training time with good generalization performance and propose a new architecture that combines multiple autoencoders that can be trained in parallel. To prevent these models from overfitting to the small training sample, we eagerly stop the training process before reaching the minimum reconstruction loss on the validation dataset. The results are combined via a combined voting algorithm and classifier. The intuition is that the random initialization of the autoencoders and the eager training will cause them to learn different features in the latent space quickly. The combination method will automatically perform bagging operation on the individual autoencoders and we study if we can get the same accuracy in lesser time or better accuracy in the same time. This way, we aim to utilize all the resources at our disposal and still have good generalization performance in our model.

We used convolutional autoencoders and trained them in parallel on MNIST and CIFAR datasets. The trained autoencoders were then fed to a classifier for accuracy prediction on these datasets. It was observed that parallel autoencoders tend to perform better than monolithic autoencoders when trained for less time (fewer epochs). Performance of both parallel and monolithic autoencoders tend to converge as the training time (epochs) increases

2 Related work

The initial versions of autoencoder [1] was composed of stacked Restricted Boltzmann Machines (RMBs). This was trained using a greedy layer wise pretraining step followed by fine tuning via gradient descent. To build upon the representational power, deeper stacked networks called deep belief nets were introduced accompanied by new training algorithms[4] that not only used layer wise pretraining but also utilized the concept of complementary priors. The model was equivalent to stacked RBMs, and used methods like contrastive divergence and Gibbs sampling. Although good performance was obtained via these methods, these models took too long (a few days) to train.

Building upon their work, Santara et al. [5] posited that greedy pretraining is usually detrimental as one may overtrain one layer causing it to lose harmony with the other layers of the network. To alleviate this issue, they developed a faster learning algorithm for deep stacked autoencoders on multi-core systems using synchronized layer-wise pre-training and obtained 26% speed-up for achieving the same reconstruction accuracy. The primary idea was to reduce the idle time of greedy layer-wise pre-training by introducing parallelism with synchronization while controlling the synchronization frequency to amortize the overheads due to parallelism.

Newer methods like the use of better activation functions (like ReLU), adaptive gradients and optimizers (like Adam) allow us to train deep networks without the need for pretraining. Thus, we do not focus on pretraining in our study.

Very similar to our parallel autoencoder architecture, Modular Auto Encoders [6] aim to target the problem of redundancy in the features learnt by the parallel autoencoders. They utilize the concept of diversity to learn a diverse collection of modular feature extractors in an unsupervised fashion. They use an ensemble of multiple smaller autoencoders and a composite loss function between reconstruction loss and diversity in learning of the auto-encoders combined with an SVD approach at the end instead of gradient descent. This achieves good results but entails overheads of synchronization between the autoencoders at every step.

Closely related to the concept of diversity, KATE [7] was a autoencoder model for text where, instead of using all neurons, the top-k activations are allowed to pass unrestrained. For the others, a fraction of the activation was allowed to pass to improve learning. This fraction could even be greater than 1.0 to amplify the learning process. The intuition behind this was to encourage diversity in the learning across neurons of the same model.

Though training autoencoders takes time, bagging (bootstrap aggregation) is very easy for parallel computing and greatly reduces generalization error. It involves duplicating the training data and training multiple models combined with a classification layer to compute the final class probabilities and the prediction from the different models are combined using voting aggregation. The hope is that due to the non linearity, different models will learn different things and this has shown to boost the classification accuracy in multiple tasks. In one case, an object retrieval task [8] involved

images being transformed using a convolutional layer and the latent space was fed to an set of bagged autoencoders. Our model aims to borrow ideas from the use of bagging coupled with autoencoders to enhance the performance of upstream tasks. Stacked convolutional autoencoders [9] combined multiple layers of pretrained convolutional autoencoders to form a powerful convolutional network. They obtained promising results on the MNIST and CIFAR-10 datasets.

Our solution space encompasses low data environments with little or no labelled data where it is difficult to assess performance by just looking at reconstruction accuracy or extrinsic supervised tasks. Data parallelism cannot be exploited due to small amount of data and model parallelism cannot be used either because training larger models might lead to strong overfitting or memorization of examples without learning salient features of the inputs.

3 Methodology

3.1 Use of CNN

The drawback in the use of fully connected autoencoders is that it not only ignores the spatial information implicit in the 2D structure of images, but also leads to issues in dealing with larger sized images due to the sheer number of parameters to be learnt. In recent years, convolutional neural networks (CNNs) and CNN based architectures have achieved state of the art results in vision and related tasks. This is due to the learning of localized features captured via filters and their detection even when the position of an object is distorted within the input. The convolution operation done by these filters capture spatial information and are more translation invariant.

3.2 Convolutional autoencoders

Convolutional autoencoders build upon conventional autoencoders by utilizing these ideas. They have weights that are shared across all locations in the input, thus preserving spatial locality. The unsupervised nature of autoencoders allows easy training and the convolutional features make it a strong fit for assessing performance of our model on the MNIST and CIFAR-10 datasets.

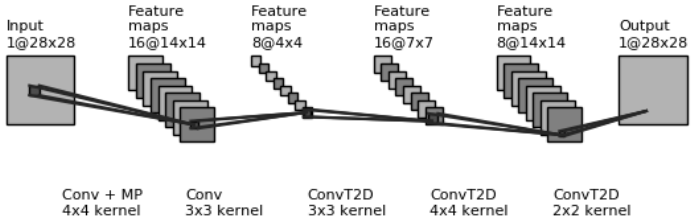

Figure 1: Architecture diagram of a single convolutional autoencoder for the MNIST data.

The convolutional autoencoder make up the basic building block of our solution. We use the above configuration while performing our study. We train the network on different sized subsets of the MNIST and CIFAR-10 datasets. The loss used for the reconstruction error was the squared error averaged over all training examples, as follows:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2^2 = ||\mathbf{x} - h(g(\mathbf{x}))||_2^2 \tag{1}$$

$$\mathcal{L}(\mathbf{X}, \mathbf{X}') = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{x_i}, \mathbf{x_i'}) \tag{2}$$

where $\mathbf{x}$ is the input image, $\mathbf{x}'$ is the reconstructed image, g is the encoder function and h is the decoder function.

While the training is being done, we log additional information for the purpose of profiling and the results in terms of accuracy, loss and time are interpreted and presented in the experiments section.

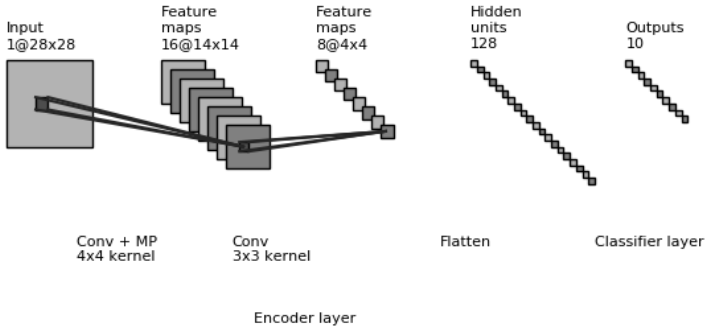

Figure 2: Architecture diagram of the classifier for the MNIST data.

3.3 Classifier architecture

The next stage of our project involves creation of the classifier. The encoder is taken out from the autoencoder and the input images are passed through it to obtain the activations. These activations are fed to a classifier that learns to predict in a supervised manner.

3.4 Multi CAE classifier architecture

Once we are ready with the basic version of the convolutional autoencoder and classifier, we design the multiple convolutional autoencoder architecture. The multiple autoencoders can be trained in an embarrasingly parallel way but the recombination step is expensive as the classifier for the next layer takes in more input features and hence needs to do more work to grasp what the input features really mean. Figure 3 shows how the multiple convolutional autoencoders are combined to obtain results via a voting

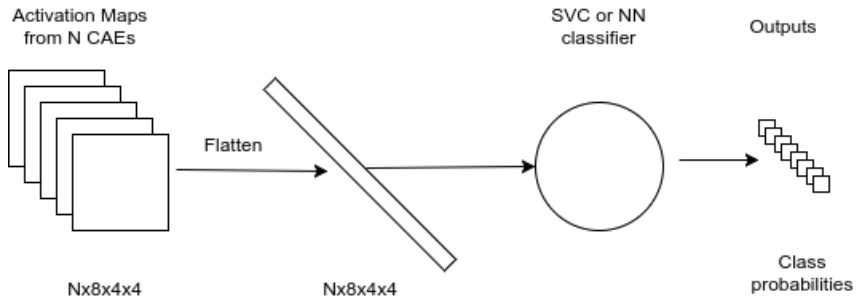

Figure 3: Architecture diagram of the classifier for the multiple CAEs

3.5 Motivation

We hypothesize the following:

- For low amounts of data, the performance of our proposed model will be much better than that of the monolithic autoencoder as we leverage the benefits of bagging in our model.
- As the dataset size gets larger, the effect of bagging is mitigated if the monolithic network is powerful enough to capture the underlying features in the dataset.
- We should be able to get same performance in lesser time or better performance in the same time.

The timing diagram in figure 4 shows what results we hope to achieve.

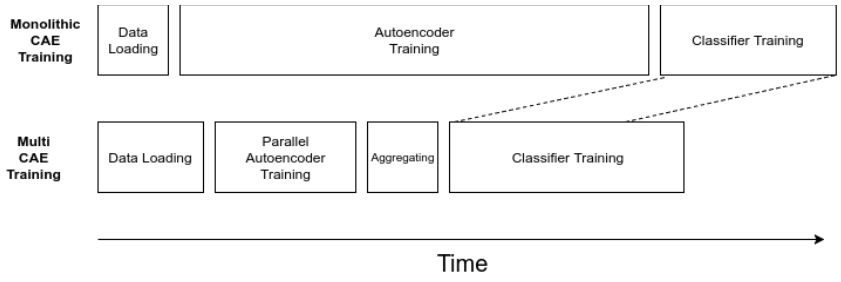

Figure 4: Timing diagram for monolithic CAE training vs multi CAE training

The experiments section highlights what we have carried out to assess the results of our model. For every graph, we run the program for three times and take the average result to get a smoother and more interpret able results.

3.6 Implementation details

- We picked PyTorch in favour of TensorFlow as both of them provided nearly equal expressivity in terms of the features that we wanted to tweak.
- The convolutional network architecture was inspired by the LeNet architecture.
- The layers of the convolutional network had max-pooling to iteratively downsample the image, but also increased the number of filters and thus feature maps to capture more information.
- ReLU non linearity was used used in favour of Sigmoid or other non linearities as it counters the issue of vanishing gradients.
- Inbuilt weight initialization provided by PyTorch seemed to perform reasonably well
- Adam was used as an optimizer due to its adaptive nature and robustness compared to stochastic gradient descent.
- As a convergence criteria for the monolithic autoencoder, we looked at the change in validation accuracy. The criteria for stopping was a strong increase in the validation accuracy for consecutive iterations.
- For the eager autoencoders, we trained the autoencoders for multiple epochs and dumped the models. Then based on how much time the classifier would take, we would pick an older model to compare.
- Thus the convergence criteria for the multiple autoencoders was judged after conducting an initial round of experiments.

4 Dataset

We have carried out our analysis using the MNIST[10][1] and CIFAR-10[11][2] datasets.

The MNIST dataset contains 28x28 sized grayscale images of handwritten digits labelled from 0 to 9. It contains 60,000 training examples and 10,000 test examples. For the purpose of our study, we do not perform any form of preprocessing on the images. Given the small size of the input features, fully labelled dataset and the availability of benchmarks to compare it against, the MNIST dataset makes for an ideal fit for our study. The state of the art result is 0.21% error [12] and can be used as an indicator helping us ascertain convergence of our models.

The CIFAR-10 dataset is similar to MNIST – it contains 60,000 coloured images of size 32x32 belonging to 10 mutually exclusive classes. The split between training and testing data is 50,000 and 10,000. The same reasons for picking the MNIST dataset apply to the CIFAR-10 dataset as well. The state of the art performance on this dataset is is 2.72% error. [13]

[1] `http://yann.lecun.com/exdb/mnist/`

[2] `http://www.cs.utoronto.ca/~kriz/cifar.html`

Figure 5: Sample images from the MNIST and CIFAR-10 datasets.

Both these datasets have user friendly wrappers in top machine learning frameworks making the data loading process easier. Thus we can shift our focus to tackling our actual problem in hand instead of having to speed much time on unnecessary details. For the purpose of assessing performance in low data environments, we utilize subsets of these datasets.

We randomized the datasets, split into training, validation and test datasets and performed minibatching with a batch size of 100.

5 Experiments

The primary focus was not just the implementation of the architecture but to design a conceptually strong set of cohesive experiments to correctly assess its performance. To this end, we made our code modular and have saved a lot of information during training so that post training analyses can be done much more effectively.

While the autoencoders are being trained, reconstruction loss on the train data and the validation data are calculated and saved along with the time taken on a per epoch basis. Once logged, this information can be read by another script for the purpose of running analyses.

The experimental setup is as follows:

- The monolithic autoencoder is trained for 100 epochs. The autoencoder models are saved after each epoch for later use.
- Multiple autoencoders are trained in parallel or one after the other. The results are aggregated based on the effective wall clock time required to train them.
- After optimizing for reconstruction, the monolithic autoencoder's encoder is used to generate activations for the training data. These activations are used as input features to run supervised learning to classify the digits.
- For multiple autoencoders, some additional overhead is involved in stacking together the activations from all the separate encoders. Another classifier that takes as input these activations is trained.
- All of the operations are timed and assessed post training.

6 Results

The results for MNIST, as expected were different for large(48k training, 12k validation) and small(4.8k training, 1.2k validation). We test for 5 parallel autoencoders. For large datasets, both the monolithic and parallel autoencoders learn and perform almost equally well on the validation set. Figure 8 shows the reconstruction and validation accuracy for Monolithic and Parallel autoencoders.

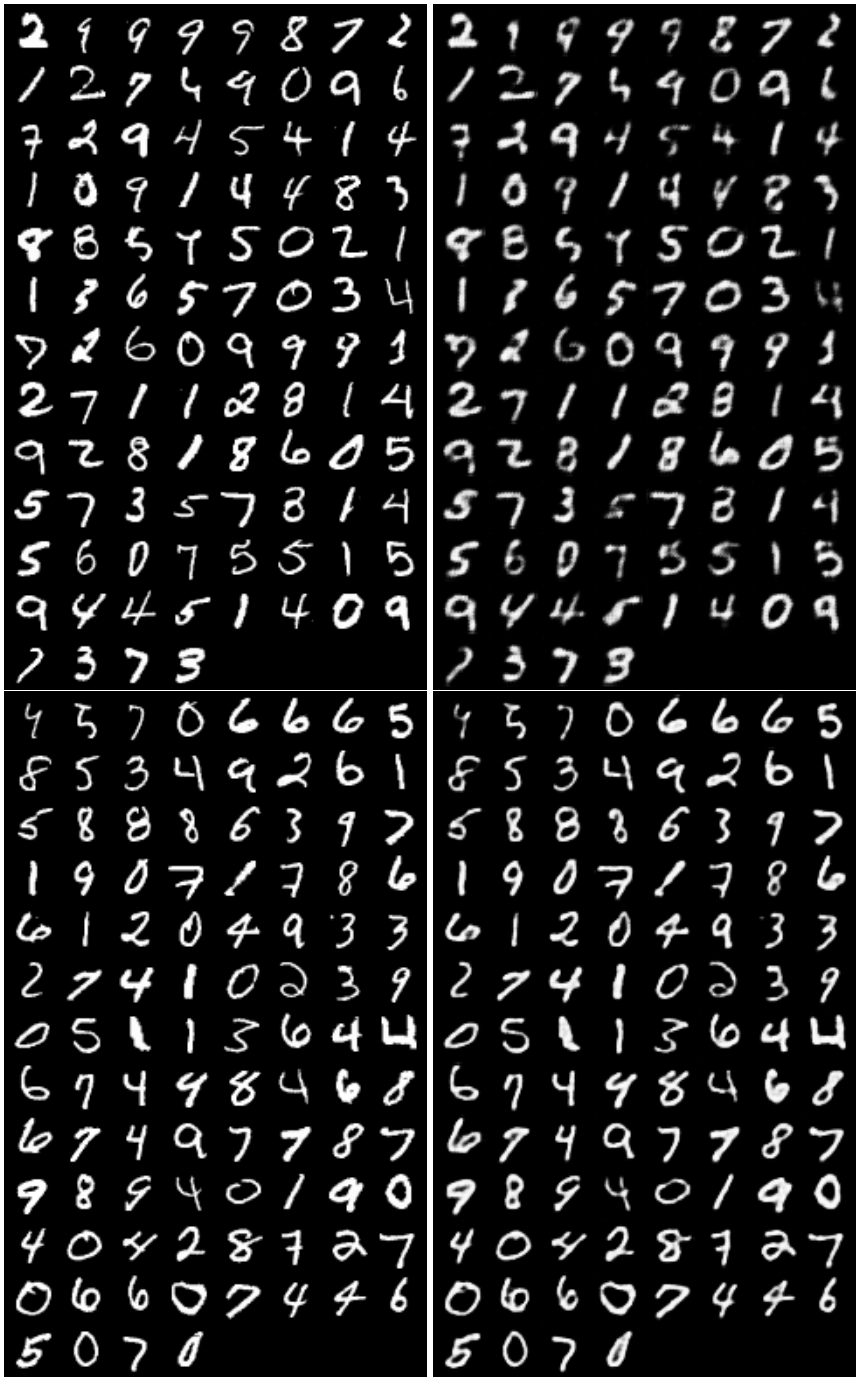

Figure 6: Original and Reconstruction for MNIST for 1st and 10th epochs

For small datasets, the parallel autoencoders tend to have a higher classification accuracy when the number of epochs are low. It should also be noted that for low number of epochs 3/5 parallel autoencoders have a slightly better learning curve than the monolithic autoencoder, while 2/5 have a much worse learning curve. We attribute this to the random initialization that caused those models to being from an unfavourable spot on the objective function. As the number of epochs increases, the reconstruction loss and classification accuracy for parallel and monolithic autoencoders tend to converge.

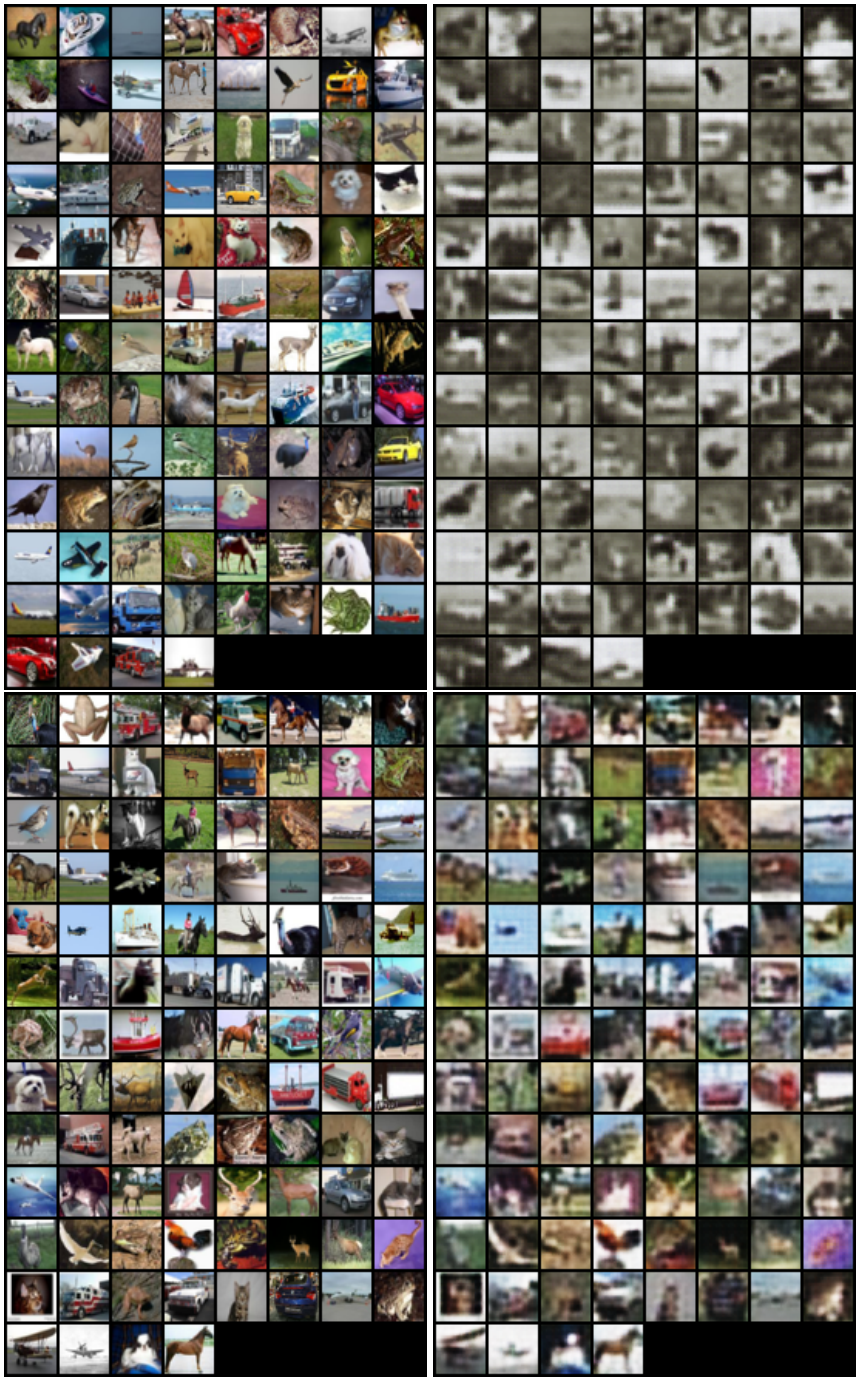

Figure 7: Original and Reconstruction for CIFAR10 for 1st and 10th epochs

For CIFAR-10 dataset, our Convolutional Autoencoder did not perform very well. The classification accuracy for reconstructed outputs was around 40% and monolithic and parallel autoencoders performed equally badly.

This highlighted a very strong point against our intuitions. We had hoped that while trying to optimize for the reconstruction loss, the autoencoders would learn important features of the input dataset. But in the absence of backpropagation of the error gradients across the autoencoder, as is the case for end to end training with fine tuning, the autoencoders had discarded information that was important to distinguish between classes.

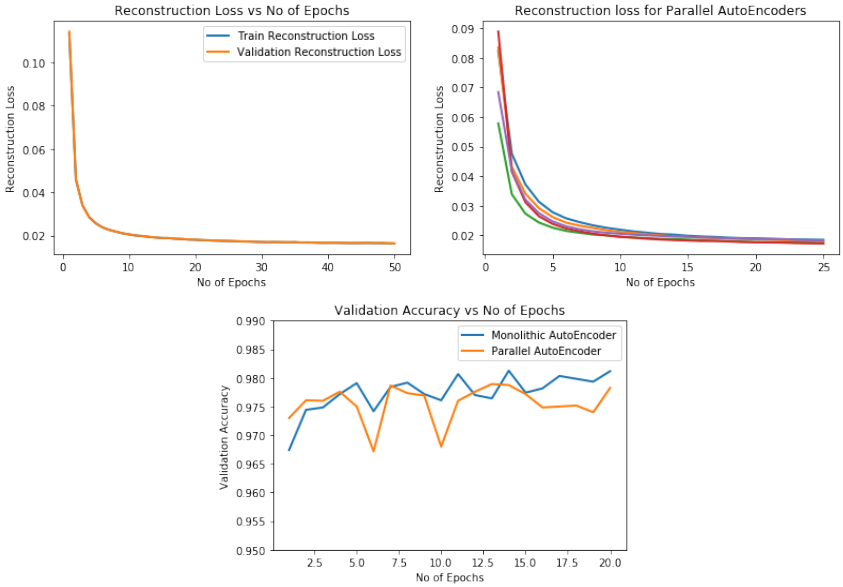

Figure 8: a) Large dataset reconstruction loss for Monolithic and Parallel Autoencoders b) Small dataset classification accuracy for monolithic and parallel autoencoders

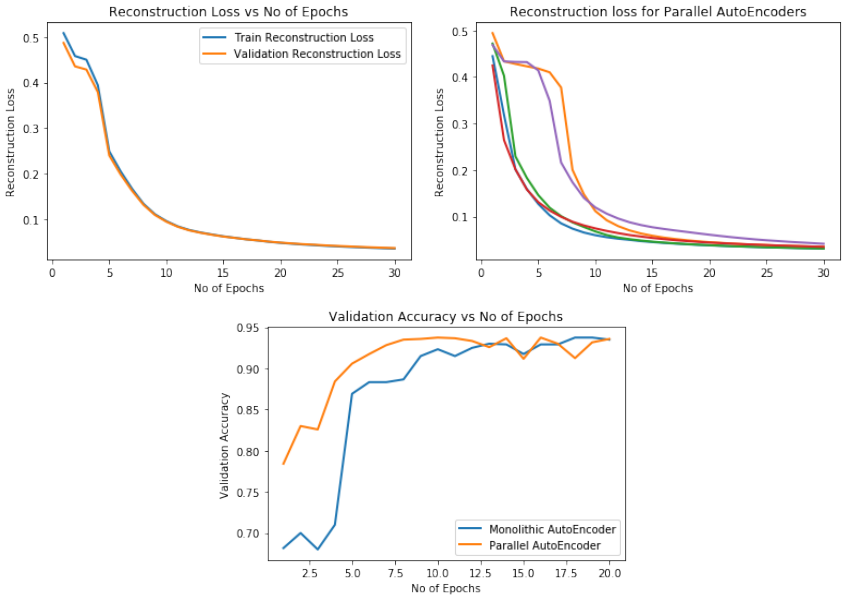

Figure 9: a) Small dataset reconstruction loss for Monolithic and Parallel Autoencoders b) Small dataset validation accuracy for monolithic and parallel autoencoders

7 Discussion and conclusions

We show that although the intuition behind the eager training has merit, it is hard to obtain good performance without performing of fine tuning. The optimization of reconstruction in an agnostic way leads to it remembering features useful for reconstruction but not for any other upstream class. It is still hard to assess the performance of a feature extractor without streamlining it to an upstream task.

The high dimensionality reduction does not work well and we need to find the right balance between dimensionality reduction and feature extraction. Non convolutional autoencoders were harder to train and performed worse than convolutional autoencoders as expected.

In the future we would like to assess how other variants like variational convolutional autoencoders, denoising convolutional autoencoders and other models in general perform when they are eagerly trained. There would be some class of problems that this could solve in the future, but we have not found them. Yet.

Acknowledgments

We thank our course instructor Prof. Benjamin M. Marlin for giving us this opportunity to build upon the concepts learnt in class through this project. His guidance helped us during the formulation of the proposal. We also wish to thank Steven Li, who provided us with invaluable feedback and suggestions regarding how to take our project forward. We also thank our friends and classmates for their continuous support during the course of the project.

References

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[2] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[3] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[4] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[5] Anirban Santara, Debapriya Maji, DP Tejas, Pabitra Mitra, and Arobinda Gupta. Faster learning of deep stacked autoencoders on multi-core systems using synchronized layer-wise pre-training. *arXiv preprint arXiv:1603.02836*, 2016.

[6] Henry Reeve and Gavin Brown. Modular autoencoders for ensemble feature extraction. *Journal of Machine Learning Research*, 44:242–259, 2015.

[7] Yu Chen and Mohammed J. Zaki. Kate: K-competitive autoencoder for text. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 85–94, New York, NY, USA, 2017. ACM.

[8] Haiyun Guo, Jinqiao Wang, and Hanqing Lu. Learning deep compact descriptor with bagging auto-encoders for object retrieval. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3175–3179. IEEE, 2015.

[9] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59, 2011.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[13] Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. *ICLR*, 2017.