Autoencoder for Extracting Problem-Specific Features

Hoang Chuong Nguyen

Research School of Computer Science, Australian Nation University
U7076755@anu.edu.au

Abstract. Extracting good features is essential to achieve a good performance on a classification problem. Neural networks in recent years have become the first choice of many people for learning good features. However, neural networks still sometimes fail to extract good features for difficult tasks such as fine-grained classification. An autoencoder is one method in the neural network family that could be used to achieve this goal. Nevertheless, representations learnt by the autoencoder tends to be too general and not specific to a particular task. In this work, we propose a new variation of the standard autoencoder that helps to learn good features for a particular classification problem. Our method implicitly injects supervision signals into a process of training the autoencoder, thus helping the leant representations to be more useful for the classification problem. The result of this new variation yields not only an improvement in statistical metrics, but also a better generalization of our model.

Keywords: Neural network, feature extraction, classification, transfer learning, autoencoder, distinctiveness pruning.

1 Introduction

Input features are one of the most important factors that affect the performance of a machine learning (ML) model, thus learning a good representation of data is crucial for building a good ML model. In the past, hand-engineered features were mostly used. There are plenty of methods to engineer data such as Scale Invariant Feature Transform (SIFT) (Lowe, 2004), histogram of oriented gradient (Dalal and Triggs, 2005) in computer vision; or term frequency vector (Leskovec, Rajaraman and Ullman, 2014), Pairwise Mutual Information (PMI) (Chruch and Hanks, 1990) in natural language processing. These features make senses to humans; however, it has been shown that this is not the best way to learn a good representation as machines do not see the world the same way as humans do. Moreover, this approach usually requires the pre-engineered input features to be understandable by humans so that some techniques and statistics could be done to engineer the data. Unfortunately, this is not always the case as data is not always analysable.

In recent years, neural networks are a good choice to overcome a problem of hand-engineered features as these models have an ability to learn good features by itself. A lot of effort has been put into creating neural network architectures that could help to extract features from a wide range of different structures such as grid data (Krizhevsky, Sutskever and Hinton, 2017), or sequential data (Hochreiter, and Schmidhuber, 1997). Nevertheless, a lesser amount of work has been carried out to extract good features from raw vectorized input data. Indeed, some prior works in the field of neural networks still rely on hand-engineering features for neural networks (Milne, Gedeon, and Skidmore, 1995.). However, the extracted features in some cases are still not a good representation; and hand-engineering features become much more challenging when data is abstract and not understandable to humans. Autoencoder (i.e. autoregressive network) is one approach that makes use of neural networks to extract useful features from raw input data. Nonetheless, this method still has weaknesses as the extracted features are too general and not problem-specific.

To overcome the disadvantages of hand-engineering features and standard neural networks, we propose a new variation of the autoencoder's architecture in conjunction with a new training procedure. In contrast with the standard autoencoder whose purpose is just to minimize a reconstruction loss, our approach could help to inject supervision signals which help to extract useful features for a specific classification task. The effectiveness of our method is evaluated on a fine-grained classification task using the Vehicle-x dataset (Yao, Zheng, Yang, Naphade, and Gedeon, 2019). This dataset is suitable for evaluating our method as it has characteristics of the aforementioned problems, therefore we could easily observe the superiority of features produced by our proposed method compared to features extracted by standard neural networks and hand-engineering features.

Our investigation in this paper is conducted following the steps below:

- Propose a new variation of autoencoder in conjunction with a new training procedure.
- Evaluate the effectiveness of the proposed method and compare the result of the proposed method with a fully connected neural network on a fine-grained classification task using the Vehicle-x dataset.

- Perform distinctiveness pruning (Gedeon, and Harris, 1992) to illustrate the robustness of the new method in terms of feature extractions and data compression.

This paper is organized as follows. First, **Section 2** gives an overview on the Vehicle-X dataset and how decisions regarding data preprocessing were made. **Section 3** describes the proposed method in detail, which is followed by an analysis on the results in **Section 4**. Finally, **Section 5** is a wrap-up of our research, together with some future works that could help to enhance our method.

2 Dataset

The dataset used in our research is a synthesised version of a Vehicle-X dataset. Specifically, each sample in the dataset is generated by feeding an image from the original Vehicle-X dataset into a Residual Network (He, Zhang, Ren, and Sun, 2016) which was pretrained on ImageNet. Each sample is a 2,048-dimensional vector. The dataset is split into three smaller sets for training (including 45,438 samples), validating (including 14,936 samples) and testing (including 15,142 samples). A model is trained using the training set, whereas the validation set is used to tune hyper-parameters and the testing set is only used to evaluate the best model. Vehicle id is a target attribute for this classification problem. There are totally 1,362 classes that needed to be classified. As shown in Figure 1, the distributions of number of samples per each class in all three datasets are balanced both within each dataset and between datasets.

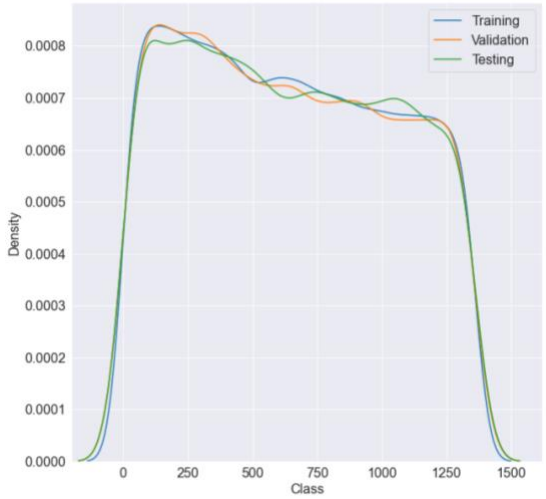

Figure 1. Distributions of number of samples per each class in training, validation and testing set. Number of samples per each class in all three datasets are balanced both within each dataset and between datasets.

As this is a synthesis dataset, it is not easy to understand and gain an insight from the data by purely looking at statistical measurements. *Table 1* shows the average statistical measurements across all features, it can be seen that the data are equally distributed between 0 and 1 with the mean value of 0.5. One thing that is worth noting from this is that all values in the dataset are non-negative, which could cause some problems while training a neural network. Specifically, all positive data makes gradients produced by a non-zero center activation functions such as ReLU or sigmoid to be all positive or all negative. This effect makes the training process becomes much slower as an optimizer tends to move in only one direction at a time (Nwankpa, Ijomah, Gachagan, and Marshall, 2018). There are two ways to overcome this problem: (1) use activation functions that are not non-zero centers such as tanh or leaky ReLU; (2) normalize the data so that it centers around zero and have a standard deviation of 1. Decisions for which approach to be used in this work are determined on a case-by-case basis. More clarification about this choice will be made clearer in **Section 4**. Additionally, the raw features are used in our proposed method without any further pre-processing. This could help to increase the difficulty of the task of extracting features, helping us to have a better evaluation of how useful our method is. Furthermore, since the raw features are extracted from a neural network, we could expect them to be suitable for a new neural network to learn on without further pre-processing (Jason, Jeff, Bengio, Hod, 2014).

Max	Min	Mean	Variance
1.00	0.002	0.46	0.02

Table 1. Average statistical measurement across all features in the dataset. Max and min value in the table shows that all data are non-negative.

The chosen dataset is suitable for our work because it could clearly indicate the weaknesses of hand-engineered features and features extracted by a normal fully-connected network. In particular, the dataset is not easily analysable or understandable to be hand-engineered. Furthermore, fine-grain classification is a difficult task that could expose weaknesses of a normal neural network in terms of feature extractions. Therefore, evaluating models on this dataset could help to clearly illustrate the outperformance of our method.

3 Proposed Method

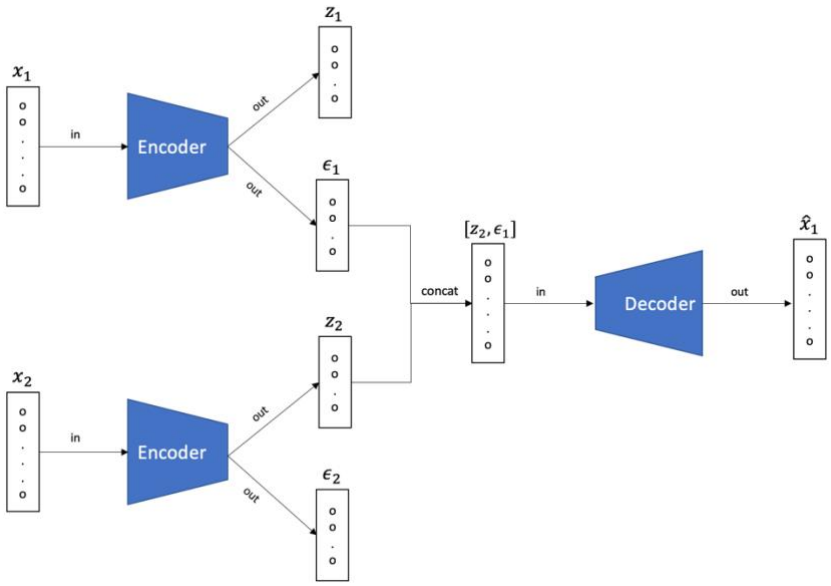

Figure 2. The extractor's architecture. The encoder is first used to extract latent variable z_1, z_2 and noise ϵ_1, ϵ_2. Then the decoder uses the concatenation of z_2 and ϵ_1 to reconstruct x_1. Note that there is only one encoder. The two encoders shown in the figure are actually the same one.

3.1 Model Architecture

The new variation of the autoencoder presented in this paper is inspired from a work done by Denton and Fergus (2018). This work proposed a Stochastic Video Generation (SVG) model which disentangles encoded features by exchanging the common features between two images and forces an autoencoder to reconstruct the original image by using the exchanged features. Specifically, the main idea is that as two static images are in a same sequence (i.e. video), they share same background features and difference between them only comes from different motion features in the two images. Therefore, by forcing the autoencoder to reconstruct one image using the original motion feature and the exchanged background features, the model could eventually learn to not only extract background and motion features, but also to disentangle them. This method has been proven to be useful as it could be applied to different problems, or even different training procedures (Xu, Xu, Ni, Yang and Darrell 2020).

Likewise, our method is modelled as follows. In a classification problem, samples that are in the same class share some common latent variables and have some different variables. Thus, by paring these samples together, then exchanging the latent variables and finally forcing the autoencoder to reconstruct one sample using the exchanged variables, the autoencoder would ultimately be able to extract useful features for classifying. In other words, the extracted features are encoded with information that is only shared between samples that are in the same class. This procedure is described pictorially in *Figure 2* and it can be formularized as below.

$$z_1, \epsilon_1 = Encoder(x_1)$$

$$z_2,\ \epsilon_2 = Encoder(x_2)$$

$$\hat{x}_1 = Decoder([z_2, \epsilon_1])$$

$$Loss(x_1, \hat{x}_1) = \left|\left|x_1 - \hat{x}_1\right|\right|_2^2$$

Where $Encoder$ and $Decoder$ are two separated neural networks; x_1 and x_2 are two input samples having the same class label; z_1 and z_2 are two latent variables extracted by the encoder; ϵ_1 and ϵ_2 are some noise (i.e. different features) extracted by the encoder; $\hat{x}_1$ is the reconstruction of x_1 by feeding a concatenation of the latent variable z_2 and noise ϵ_1 to the decoder. The reconstruction loss is l_2 loss which is the same as that one of the standard autoencoder. As observed from the equations, using z_2 and ϵ_1 to reconstruct x_1 implicitly implies that x_1 and x_2 share some common latent variable z. Hence, it could be said that our method helps to indirectly inject supervision signals into the process of extracting features, which is then useful for a classification problem. One thing that needs to be noticed is that our method requires there must be at least two samples per each class in a whole dataset.

3.2 Training with Triplet Loss

In this work, we also used triplet loss in conjunction with our proposed model in order to understand whether they could be used together or not. The main idea is very similar to the previous section. Given the two samples that are in the same class, one should be able to construct these samples by exchanging their latent variables. Additionally, given two samples that belong to different classes, one sample must not be reconstructed by using the latent variable of the other sample, making the reconstruction error considerably high. This idea can be formularized as below:

$$z_a, \epsilon_a = Encoder(x_a)$$

$$z_p,\ \epsilon_p = Encoder(x_p)$$

$$z_n,\ \epsilon_n = Encoder(x_n)$$

$$\hat{x}_{ap} = Decoder([z_p, \epsilon_a])$$

$$\hat{x}_{an} = Decoder([z_n, \epsilon_a])$$

$$Loss(x_a, \hat{x}_{ap}, \hat{x}_{an}) = \left|\left|\hat{x}_{ap} - \hat{x}_a\right|\right|_2^2 + \max\left(\left|\left|\hat{x}_{ap} - \hat{x}_a\right|\right|_2^2 - \left|\left|\hat{x}_{an} - \hat{x}_a\right|\right|_2^2, \alpha\right)$$

With x_a and x_p are two samples belonging to the same class, whereas x_n is a sample that has a different class from that one of x_a and x_p. Correspondingly, z and ϵ denote the sample's latent variables and noises. $\hat{x}_{ap}$ and $\hat{x}_{an}$ are the reconstruction of x_a using the pair (z_p, ϵ_a) and the pair (z_n, ϵ_a) respectively. The loss function includes two terms: (1) the positive reconstruction error using $\hat{x}_{ap}$; (2) the difference between the positive reconstruction error using $\hat{x}_{ap}$ and the negative reconstruction error using $\hat{x}_{an}$. The first term is similar to the reconstruction loss in the **Section 3.2** since our goal is still to extract latent variables. For the second term, our intuition is that the positive reconstruction error should be much higher than the negative reconstruction error, thus the difference between them should be large. The hyper-parameter α acts as a margin between the two reconstruction errors. In other words, it controls a trade-off between how much we want to minimize positive reconstruction loss and how different between the positive reconstruction error and the negative reconstruction error that we want.

3.3 Training Procedure

The training procedure of our method is divided into two phases. We need two different networks for two different purposes: an autoencoder for extracting useful features, and a fully connected neural network for classifying classes. For convenience, the autoencoder is called an extractor and a fully connected is called a classifier from now on.

The first step in the training procedure is to construct a training data for the extractor. This can be easily done by finding all pairs of samples in the dataset that have the same label. Therefore, given a dataset having K classes and m_k samples per each class k, the number of total samples n is calculate as below.

$$n = \sum_{k=1}^{K} \frac{m_k!}{(m_k - 2)!}$$

Similarly, in case the triplet loss is used, the first step is to find all pairs of samples having the same class labels. Then for each pair, we could assign to it a random sample that has a different label. It can be seen that the number of training samples increases significantly which helps to add even more supervision signals into the training process of the extractor. It is worth noting that the data pair (x_1, x_2) are different from the data pair (x_2, x_1) as the reconstructed outputs produced by the decoder are different. The decoder could be made to output $\hat{x}_1$ and $\hat{x}_2$ as once when it receives the data pair (x_1, x_2). But by differentiating the two pairs, there would be a stronger stochastic effect added to the training process, which could help to improve the convergence of the model in general (Gedeon, 1998).

After finish training the extractor, the extractor will be then used to extract features from the raw input data in order to train the classifier. During this phase, the weight of the extractor is completely frozen since it is not necessary to train the extractor anymore. On the other hand, the classier is trained using the extracted feature. The process of training the classifier is formularized as below.

$$z_n, \epsilon_n = Encoder(x_n)$$
$$\hat{y} = Classifier(z_n)$$
$$Loss = CrossEntropy(y, \hat{y})$$

With z_n, ϵ_n are latent and noise features of input vector x_n respectively; $\hat{y}$ is the output of the classifier, which is then used together with a ground truth label y to calculate cross-entropy loss.

4 Experiments and Results

4.1 Baseline Model

A simple feed-forward neural network with six hidden layers is chosen as a baseline model to compare with our models on a fine-grained classification task. We consider this a strong competitor as the neural network could also learn to extract features that is useful for a specific task. Unlike the classifier in our proposed method, this baseline model receives features that are manually pre-processed from the raw input data. By doing this, we can have a good insight of how good the extracted features are compared to a hand-engineered features that are fed to a simple neural network.

To avoid the problem of all positive input features described in **Section 2**, features are normalized to have zero mean and unit standard deviation. Other methods had also been tried but the outcomes were not good. Additionally, in order to have a fair comparison, hyper-parameter tuning was carried out carefully for the baseline model to make its performance as good as possible. Below are details of the baseline's model setting:

- Input features is normalized to have zero mean and unit standard deviation.
- Six hidden layers with 2048, 2048, 1024, 512, 256, 128 hidden units respectively. The output unit has 1362 units with a SoftMax activation function at the end. The dimension of the input layer is 2048.
- Leaky ReLU activation function with a negative slope of 0.5 is used in each hidden layer.
- As observed during the hyper-parameter tuning process, the model overfitted fairly quickly, therefore, three dropout layers with dropping rate of 0.45 are used for the last three hidden units.
- Adam optimizer with an initial learning rate of 0.002 is used, which then decreases to 0.001 after the tenth epochs to make sure the model converges in a good local minimum.
- Batch size used for all experiments is 256.

4.2 Extractor and Classifier

The model described in this section corresponds to the method described in **Section 3.1** (i.e. triplet loss is not used for this model). For a purpose of a fair comparison, the whole architecture of the proposed model is set to be the same as the architecture of the baseline model. Below is the detailed setting of the extractor:

- The extractor contains two components which are the encoder and the decoder. Each component has two hidden layers with 2048 hidden neurons.
- The input of the encoder is raw input data whose dimension is 2048. The encoder also has two separated output layers which output latent variable z and noise ϵ respectively. The dimensions of z and ϵ are 1024.
- As illustrated in the equation in **Section 3.1**, the decoder receives latent variable z_2 of one sample and noise ϵ_1 of another sample and then concatenate them. The concatenated feature has a dimension of 2048, so that all input layer, hidden layers and output layer of the decoder have a dimension of 2048.
- Adam optimizers with an initial learning of 0.0001 is used to train the extractor. The learning rate is reduced by half at epoch 10^{th}, 20^{th}, 50^{th}, 100^{th}.
- Both the encoder and the decoder use tanh activation functions in the hidden layers. For the output layer of the encoder, it also uses tanh activation, whereas that one of the decoder is a simple linear layer.
- The extractor is trained using l_2 reconstruction loss.

As for the main purpose of the exactor is to extract useful features, there is no need to for dimensionality reduction while encoding or decoding the features. This is the reason why the extractor setting is designed as above. It is also worth

mentioning that we let the extractor receives raw input features instead of the preprocessed ones. This slightly increases of the difficulty of the task, helping us to get more idea of how effective our method is in terms of feature extraction.

Regarding to the classifier, its setting is:

- The extractor is first used to transform raw input features into features of 1024 dimensions. Note that only the encoder of the extractor is used in this phase, whereas the decoder is discarded. Thus, the input feature of the classifier has 1024 dimensions.
- The classifier has three hidden layers which are exactly the same as the last three hidden layers of the baseline model. Specifically, they have 512, 256 and 128 hidden units respectively. Leaky ReLU activation function with a negative sloop of 0.5 and dropout with dropping rate of 0.45 are also used for these layers. Additionally, the output layers and the loss function are also similar to that of the baseline model.
- Lastly, Adam optimizer is also used for the classifier with the same method of decaying the learning rate as described in the setting for the baseline model.

By designing our experiment like this, at the stage of training the two predictors, both our network and the baseline model have the same architecture. The only difference is that the first three hidden layers of our network is the encoder whose parameters are frozen, whereas parameters of the first three hidden layers of the baseline model are still be updated.

4.3 Extractor and Classifier with Triplet loss

The model in this section corresponds to the triplet loss described in **Section 3.2**. The architecture of this model is exactly similar to the one described in section **Section 4.2**. The only difference is that the extractor is now trained using triplet loss instead of a normal l_2 reconstruction loss. Additionally, the hyper-parameter α used for the triplet is 0.0001.

4.4 Evaluation and Comparison

In this section, we evaluate the performance of the baseline model and our models on the fine-grained classification task using the Vehicle-x dataset. Comparison and analysis are then showed to illustrate the strength of our models compared to the baseline model.

All three models including the baseline model, our model with l_2 reconstruction and our model with triplet loss are trained for 50 epochs and then evaluated on the testing set. The first three rows in *Table 2* shows the cross-loss entropy loss, top-1 accuracy, top-10 accuracy and balanced accuracy of these models. It is interesting that even the loss of our models are larger than that of the baseline model, their accuracies in classifying data are considerably better than the baseline model's. This is because the only supervision signals in the training process of the baseline model comes from the cross-entropy loss function, and as the model has more learnable parameters, it does a good job in reducing the loss function. Nevertheless, due to the low quality of the input features, it fails to make good predictions. On the other hand, although the classifiers of our models have less learnable parameters, it receives useful input features which are extracted by the extractor, thus making them have higher loss but higher accuracy. Moreover, cross-entropy loss function is not the only source of supervision signals for our models as the process of training the extractors also indirectly produces useful signals needed for classifying classes. This shows the usefulness of our proposed method in terms of feature extractions. It is worth mentioning that as the predictions of all three models include a variety of different classes, plus that their balanced accuracy is similar to the top-1 accuracy, therefore the models do not simply predict the major class. Hence, the result is a valid and can be used for comparing models.

It's unexpected that the performance of the model using the triplet loss is slightly worse than that of the model using normal l_2 reconstruction loss. The reason could be because fine-grained classification is a complicated task such that using only the triplet loss in most cases would not help to improve the performance. Specifically, the triplet loss sometimes causes inter-class distance to be smaller than intra-class distance since this loss does not have globally optimal constraint (Luo et al., 2019). In order to make the best use of triplet loss, one approach is to use it in conjunction with ID loss. By doing this, the model could be able to learn more useful discriminative features. This leaves us rooms for future improvements of our method.

	cross-entropy loss	**top-1 accuracy**	**top-10 accuracy**	**balanced accuracy**
Baseline model	4.15	15.83	47.52	15.11
Our model	4.40	20.35	52.80	19.80
Our model (triplet loss)	4.51	19.84	49.12	19.55
Our pruned model	3.87	21.26	55.00	21.13

Table 2: The performance of different models evaluated using cross-entropy loss, top-1 accuracy, top-10 accuracy and balanced accuracy. All of our models outperformance the baseline model in all metrics. Moreover, pruning our model using the distinctiveness pruning method also helps to improve its performance.

Additionally, our model is also better in term of generalization. To illustrate this, we retrained the baseline model and our model (which uses l_2 reconstruction loss) for 200 epochs and visualized the learning curve of the two models. Figure 3 includes two plots showing two curves in terms of loss (the left plot) and top-1 accuracy (the right plot). Despite of the strong regularization effect used for both models, the baseline still overfits fairly easily. It can be observed that the baseline models start to overfit at epoch 50th as the validation loss increase considerably. In contrast to this, the validation loss of our model remains stable after epoch 50th. Moreover, the right plot shows that the accuracy of the baseline model reduces gradually after overfitting, whereas there is still a small improvement in the accuracy of our model despite of its loss remaining stable after epoch 50th.

The performance of our model is still bad due to the fact that the fine-grained classification is a difficult task, so that using a simple neural network as a classifier is not enough to achieve a good performance. Especially, this synthetic version of the Vehicle-X dataset is not meaningful for classifying classes. In particular, as the model used to extract the vectorized data is deep residual network pretrained on ImageNet, the final output layer is only able to extract useful features from images in the ImageNet dataset, but not other datasets. In short, since this is a deep network, the extracted features are not general (Jason, Jeff, Bengio, Hod, 2014). This is the reason why it cannot extract useful features from images in Vehicle-X dataset. Nonetheless, improving quantitative metrics is not main point of our work. Instead, we have shown the effectiveness of the proposed method in extracting features from raw input data. Our method outperforms the simple neural network in many aspects.

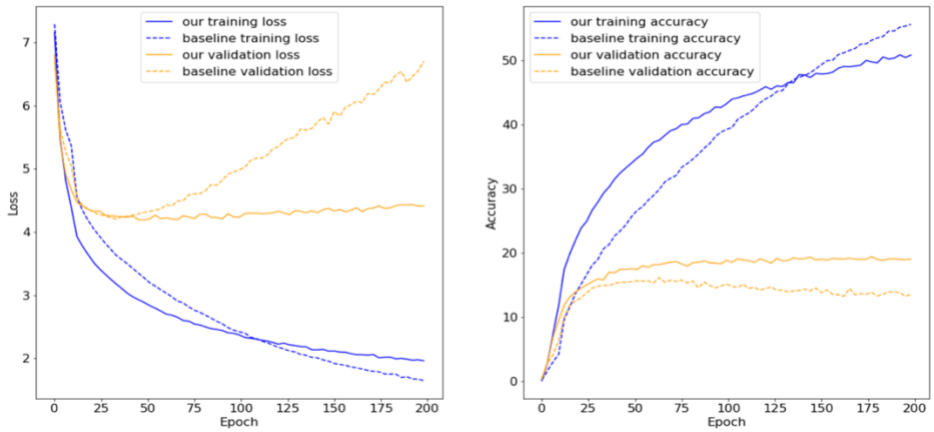

Figure 3: Learning curve of our model (with the extractor using only l_2 reconstruction loss) and the baseline model in terms of cross-entropy loss (left plot) and top-1 accuracy (right plot). It can be seen that the generalization of our model is better than the baseline model's.

4.5 Pruning the Extractor

The purpose of this section is to proves the robustness of our method in encoding and compressing the latent variables z. Specifically, we make use of distinctiveness pruning technique to reduce the dimension of the latent variables gradually and observe how dimensionality reduction impacts the quality of the extracted features. Regarding distinctiveness pruning, it is a method to prune a neural network based on a similarity measured by an angle between vectors of two hidden units. A vector of a hidden unit is determined by stacking all of its output for every sample in the dataset into one single vector. An angle between two hidden units is then calculated and one unit will be pruned if the angle between it and another unit is less than 15 degrees. Similarly, both hidden units will be pruned if their angle is larger than 165 degrees. We followed exactly the same method presented by Gedeon, and Harris (1992) with only one modification regarding to the activation function. Since sigmoid activation function is used in the original paper of this method, normalizing all vectors to 0.5, 0.5 is necessary as all the original output ranges between 0 and 1. As the extractor uses tanh activation function, normalizing is not necessary in our experiment.

Recall that the job of the extractor is not only to extract latent variables z, but also to leave out the noise ϵ. The more amount of noise the extractor can extracts, the better the quality of the latent variable is. Hence, to have a good evaluation on how dimensionality reduction impacts the quality of the latent variables, we decided not to limit the ability of extracting noise ϵ of the extractor. Therefore, the setting for this experiment is that we only prune the hidden layers of the encoder which is responsible for outputting the latent variables z; whereas other components of the extractor are kept as the same design described in **Section 4.2**. The starting dimension of the latent variables is 1024 and it is pruned until there is no similar or complementary neurons. At each epoch, at most one similar neuron is pruned, or at most two complementary neurons are pruned. After finishing pruning, one more neuron is added to make sure the network has a full capacity.

Lastly, the extractor is trained for another 20 epochs. Note that the loss used for the extractor in this section is just a normal l_2 reconstruction loss since it has the best performance as shown in the previous sections. Apart from the pruned extractor, we also trained another three extractors with dimensions of the latent variables of 128, 256 and 768 and compare them with the pruned network. As for the classifier, its design is kept as the same as described in **Section 4.2** with the dropout rate adjusted to 0.25, 0.3, 0.4 for the extractors with 128, 256 and 768 latent dimensions respectively.

Figure 4 shows the resultant top-1 accuracy corresponding to each extractor in this experiment. Note that only the extractor with 490-dimensional latent variable is pruned, whereas the rest are left unpruned. The pattern of the solid line in the figure matches our expectation since it is obvious that the reduction in the dimension of the latent variable leads to the worst performance of the model. Interestingly, pruning the network does not result in information loss as in image compression, instead, it slightly improves performance of the model. The final pruned network has a latent variable of 490 dimensions, but its performance is better than the best unpruned network which has 1024-dimensional latent variable. A possible justification for this phenomenon is that pruning could help our method to gradually compress useful information into the latent variable every time one neuron is discarded. Moreover, unlike image compression whose purpose is mainly to compress data, the task of our extractor is relatively easier as it only needs to construct useful features. Ideally, the extractor only needs to extract K latent variables for K different classes, whereas in data compression, the autoencoder needs to learn how to reconstruct N different data points appropriately.

Not only that, but the pruned extractor also results in a much lower loss compared to all other models as shown in the fourth row in *Table 2*. Improvements in terms of generalization and stability while training the classifier could be observed from the learning curves in *Figure 5*. From the left plot, it could be seen that the learning curve of the pruned network has less oscillation effect then the unpruned ones. For both accuracy and cross-entropy loss, the training and validation curve of the pruned network are closer to each other compared to the unpruned network. Our interesting finding in this section is that distinctiveness pruning technique could help to regularize that our model, thereby, improves its performance. Thus, distinctiveness pruning technique and our method could be used in conjunction with each other.

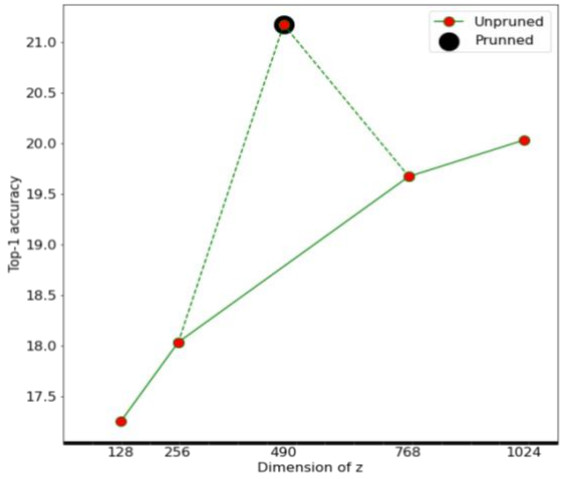

***Figure 4**:* Top-1 accuracy of the models with the different dimensions of the latent variables. Only the model corresponds to the black dot is pruned, whereas the rest are left unpruned. The pruned network has a best performance compared to the others.

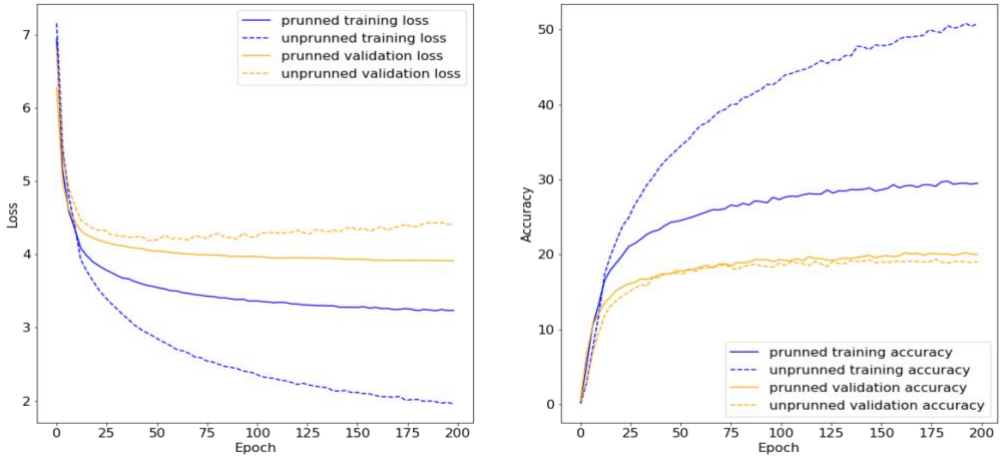

Figure 5: Learning curve of the pruned and unpruned model. The pruned outperform the other in term of generalization and training stability

4.6 Ablation Study

To confirm the key insight of our work, we reduce the dimension of the latent variables into 2 and visualize them in a two-dimensional plane. Only the first ten classes are visualized as the plot would look cluttered if all 1362 classes were plotted. Figure 6 shows the resultant datapoints of at the 1st, 10th, 20th, 30th epoch. It could be seen that there are huge overlaps between classes at the beginning. After that, datapoints having the same class labels tends to move closer to others gradually. At epoch 30th, there are some classes that are completely separable from the others. As this visualization is in two-dimensional plane, there are still large overlaps between classes, but the situation could be improved as dimension of the latent variables increases. Nevertheless, the overlaps also indicate one weakness of our method. In particular, the $l2$ reconstruction loss only helps to move datapoints having the same classes closer to each other; but fails to separate them from other classes. We expected the triplet loss could help to cope with this problem, but it did not work as expected as shown in Table 2. This leaves us rooms for future improvements of our method.

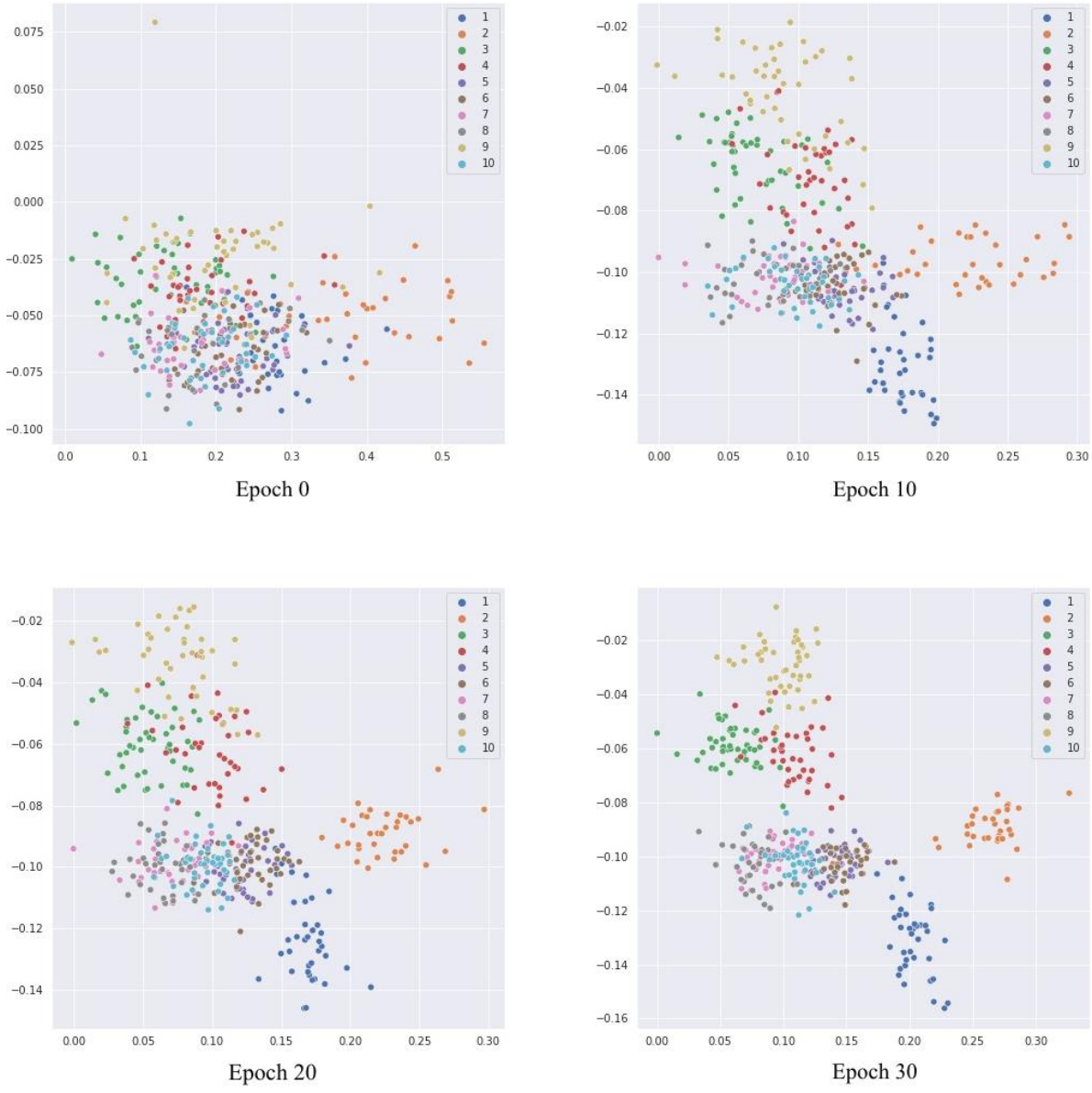

Figure 6: Figure shows the latent variables of the first ten classes in two-dimensional plane during the training process. The datapoints having the same class labels tends to move closer to others gradually as the extractor is being trained.

5 Conclusion and Future Work

In this work, we propose a new method to extract useful features for a classification problem from raw input data. Although the accuracy of our model still needs to be improved, we have proven that our model is superior to a simple neural network. Our proposed method is useful specifically in case the dataset is not analytical or there is lack of information for human to engineer features manually. The triplet loss is found to not be suitable to our method as it has

some limitations such as not having globally optimal constraint. Moreover, we also show that the proposed method could be used together with distinctiveness pruning to increase its effectiveness.

There are plenty of rooms for improvement of our method. Firstly, the noise ϵ variables could be decomposed further into different latent variables using another set of autoencoder. The idea is that we could recursively extract many latent variables by stacking many autoencoders together. This could be useful to extract features for a multi-label classification task. Secondly, the current reconstruction loss used for the extractor implies that two samples having the same class should have the same latent variable, but it still cannot help the extractor to differentiate between the latent variable of two different class. Triplet loss was expected to help our method to overcome this problem, however, the result is not as expected. Using a triplet loss together with ID loss has been proven to be useful to cope with this issue. This method is potential solution to help us enhance our method.

References

Church, K. and Hanks, P., 1990. Word association norms, mutual information, and lexicography. *Computational linguistics*, *16*(1), pp.22-29.

Dalal, N. and Triggs, B., n.d. Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*.

Denton, E. and Fergus, R., 2018, July. Stochastic video generation with a learned prior. In *International Conference on Machine Learning* (pp. 1174-1183). PMLR.

Gedeon, T.D. and Harris, D., 1992, June. Progressive image compression. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks* (Vol. 4, pp. 403-407). IEEE.

Gedeon, T.D., 1998, October. Stochastic bidirectional training. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)* (Vol. 2, pp. 1968-1971). IEEE.
He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

Krizhevsky, A., Sutskever, I. and Hinton, G., 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), pp.84-90.

Leskovec, J., Rajaraman, A. and Ullman, J., 2014. Data Mining. *Mining of Massive Datasets*, pp.1-18.

Lowe, D.G., 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), pp.91-110.

Luo, H., Gu, Y., Liao, X., Lai, S. and Jiang, W., 2019. Bag of Tricks and a Strong Baseline for Deep Person Re-Identification. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

Milne, L.K., Gedeon, T.D. and Skidmore, A.K., 1995. Classifying Dry Sclerophyll Forest from Augmented Satellite Data: Comparing Neural Network, Decision Tree & Maximum Likelihood. *training*, *109*(81), p.0.

Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S., 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Xu, J., Xu, H., Ni, B., Yang, X. and Darrell, T., 2020, November. Video Prediction via Example Guidance. In *International Conference on Machine Learning* (pp. 10628-10637). PMLR.

Yao, Y., Zheng, L., Yang, X., Naphade, M. and Gedeon, T., 2019. Simulating content consistent vehicle datasets with attribute descent. *arXiv preprint arXiv:1912.08855*.

Yosinski, J., Clune, J., Bengio, Y. and Lipson, H., 2014. How transferable are features in deep neural networks?. *arXiv preprint arXiv:1411.1792*.