CPS 803 Final Project Report

Ian MacPherson
dept. of Computer Science
Ryerson University
Toronto, Canada
ian.macpherson@ryerson.ca

***Abstract*—This project report investigates auto-encoders and their relevance in image clustering for unsupervised image classification. The most promising results come from the experiments training a linear classifier, where it is trained on top of a randomly initialized auto-encoder, a pre-trained auto-encoder, and trained end-to-end (auto-encoder & classifier). The end-to-end method achieves the highest accuracy with fully supervised image classification as hypothesized. Insights of this investigation are extrapolated from the comparison between the results of the experiments, and conclude that auto-encoder methods are a viable architecture to increase training accuracy for unsupervised image classification. This project provides plenty more questions worth investigating, and are briefly discussed at the end of this report.**

I. Introduction

The deep learning revolution was made possible and accelerated in large part due to the vast amounts of data that became readily available through user generated content uploaded to the internet. The performance of neural networks have been closely tied to the quality and quantity of the datasets available and utilized during training time. However, it is not always the case that datasets for specific tasks are readily available, or that pre-trained networks will generalize well to niche tasks.

There is a growing area of research that aims to solve this problem. The current methods are to train neural networks to achieve a higher classification accuracy with a smaller training datasets. This area of research is known as *Few Shot Learning*, and has been investigated further over the course of this project.

Auto-encoders are a type of neural network which take as input image pixels, scale the image down to its features, and reconstruct the image by transposing the operations to encode the image. The encoded feature maps provide useful data for the image, representing features such as lines of the image, colours, and other patterns. The feature map information may lead to important insights about which images belong to the same image class. The concept of taking image features to identify class may be accurate enough to show few images to a neural network and still attain a high classification accuracy.

One can relate this to how humans learn to identify a certain objects when only shown one-example of the class. For instance, in the common memory game where players identify pairs of the same class, players could flip over two different images of cats and easily classify both images as belonging to class ”cat”. Swap this with patterns, and players of average cognitive ability can identify a pattern when only being exposed to it once. If this same sort of learning can be developed in a neural network, then image classification will be able to achieve human accuracy when exposed on small batches of training data.

II. Problem Statement and Dataset

The main question this project asks is if there are methods or architectures that could achieve a high accuracy with minimal training. The intention behind this is to discover whether the image classes are separable based on their features alone. If the image features are separable, images from the same class would cluster together and therefore be classifiable through unsupervised learning. The dataset used is the CIFAR-10 [1] dataset, which consists of 60,000 32x32 colour images across 10 image classes. Multiple experiments are run to compare and contrast classification performance on the same architecture trained by different methods.

An auto-encoder is used to try and achieve this separability, as auto-encoders are used to compress images to a feature-representation, and reconstruct the images from the feature representation by reversing the encoding steps. The feature representations or ”feature maps” can then be used to determine how image classes cluster together. This hypothesizes that feature representations of the same class will be mathematically closer together than feature representations of another class.

The feature representations are used rather than simply comparing images on a per pixel basis. Comparing images pixel-by-pixel is not as robust to variances in the images such as scale, rotation, or mirroring. By using the learned features of the image, classification accuracy is expected to be increased as features in class horse will be the same whether the horse is on the left side of the image vs. the right side. When comparing feature maps of the two aforementioned images, they should be relatively similar however this will not reliably be the case with pixel by pixel comparison.

III. Methods & Models

A. Auto-encoder

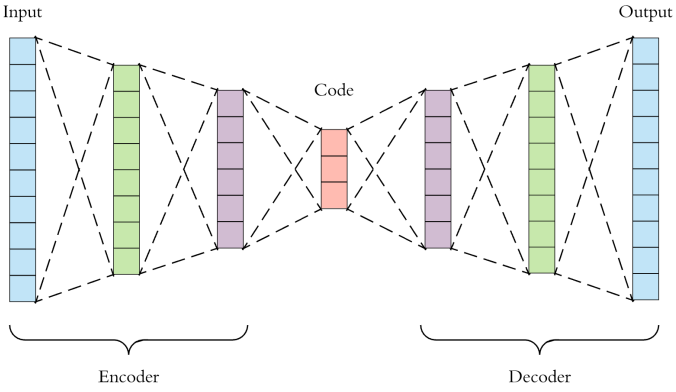

Fig. 1. An example architecture of an auto-encoder network taken from the article *Applied Deep Learning - Part 3: Autoencoders*. Note how the input image is converted into a 'code' representation and passed to the decoder network. This encoding will be important throughout the rest of the report as this feature representation is what is used to perform principal component analysis, and train various models of a classification network, [2]

An auto-encoder is first trained on the CIFAR-10 dataset for 100 epochs with binary cross entropy (BCE), L1, and MSE losses for comparison. The code [3] for the auto-encoder was found online, and was chosen due to its simplicity. The auto-encoder consists of 3 convolutional layers for the encoder network, and 3 transposed convolutional layers for the decoder network. All intermediate activation functions are Relu, and the final activation from the decoder is a sigmoid function.

When running experiments with the these loss functions, there did not seem to be a large enough perceptual difference to make the assertion that one loss function is more optimal for the auto-encoder task than the others. The BCE loss was only feasible for this application as the input images are scaled within the range [0,1] on input due to torch.toTensor(), and on output as per the sigmoid activation. The following experiments and tests that include a pre-trained auto-encoder were run with weights trained on the BCE loss unless otherwise stated.

Fig. 2. Diagram of results of the Auto-encoder trained on different loss functions for 100 epochs each. This diagram is meant to be zoomed in on with an e-reader or in browser. This diagram illustrates that there is a slight or negligible difference between networks trained on the respective loss functions.

B. Principal Component Analysis

Once the auto-encoder was producing perceptually acceptable results, the next step is to find if the features are separable. A technique that was found was to perform principal component analysis (PCA) on the feature maps for each image. The intended outcome of this experiment is to visualize and represent the data in a way that could be separable through clusters of image features projected in two-dimensions. Expected results are nicely defined regions between each class, or 10 data clusters on the plot as per the amount of CIFAR-10 classes. For this, the feature-maps of the encoder network were vectorized and run through the sklearn.decomposition.PCA() function.

It seems that the features when trained on an auto-encoding task are not as separable as PCA or t-SNE [4] diagrams for feature maps trained for supervised classification tasks. T-SNE was tested on the same feature maps and did not provide consistent results due to the hyper-parameter tuning required. For simplicity and consistency of results, PCA was chosen for projecting the high dimensional feature vectors to two-dimensional space. The plot below shows the PCA of feature maps of 10 000 CIFAR-10 Images, each point colour coded according to its class.

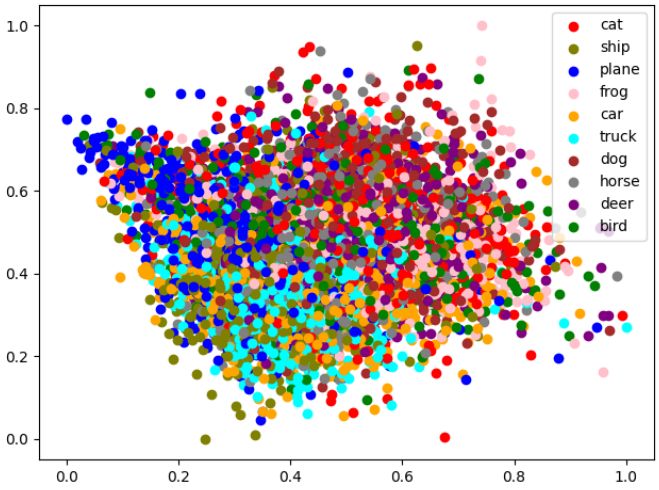

Fig. 3. Feature space reduction to two dimensional plot points for 10 000 CIFAR-10 images. The plot points were generated using the feature maps from the encoder network of a simple 3-layer neural network.

While Fig. 2 does not provide any clearly separable features, it is still beneficial to note how the points are clustering. Plane and truck seem to dominate the left-half of the cluster, and more animal classes cluster toward the right-half. It makes sense a priori that plane and truck would be somewhat similar, as they are both machines made of metal and would share similar features in this respect. The car class seems to be evenly distributed across the cluster, which may suggest that there are many images of cars taken in a nature setting. This anecdote may be why the car is clustered between images of animals as well as machines such as planes and trucks.

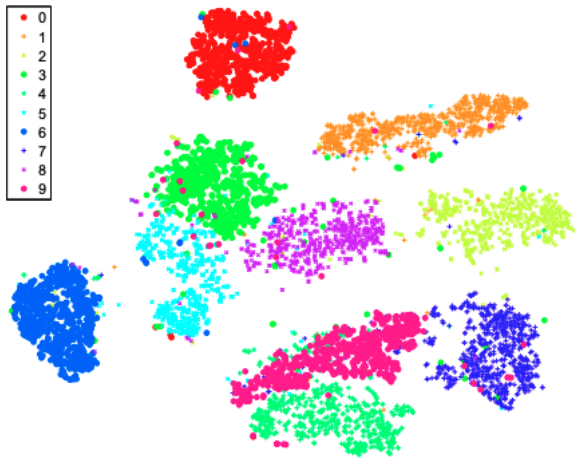

Fig. 4. Diagram from T-SNE visualization of 60 000 MNIST hand-written digits [4]. *Note:* this visualization is simply to demonstrate separability across a 10-class dataset from a neural network trained for classification. PCA and the CIFAR-10 dataset are used in this project, not t-SNE and MNIST.

Unfortunately, this is nowhere near as separable as intended, when a diagram much like the MNIST and t-SNE example was expected from PCA.

A hypothesis for this outcome is that when a model is trained for classification tasks, the model parameters must be much more specific for classification, and produce more separable features. The classification feature-maps generated before classification would provide the nicely separable clusters when projecting the to two-dimensions definable for each image class.

Since this network was trained as an encoder portion of an auto-encoder network, this may show that features for auto-encoders do not need to be as pronounced for subjective perceptually pleasing results. Projection of features from classification tasks may be more separable as the training will modify the weights to be more generalizable across a class. The auto-encoder seems to not be as separable due to the lack of emphasis on the class of image, but rather the pixel-wise difference in input and output images with respect to the BCE loss.

An example of expected separability is shown in Fig. 3.

C. Classifier

Since principle component analysis does not seem to be the best way to separate the feature maps, it was instead decided to train a classifier directly on the feature maps of the encoder portion of the auto-encoder network. A one-layer linear classifier was used to test what effect the auto-encoder network may have on classification accuracy. The first experiment is to train a linear classifier on a randomly initialized auto-encoder, and freeze the auto-encoder weights during training. This first experiment may act as a sort of baseline in comparison to the experiments that follow it. The experiments that follow train a classifier on a pre-trained auto-encoder, and a classifier and auto-encoder end to end. Each of the experiments are run with the cross entropy (CE) loss to train the classifier using the Adam optimizer. For the pre-trained auto-encoder, the BCE weights are used, and each classifier is trained over 100 epochs.

The one-layer linear classifier was chosen due to its simplicity, and is assumed to have a smaller impact on classification accuracy. Since the main goal of these experiments is to see if the data is separable, it follows that a weaker classifier would provide a good indication of separability. In contrast to PCA, the hypothesis is that the separability is not easily represented in two-dimensions. It may be that even when the data seems noisy from PCA, that a simple classifier is able to find patterns in the data.

1) Classifier trained on a randomly initialized auto-encoder: The intention behind this experiment is to provide a baseline to see what level of accuracy the linear classifier can achieve on random feature maps. The weights of the auto-encoder were randomly initialized and frozen throughout training to achieve such results. This baseline is important, as it provides a greater understanding of the power of the linear classifier vs. the feature representations, and how large of a role the auto-encoder plays in classification accuracy in the subsequent experiments.

A random classification on CIFAR-10 would result in an overall accuracy of 10% across all classes. Examples of this sort of case would include if the classifier always predicts the same class, or predicts all classes evenly. If the linear classifier can achieve a classification average greater than random, the logic follows that the linear classifier is able to find some sort of separability from even the random outputs of the encoder. If not, the question arises as to whether the auto-encoder can represent features in a separable way without training, or if a one-layer classifier is powerful enough to find separations in the seemingly random data. The answer may also exist somewhere between the two previous assertions, with the auto-encoder and classifier being symbiotic to one-another. If this is the case, the encoder and classifier each contribute significantly to the classification accuracy.

The results of this experiment are outlined in the second column of table 1.

2) Classifier trained on pre-trained auto-encoder: For this experiment, the pre-trained weights from the BCE loss function are loaded into the auto-encoder network. As in the experiment above, the auto-encoder weights are frozen during training as to only optimize the weights of the classifier. With a trained auto-encoder, it is expected that the results will be better than the classification on the randomly initialized auto-encoder weights.

This experiment in comparison to the previous gives a particular insight into what increase in accuracy a trained auto-encoder gives to classification. Or more specifically, what accuracy an auto-encoder trained for an auto-encoder task can provide to image classification. It is key to remember that the auto-encoder gives no importance to the image class, but

simply reproduces the pixels from the feature maps of an image. If the features are much more separable than random, the auto-encoder can provide a useful tool in unsupervised learning tasks for image classification. If the experiments show that images can be separated into classes based on their features alone, more robust models can be developed to perform a range of classification tasks.

Robustness meaning in this instance that a network could be trained on one dataset, fed completely different data than what it was trained on, and have the classifier determine which images belong into which class. Or at the very least, predict if the images belong to the same or different classes. An example could be training a network on a dataset of animals, and later testing its accuracy when classifying cars, or at the very least if cars are different from one-another. Unfortunately this is beyond the scope of this current assignment, but is an interesting topic to investigate and experiment with further.

The results of this experiment are outlined in the third column of table 1.

3) Classifier and auto-encoder trained end-to-end: The end-to-end method trains both the auto-encoder and the linear classifier. This experiment will likely produce the highest classification results of the three, as it is a fully-supervised method of training a classifier. Due to the low depth of both the auto-encoder and classifier networks, the network is not expected to beat or even meet industry standards. However, it seems reasonable to make the hypothesis that this network will have the highest overall accuracy out of all of the experiments.

It must be noted that for this example, only the cross entropy loss is used. The loss is calculated based on the predicted vs. expected labels and back-propagated through both the linear classifier and auto-encoder weights. It was decided to use this loss, as all of the other experiments used solely the cross entropy loss when training the classifier. Other methods could include adding the cross entropy from classification, and BCE from the auto-encoder losses together. Back-propagating separately for each loss may also be a viable option, and leaves a door open for more experiments beyond this project. For the sake of consistency, these options are avoided to limit the number of variables between each experiment.

The results of this experiment are outlined in the fourth column of table 1.

IV. Results

As hypothesized, the initial experiment produced the worst results, the second produced better results, and the final produced the best results in respect to classification accuracy. Interestingly, the difference in accuracy of classification on random encoder weights vs. pre-trained encoder weights was not as large as expected.

Table 1 outlines the results from each experiment, with the last row displaying the average precision across each class for the respective method. Each of the following accuracies are the results of each model tested over a test batch of 500 images.

It is particularly interesting to note that the random and pre-trained accuracies have consistent 0% scores for some of the same classes across experiments. This may be due to certain classes overlapping more heavily with other classes, and the other classes being exclusively predicted.

TABLE I
Classification accuracy per class across experiments.

Classes	**Random %**	**Pre-trained %**	**End-to-end %**
Plane	50	50	75
Car	22	66	88
Bird	0	0	44
Cat	0	25	50
Deer	25	41	50
Dog	33	16	50
Frog	36	72	81
Horse	16	0	83
Ship	73	26	53
Truck	0	0	75
AVG	25.50%	29.60%	64.90%

A case could be made for this in the situation where the randomly initialized auto-encoder produces 0% accuracy for the bird and cat classes. It may be the case that the fur of the cat and feathers of the bird are being grouped with the features for the dog, deer, frog, and horse classes which have attained higher accuracies with this model. This seems reasonable as for the pre-trained auto-encoder experiment, some of the 0% classes in comparison to the randomly trained experiment have a small increase in accuracy. The increase in accuracy may likely be due to the more concise feature representations from the trained auto-encoder.

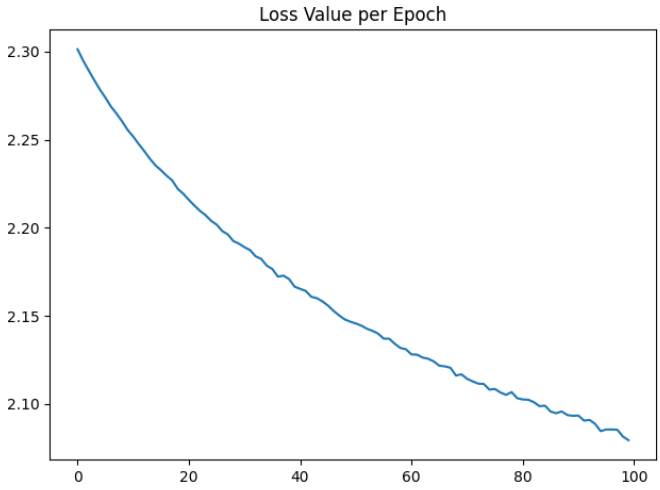

Fig. 5. Loss curve generated from the linear classifier on the randomly initialized encoder feature maps.

When considering the loss curves from each experiment, the main distinction to note is the smoothness of the end-to-end loss. The loss curves from the linear classifier trained on the random auto-encoder weights seems to be almost linear, while the loss curve trained on the pre-trained auto-encoder weights seem to be more volatile from the fifth epoch onward. Again, this opens the opportunity to research how training the

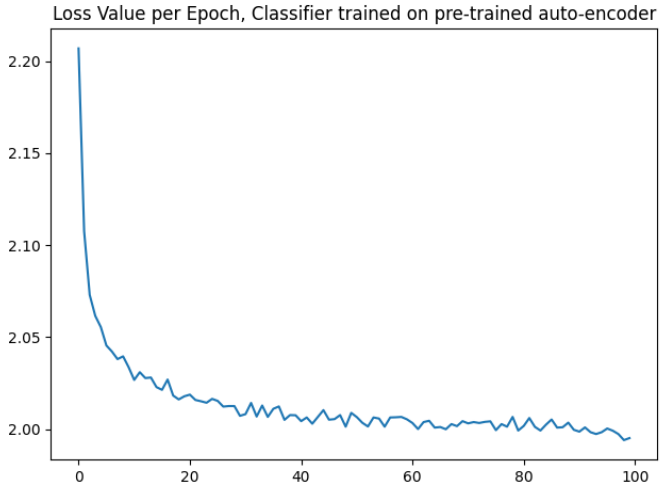

Fig. 6. Loss curve generated from the linear classifier on the pre-trained auto-encoder feature maps.

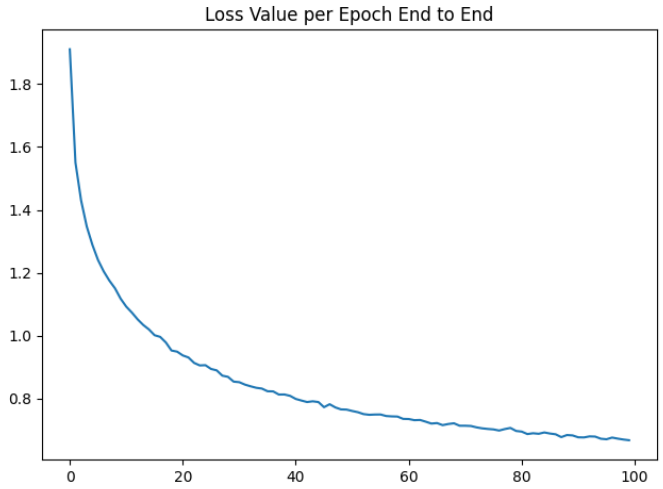

Fig. 7. Loss curve generated from the linear classifier on the randomly initialized encoder feature maps.

auto-encoder on different loss functions may serve to optimize classification. Smoothness in the end-to-end loss curve is seen as more of an ideal, as the smoothness suggests consistent steps toward a global or local minima.

Another interesting result is the comparison of the decoded images in both the randomly initialized auto-encoder and auto-encoder trained end-to-end with the classifier. The pre-trained auto-encoder diagram has been left out as the results are indistinguishable from those in Fig. 1. The results of the randomly initialized auto-encoder are not surprising, as it is makes sense that the encoder network can not learn any meaningful features from random weights. Since the weights are initialized and frozen upon training, it is clear that the images in Fig. 7 are produced at a high loss if the loss were to be calculated. What is most surprising with the randomly initialized auto-encoder is the fact that the one-layer classification network is able to

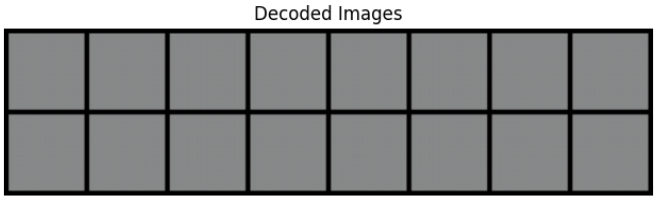

Fig. 8. Decoded images from the randomly initialized auto-encoder. This result is unsurprising as the images are being reconstructed at a high BCE loss.

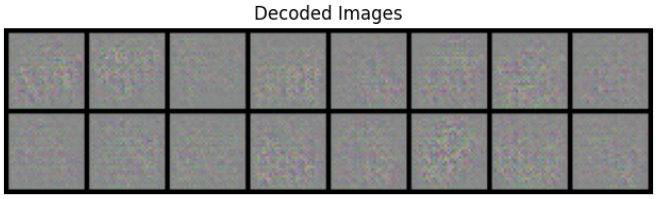

Fig. 9. Decoded images from the auto-encoder trained end-to-end. Although subtle, can begin to see some parts of the image that are being decoded. Diagonal lines, blobs, regions of the image where the object may exist, etc.

achieve a 25.50% average precision across all classes which is greater than theoretical random results of 10% accuracy.

The end-to-end decoded images are interesting, as it seems to be decoding some sort of image features. In some images it seems like the blob of the object is being decoded, or the specific region where the object exists. This is particularly interesting as the decoder network did not have its weights updated during the training time of the end-to-end auto-encoder/classifier model. This could provide evidence in support of the encoder being more responsible in attaining the higher accuracy in the end-to-end model. A future experiment to run would be to back-propagate through the decoder network as well and see if the results of both the decoder and classifier will be greater, or hindered by the other.

V. Discussions

Throughout this project, techniques for representing image data for higher classification accuracy have been explored. This was done with the intention of finding methods for separating image features into data clusters to eventually run unsupervised learning or clustering algorithms on. Future work from this point onward would include training a neural network on smaller batches of data, and classifying images based on which cluster the unseen images are closest to. With the insight that the pre-trained auto-encoder increases classification accuracy, this leads to the conclusion that the pre-trained auto-encoder learns features that can be separable by the linear classifier in comparison to the random initialization of auto-encoder weights.

Another interesting experiment would be to train both the complete auto-encoder and the linear classifier at once, using the BCE and CE loss function to train the network at once. It

would be interesting to see if the encoder (as it is the common factor between the decoder and classifier) can be optimized to support both tasks with any sort of accuracy. Using two losses to train a network involved in two tasks does not seem like it would be the most sensible thing to do or produce the best results. However, if the results of this happened to be somewhat positive, there could be more research opportunities in this area for generalizing a network over multiple tasks.

Some additional modifications to the network which seem promising are using *Local Aggregation Loss* (LA) [5] for training the auto-encoder. From some high-level research it seems that this loss function optimizes the auto-encoder network for greater separability of image features for image clustering. An informative article written on this topic was found by Anders Ohrn [6] and describes this topic in a concise manner.

Using industry standard networks and running the same tests is also a viable next step. If there is a correlation between this simple network and industry standard architectures, it could lead to insights for whether this problem is unique to the given architecture. If this is the case, perhaps new architectures paired with the LA loss above can be developed to eventually train a neural network on one (or significantly few) image(s) of each class and still achieve a high classification accuracy.

In summary, this project began as an investigation into few shot learning and auto-encoders. From this, PCA and a linear classifier were used to find separability or patterns within the encoded image data. Three experiments were run with the auto-encoder and classifier networks to provide greater insight on the role of the auto-encoder vs. the classifier. While the classifier trained on the pre-trained auto-encoder did not provide as large of an accuracy increase from the random initialization, the results provide many paths forward for future areas of investigation.

VI. Implementation and Code

```
self.classifier = nn.Linear(48, 10)
```

Fig. 10. PyTorch implementation of one-layer linear classification network. Code snippet shown here to celebrate the simplicity and ease-of-use of PyTorch. This is trained from the feature maps of three differently initialized and trained auto-encoders

The *PyTorch* package is used throughout the project which consists of the *torch* and *torchvision* packages upon installation and use within python. Other packages such as *sklearn* were used for PCA, and *matplotlib* for plotting the diagrams. The auto-encoder was taken directly from Github [3], and the linear classifier was a simple one-layer network that takes the 48 dimensions of the auto-encoder feature representation and outputs 10 class scores. This was convenient to implement with PyTorch and consisted of a one-line definition shown in Fig 9.

Various research papers and blogs were consulted for diagrams, ideas for techniques, and gaining general background knowledge. The papers are cited accordingly in the following references section, and cited where relevant throughout the project report.

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.
[2] A. Dertat, "Applied Deep Learning - Part 3: Autoencoders."
[3] C. Ni, "PyTorch CIFAR-10 Autoencoder."
[4] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.
[5] C. Zhuang, A. L. Zhai, and D. Yamins, "Local aggregation for unsupervised learning of visual embeddings," 2019.
[6] A. Ohrn, "Image Clustering Implementation with PyTorch."
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.