Assignment Report

1. Problem Definition

1.1 Dataset Description

Design a network that combines supervised and unsupervised architectures in one model to achieve a classification task. The dataset used in this task is Cifar-10 dataset under the condition that only 50% of the following classes for training (bird, deer and truck) can be used.
The training data has size of 50000, with 10 classes each consisting 5000 small RGB images of size 32 by 32. Since three classes will only preserve half of the training image, the costumed training dataset is 42500 images. The testing data has 10000 images.

1.2 Model Description

Requirement: The Architecture model must start with autoencoder(s) that is connected through its hidden layer to another network, as shown in the figure below. The autoencoder takes an input image at node 1 and reconstructs it at its output at node 3. It creates valuable features at its hidden layers (node 2) during this process. it is hypothesized that if node 2 is used as input for the CNN (node 4) then the classification can be improved. The model can be trained in an end to end style with the classifier, or the autoencoder and classifier can be trained separately.

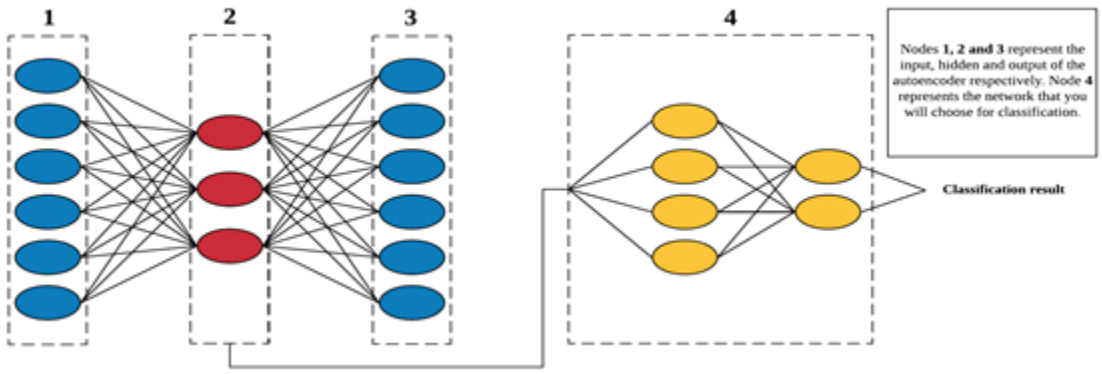


Fig. 1

1.3 Training Description

The model will be trained on Google Colab with one GPU. Pytorch framework will be used due to its good flexibility.

2. Experiments Design

2.1 Data Preparation

To prepare the images for the network, I have normalized each color channel by subtracting a mean value and dividing by a standard deviation. I will choose to augment the training data in this stage because of the imbalance situation. These operations are done using image transforms (including

horizontal flipping and a small degree rotation), it will effectively provide the neural network with many different versions of the same image. The example is shown in the figure below.

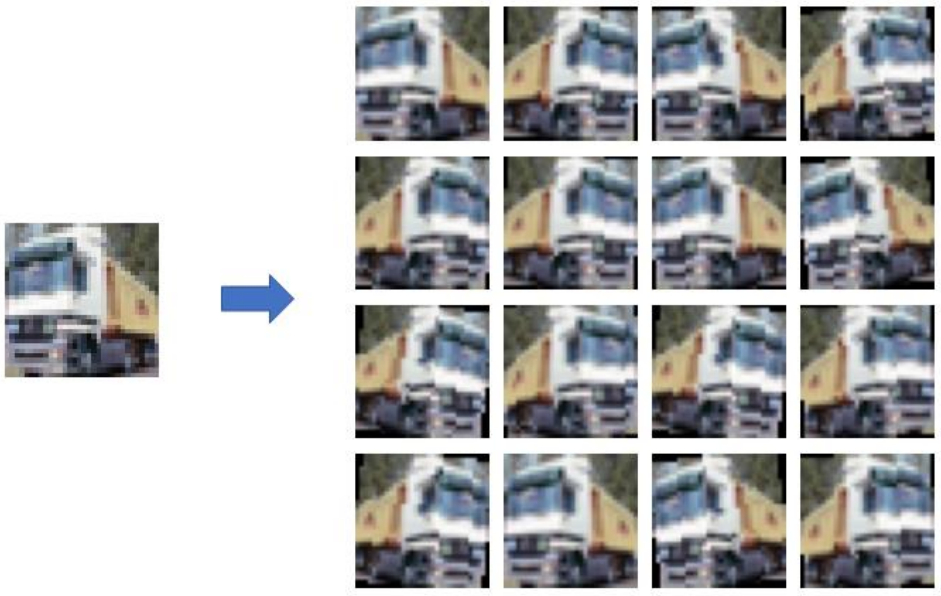


Fig.2 Data Augmentation Example

The test dataset will be spilt to 5000 images as validation set and 5000 images as final testing set. As the result, the dataset has been arranged in three sets: train, valid, test with a split of roughly 80%, 10%, and 10%.

2.2 Model design

2.2.1 Problem formulation

Given an image, our goal is to through training and autoencoder's reconstruct the image to predict the class label.

2.2.2 Network Architecture Experiments

2.2.2.1 Prototype Model

To start prototyping the autoencoder model, I have built the following simple structure.
The encoder consists of three blocks of convolutional layers with 4×4 filters. ReLU activation function used for all the layers, as it's the standard with deep neural networks. the number of channels in each block is increased with the depth of the network. The output of encoder is 48 × 4 × 4, which has a very high compress factor.
The decoder consists of three blocks of convtranspose layers with 4×4 filters to reconstruct the image from its compressed version. The last layer uses the sigmoid activation so the outputs to be between [0, 1] since the input is also standardized in the same range. The structural representational is shown in the figure below.

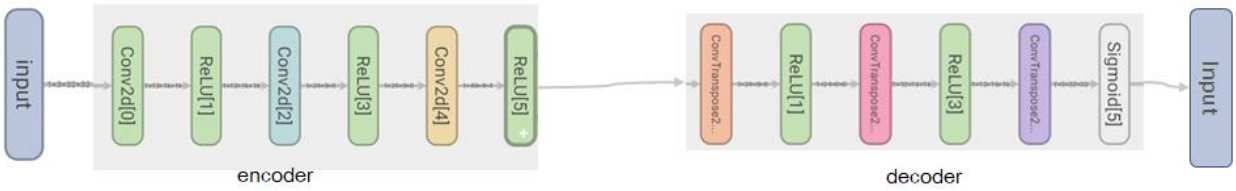


Fig3. Prototype Autoencoder structure

The Classifier branch will take encoder's output as its input and generate the corresponding label. The classifier has two parts. First, the convolution layers consist of stacking conv layers with batch normalization as a modest regularization method as an effort to stabilize and accelerate the learning process. The fully connected layer is started with dropout regularization.

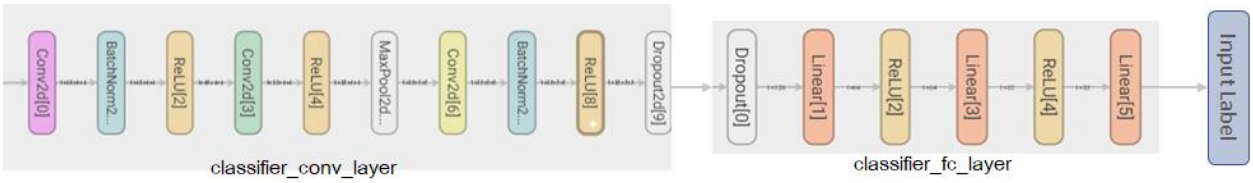


Fig4. Prototype Classifier structure

The model is will be trained end to end, which means the autoencoder and classifier are jointly optimized and fully differentiable. The structure is shown in the figure below. I choose Adam optimizer because it achieves good results fast. The loss of the whole model is designed as the addition of Mean squared error (MSE) loss of the autoencoder branch and the cross-entropy loss of the classification branch (weighted addition is also explored but it doesn't show a lot impact to the result).

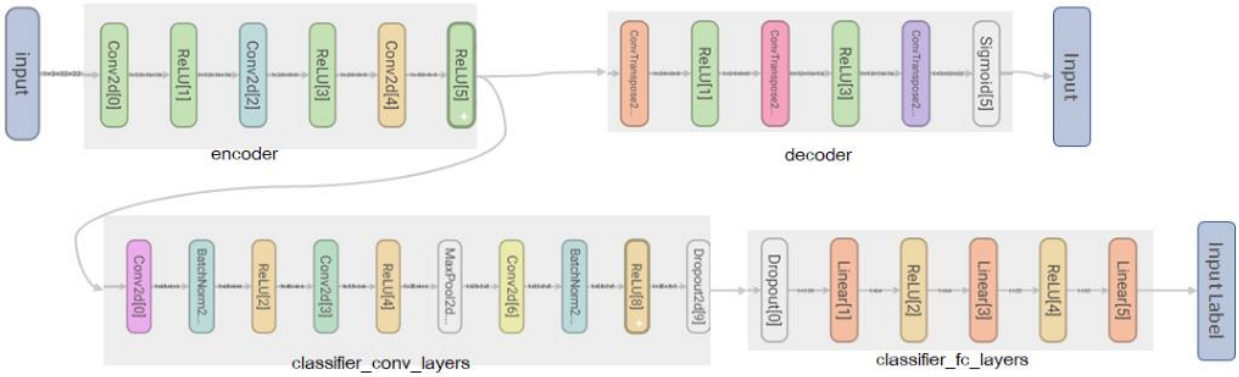


Fig5. Prototype end-to-end Autoencoder-classifier structure

Another set of experiment is also provided where autoencoder and classifier are trained separately, in the manner of training autoencoder first, and saving trained model and weights, and second, I will Freeze all the weights for autoencoder when training classifier. The input of the classifier is coming from autoencoder's encoder's output. But I will not include in the discussion due to poor performance and lack of further investigation. The future work on this experiment is listed in future works. The structure is shown in the figure below.

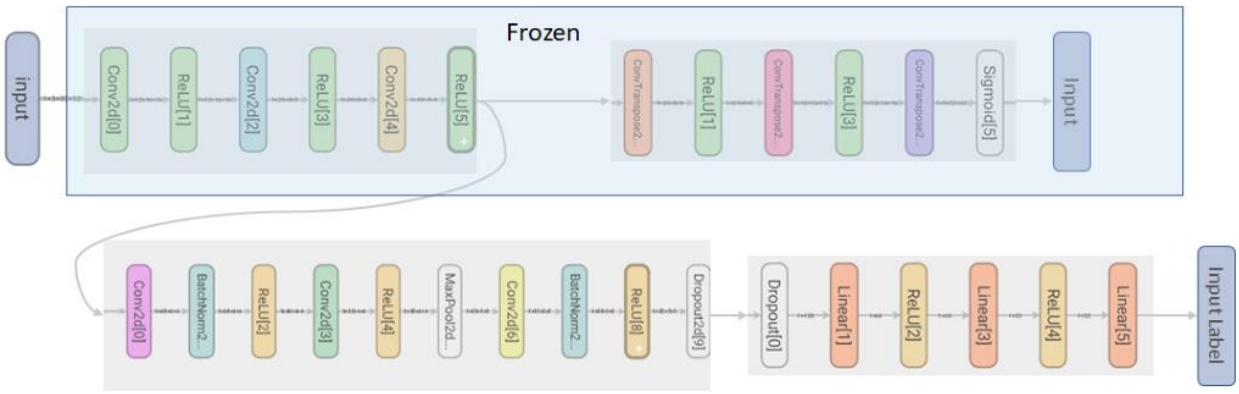


Fig6. Prototype separately trained Autoencoder-classifier structure

2.2.2.2 Improved Model Structure

Based on the training result of the prototype model and some further experiments, I have made several changes to the prototype, the new structure is showed in the figure below.

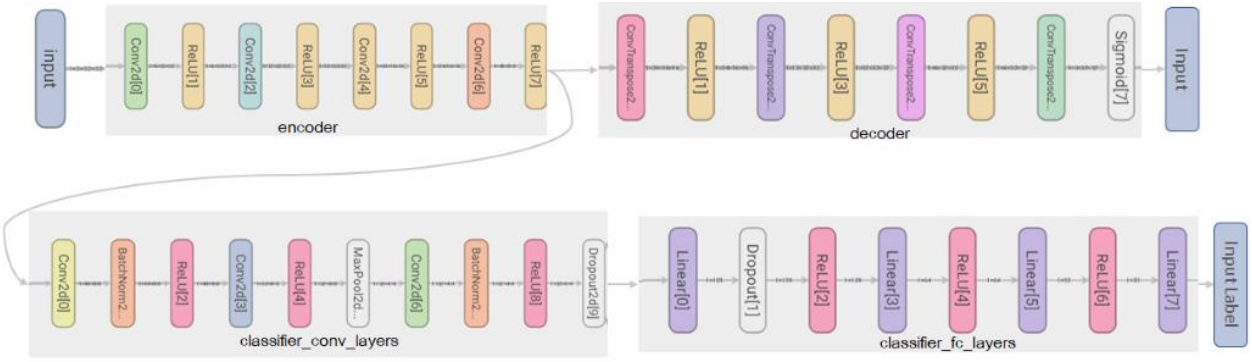


Fig8. Improved end-to-end Autoencoder-classifier structure

The changes I made are:

1. Change all kernel size from 4×4 to 3×3, as it's common choice to keep the kernel size at 3x3 or 5x5, also stacking smaller convolutional layers shows better result than stacking bigger ones.
2. Make the autoencoder deeper to learning the complex structure of images.
3. The output of encoder has increase from 48 × 4 × 4 to 48 × 8 × 8, hence more feature could be preserved in the compression.
4. To offset the more complex structure of autoencoder, I used a MaxPooling layer to effective reducing the dimensionality in classifier branch.

The overall result shows great improvement; therefore, this improved model will be used as the baseline in further experiments (from this point forward this improved model will be refer to as baseline model.)

2.2.2.3 Other Experiments for Performance Improvement

I have also investigated the following modificationa to seek to improve on the baseline model.

1. Try different optimizers: such as SGD optimizer and Adabound optimizer (which had claimed to train as fast as Adam and as good as SGD).
2. Try other kernel weight initializer, such as Xavier uniform initializer.

I provide individual notebook for each of the experiment. The results are discussed in the next section.

2.2.3 Model Training

For training, I iterate through the train DataLoader , each time passing one batch through the model. One complete pass through the training data is known as an epoch, and I train for a set number of epochs. After each batch, I calculate the loss (with criterion (output, targets)) and then calculate the gradients of the loss with respect to the model parameters with loss.backward(). This uses auto differentiation and backpropagation to calculate the gradients.

After calculating the gradients, optimizer.step() is called to update the model parameters with the gradients. This is done on every training batch, so I will be implementing Adam Optimization (or stochastic gradient descent). For each batch, the accuracy is calculated for monitoring and after the training loop has completed, then the validation loop starts.

Early stopping is implemented in reference to other project's code but not used so far in the experiment.

3. Results and Discussion

I trained the model on 42500 training images and validated on 5000 images. The training and validating dataset's accuracy, classification branch's loss (CLFloss) and autoencoder branch's loss (AEloss) of are plotted respectively.

The Prototype model and the Baseline model are compared below, the baseline model shows great improvement in accuracy starting from the second epoch, and after 30 epochs, the improvement on both training and validating dataset have shown constant improvement for more than 5%, and the CLF loss's minimization also has improved steadily. The AEloss pf both models converge at around 5 epochs and the improvement is also significant.

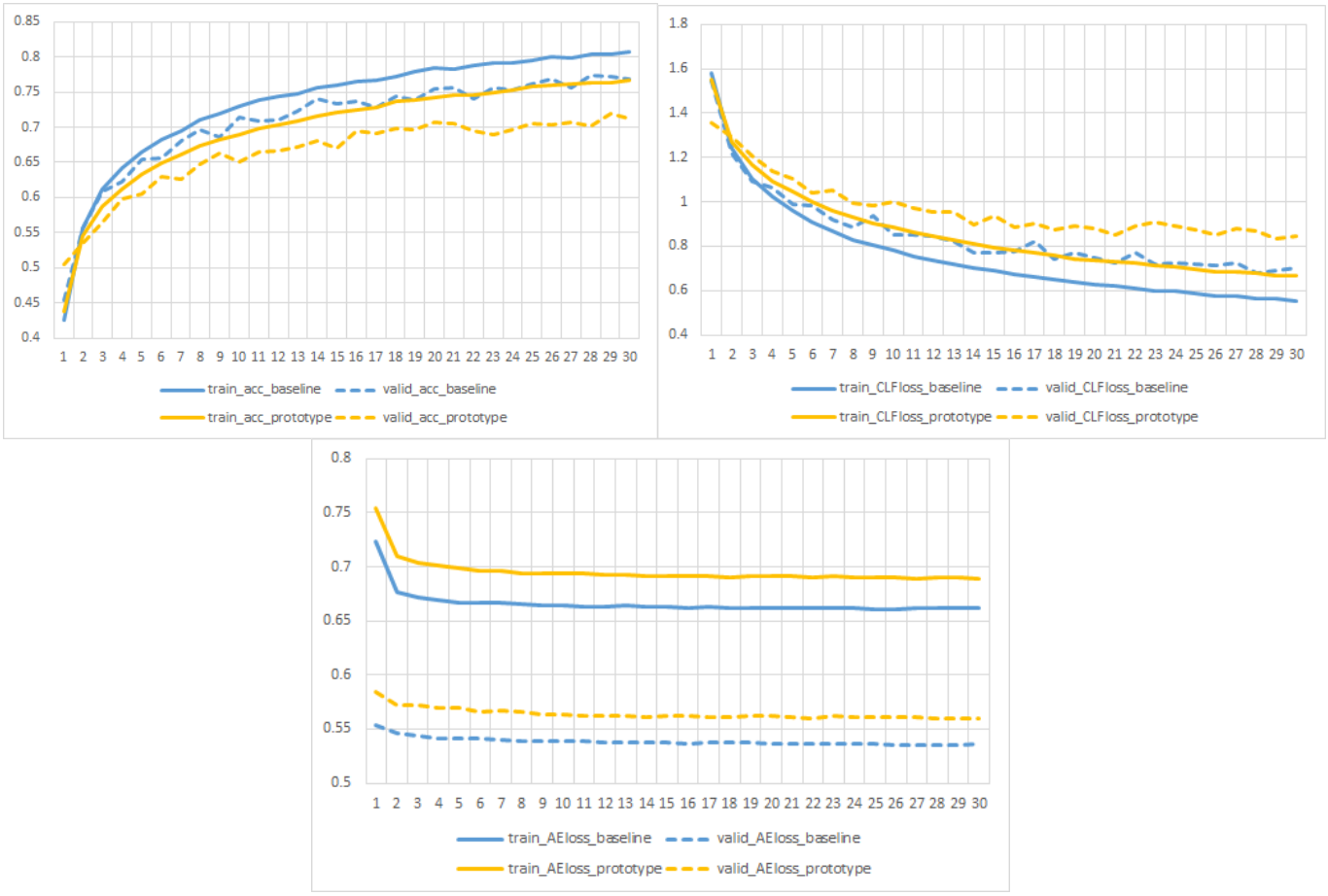
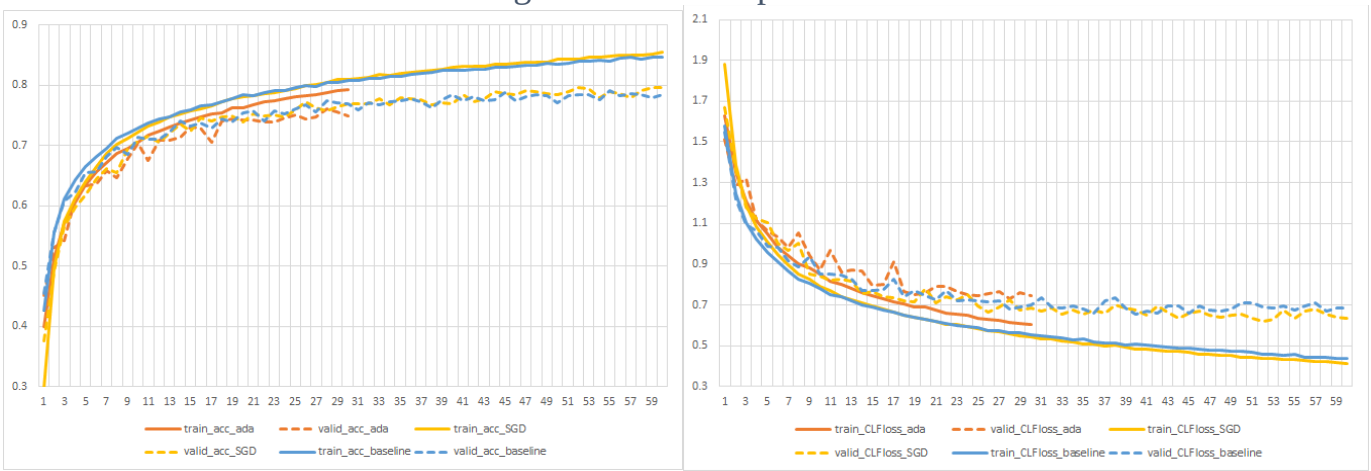


Fig9. Prototype model vs Baseline model on training(solid) & validating(dash) dataset's accuracy, CLFloss and AEloss

The comparison among three optimizers' result is very interesting. The baseline model used Adam optimizer, which shows slight superiority than SGD optimizer during beginning of the training process on the training set, but after 50 epoch SDG shows slight better accuracy on both training and validation dataset. Adabound is only trained for 30 epochs but the result not as good as I expected. For the CLF loss minimization, the pattern is similar to the accuracy's result, it looks that SGD is more promising in longer training period. For AEloss, the SGD is significantly higher than using the other two optimizers.



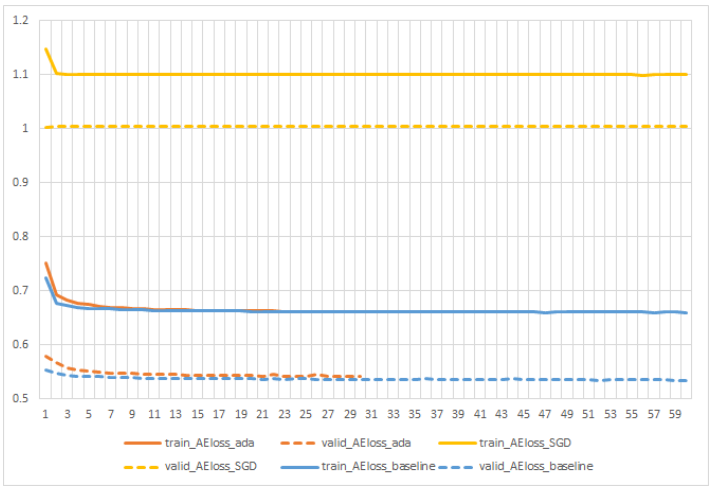


Fig9. Three optimizer comparison on training(solid) & validating(dash) dataset's accuracy, CLFloss and AEloss

Compare the baseline model whose kernels has been initialized use He initialization, the xavier uniform kernel initialization gives better validation accuracy for the whole training period and has better CLFloss result.

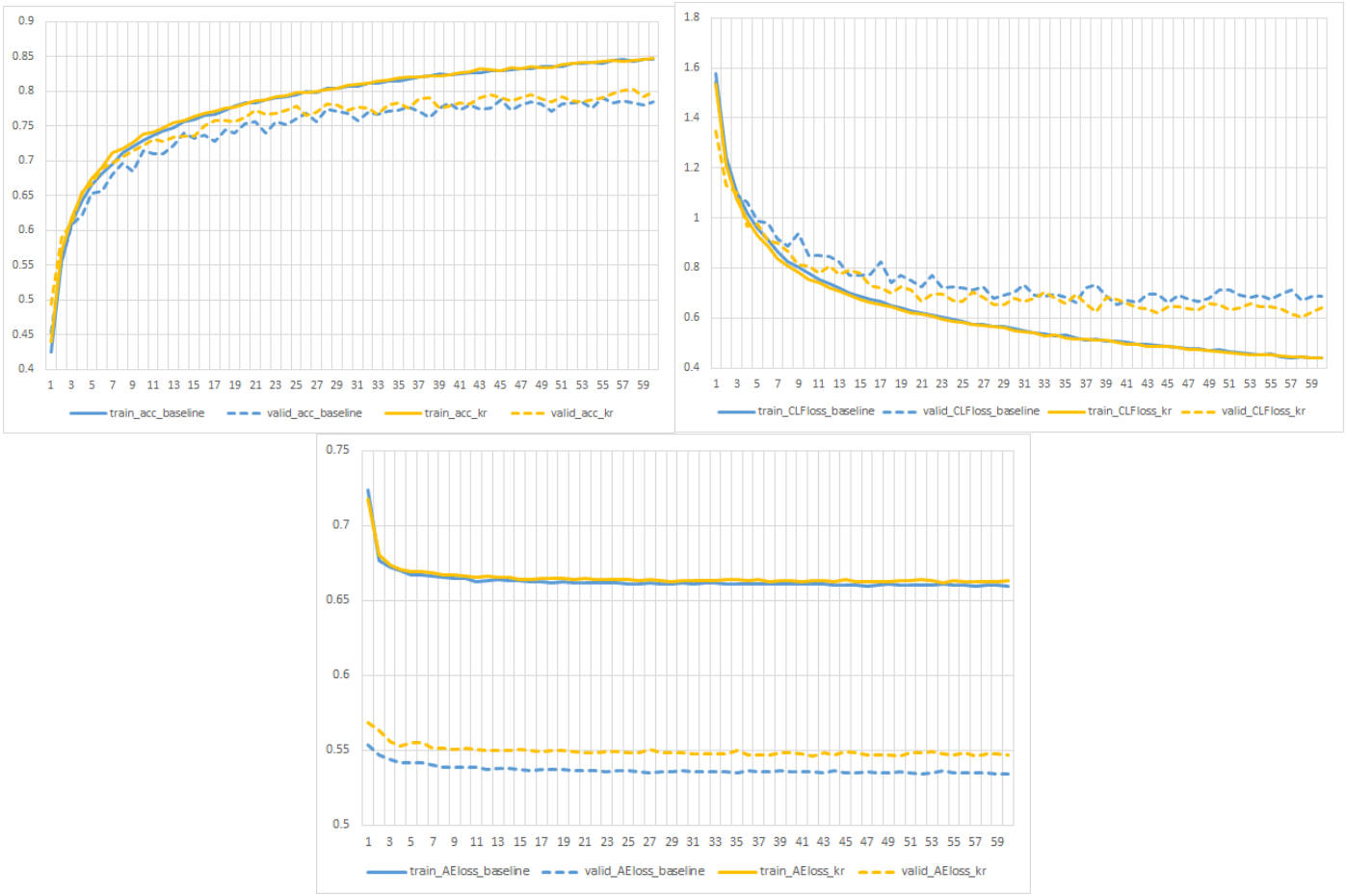


Fig9. He initialization(baseline) vs Xavier uniform initialization (kr) on training(solid) & validating(dash) dataset's accuracy, CLFloss and AEloss

The models are tested on 5000 hold out test dataset after training, the results are follows, The baseline model's result is satisfying in most categories but SGD seems perform uniformly good on all categories. Xavier uniform kernel initialization also gives a boost on baseline model's performance in most categories.

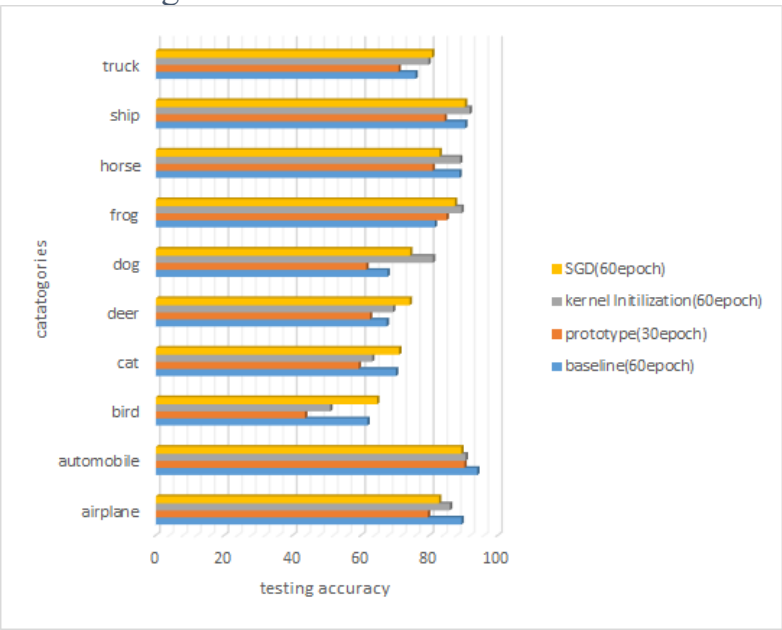


Fig10. Testing Accuracy

3. Conclusion

In this assignment I has designed an autoencoder classification model for Cifar10 datasets, which is fully differentiable that can be trained end-to-end. I first purposed a prototype model and worked on several aspects to improve it, and I further introduced experiments on difference optimizer and weight initialization for improving the accuracy for classification.

4. Future Work

As our result and discussion shows, both SGD optimizer and Xavier uniform kernel initialization could improve the current baseline model, it will be promising to combine both in the next training experiment.

Also, as the result shows, the autoencoder's loss stops improving at very early stage of training, since autoencoder only learn a "compressed representation" of input and decompress it back to match the original input. I would suggest a better loss function to improve the featured learned at the hidden layer, like for example using variational autoencoders to learn the parameters of a probability distribution representing the data at the hidden layer.

Last, it will be a very interesting task to compare the end-to-end trained model and separately trained model's performance, weight histogram and image reconstruction, therefore I would like to purpose it as a future task.