AI Capstone: Project 1 Report

110550088 李杰穎

March 11, 2023

1 Public Image Datasets: CIFAR-10

1.1 Datasets Description

CIFAR-10 is a popular image classification datasets that is frequently used as a benchmark for computer vision and deep learning tasks. It contains 60,000 32x32 color images of 10 different objects, with 6,000 images per class. The classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The datasets is divided into 50,000 training images and 10,000 testing images. CIFAR-10 is challenging due to the small image size and the high variability of the objects within each class. It is often used for developing and testing new machine learning algorithms and architectures for image classification.

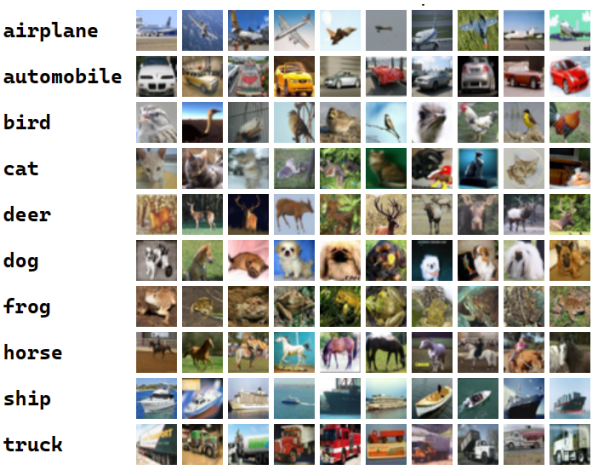

Figure 1: Ten randomly select images from 10 classes.

In this project, I will directly use the datasets provided in the `torchvision` library, which is part of the PyTorch deep learning framework. Torchvision is a library consists of lots of datasets (including MNIST, CIFAR-10, ImageNet, ...) and pretrained model (including ResNet, VGGNet, BERT, ...).

As for the test set, I will use the original train/test split of the CIFAR-10 datasets. The datasets is divided into 50,000 training images and 10,000 testing images, with an equal number of images from each class in both sets. This train/test split is commonly used in the literature and provides a fair evaluation of the performance of machine learning models on the CIFAR-10 datasets.

It's also worth noting that cross validation can be also used for the CIFAR-10 datasets. However, the number of images in the datasets is large and most of the research paper use the original train/test split. Therefore, it's reasonable for this project to use the original train/test split.

Noted that I also use torchvision to download and process the datasets.

1.2 Algorithms

For the CIFAR-10 datasets, I use two classifier, ResNet-18 and Multilayer Perceptron (MLP).

1.2.1 ResNet-18

ResNet-18 is a popular deep convolutional neural network architecture used for image classification tasks. It was introduced by Microsoft Research in 2015 and is part of a family of ResNet models, which are designed to tackle the problem of vanishing gradients in very deep neural networks. ResNet has different variance, from ResNet-18 all the way to ResNet-152. The number after ResNet indicates the number of layers. Therefore, ResNet-18 is a relatively shallow model, with 18 layers, but it still achieves state-of-the-art performance on a range of image classification benchmarks, including ImageNet, CIFAR-10, and CIFAR-100.

For the sake of convenience, I directly use the ResNet-18 models which provides in torchvision library. Torchvision library also provides the pretrained weights. However, those weights are trained on ImageNet Datasets, not CIFAR-10. Therefore, I also compare the performance with or without using pretrained weights. There are more details in appendix.

It's worth noting that because the image size of CIFAR-10 is only 32 x 32. And ResNet is original designed for ImageNet, which image size is 224 x 224. Therefore, the first convolution layer with kernel size equals to 7 is too large for CIFAR-10 datasets, this will make lots of features lost from the first convolution layer. Moreover, the max pooling layer may also make features lost. Therefore, I change the kernel size from 7 to 3 and also remove the max pooling layer. In the analysis section, I will discuss the performance difference after changing these two layers.

1.2.2 MLP

A multilayer perceptron (MLP) is a type of artificial neural network (ANN) that consists of multiple layers of interconnected nodes (also called neurons) in a feedforward network architecture. It is one of the most commonly used neural network models for supervised learning tasks such as classification, regression, and pattern recognition.

An MLP typically consists of an input layer, one or more hidden layers, and an output layer. The number of neurons in the input layer is determined by the number of features in the input data, while the number of neurons in the output layer corresponds to the number of output classes or the number of output variables in a regression task. The number of neurons in the hidden layers is a hyperparameter that can be tuned during the training process.

We know that MLP has one input layer, one output layer and several hidden layers. In this project, I will only use an MLP that consists of two hidden layers with same number of neurons. The number of neurons is also a hyperparameters that will compare in the analysis section. As for the input layer, I just flatten the images into an one dimensional vector, with $3 \times 32 \times 32 = 3072$ elements as input. For the output layer, MLP will output a one dimensional vector with 10 elements, corresponding to 10 classes.

1.2.3 Epochs, Loss Function, Optimizer and Learning Rate Scheduler

In the following experiments, I will use the below configuration for both MLP and ResNet-18.

- Epochs: 200
- Learning Rate: 0.1
- Loss Function: `CrossEntropyLoss`
- Optimizer: `SGD`
- Learning Rate Scheduler: `ReduceLROnPlateau` (monitor on `val_acc,patient=10,factor=0.1`)
- Batch Size: 128

Learning rate scheduler is a tool to change learning rate while training. And the LR scheduler I use will monitor on `val_acc`. When `val_acc` doesn't increase for 10 epochs, the learning rate will be reduced by 90%.

1.3 Analysis

First of all, because CIFAR-10 already has provided a train/test split, I will not use cross-validation for this datasets.

1.3.1 Compare Different Architecture of ResNet-18

As I mentioned earlier, because the kernel size of first convolution layer is too large for images in CIFAR-10 datasets, I change the kernel size and also remove the first max pooling layer. This version is called "Modified ResNet-18", and the unmodified one is called "Original ResNet-18". As we can see in the chart from Figure 3 to Figure 6, the modified one is significant better than the unmodified one. The highest test accuracy for the modified one is 0.9227, in the contrast, the unmodified one is 0.8766.

In conclusion, changing the architecture of ResNet can indeed significantly improve the model's accuracy.

1.3.2 Comparing whether there is a pre-trained weight for ResNet-18

PyTorch provides a pretrained weight for ResNet-18. However, the pretrained weight is trained on ImageNet datasets. Therefore, it might perform well on CIFAR-10 datasets. Because of this I make an experiment to test how the existence of pretrained weight affect the performance of model.

As we can see in Figure 7 and Figure 8, the ResNet-18 with pretrained weight doesn't perform very well. The highest test accuracy of ResNet-18 without pretrained weight is 0.9227, as for the model with pretrained weight, this number is 0.9177. I will discuss more in 1.4.1.

1.3.3 Compare Different Architecture of MLP

For the MLP, I fixed the number of hidden layer to two. And by setting the number of neurons in hidden layers to 100 and 2500, I trained two MLP model.

As we can see in the chart (Figure 11 to Figure 14), the MLP with 2500 hidden neurons is significant better than the 100 ones. The highest testing accuracy for the 2500 one is 0.5886, as for the 100 ones is 0.5324.

1.3.4 Compare Different Amount of Training Data

In this section, I will first analysis ResNet-18 case, and then analysis MLP case.

Noted that though the number of training data is different, the testing data for both model is exactly the same.

First, for ResNet-18, we can look at Figure 15 and Figure 16. As we can observe, the "half" one train much more

slower than the ResNet-18 that trained with full training CIFAR-10 datasets, which contains 50000 images. And the final test accuracy, the "half" one's accuracy is 0.8782. Which is less than the full-trained model by around 5%.

Second, let's look at the MLP. Also we can observe that in Figure 17 and Figure 18, the training accuracy of the MLP with full training datasets is slightly better, and for the testing accuracy, it is nearly the same. The "Half MLP" is 0.5816, and the "Full MLP" is 0.5886.

1.3.5 Compare with SOTA Model

After researching, the state-of-the-art model on CIFAR-10 is Vision Transformer (ViT). In particularly, the ViT-H/14 model can achieve 99.5% test accuracy, which is astonishing results. However, this model require 632M parameters, in the contrast, ResNet-18 only requires 11M parameter.

In conclusion, ViT is 57 times larger than ResNet-18, but it only increase 7% of accuracy.

1.3.6 Overall Comparison

From the above comparing, we know that the best performance for ResNet-18 is the modified ResNet-18 without pretrained weights. As for MLP, the best model is with 2500 hidden neurons. And the SOTA model for CIFAR-10 is ViT-H/18.

For the overall comparison table, please refer to Table 1.

1.4 Discussion

1.4.1 Assessing Expected Results and Behaviors in Experiments

1. **Using pretrained weights or not**

   In 1.3.2, I compare the performance of the ResNet-18 with or without pretrained weights. And the result is the model without pretrained weights has better performance.

   One can guess the model with pretrained weights might be better, but it's not. I think one main reason is that I already changed the model architecture, therefore the idea of "transfer learning" didn't work well here. And also, the distribution of CIFAR-10 and ImageNet is also difference. Therefore, trained the model from scratch might be a better idea.

   In conclusion, I think these are the two main reasons why the model without pretrained weights is better.

1.4.2 Factors Affecting Performance

1. Model architecture
2. Different hyperparameters
3. Number of epochs count
4. Number of training data
5. LR scheduler

   Using LR scheduler is actually really important for machine learning, especially when SGD is used. Because larger LR in the later stage of training might make the model can't converge to optimal point. Therefore it's necessary to have LR scheduler.

1.4.3 Future Experiments

1. **Using adam optimizer instead of SGD.**

   The adam optimizer can change the learning rate of each weights, which is might be a better method compared with LR decay. The SGD along with LR decay method may take longer time to converge.
2. **Apply features extractions before sending into MLP.**

   Apply features extractions on images can extract useful features, and using these extracted features might be able to improve the MLP performance.

1.4.4 Findings and Open Questions from Experiments

1. **MLP doesn't perform very well**

   Before conducting the experiment, I thought it may achieve an accuracy around 70%. But it turns out only get 60%.
2. **Changing the architecture of the CNN is useful**

   I improved the model's accuracy by 5% by only changing the kernel size of one layer and removing one max pooling layer. This is a new inspiration for me that sometimes we need to compare the difference of the images we need to classify now with the designated image, and based on the input images to design the proper architecture.

2 Public Non-Image Datasets: 20 Newgroups

2.1 Datasets Description

The 20 Newsgroups datasets is a popular datasets used in natural language processing and machine learning research. It consists of a collection of approximately 20,000 documents, partitioned into 20 different newsgroups,

each representing a different topic. The datasets was first collected by Ken Lang and others at the University of California, Irvine, and has since been widely used in research and experimentation.

Each document in the 20 Newsgroups datasets is a posting to one of the 20 newsgroups, and is represented as a single text file. The datasets includes a variety of topics, including politics, religion, sports, science, and technology, among others.

In this project, I will use vectorize version of this datasets. This is done by using the `CountVectorize` in scikit-learn library. In this way, the text data is transformed to real-valued vector, which can be processed by `RandomForestClassifier`, `GradientBoostingClassifier`, `AdaBoostClassifier`, etc.

As for the cross-validation, because scikit-learn has alreaday provided a train/test split, which contains 11314 train data and 7532 test data.

2.2 Algorithms

In this datasets, I use four kinds of ensemble learning algorithms, Random Forest Classifier, Gradient Boosting Classifier, and AdaBoost Classifier. One SVM algorithm, linear SVM. And a MLP Classifier, which is same MLP architecture in the first datasets. Therefore, I will only talk about the four ensemble learning algorithms and the SVM algorithm.

Random Forest Classifier, Gradient Boosting Classifier, and AdaBoost Classifier are all powerful machine learning algorithms commonly used for classification tasks. These algorithms are known for their ability to handle complex datasets and provide accurate predictions.

And SVM (Support Vector Machines) is also a machine learning algorithm used for classification and regression analysis. It was developed by Vapnik and his colleagues in the 1990s. SVM is a supervised learning algorithm that works by finding the optimal hyperplane that separates the data into different classes. The hyperplane is chosen in a way that maximizes the margin between the closest data points from each class. These closest data points are known as support vectors.

These five classifiers can be found in the `scikit-learn` library. In the following experiments, I directly apply the default value of `scikit-learn` provides. I state the option clearly in different classifier.

2.2.1 Random Forest Classifier

Random Forest Classifier is an ensemble learning algorithm that creates multiple decision trees and combines their outputs to make a final prediction. Each tree is trained on a random subset of the features and samples of the datasets, making the algorithm robust to overfitting. The final prediction is made by taking the mode of the predictions of all trees in the forest.

- The number of tree: 1000 or 500 (compare the difference)
- Criterion: Gini Impurity

2.2.2 Gradient Boosting Classifier

Gradient Boosting Classifier is another ensemble learning algorithm that creates multiple weak learners and combines their outputs to make a final prediction. Unlike Random Forest Classifier, it creates trees sequentially, with each new tree correcting the errors of the previous tree. This iterative process continues until a stopping criterion is met, resulting in a strong learner that makes accurate predictions.

Gradient Boosting algorithm is widely used in many machine learning competitions in Kaggle. This is reason why I choose this algorithms in this project.

- The Number of Tree: 100 or 50 (compare the difference)
- Loss Function: `log_loss`
- Criterion: `friedman_mse`
- Learning Rate: 0.1
- Max Depth: 3

2.2.3 AdaBoost Classifier

AdaBoost Classifier, short for Adaptive Boosting Classifier, is also an ensemble learning algorithm that creates multiple weak learners and combines their outputs to make a final prediction. It is similar to Gradient Boosting Classifier in that it creates trees sequentially, but it assigns weights to each sample in the datasets based on how difficult it is to classify correctly. Samples that are misclassified by a weak learner are given higher weights, making them more likely to be correctly classified by the next weak learner.

- Estimator: Decision tree with maximum depth 1
- Number of Estimators: 50 or 500 (compare the difference)
- Algorithm: SAMME.R

2.2.4 Linear SVM Classifier

Linear SVM is a specific type of SVM that assumes the data is linearly separable, meaning that a straight line can be drawn to separate the data points into different classes. Linear SVM is used when the data is linearly separable, which means that the data can be classified into two or more classes by a straight line. In this case, the SVM algorithm finds the optimal hyperplane in a way that maximizes the margin between the closest data points from each class while keeping the classification error rate as low as possible.

Linear SVM is a simple and effective algorithm that works well on a wide range of problems. However, it may not work well when the data is not linearly separable. In this case, other types of SVMs or non-linear methods may be more appropriate.

- Penalty: L2
- Loss: squared hinge
- dual: True or False (compare the difference)

2.2.5 MLP Classifier

Noted that this MLP Classifier is the built-in version in scikit-learn.

- Epochs: 200
- Optimizer: Adam
- Learning Rate: 0.001
- Momentum: 0.9
- Loss Function: `CrossEntropyLoss`
- Tolerance: 0.0001
- Patient: 10
- Number of hidden layer: 1
- Number of hidden neurons: 100

2.3 Analysis

2.3.1 Compare Different Random Forest Classifier

As we can see in the Figure 19 to Figure 22, obviously, the one with more estimators has higher accuracy. However, the difference between the accuracy is not much, only 0.46%. The accuracy random forest classifier with 1000 estimators is 74.65%.

2.3.2 Compare Different Gradient Boosting Classifier

As we can see in the Figure 23 to Figure 26, the accuracy of gradient boosting classifier with 100 estimators is 70.74%, which is higher than one that with 50 estimators.

2.3.3 Compare Different AdaBoost Classifier

As we can see in the Figure 29 to Figure 28, the accuracy of Adaboost classifier with 50 estimators is 46.52%, which is higher than one that with 500 estimators. This is not the results I expected, I will discuss about it in the discussion section.

2.3.4 Compare Different Linear SVM Classifier

As we can see in the Figure 31 to Figure 32, using dual or primal optimization is actually the same for 20 newsgroups datasets. And also the accuracy of SVM is 78.47% and AUC is 47.85%, which is a high accuracy and AUC.

2.3.5 Compare Different MLP Classifier

As we can see in Figure 35 to Figure 38, the MLP with 50 hidden neurons perform better than one that with 100 hidden neurons, with accuracy is 78.57%.

2.3.6 Compare Different Amount of Training Data

In this section, I will only test on the best model, which is the MLP model with 50 hidden neurons. Recall that the accuracy of linear SVM model with full train datasets is 78.57%.

As we can see in the Figure 39 and Figure 40, the testing accuracy is 0.7767 which is a little bit less than the full trained one.

2.3.7 Compare with SOTA Model

Currently, to my best knowledge, the SOTA of 20 newsgroups datasets is LinearSVM+TFIDF. TFIDF is a kind of vectorizer and LinearSVM is the same algorithms as in 2.2.4.

2.3.8 Overall Comparison

You can refer to Table 2.

By this table, we know that best model is MLP model with 78.57% test accuracy.

2.4 Discussion

2.4.1 Assessing Expected Results and Behaviors in Experiments

Before conducting the experiment, I guess the gradient boosting or Adaboost algorithm may perform pretty well. Because I heard that GB algorithms perform really well on lots of Kaggle competition. However, after the experiments, I found that the best model is simple linear SVM classifier, and it only took around 2 minutes to finished training, which is really surprising.

Second, I also expect adaboost with more estimators[1] would perform better. After some researching, I found that more estimators may lead to overfitting problem, which made the testing accuracy worse than the less one.

Third, the MLP case is also surprised me, the 50 one perform better than the 100 one. I guess this is also because of overfitting problem.

To overcome overfitting problem, one usual strategy is using "early-stopping". Stopping the training process before the model becoming to be overfitted.

2.4.2 Factors Affecting Performance

1. Different algorithms
2. The latent distribution of data
3. Different vectorizer and preprocessing method

2.4.3 Future Experiments

1. **Try to use different vectorizer**
2. **Try more text preprocessing**
   Including removing stop words, stemming, etc.
3. **Try to use NLP Model like BERT**
4. **Using early-stopping to avoid overfitting**

2.4.4 Findings and Open Questions from Experiments

1. Linear SVM classifier is a great model for some cases
2. Histogram-based gradient boosting is time consuming
3. For simple classification task, the traditional machine learning can already perform really well

[1] Estimator is actually a decision tree, which is a weak classifier. Therefore more classifier may make adaboost algorithm to overfit the training data.

3 Self-made Datasets: Satellite Images of 5 Regions

3.1 Datasets Description

In this datasets, I collect the satellite images of mountain area from 5 regions over the world, including Taiwan, Canada, Himalaya, Hengduan and Argentina. The task is to classify the satellite images to these five regions. The satellite images are from MapTiler, a tile map service. I first calculate which tiles should be download, then combine those tiles into a PNG file. Each image is an 256×256 RGB images.

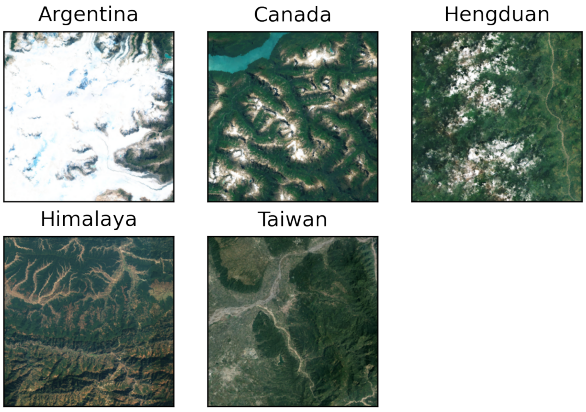

Figure 2: This figure shows 5 satellite images from 5 regions

3.2 Algorithms

Because this datasets is also a image datasets, I just use the two classifiers (ResNet-18 and MLP) which used in the CIFAR-10 datasets with the exact same hyperparameters and configurations.

For MLP, because directly use $3 \times 256 \times 256$ as input layer is too large, I down-sampling the image from 256 x 256 to 128 x 128.

3.3 Analysis

Because I've already conducted the experiments of comparing different architecture of the same classifier, I will only compare the difference between ResNet-18 and MLP. And also compare different amount of training data.

For the validation, because this datasets is self-made datasets, I use 5-fold cross-validation to evaluate the model performance.

3.3.1 Compare ResNet-18 and MLP

As we can see in Table 3, MLP outperform ResNet-19. The mean accuracy of MLP is 98.032%. As for ResNet-18, it's 93.186%.

3.3.2 Compare with SOTA Model

Because this datasets is a self-made datasets, I can't compare to other models. However, there is some similar tasks, using CNN and ViT.

3.4 Discussion

3.4.1 Assessing Expected Results and Behaviors in Experiments

I am actually surprised by the performance of ResNet-18, I thought it can only achieve an accuracy around 90%. However, most of the fold can achieve 100% of testing accuracy.

Furthermore, simple MLP can even outperform ResNet-18, which is also a big surprise for me.

3.4.2 Factors Affecting Performance

1. Size of datasets
2. Different model

3.4.3 Future Experiments

1. **More robust training and testing data**
   This classify problem is too simple for ResNet-18, an unmodified model can achieve near 100% of accuracy. Therefore, I would like to introduce more data or regions to made this classify problem more challenging.
2. **Optimized the model architecture**
3. **Try to use ViT**

3.4.4 Findings and Open Questions from Experiments

1. Small model may perform well
2. Why this classification problem is that easy

Appendix A Figures, Charts and Tables

A.1 Public Image Datasets: CIFAR-10

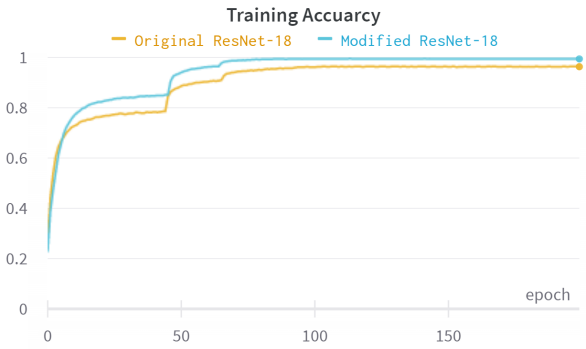

Figure 3: Training accuracy of two ResNet-18 models on CIFAR-10 datasets

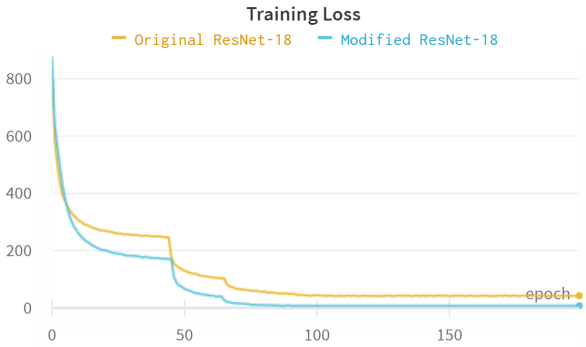

Figure 4: Training loss of two ResNet-18 models on CIFAR-10 datasets

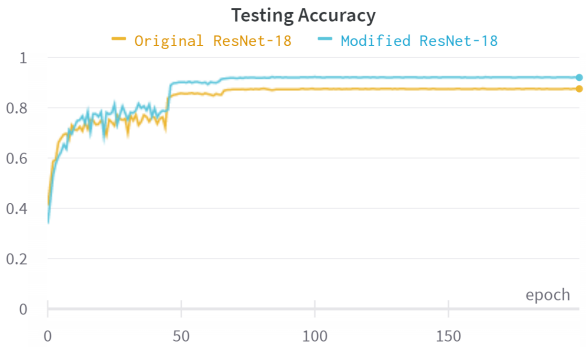

Figure 5: Testing accuracy of two ResNet-18 models on CIFAR-10 datasets

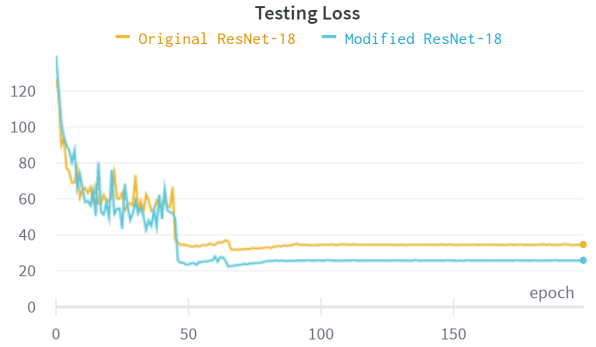

Figure 6: Testing loss of two ResNet-18 models on CIFAR-10 datasets

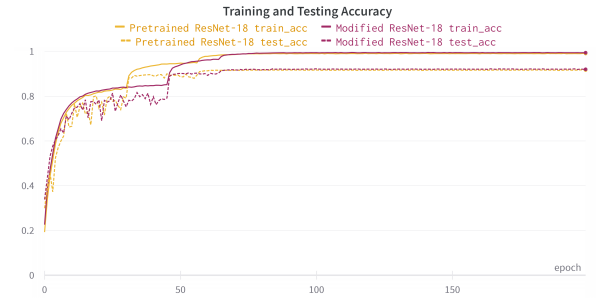

Figure 7: Training and testing accuracy of two ResNet-18 model with and without ImageNet pretrained weight

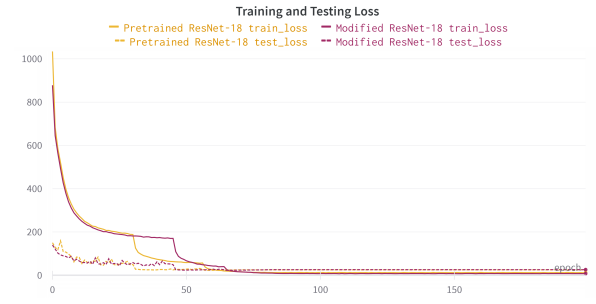

Figure 8: Training and testing loss of two model with and without ImageNet pretrained weight

	precision	recall	f1-score	support
airplane	0.92	0.93	0.92	1000
automobile	0.96	0.96	0.96	1000
bird	0.89	0.90	0.90	1000
cat	0.81	0.83	0.82	1000
deer	0.91	0.93	0.92	1000
dog	0.88	0.84	0.86	1000
frog	0.94	0.94	0.94	1000
horse	0.96	0.94	0.95	1000
ship	0.96	0.95	0.95	1000
truck	0.95	0.95	0.95	1000
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

Figure 9: Accuracy table of pretrained ResNet-18

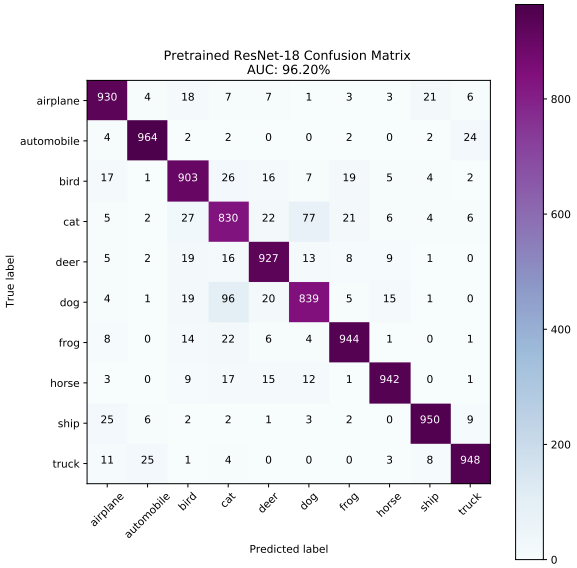

Figure 10: Confusion matrix of pretrained ResNet-18

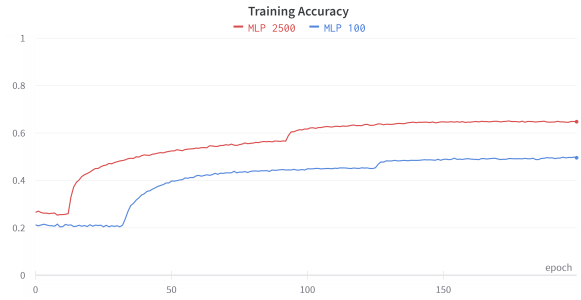

Figure 11: Training accuracy of two MLP models on CIFAR-10 datasets

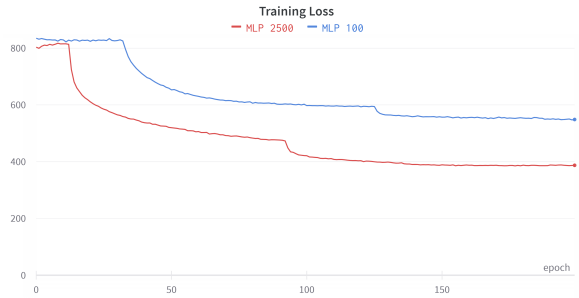

Figure 12: Training loss of two MLP models on CIFAR-10 datasets

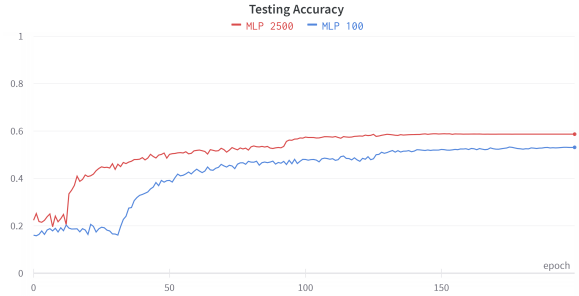

Figure 13: Testing accuracy of two MLP models on CIFAR-10 datasets

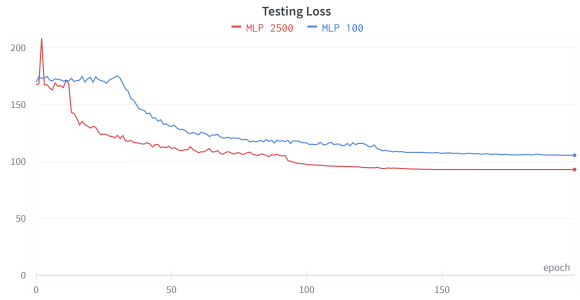

Figure 14: Testing loss of two MLP models on CIFAR-10 datasets

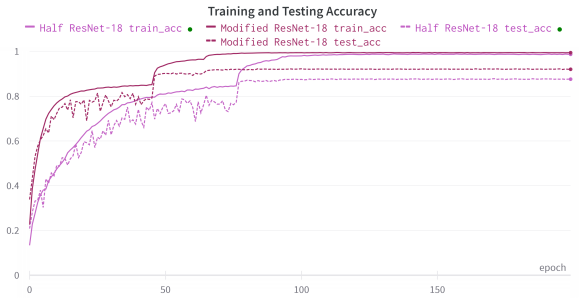

Figure 15: Training and testing accuracy of ResNet-18 with different training data amount. The "Half-ResNet-18" is trained with only 25000 training images.

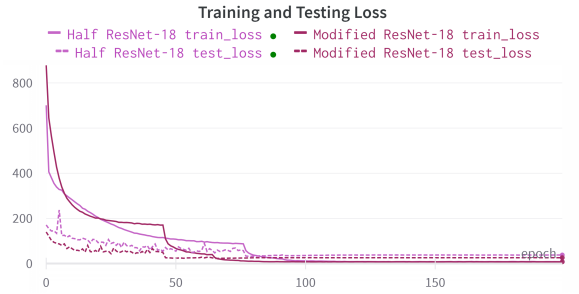

Figure 16: Training and testing loss of ResNet-18 with different training data amount. The "Half-ResNet-18" is trained with only 25000 training images.

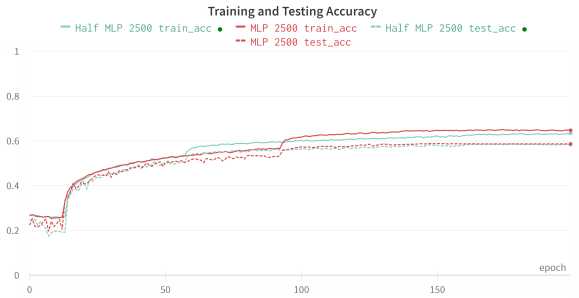

Figure 17: Training and testing accuracy of MLP with different training data amount. The "Half MLP" is trained with only 25000 training images.

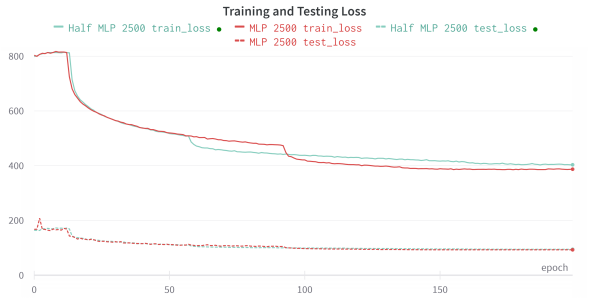

Figure 18: Training and testing accuracy of MLP with different training data amount. The "Half MLP" is trained with only 25000 training images.

A.2 Public Non-image Datasets: 20 Newsgroups

Model AUC 50.75%, Accuracy 74.19% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.60	0.65	0.62	295
comp.graphics	0.70	0.60	0.65	457
comp.os.ms-windows.misc	0.74	0.69	0.71	425
comp.sys.ibm.pc.hardware	0.68	0.68	0.68	397
comp.sys.mac.hardware	0.73	0.71	0.72	392
comp.windows.x	0.68	0.78	0.73	348
misc.forsale	0.88	0.68	0.77	505
rec.autos	0.79	0.79	0.79	394
rec.motorcycles	0.89	0.90	0.90	393
rec.sport.baseball	0.89	0.76	0.82	466
rec.sport.hockey	0.91	0.88	0.90	412
sci.crypt	0.87	0.85	0.86	406
sci.electronics	0.52	0.65	0.58	315
sci.med	0.66	0.80	0.73	327
sci.space	0.86	0.78	0.82	438
soc.religion.christian	0.90	0.64	0.75	564
talk.politics.guns	0.79	0.63	0.70	457
talk.politics.mideast	0.78	0.94	0.85	310
talk.politics.misc	0.47	0.85	0.60	171
talk.religion.misc	0.20	0.82	0.32	60
accuracy			0.74	7532
macro avg	0.73	0.75	0.72	7532
weighted avg	0.77	0.74	0.75	7532

Figure 19: The test accuracy of each classes of the random forest classifier with 500 tress.

Table 1: Overall Comparision of Models Trained on CIFAR-10 Datasets

Model	Feature	Test Accuracy
ResNet-18	w/ pretrained weights	0.9177
	w/o pretrained weights	**0.9227**
	w/o pretrained weights and half training size	0.8782
MLP	2500 hidden neurons	**0.5886**
	100 hidden neurons	0.5324
	2500 hidden neurons and half training size	0.5816
ViT-H/14		**0.995**

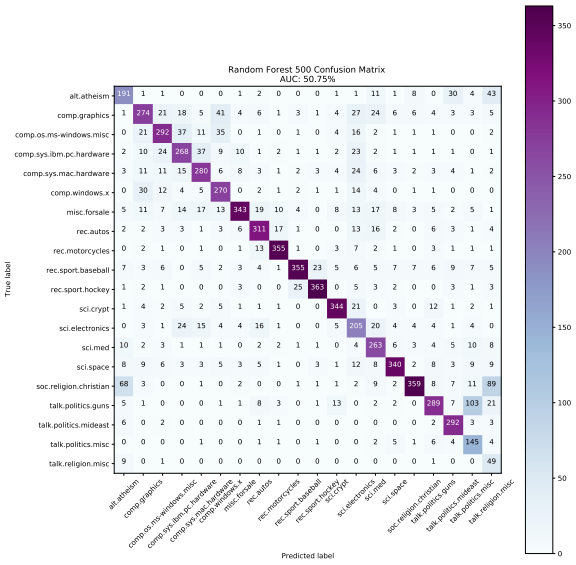

Figure 20: The confusion matrix of each class of the random forest classifier with 500 tress.

Model AUC 50.39%, Accuracy 74.65% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.59	0.65	0.62	292
comp.graphics	0.71	0.60	0.65	463
comp.os.ms-windows.misc	0.75	0.70	0.72	421
comp.sys.ibm.pc.hardware	0.70	0.66	0.68	413
comp.sys.mac.hardware	0.76	0.74	0.75	399
comp.windows.x	0.69	0.80	0.74	342
misc.forsale	0.86	0.69	0.76	487
rec.autos	0.79	0.82	0.80	380
rec.motorcycles	0.89	0.90	0.90	394
rec.sport.baseball	0.88	0.77	0.82	455
rec.sport.hockey	0.93	0.87	0.90	425
sci.crypt	0.87	0.83	0.85	416
sci.electronics	0.53	0.66	0.59	317
sci.med	0.67	0.80	0.73	335
sci.space	0.86	0.78	0.82	434
soc.religion.christian	0.90	0.64	0.75	564
talk.politics.guns	0.80	0.64	0.71	457
talk.politics.mideast	0.76	0.93	0.84	306
talk.politics.misc	0.47	0.86	0.61	169
talk.religion.misc	0.22	0.86	0.34	63
accuracy			0.75	7532
macro avg	0.73	0.76	0.73	7532
weighted avg	0.78	0.75	0.75	7532

Figure 21: The test accuracy of each class of the random forest classifier with 1000 trees.

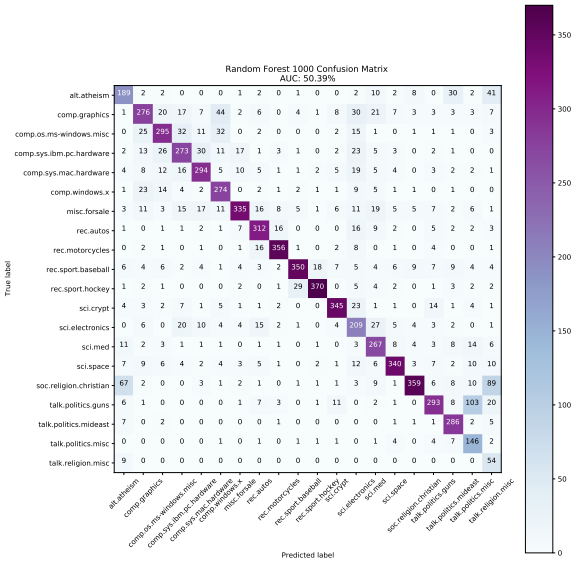

Figure 22: The confusion matrix of each class of the random forest classifier with 1000 tress.

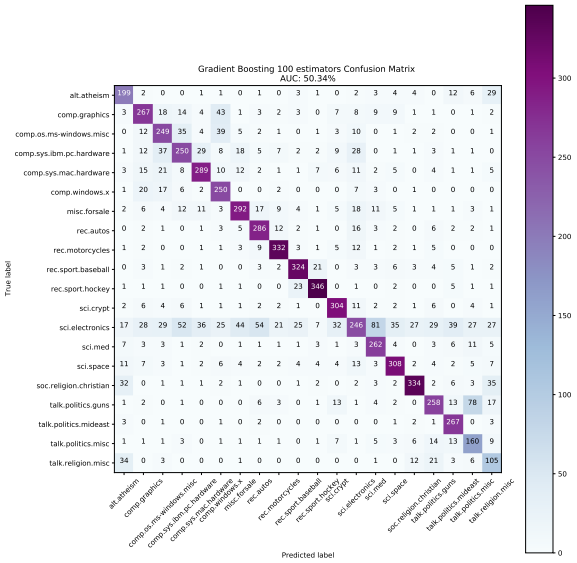

Figure 24: The confusion matrix of each classof the gradient boosting classifier with 100 estimators.

Model AUC 50.34%, Accuracy 70.74% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.62	0.74	0.68	268
comp.graphics	0.69	0.67	0.68	396
comp.os.ms-windows.misc	0.63	0.68	0.65	367
comp.sys.ibm.pc.hardware	0.64	0.60	0.62	415
comp.sys.mac.hardware	0.75	0.72	0.73	404
comp.windows.x	0.63	0.81	0.71	309
misc.forsale	0.75	0.72	0.73	407
rec.autos	0.72	0.83	0.77	345
rec.motorcycles	0.83	0.88	0.85	379
rec.sport.baseball	0.82	0.84	0.83	384
rec.sport.hockey	0.87	0.90	0.88	384
sci.crypt	0.77	0.85	0.81	357
sci.electronics	0.63	0.28	0.39	881
sci.med	0.66	0.82	0.73	318
sci.space	0.78	0.78	0.78	394
soc.religion.christian	0.84	0.78	0.81	428
talk.politics.guns	0.71	0.65	0.68	400
talk.politics.mideast	0.71	0.95	0.81	281
talk.politics.misc	0.52	0.70	0.59	229
talk.religion.misc	0.42	0.56	0.48	186
accuracy			0.71	7532
macro avg	0.70	0.74	0.71	7532
weighted avg	0.71	0.71	0.70	7532

Figure 23: The test accuracy of each class of the gradient boosting classifier with 100 estimators.

Model AUC 50.58%, Accuracy 69.29% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.59	0.76	0.66	247
comp.graphics	0.65	0.67	0.66	377
comp.os.ms-windows.misc	0.62	0.67	0.65	366
comp.sys.ibm.pc.hardware	0.64	0.61	0.62	412
comp.sys.mac.hardware	0.73	0.71	0.72	398
comp.windows.x	0.61	0.79	0.68	303
misc.forsale	0.75	0.73	0.74	397
rec.autos	0.69	0.83	0.75	331
rec.motorcycles	0.82	0.88	0.85	369
rec.sport.baseball	0.80	0.83	0.82	380
rec.sport.hockey	0.86	0.87	0.87	395
sci.crypt	0.77	0.86	0.81	352
sci.electronics	0.65	0.25	0.36	1011
sci.med	0.60	0.80	0.69	296
sci.space	0.76	0.80	0.78	376
soc.religion.christian	0.84	0.75	0.79	446
talk.politics.guns	0.68	0.64	0.66	390
talk.politics.mideast	0.72	0.95	0.82	282
talk.politics.misc	0.50	0.71	0.59	220
talk.religion.misc	0.43	0.58	0.49	184
accuracy			0.69	7532
macro avg	0.68	0.73	0.70	7532
weighted avg	0.70	0.69	0.68	7532

Figure 25: The test accuracy of each class of the gradient boosting classifier with 50 estimators.

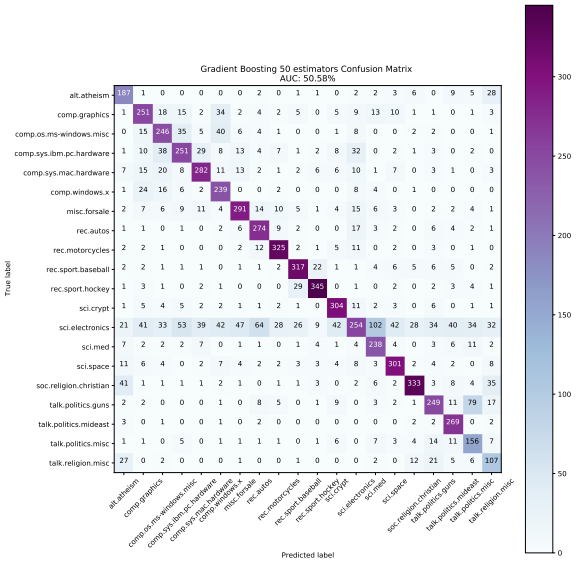

Figure 26: The confusion matrix of each class of the gradient boosting classifier with 50 estimators.

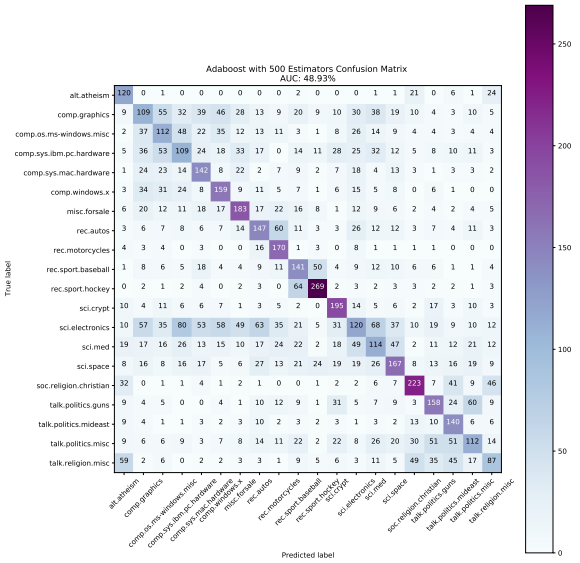

Figure 28: The confusion matrix of each class of the Adaboost classifier with 500 estimators.

Model AUC 48.93%, Accuracy 39.52% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.38	0.68	0.48	177
comp.graphics	0.28	0.22	0.25	498
comp.os.ms-windows.misc	0.28	0.30	0.29	372
comp.sys.ibm.pc.hardware	0.28	0.24	0.26	454
comp.sys.mac.hardware	0.37	0.46	0.41	308
comp.windows.x	0.40	0.48	0.44	333
misc.forsale	0.47	0.49	0.48	375
rec.autos	0.37	0.42	0.39	353
rec.motorcycles	0.43	0.79	0.55	216
rec.sport.baseball	0.36	0.45	0.40	310
rec.sport.hockey	0.67	0.73	0.70	368
sci.crypt	0.49	0.63	0.55	310
sci.electronics	0.31	0.15	0.20	782
sci.med	0.29	0.24	0.26	467
sci.space	0.42	0.37	0.39	457
soc.religion.christian	0.56	0.58	0.57	386
talk.politics.guns	0.43	0.44	0.44	361
talk.politics.mideast	0.37	0.62	0.47	224
talk.politics.misc	0.36	0.26	0.30	431
talk.religion.misc	0.35	0.25	0.29	350
accuracy			0.40	7532
macro avg	0.39	0.44	0.41	7532
weighted avg	0.38	0.40	0.38	7532

Figure 27: The test accuracy of each class of the Adaboost classifier with 500 estimators.

Model AUC 50.60%, Accuracy 46.52% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.31	0.47	0.37	208
comp.graphics	0.24	0.61	0.34	152
comp.os.ms-windows.misc	0.46	0.66	0.55	276
comp.sys.ibm.pc.hardware	0.15	0.61	0.23	94
comp.sys.mac.hardware	0.51	0.61	0.55	319
comp.windows.x	0.47	0.72	0.57	256
misc.forsale	0.35	0.77	0.49	179
rec.autos	0.53	0.81	0.64	262
rec.motorcycles	0.71	0.86	0.78	330
rec.sport.baseball	0.51	0.59	0.55	340
rec.sport.hockey	0.56	0.91	0.69	245
sci.crypt	0.57	0.78	0.66	291
sci.electronics	0.74	0.10	0.18	2809
sci.med	0.23	0.79	0.36	115
sci.space	0.54	0.70	0.61	304
soc.religion.christian	0.66	0.67	0.67	392
talk.politics.guns	0.54	0.51	0.53	386
talk.politics.mideast	0.56	0.93	0.70	228
talk.politics.misc	0.39	0.43	0.41	280
talk.religion.misc	0.10	0.38	0.16	66
accuracy			0.47	7532
macro avg	0.46	0.65	0.50	7532
weighted avg	0.59	0.47	0.42	7532

Figure 29: The test accuracy of each class of Adaboost with 50 estimators.

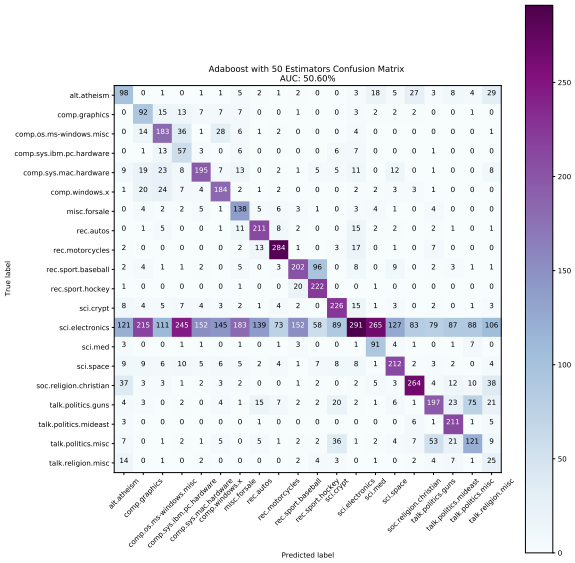

Figure 30: The confusion matrix of each class of the gradient boosting classifier with 50 estimators.

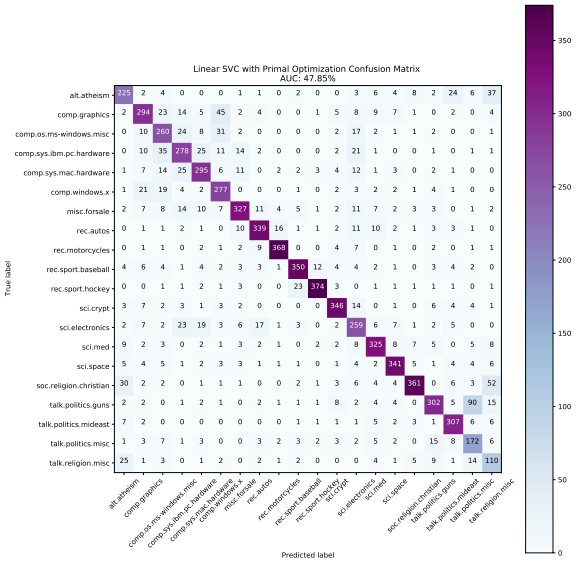

Figure 32: The confusion matrix of each class of the linear SVM classifier with primal optimization.

Model AUC 47.85%, Accuracy 78.47% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.71	0.69	0.70	325
comp.graphics	0.76	0.69	0.72	426
comp.os.ms-windows.misc	0.66	0.72	0.69	361
comp.sys.ibm.pc.hardware	0.71	0.69	0.70	402
comp.sys.mac.hardware	0.77	0.76	0.76	389
comp.windows.x	0.70	0.81	0.75	340
misc.forsale	0.84	0.77	0.80	427
rec.autos	0.86	0.84	0.85	405
rec.motorcycles	0.92	0.92	0.92	400
rec.sport.baseball	0.88	0.85	0.87	410
rec.sport.hockey	0.94	0.90	0.92	414
sci.crypt	0.87	0.87	0.87	397
sci.electronics	0.66	0.71	0.68	365
sci.med	0.82	0.83	0.82	393
sci.space	0.87	0.87	0.87	394
soc.religion.christian	0.91	0.75	0.82	479
talk.politics.guns	0.83	0.68	0.75	443
talk.politics.mideast	0.82	0.90	0.85	343
talk.politics.misc	0.55	0.72	0.63	238
talk.religion.misc	0.44	0.61	0.51	181
accuracy			0.78	7532
macro avg	0.78	0.78	0.77	7532
weighted avg	0.79	0.78	0.79	7532

Figure 31: The test accuracy of each class of the linear SVM classifier with primal optimization.

Model AUC 47.85%, Accuracy 78.47% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.71	0.69	0.70	325
comp.graphics	0.76	0.69	0.72	426
comp.os.ms-windows.misc	0.66	0.72	0.69	361
comp.sys.ibm.pc.hardware	0.71	0.69	0.70	402
comp.sys.mac.hardware	0.77	0.76	0.76	389
comp.windows.x	0.70	0.81	0.75	340
misc.forsale	0.84	0.77	0.80	427
rec.autos	0.86	0.84	0.85	405
rec.motorcycles	0.92	0.92	0.92	400
rec.sport.baseball	0.88	0.85	0.87	410
rec.sport.hockey	0.94	0.90	0.92	414
sci.crypt	0.87	0.87	0.87	397
sci.electronics	0.66	0.71	0.68	365
sci.med	0.82	0.83	0.82	393
sci.space	0.87	0.87	0.87	394
soc.religion.christian	0.91	0.75	0.82	479
talk.politics.guns	0.83	0.68	0.75	443
talk.politics.mideast	0.82	0.90	0.85	343
talk.politics.misc	0.55	0.72	0.63	238
talk.religion.misc	0.44	0.61	0.51	181
accuracy			0.78	7532
macro avg	0.78	0.78	0.77	7532
weighted avg	0.79	0.78	0.79	7532

Figure 33: The test accuracy of each class of the linear SVM classifier with dual optimization.

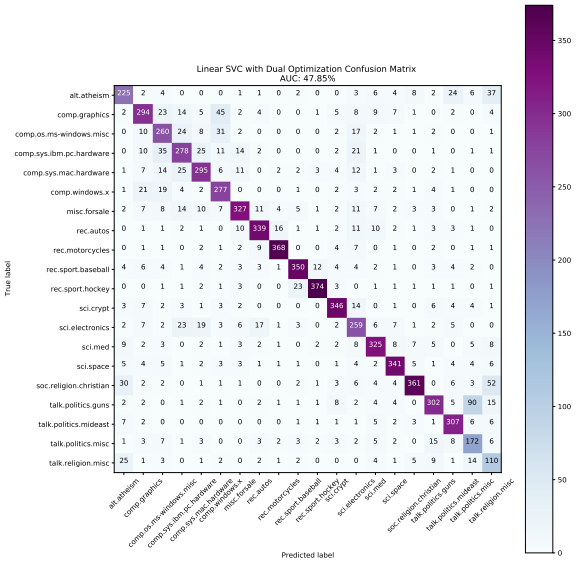

Figure 34: The confusion matrix of each class of the linear SVM classifier with dual optimization.

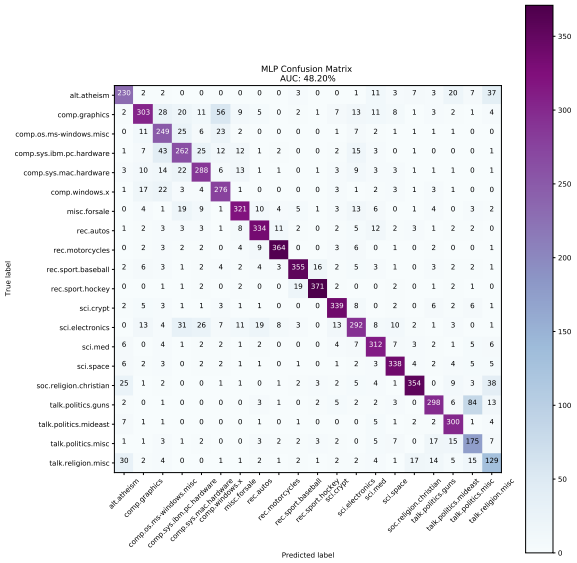

Figure 36: The confusion matrix of each class of the MLP classifier with 100 hidden neurons.

Model AUC 48.20%, Accuracy 78.20% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.72	0.71	0.71	326
comp.graphics	0.78	0.62	0.69	487
comp.os.ms-windows.misc	0.63	0.75	0.69	330
comp.sys.ibm.pc.hardware	0.67	0.68	0.67	387
comp.sys.mac.hardware	0.75	0.76	0.75	381
comp.windows.x	0.70	0.82	0.75	338
misc.forsale	0.82	0.79	0.81	407
rec.autos	0.84	0.84	0.84	397
rec.motorcycles	0.91	0.91	0.91	399
rec.sport.baseball	0.89	0.85	0.87	417
rec.sport.hockey	0.93	0.92	0.92	404
sci.crypt	0.86	0.89	0.87	381
sci.electronics	0.74	0.65	0.69	452
sci.med	0.79	0.85	0.82	365
sci.space	0.86	0.88	0.87	382
soc.religion.christian	0.89	0.78	0.83	452
talk.politics.guns	0.82	0.71	0.76	422
talk.politics.mideast	0.80	0.92	0.85	326
talk.politics.misc	0.56	0.71	0.63	246
talk.religion.misc	0.51	0.55	0.53	233
accuracy			0.78	7532
macro avg	0.77	0.78	0.77	7532
weighted avg	0.79	0.78	0.78	7532

Figure 35: The test accuracy of each class of the MLP classifier with 100 hidden neurons.

Model AUC 48.32%, Accuracy 78.57% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.74	0.70	0.72	337
comp.graphics	0.76	0.67	0.71	439
comp.os.ms-windows.misc	0.64	0.71	0.67	356
comp.sys.ibm.pc.hardware	0.71	0.65	0.68	427
comp.sys.mac.hardware	0.77	0.70	0.73	423
comp.windows.x	0.69	0.82	0.75	333
misc.forsale	0.85	0.79	0.82	421
rec.autos	0.87	0.82	0.84	421
rec.motorcycles	0.90	0.91	0.91	394
rec.sport.baseball	0.89	0.87	0.88	409
rec.sport.hockey	0.93	0.92	0.93	402
sci.crypt	0.87	0.88	0.88	391
sci.electronics	0.70	0.75	0.72	367
sci.med	0.79	0.86	0.82	366
sci.space	0.85	0.88	0.86	377
soc.religion.christian	0.88	0.80	0.84	437
talk.politics.guns	0.81	0.71	0.76	417
talk.politics.mideast	0.81	0.93	0.87	329
talk.politics.misc	0.56	0.70	0.62	249
talk.religion.misc	0.53	0.56	0.54	237
accuracy			0.79	7532
macro avg	0.78	0.78	0.78	7532
weighted avg	0.79	0.79	0.79	7532

Figure 37: The test accuracy of each class of the MLP classifier with 50 hidden neurons.

Model AUC 48.83%, Accuracy 77.67% on Test Data

	precision	recall	f1-score	support
alt.atheism	0.70	0.71	0.70	315
comp.graphics	0.74	0.65	0.69	441
comp.os.ms-windows.misc	0.67	0.69	0.68	385
comp.sys.ibm.pc.hardware	0.72	0.63	0.67	452
comp.sys.mac.hardware	0.76	0.73	0.75	402
comp.windows.x	0.69	0.83	0.75	328
misc.forsale	0.82	0.73	0.78	437
rec.autos	0.86	0.83	0.84	410
rec.motorcycles	0.91	0.89	0.90	410
rec.sport.baseball	0.90	0.85	0.87	420
rec.sport.hockey	0.91	0.94	0.92	387
sci.crypt	0.84	0.89	0.86	373
sci.electronics	0.67	0.74	0.70	355
sci.med	0.75	0.84	0.79	352
sci.space	0.84	0.89	0.87	370
soc.religion.christian	0.89	0.76	0.82	465
talk.politics.guns	0.82	0.71	0.76	419
talk.politics.mideast	0.82	0.92	0.87	337
talk.politics.misc	0.59	0.69	0.64	262
talk.religion.misc	0.48	0.57	0.52	212
accuracy			0.78	7532
macro avg	0.77	0.77	0.77	7532
weighted avg	0.78	0.78	0.78	7532

Figure 39: The test accuracy of each class of the MLP classifier with 50 hidden neurons and half training data size.

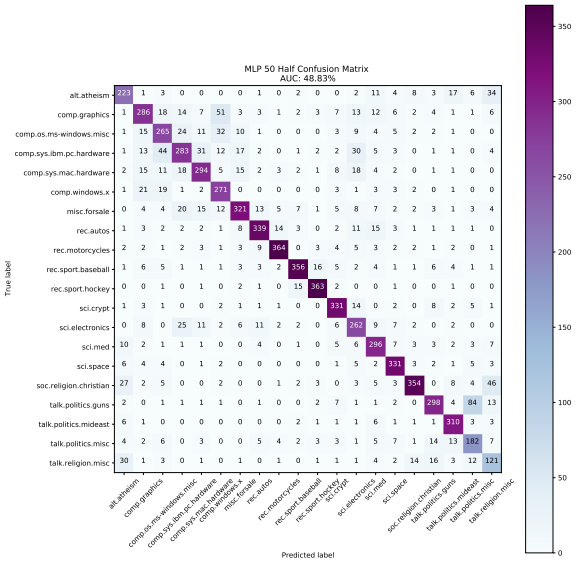

Figure 40: The confusion matrix of each class of the MLP classifier with 50 hidden neurons and half training data size.

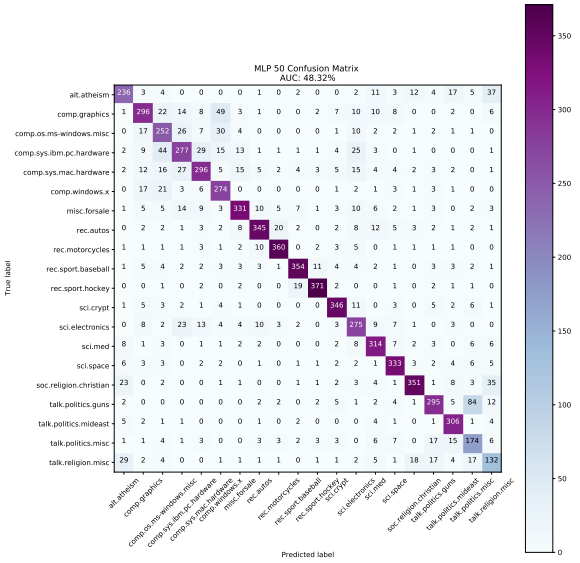

Figure 38: The confusion matrix of each class of the MLP classifier with 50 hidden neurons.

A.3 Self-made Datasets: Satellite Images of 5 Regions

Table 3: The 5-fold test accuracy for ResNet-18 and MLP. Can see that MLP has better test accuracy.

Model	Fold	Test Accuracy	Mean Accuracy
MLP	1	1.0	**0.98032**
	2	1.0	
	3	0.9094	
	4	0.9969	
	5	0.9953	
ResNet-18	1	0.8375	0.93186
	2	0.9703	
	3	0.9703	
	4	1.0	
	5	0.8812	

Appendix B Some Note about the Python Scripts

Because I'm using jupyter notebook for training and testing, the code below is actually the content of a jupyter notebook. Therefore, it can't be directly execute as an python code. Each `#In[...]:` is the beginning of a cell. Each cell should be execute independently and the

Table 2: Overall Comparision of Models Trained on 20 Newsgroups Datasets

Model	Feature	Test Accuracy	AUC	Precision	Recall	F1-Score
Random Forest	500 esimators	0.7419	**0.5075**	0.73	0.75	0.72
	1000 estimators	0.7465	0.5039	0.73	0.76	0.73
Gradient Boosting	50 estimators	0.6929	0.5058	0.68	0.73	0.70
	100 estimators	0.7074	0.5034	0.70	0.74	0.71
Adaboost	50 esimators	0.4652	0.5060	0.46	0.65	0.50
	500 esimators	0.3952	0.4893	0.39	0.44	0.41
Linear SVM	Primal Optimization	0.7847	0.4785	**0.78**	**0.78**	0.77
	Dual Optimization	0.7847	0.4785	**0.78**	**0.78**	0.77
MLP	50 neurons	**0.7857**	0.4832	**0.78**	**0.78**	**0.78**
	50 neurons w/ half training data	0.7767	0.4883	0.77	0.77	0.77
	100 neurons	0.7820	0.4820	0.77	0.78	0.77

execution order of cell is not determine by the order.

Appendix C Code of CIFAR-10

Code 1: Training and Testing of CIFAR-10

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn

import torchvision as tv
import torchvision.transforms as transforms

import wandb

import os
import numpy as np

from tqdm import *


# In[2]:


wandb.login()


# In[3]:


lr = 0.1
best_acc = 0.0
start_epoch = 0


# In[ ]:




# In[4]:


device =
↪  'cuda' if torch.cuda.is_available() else 'cpu'


# In[5]:


# Transforms

transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914,
    ↪  0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914,
    ↪  0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])


# In[6]:


# Datasets
train_dataset = tv.datasets.CIFAR10(root='./
↪  datasets', train=True,
↪  transform=transform_train, download=True)
```

```
train_dataset_half = torch.utils.data.random_split(train_dataset, [25000, 25000])[0]
test_dataset = tv.datasets.CIFAR10(root='./datasets', train=False, transform=transform_test, download=True)

# Data Loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=128, shuffle=True, num_workers=35)
train_loader_half = torch.utils.data.DataLoader(dataset=train_dataset_half, batch_size=128, shuffle=True, num_workers=35)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=128, shuffle=False, num_workers=35)


# In[8]:


print(len(train_dataset))
print(len(test_dataset))
print(len(train_dataset_half))


# In[9]:


# Define Model
net = tv.models.resnet18(pretrained=True)
net.fc = nn.Linear(in_features=512, out_features=10, bias=True)

net.conv1 = nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(3, 3), bias=False)
net.maxpool = nn.Identity()


# In[26]:


class MLP(nn.Module):
    def __init__(self, n_hidden_nodes):
        super(MLP, self).__init__()
        self.n_hidden_nodes = n_hidden_nodes
        # Set up perceptron layers and add dropout
        self.fc1 = nn.Linear(3 * 32 * 32,
                             n_hidden_nodes)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(n_hidden_nodes, n_hidden_nodes)
        self.fc2_drop = nn.Dropout(0.2)
        self.out = nn.Linear(n_hidden_nodes, 10)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc1_drop(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc2_drop(x)
        return self.out(x)
```

```
net = MLP(100)


# In[10]:


net = net.to(device)


# In[11]:


lr = 0.1
best_acc = 0.0
start_epoch = 0

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=lr,
                      momentum=0.9, weight_decay=5e-4, nesterov=True)
# scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.1, patience=10,
                                                       threshold=0.001, mode="max")

wandb.init(
    project="AI Capstone Project 1",
    config = {
        "lr": 0.1,
        "arch": "resnet18",
        "dataset": "CIFAR-10",
        "epochs": 200,
        "patient": 10,
        "pretrained": True,
        "training data": 25000
    }
)

# Training
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in tqdm(enumerate(train_loader_half)):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
```

```
        correct
        ↪  += predicted.eq(targets).sum().item()

    wandb.log({
        "epoch": epoch,
        "train_loss": train_loss,
        "train_acc": 1.*correct/total,
        "lr": scheduler.optimizer.param_groups[0]
        ↪  ['lr'],
    })



def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets)
        ↪  in tqdm(enumerate(test_loader)):
            inputs, targets =
            ↪  inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct
            ↪  += predicted.eq(targets).sum().item()

#
↪  progress_bar(batch_idx, len(testloader),
↪  'Loss: %.3f | Acc: %.3f%% (%d/%d)'
#                     % (test_loss/(batch_idx+1),
↪  100.*correct/total, correct, total))

    # Save checkpoint.
    acc = 1.*correct/total
    wandb.log({
        "epoch": epoch,
        "test_loss": test_loss,
        "test_acc": acc,
    })
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, './checkpoint/ckpt.pth')
        wandb.save('./checkpoint/ckpt.pth')
        best_acc = acc
    scheduler.step(acc)


for epoch in range(start_epoch, start_epoch+200):
    train(epoch)
    test(epoch)
```

```
    print(best_acc)



# In[30]:


wandb.finish()


# In[16]:


best_model = wandb.restore('checkpoint/
↪  ckpt.pth', run_path="jayinnn/
↪  AI Capstone Project 1/1zzr5u3j")
print(best_model)


# In[26]:


net = net.to("cpu")
net.load_state_dict(torch.load("checkpoint/
↪  half-resnet.pth")["net"])
net.eval()


# In[97]:


from sklearn import metrics
from sklearn.metrics import *
import matplotlib.pyplot as plt
import itertools

def plot_training_metrics(model,
↪  y_actual, y_pred, classes, model_name='model'):
    """
    Input: trained
↪  model history, model, test image generator,
↪  actual and predicted labels, class list
    Output: Plots loss
↪  vs epochs, accuracy vs epochs, confusion matrix
    """
    fpr, tpr, thresholds = metrics
    ↪  .roc_curve(y_actual, y_pred, pos_label=9)
    AUC      = metrics.auc(fpr, tpr)*100
    Acc      = metrics.accuracy_score(y_actual,
    ↪  y_pred)*100
    results_title =(f"\n Model AUC {AUC:.2f}%,
    ↪  Accuracy {Acc:.2f}% on Test Data\n")
    print(results_title.format(AUC, Acc))


    # print classification report
    print(classification_report(y_actual,
    ↪  y_pred, target_names=classes))


    plt.subplots(figsize=(12,12))
```

```
    # calculate Confusion Matrix
    cm = confusion_matrix(y_actual, y_pred)

    # create confusion matrix plot
    plt.imshow(cm,
    ↪  interpolation='nearest', cmap=plt.cm.BuPu)
    plt.title(f"{model_name}
    ↪  Confusion Matrix \nAUC: {AUC:.2f}%")
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    # loop through matrix, plot each
    threshold = cm.max() / 2.
    for r, c in itertools.product(range(cm
    ↪  .shape[0]), range(cm.shape[1])):
        plt.text(c, r, format(cm[r, c], 'd'),
                horizontalalignment="center",
                color="white" if
                ↪  cm[r, c] > threshold else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.savefig(f"{model_name}.pdf")

    plt.show()

def accuracy(predicted, actual):
    _, predictions = torch.max(predicted, dim=1)
    return torch
    ↪  .tensor(torch.sum(predictions==actual)
    ↪  .item()/len(predictions))

@torch.no_grad()
def get_all(model, loader):
    all_preds = torch.tensor([])
    all_truths = torch.tensor([])
    for batch in loader:
        images, labels = batch

        preds = model(images)
        all_preds
        ↪  = torch.cat((all_preds, preds) ,dim=0)
        all_truths
        ↪  = torch.cat((all_truths, labels), dim=0)

    return all_preds, all_truths


# In[91]:


net = net.to("cpu")
net.load_state_dict(torch.load("checkpoint/
↪  resnet18-cifar.pth")["net"])
net.eval()


# In[92]:


y_pred, y_truth = get_all(net, test_loader)

```

```

# In[63]:


classes = '''airplane automobile bird
↪  cat deer dog frog horse ship truck'''.split()


# In[93]:


_, y_pred_one = y_pred.max(1)


# In[77]:


print(y_pred_one)


# In[98]:


plot_training_metrics(net, y_truth, y_pred_one,
↪  classes, model_name="Pretrained ResNet-18")
```

Appendix D Code of 20 Newsgroups

Code 2: Training and Testing of 20 Newsgroups

```
#!/usr/bin/env python
# coding: utf-8

# In[107]:


from sklearn.datasets import fetch_20newsgroups,
↪  fetch_20newsgroups_vectorized
from sklearn.linear_model import Perceptron
from
↪  sklearn.ensemble import RandomForestClassifier,
↪  GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier


# In[18]:


train_x, train_y = fetch_20newsgroups_vectorized(
    data_home='./datasets',
    subset="train",
    return_X_y=True,
    remove=("headers")
)


# In[22]:


```

```
test_x, test_y = fetch_20newsgroups_vectorized(
    data_home='./datasets',
    subset="test",
    return_X_y=True,
    remove=("headers")
)


# In[19]:


train_x


# In[118]:


print(len(train_y))
print(len(test_y))


# In[84]:


perc = Perceptron(
    max_iter = 2000,
    n_jobs = 30,
    validation_fraction = 0.1,
    n_iter_no_change = 2000,
)
perc.fit(train_x, train_y)


# In[95]:


plot_training_metrics(perc.predict(test_x),
↪  test_y, classes, model_name="Perceptron")


# In[85]:


rf = RandomForestClassifier(
    n_jobs = 30,
    verbose = 1,
    n_estimators=1000,
)
rf.fit(train_x, train_y)


# In[100]:


rf500 = RandomForestClassifier(
    n_jobs = 30,
    verbose = 1,
    n_estimators=500,
)
rf500.fit(train_x, train_y)


# In[113]:


plot_training_metrics(rf.predict(test_x), test_y,
↪  classes, model_name="Random Forest 1000")


# In[106]:


plot_training_metrics(rf500.predict(test_x),
↪  test_y,
↪  classes, model_name="Random Forest 500")


# In[87]:


gb = GradientBoostingClassifier(
    verbose = 1,
    n_estimators=100,
)
gb.fit(train_x, train_y)


# In[99]:


gb50 = GradientBoostingClassifier(
    verbose = 1,
    n_estimators=50,
)
gb50.fit(train_x, train_y)


# In[116]:


plot_training_metrics(gb50.predict(test_x),
↪  test_y, classes,
↪  model_name="Gradient Boosting 50 estimators")


# In[ ]:




# In[ ]:


mlp = MLPClassifier(
    hidden_layer_sizes=(100),
    verbose=True
)


# In[68]:


mlp.fit(train_x, train_y)


# In[134]:


mlp50 = MLPClassifier(
```

```
    hidden_layer_sizes=(50),
    verbose=True
)
mlp50.fit(train_x, train_y)


# In[141]:


train_x_half, _, train_y_half, _ = sklearn
↪  .model_selection.train_test_split(train_x,
↪  train_y, random_state=42)


# In[139]:


train_x_half


# In[ ]:


mlp50_half = mlp50 = MLPClassifier(
    hidden_layer_sizes=(50),
    verbose=True
)
mlp50_half.fit(train_x_half, train_y_half)


# In[135]:


plot_training_metrics(mlp50_half.predict(test_x),
↪  test_y, classes, model_name="MLP 50 Half")


# In[129]:


from sklearn.svm import LinearSVC

linearsvc = LinearSVC(
    verbose = 1,
    dual=True,
)

linearsvc.fit(train_x, train_y)


# In[126]:


from sklearn.cross_validation
↪  import train_test_split


# In[130]:


plot_training_metrics(linearsvc.predict(test_x),
↪  test_y, classes,
↪  model_name="Linear SVC with Dual Optimization")
```

```
# In[80]:


from sklearn.metrics import f1_score,
↪  confusion_matrix, classification_report


# In[ ]:




# In[111]:


ada = AdaBoostClassifier(
    n_estimators=500
)
ada.fit(train_x, train_y)


# In[119]:


ada50 = AdaBoostClassifier(
    n_estimators=50
)
ada50.fit(train_x, train_y)


# In[133]:


plot_training_metrics(ada50.predict(test_x),
↪  test_y, classes,
↪  model_name="Adaboost with 50 Estimators")


# In[ ]:


model = HistGradientBoostingClassifier()


# In[58]:


model.fit(train_x, train_y)


# In[59]:


model.score(test_x, test_y)


# In[91]:


classes = fetch_20newsgroups_vectorized(
    data_home='./datasets',
    subset="test",
    remove=("headers")
)["target_names"]
```

```

# In[92]:


classes


# In[94]:


from sklearn import metrics
from sklearn.metrics import *
import matplotlib.pyplot as plt
import itertools
import numpy as np

def plot_training_metrics(y_actual, y_pred, classes, model_name='model'):
    """
    Input: trained model history, model, test image generator, actual and predicted labels, class list
    Output: Plots loss vs epochs, accuracy vs epochs, confusion matrix
    """
    fpr, tpr, thresholds = metrics.roc_curve(y_actual, y_pred, pos_label=9)
    AUC      = metrics.auc(fpr, tpr)*100
    Acc      = metrics.accuracy_score(y_actual, y_pred)*100
    results_title =(f"\n Model AUC {AUC:.2f}%, Accuracy {Acc:.2f}% on Test Data\n")
    print(results_title.format(AUC, Acc))


    # print classification report
    print(classification_report(y_actual, y_pred, target_names=classes))



#    # create plots
    plt.subplots(figsize=(12,12))



     # calculate Confusion Matrix
    cm = confusion_matrix(y_actual, y_pred)

    # create confusion matrix plot
#    plt.subplot(1,3,3)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.BuPu)
    plt.title(f"{model_name} Confusion Matrix \nAUC: {AUC:.2f}%")
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    # loop through matrix, plot each
    threshold = cm.max() / 2.
    for r, c in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(c, r, format(cm[r, c], 'd'),
                horizontalalignment="center",
                color="white" if cm[r, c] > threshold else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.savefig(f"{model_name}.pdf")

    plt.show()

def accuracy(predicted, actual):
    _, predictions = torch.max(predicted, dim=1)
    return torch.tensor(torch.sum(predictions==actual).item()/len(predictions))

def get_all(model, loader):
    all_preds = torch.tensor([])
    all_truths = torch.tensor([])
    for batch in loader:
        images, labels = batch

        preds = model(images)
        all_preds
        = torch.cat((all_preds, preds) ,dim=0)
        all_truths
        = torch.cat((all_truths, labels), dim=0)

    return all_preds, all_truths
```

Appendix E Code of Satellite Image Datasets

Code 3: Collecting Satellite Image Datasets

```
import math
import urllib.request
import os
import glob
import subprocess
import shutil
# import requests
from tqdm import tqdm

# from fp.fp import FreeProxy
from tile_convert import bbox_to_xyz, tile_edges
from osgeo import gdal




def download_tile(x, y, z, tile_server):
    url = tile_server.replace(
        "{x}", str(x)).replace(
        "{y}", str(y)).replace(
        "{z}", str(z))
```

```
    path = '{}/{}_{}_{}.jpg'.format(temp_dir, x, y, z)
    if(os.path.isfile(path)):
        return path
    # print("getting proxy")
    # proxies = {"http": FreeProxy(rand=True).get()}
    # proxy = urllib.request.ProxyHandler(pro)
    # opener = urllib.request.build_opener(proxy)
    # urllib.request.install_opener(opener)
    # print(proxies)
    # r = requests.get(url)
    # print(path)
    # print(r.content)
    # with open(path, 'wb') as f:
        # f.write(r.content)
    urllib.request.urlretrieve(url, path)
    return(path)


def merge_tiles(input_pattern, output_path):
    merge_command = ['gdal_merge.py', '-o', output_path]

    for name in glob.glob(input_pattern):
        merge_command.append(name)

    subprocess.call(merge_command)


def georeference_raster_tile(x, y, z, path):
    bounds = tile_edges(x, y, z)
    filename, extension = os.path.splitext(path)
    gdal.Translate(filename + '.tif',
                   path,
                   outputSRS='EPSG:4326',
                   outputBounds=bounds)

def crop(input, output, lat, lon):
    gdal.Translate(
        output,
        input,
        outputSRS='EPSG:4326',
        projWin=[lon, lat+1, lon+1, lat]
    )

taiwan = [
    (23, 120),
    (23, 121),
    (24, 121)
]

china = [
    (26, 101),
    (26, 100),
    (26, 99),
    (26, 98),
    (29, 101),
    (29, 100),
    (29, 99),
    (28, 101),
    (28, 100),
    (28, 99),
    (28, 98),
    (27, 101),
    (27, 100)
]

hima = [
    (29, 81),
    (29, 82),
    (29, 83),
    (28, 83),
    (28, 84),
    (28, 85),
    (27, 85),
    (27, 86),
    (28, 86),
    (27, 87)
]

arge = [
    (-48, -74),
    (-49, -74),
    (-49, -75),
    (-50, -74),
    (-50, -75),
    (-51, -73),
    (-51, -74),
    (-51, -75),
    (-52, -74),
]

cana = [
    (58, -134),
    (57, -133),
    (56, -132),
    (56, -131),
    (55, -131),
]

#---------- CONFIGURATION -----------#
# tile_server = "https://mt0.google.com/vt/lyrs=s&x={x}&y={y}&z={z}"
region = "cana"
tile_server = f"https://api.maptiler.com/tiles/satellite/{z}/{x}/{y}.jpg?key={key}"
temp_dir = os.path.join(os.path.dirname(__file__), 'temp/'+region)
output_dir = os.path.join(os.path.dirname(__file__), 'output/'+region)
zoom = 15

# lon_max = 122
# lat_max = 25
#-----------------------------------#
for lat_min, lon_min in cana:
    print(lat_min, lon_min)

    lat_dir = ("N" if lat_min > 0 else "S")
    lon_dir = ("E" if lon_min > 0 else "W")

    lat_max = lat_min + 1
    lon_max = lon_min + 1
    x_min, x_max, y_min, y_max = bbox_to_xyz(
        lon_min, lon_max, lat_min, lat_max, zoom)

    print("Downloading {} tiles".format((x_max - x_min + 1) * (y_max - y_min + 1)))

```

```
    pbar = tqdm(total=(x_max - x_min + 1) * (y_max - y_min + 1))
    for x in range(x_min, x_max + 1):
        for y in range(y_min, y_max + 1):
            # print("{},{}".format(x, y))
            pbar.update(1)
            pbar.set_description("{},{}".format(x, y))
            png_path = download_tile(x, y, zoom, tile_server)
            georeference_raster_tile(x, y, zoom, png_path)

    pbar.close()
    print("Download complete")

    print("Merging tiles")
    merge_tiles(temp_dir + '/*.tif', output_dir + '/{}{}{}{}_{}.tif'.format(lat_dir, abs(lat_min), lon_dir, abs(lon_min), zoom))
    print("Merge complete")

    print("Croping")

    crop(
        output_dir + '/{}{}{}{}_{}.tif'.format(lat_dir, abs(lat_min), lon_dir, abs(lon_min), zoom),
        output_dir + '{}{}{}{}_{}_final_sat.tif'.format(lat_dir, abs(lat_min), lon_dir, abs(lon_min), zoom),
        lat_min,
        lon_min
    )
    print("Crop complete")


    shutil.rmtree(temp_dir)
    os.makedirs(temp_dir)
```

Code 4: Training and Testing of Terrain Datasets

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn

import torchvision as tv
import torchvision.transforms as transforms

from sklearn.model_selection import KFold

import wandb

import os
import numpy as np


from tqdm import *


# In[6]:


import wandb
wandb.login()


# In[7]:


lr = 0.1
best_acc = 0.0
start_epoch = 0


# In[8]:


device = 'cuda:1' if torch.cuda.is_available() else 'cpu'


# In[53]:


# Transforms

transform_train = transforms.Compose([
    transforms.Resize((128, 128)),
    # transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.2, 0.2, 0.2)),
])

transform_test = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.2, 0.2, 0.2)),
])


# In[54]:


train_dataset = tv.datasets.ImageFolder(root='./datasets/terrain_train', transform=transform_train)
test_dataset = tv.datasets.ImageFolder(root='./datasets/terrain_test', transform=transform_test)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=8, shuffle=True, num_workers=35)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=8, shuffle=False, num_workers=35)

```

```

# In[51]:


datasets = torch.utils.data.ConcatDataset([train_dataset, test_dataset])
datasets_half = torch.utils.data.random_split(datasets, [1600, 1600])[0]


# In[34]:


print(len(datasets))


# In[22]:


print(len(train_dataset))
print(len(test_dataset))


# In[30]:


# Define Model
# net = timm.create_model("hf_hub:edadaltocg/resnet18_cifar10")
net = tv.models.resnet18(pretrained=False)
net.fc = nn.Linear(in_features=512, out_features=5, bias=True)

# net.conv1 = nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(3, 3), bias=False)
# net.maxpool = nn.Identity()
net = net.to(device)


# In[55]:


class MLP(nn.Module):
    def __init__(self, n_hidden_nodes):
        super(MLP, self).__init__()
        self.n_hidden_nodes = n_hidden_nodes
        # Set up perceptron layers and add dropout
        self.fc1 = nn.Linear(3 * 128 * 128,
                             n_hidden_nodes)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(n_hidden_nodes, n_hidden_nodes)
        self.fc2_drop = nn.Dropout(0.2)
        self.out = nn.Linear(n_hidden_nodes, 5)

    def forward(self, x):
        x = x.view(-1, 3 * 128 * 128)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc1_drop(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc2_drop(x)
        return self.out(x)


net = MLP(200)


# In[56]:


net = net.to(device)


# In[24]:


lr = 0.1
best_acc = 0.0
start_epoch = 0

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=lr,
                      momentum=0.9, weight_decay=5e-4, nesterov=True)
# scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.1, patience=10,
                                                       threshold=0.001, mode="max")

wandb.init(
    project="AI Capstone Project 1",
    config = {
        "lr": 0.1,
        "arch": "mlp",
        "dataset": "terrain",
        "hidden_nodes": 1000,
        "epochs": 200,
        "patient": 10,
    }
)


# Training
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in tqdm(enumerate(train_loader)):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
```

```
        correct
        ↪  += predicted.eq(targets).sum().item()

    wandb.log({
        "epoch": epoch,
        "train_loss": train_loss,
        "train_acc": 1.*correct/total,
        "lr": scheduler.optimizer.param_groups[0]
        ↪  ['lr'],
    })


def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets)
        ↪  in tqdm(enumerate(test_loader)):
            inputs, targets =
            ↪  inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct
            ↪  += predicted.eq(targets).sum().item()

#
↪  progress_bar(batch_idx, len(testloader),
↪  'Loss: %.3f | Acc: %.3f%% (%d/%d)'
#                     % (test_loss/(batch_idx+1),
↪  100.*correct/total, correct, total))

    # Save checkpoint.
    acc = 1.*correct/total
    wandb.log({
        "epoch": epoch,
        "test_loss": test_loss,
        "test_acc": acc,
    })
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, './checkpoint/ckpt.pth')
        wandb.save('./checkpoint/ckpt.pth')
        best_acc = acc
    scheduler.step(acc)


for epoch in range(start_epoch, start_epoch+200):
    train(epoch)
    test(epoch)
    print(best_acc)
```

```


# In[31]:


split =
↪  KFold(n_splits=5,shuffle=True,random_state=42)


# In[45]:


criterion = nn.CrossEntropyLoss()
# optimizer = optim.SGD(net.parameters(), lr=lr,
#
↪  momentum=0.9, weight_decay=5e-4, nesterov=True)
optimizer = optim.Adam(net.parameters())
# scheduler = torch.optim.lr_scheduler
↪  .CosineAnnealingLR(optimizer, T_max=200)

scheduler = torch.optim
↪  .lr_scheduler.ReduceLROnPlateau(optimizer,
↪  factor=0.1, patience=10,

                                                ↪  threshold=0
                                                ↪  .001,
                                                ↪  mode="max")

def train(epoch, fold):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs,
    ↪  targets) in tqdm(enumerate(train_loader)):
        inputs, targets
        ↪  = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct
        ↪  += predicted.eq(targets).sum().item()

    wandb.log({
        "epoch": epoch,
        "train_loss": train_loss,
        "train_acc": 1.*correct/total,
        # "lr": scheduler.optimizer.param_groups[0]
        ↪  ['lr'],
    })




def test(epoch, fold):
    global best_acc
    net.eval()
    test_loss = 0
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in tqdm(enumerate(test_loader)):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

#         progress_bar(batch_idx, len(testloader), 'Loss: %.3f | Acc: %.3f%% (%d/%d)'
#                      % (test_loss/(batch_idx+1), 100.*correct/total, correct, total))

    # Save checkpoint.
    acc = 1.*correct/total
    wandb.log({
        "epoch": epoch,
        "test_loss": test_loss,
        "test_acc": acc,
    })
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, f'./checkpoint/hand_{fold}.pth')
        wandb.save(f'./checkpoint/hand_{fold}.pth')
        best_acc = acc
    # scheduler.step(acc)


# In[ ]:


for fold, (train_idx, test_idx) in enumerate(split.split(np.arange(len(datasets)))):
    print(len(train_idx), len(test_idx))
    train_sampler = torch.utils.data.SubsetRandomSampler(train_idx)
    test_sampler = torch.utils.data.SubsetRandomSampler(test_idx)
    train_loader = torch.utils.data.DataLoader(datasets, num_workers=5, sampler=train_sampler, batch_size=128)
    test_loader = torch.utils.data.DataLoader(datasets, num_workers=5, sampler=test_sampler, batch_size=128)
    lr = 0.1
    best_acc = 0.0
    start_epoch = 0


    # optimizer = optim.SGD(net.parameters(), lr=lr,
    #                       momentum=0.9, weight_decay=5e-4, nesterov=True)
    # scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)

    # scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.1, patience=10,
    #                                                        threshold=0.001, mode="max")

    net = tv.models.resnet18(pretrained=False)
    net.fc = nn.Linear(in_features=512, out_features=5, bias=True)

#     net.conv1 = nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(3, 3), bias=False)
#     net.maxpool = nn.Identity()
    net = net.to(device)


     # optimizer
     = optim.SGD(net.parameters(), lr=lr,
    #                       momentum=0.9, weight_decay=5e-4, nesterov=True)
    # scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)

    # scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.1, patience=10,
    #                                                        threshold=0.001, mode="max")
    optimizer = optim.Adam(net.parameters())


    wandb.init(
        project="AI Capstone Project 1",
        config = {
            "lr": 1e-4,
            "arch": "mlp_kfold",
            "dataset": "terrain",
            "epochs": 200,
            "fold": fold+1,
        },
    )
    print(f"========== Fold {fold+1} ==========")


    for epoch in range(start_epoch, start_epoch+60):
        train(epoch, fold)
        test(epoch, fold)
        print(best_acc)

    del train_loader
    del test_loader

    wandb.finish()
```