Image Classification using Machine Learning and Deep Learning

Xuhui Wang

April 2021

Introduction

The image classification is a classical problem in computer vision, image processing and machine learning domains. In this project, I introduce the image classification using different machine learning and deep learning methods. For machine learning, I primarily use SVM, Logistic Regression and Gaussian Naive Bayes based on the features extracted using Histogram of Oriented Gradients (HoG), Hue Histogram and Raw RGB for this purpose. For deep learning, I primarily use convolutional neural networks for this purpose. The dataset for machine learning is selected from the ImageNet database for the classification purpose. The dataset for deep learning is MNIST. I achieved up to about 90% accuracy using machine learning methods and up to 98.5% accuracy using deep learning method. The results show the effectiveness of machine learning and deep learning in this difficult task, image classification.

Machine Learning Methods

Data Preprocessing

(Note that, in this report, the wordings that describe the algorithms and dataset are all from the course material by Dr. Marques and I will give the appropriate citation at the end of each corresponding paragraph. On the other hand, all resulted figures, methods implementation, statistics, analysis, explanations and conclusions are all from my personal work.) We firstly introduce the machine learning methods and the dataset used in this project for image classification. Commonly, different datasets should be handled distinctly. We often need to create our own custom dataloader to correctly retrieve the data and its labels following a given format. For this purpose, we firstly code a custom dataloading scheme here based on a simple dataset of images, selected from ImageNet, from three classes. This dataloder is going to read sub-folders from a root directory and create data and labels structures based on them. The dataloading process also involves the shuffling of the training and testing sets.[1]

After creating the dataloader, we load a complete, unified dataset and randomly partition it into training, validation and testing subsets. Distinct splits of a single dataset into different subsets are the basis of cross-fold validation. The validation subset helps evaluating the performance of the system while the training process takes place without influencing in its parameter's values (as the samples in the training subset are supposed to do via backpropagation).[1]

The dataset we use has three classes, srkw, moose and beaver. For the ease and convenience of the feature extraction, we resize the data image using the dimensions as user-defined. In this part we choose a dimension of [64, 64, 3]. The split preprocessed training set with the size of 80% and test set with the size of 20% are shown in Figure 1.

Figure 1: Some images in training set and test set

Feature Extraction

After the training and test data are correctly loaded and pre-processed, we need to extract meaningful visual features from them. The paragraph that states the important of feature extration come from the course material by Dr. Marques. These features are going to drive machine lenarning-based image classifications tasks. Since such classifiers are trained and tested based on the features extracted, their design and quality (i.e., how well they represent/generalize the data) are paramount in the performance of the system. In this part we primarily work with three generic features (i.e., not specific to a certain target class) and analyze the classification performance based solely on each of them.[1]

Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. For each image, we create a row vector representing the HOG feature. Specifically, we 9-bin HOGs, cells of 8 x 8 pixels and blocks of 3 x 3 cells. Using these settings, we could extracting HOG features from the training dataset and test set respectively.

Hue Histogram

The histogram of a hue image represents the relationship between each hue value and the number of pixels that have the same hue value. For each image, we create a row vector representing the Hue value histogram feature. Specifically, we 16-bin Hue histogram. Using these settings, we could extracting Hue histogram features from the training dataset and test set respectively.

Raw RGB

For this feature, we only need to flatten all the pixel intensities of the images in the training set and test set. For each image, we create a row vector representing the raw RGB feature. Using these settings, we could extracting raw RGB features from the training dataset and test set respectively.

Methods

In this part, we will apply three different machine learning methods as the classifier using the features extracted above and test their performances. The methods include support vector machine, logistic regression and Gaussian naive Bayes.

Note that we are working with a classification task: the independent variables (i.e., features) are going to determine the probability that a discrete value (class) happens. The paragraph that states how classifiers use features come from the course material by Dr. Marques. In the end, the dependent variable (output) can only assume a fixed number of values—the classes of the training data (for our datasets of three classes: SRKW, beaver and moose, "3"). Another common machine learning-based task is regression, where the independent variables result in a continuous value for the dependent variable (output). A common supervised machine learning algorithm used in such cases is the linear regression, where a line is fit to the data to map the values

from the features to the predicted, continuous output.[1]

We mix the three features extracted and three classifiers to generate 9 different combinations firstly, and then we train the different models using the training set.

Evaluation

To assess the performance of different methods and features, there are many standard criterion that could be used here and we choose the following performance metrics:

- Average recall
- Average precision
- Average F1-Score

After the prediction using the different models, we present the performance table as shown in Figure 2.

	AvgR	AvgP	F1
SVM_HOG	0.7520467836257311	0.7601540616246499	0.7560786900766477
SVM_HHist	0.6760233918128655	0.762962962962963	0.7168668533068424
SVM_Raw	0.6947368421052632	0.7154713241669763	0.7049516521842044
NB_HOG	0.8131578947368422	0.8113175094599243	0.8122366596024616
NB_HHist	0.6780701754385965	0.6879668534080299	0.6829826646502974
NB_Raw	0.5070175438596491	0.5040849673202614	0.5055470028039795
LogR_HOG	0.7742690058479532	0.7794117647058822	0.7768318738860414
LogR_HHist	0.623391812865497	0.6335978835978836	0.6284534143678776
LogR_Raw	0.7549707602339181	0.7524658348187759	0.7537162162940896

Figure 2: Accuracy table

From the table, it could be seen that the highest accuracy is achieved by SVM using raw RGB features, and the accuracy by most combinations are above 0.7, which shows that these machine learning methods and features are appropriate to achieve the image classification task.

A confusion matrix could also be used here to visualize the classification results. It demonstrates the predictions of a model while specifying if they were correct or not. The values of such matrix can be used to infer the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN).[1] We only show the confusion matrices for SVM with different features in Figure 3.

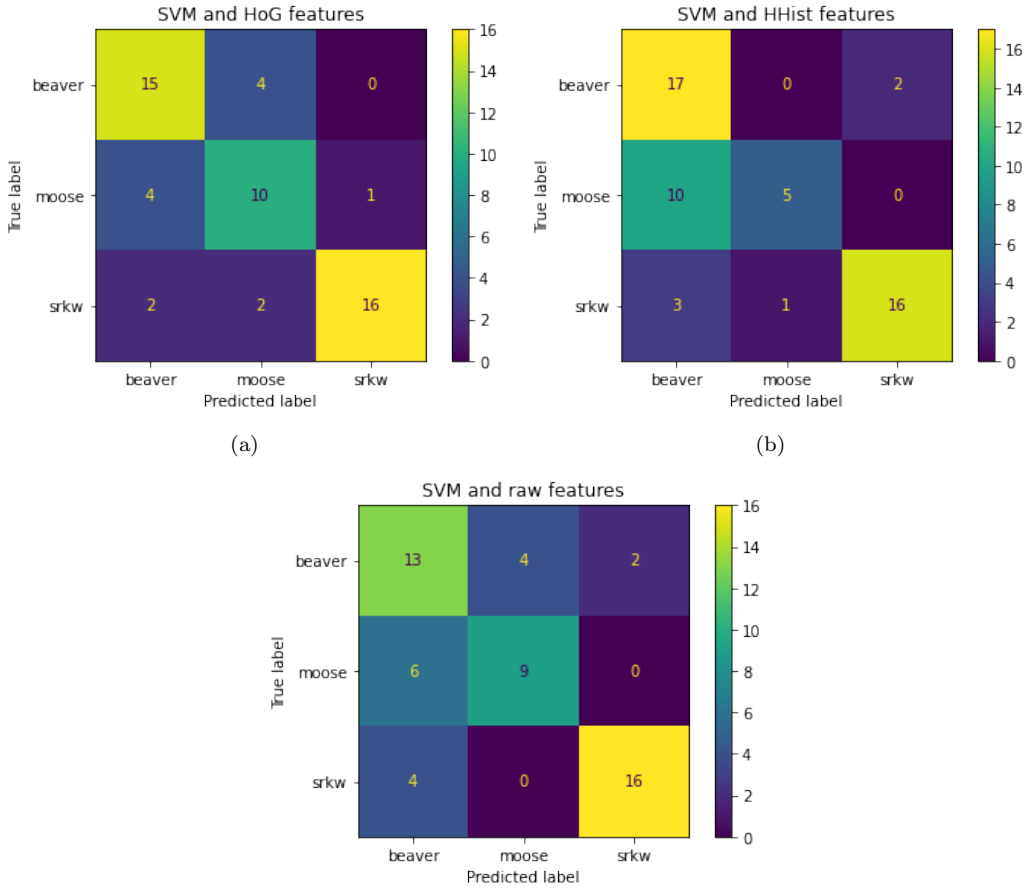


Figure 3: Confusion matrices for SVM with different features

From the above table and confusion matrices, we conclude that SVM is a better method than the other two methods for the image classification task.

Optimization

In this part, to further enhance the accuracy of the image classification, we make some modifications in the process of training and inference and our aim is to reach a layout that surpasses 0.85 of F1-score using any combination of the aforementioned modifications in the training/testing phases. There are a variety of design choices when curating a dataset and training a model. These include, but are not limited to: Size of the input images (if the data is resized), Data standardization: turn the data (either raw of feature-like) into a zero-mean, unit variance set, Data augmentation: modify the input data to train the models using more generic and representative data, Solvers: supervised machine learning algorithms use specific techniques to carry out their optimization problems. Changing these techniques influences in the models' performance.[1]

Now I'm using the combination of 'adjusting the size of input images to the dimension of [130,130,3]' and 'data standardization: turn the data (either raw of feature-like) into a zero-mean, unit variance set' as my first layout to enhance the model precision.

For 'data standardization: turn the data (either raw of feature-like) into a zero-mean, unit variance set' to enhance the model precision, the HoG, HHist and flatten pixel intensity (raw) are all raw information of feature-like and could be directly fed to train a model, so I choose to standardize these features by removing the mean and scaling them to unit variance set, using StandardScalar() from sklearn library.

In the interest of preventing information about the distribution of the test set leaking into the model, I fit the scaler on the training data only, and then standardize both training and test sets with that scaler. After standardizing all 3 kinds of features, I could use them to train and test the different models.

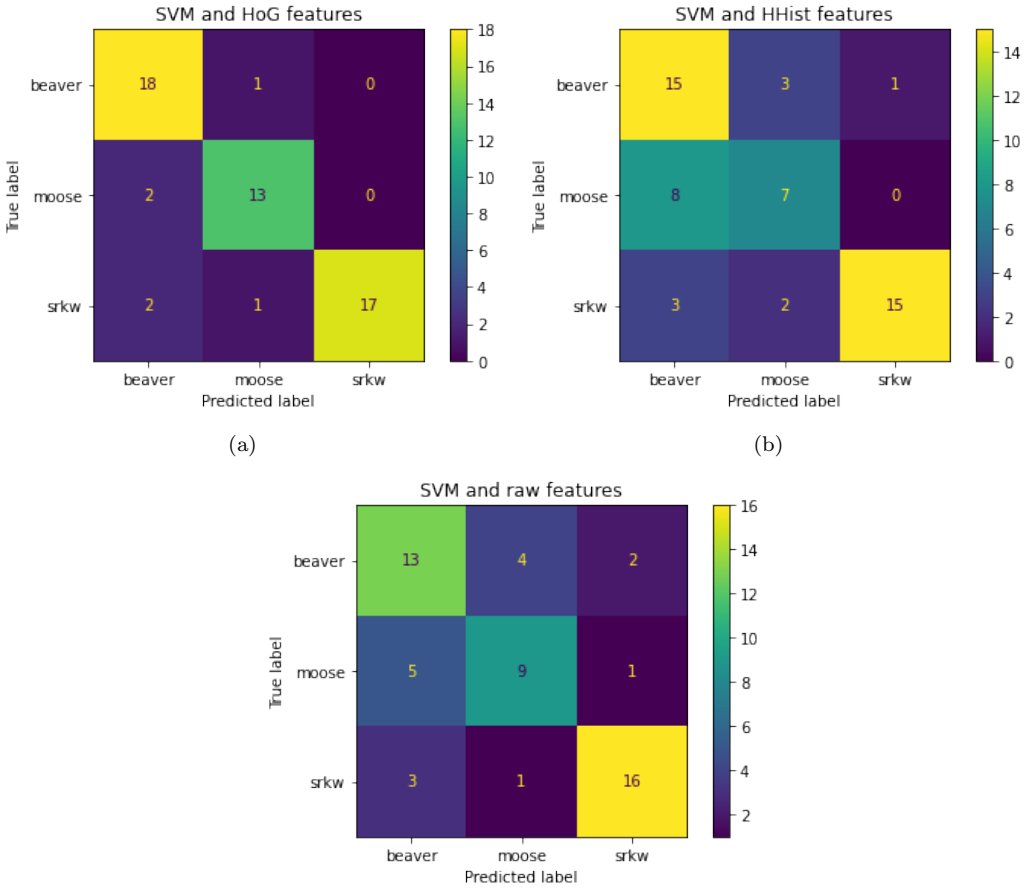
After the model training and inference, again we present the evaluation metrics in Figure 4.

	AvgR	AvgP	F1
SVM_HOG	0.8880116959064327	0.894949494949495	0.8914670973619913
SVM_HHist	0.6687134502923976	0.6992521367521367	0.6836419182182688
SVM_Raw	0.6947368421052632	0.7013366750208855	0.6980211584553508
NB_HOG	0.8011695906432749	0.8019607843137254	0.8015649922394122
NB_HHist	0.7084795321637426	0.7435897435897436	0.7256101653784395
NB_Raw	0.5070175438596491	0.5064814814814814	0.5067493709028547
LogR_HOG	0.8657894736842106	0.8727272727272727	0.8692445300674281
LogR_HHist	0.69093567251462	0.7113095238095237	0.700974587736479
LogR_Raw	0.7391812865497075	0.7421296296296296	0.7406525239565227

Figure 4: Accuracy table

From the table, it could be seen that many of the F1 accuracy values are greater than 0.85, and the highest accuracy, about 0.9, is achieved by SVM using HoG features, which shows that the combination of SVM and HoG features is the best model to achieve the image classification task.

Again, we present the confusion matrices for SVM with different features in Figure 5.



(a) (b)

(c)

Figure 5: Confusion matrices for SVM with different features

Custom Dataset

In order to test the effectiveness of our models, we create and use our own dataset to perform the previous steps and give the performance evaluation. The new dataset has also three classes, panda, dog and human with the similar validation and size settings as the original dataset. The split preprocessed training set with the size of 80% and test set with the size of 20% are shown in Figure 6.

Figure 6: Some images in training set and test set

After the model training and inference, again we present the evaluation metrics in Figure 7.

	AvgR	AvgP	F1
SVM_HOG	0.7593567251461989	0.774270482603816	0.7667410894593851
SVM_HHist	0.6502923976608187	0.7228835978835978	0.6846692770958777
SVM_Raw	0.6994152046783627	0.7139265962795375	0.7065964031117669
NB_HOG	0.7409356725146199	0.7380952380952381	0.7395127278204549
NB_HHist	0.5640350877192982	0.5826210826210826	0.5731774562303983
NB_Raw	0.622514619883041	0.6281440781440781	0.62531667936981
LogR_HOG	0.7982456140350878	0.7985620915032681	0.7984038214072339
LogR_HHist	0.5345029239766083	0.5603508771929825	0.547121783735056
LogR_Raw	0.7216374269005849	0.7527472527472527	0.7368641279001347

Figure 7: Accuracy table

From the table, it could be seen that many of the F1 accuracy values are greater than 0.7, and the highest accuracy, about 0.8, is achieved by logistic regression using HoG features, which again verify the the power of HoG feature in image classification task.

Deep Learning Method

Data Preprocessing

We now introduce the deep learning method and the dataset used in this project for image classification. In this project we are going to work with a real-world dataset called MNIST. This popular dataset of handwritten digits contains 50,000 training samples and 10,000 testing samples.[1]

Downloading and preparing the MNIST dataset in this first segment of this part are asked to download MNIST and create indices to divide it into training, validation and testing. These indices will drive the creation of dataloaders aiming to facilitate the access to batches of these three subsets (an important ability when working with larger datasets). We are going to read the data as Tensors (similar to numpy arrays, but they are more adequate to handle using either the CPU or GPU—as we know, GPUS are the preferred processing unit when training neural networks). Therefore, when reading the dataset, set the parameter "transform" such that the data is read as a Tensor. [1]

We randomly split the dataset, set the size of training set as 70% and the size of test set as 30%. Specifically, we set the minibatch number = 16 (number of images to be considered at a time), and then we present the images in a minibatch in Figure 8.

Figure 8: Some images in dataset

Convolutional Neural Network

We create a small Convolutional Neural Network (CNN) to perform an image classification task on the handwritten digits dataset. Firstly, we need to define the architecture and the forward passes (the backward pass, where backpropagation happens, is automatically implemented by torch.autograd).[1]

In this part, we will define a custom CNN (architecture) and perform an initial analysis on its ability to receive a batch of inputs and create corresponding (untrained) predictions. The architecture we are creating is shown in Figure 9.

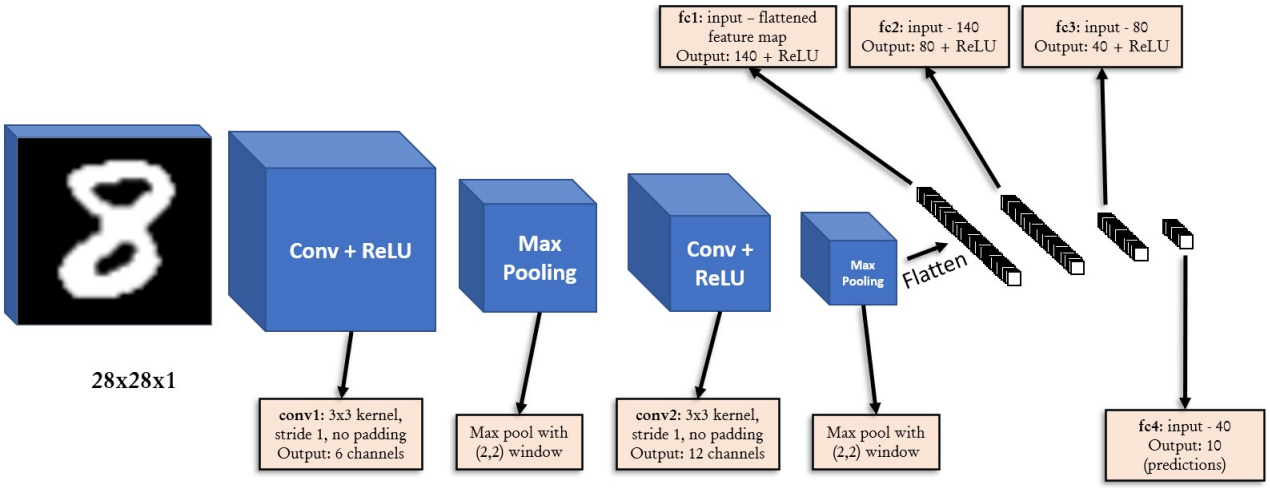


Figure 9: The architecture of the convolutional neural network [1]

Specifically, the architecture could be verified by a summary function as shown in Figure 10.

```
my_network(
  (conv1): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=300, out_features=140, bias=True)
  (fc2): Linear(in_features=140, out_features=80, bias=True)
  (fc3): Linear(in_features=80, out_features=40, bias=True)
  (fc4): Linear(in_features=40, out_features=10, bias=True)
)
Requirement already satisfied: torchsummary in /usr/local/lib/pyt
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [16, 6, 26, 26]              60
            Conv2d-2          [16, 12, 11, 11]             660
            Linear-3                [16, 140]          42,140
            Linear-4                 [16, 80]          11,280
            Linear-5                 [16, 40]           3,240
            Linear-6                 [16, 10]             410
================================================================
Total params: 57,790
Trainable params: 57,790
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.71
Params size (MB): 0.22
Estimated Total Size (MB): 0.97
----------------------------------------------------------------
```

Figure 10: The architecture of the convolutional neural network

Here we make predictions in one minibatch of the training set and present the predictions of the model, in which softmax function is used to present the probabilities associated with the predictions of the model, in Figure 11. It could be seen that the initial test gives the incorrect result for almost every image.

```
The prediction result is: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
The probabilities associated with the predictions of my model for the sample minibatch that I calculated earlier:
[[0.11783971 0.12007307 0.09562321 0.09437446 0.10395323 0.08554069
  0.10088903 0.09632445 0.09660199 0.08878017]
 [0.11744003 0.11975098 0.09554777 0.0941678  0.10395577 0.08562969
  0.10065305 0.0968886  0.09697177 0.08899461]
 [0.11745682 0.12011024 0.09566038 0.09431881 0.1040276  0.08570126
  0.10047814 0.09655456 0.0967668  0.08892544]
 [0.11741453 0.12011831 0.09532753 0.09435272 0.1040073  0.0857067
  0.10078157 0.09678204 0.09674459 0.08876474]
 [0.11764387 0.11988863 0.09565563 0.09422575 0.1038027  0.08569627
  0.100692   0.09662036 0.09691159 0.08886319]
 [0.11742351 0.11986009 0.09565956 0.09411497 0.10384894 0.08592367
  0.10066527 0.09669785 0.09692252 0.08888368]
 [0.11755737 0.1202388  0.09515228 0.09439184 0.10401057 0.0854789
  0.10069787 0.09665779 0.0969049  0.08890967]
 [0.11757977 0.1199368  0.09576576 0.09429996 0.10404833 0.08525237
  0.10085166 0.09644632 0.09679891 0.08902015]
 [0.11762936 0.12023969 0.09567416 0.09428293 0.10417599 0.08558576
  0.10071413 0.09628189 0.09653075 0.08888531]
 [0.11757375 0.11997273 0.09558842 0.09428029 0.103596   0.08567443
  0.10090358 0.0964333  0.09694148 0.08903604]
 [0.11773729 0.11990895 0.09566495 0.09439138 0.10393615 0.08581179
  0.10010415 0.0965042  0.09686264 0.08907853]
 [0.11764913 0.12025577 0.0957425  0.09444641 0.10382351 0.08547263
  0.10058418 0.09637518 0.09684563 0.08880506]
 [0.11747069 0.1199766  0.09519489 0.09435105 0.10391297 0.08569028
  0.1008096  0.09690631 0.0968491  0.08883846]
 [0.11730387 0.11964062 0.09545968 0.09407648 0.10379726 0.08589602
  0.1007701  0.09685644 0.09721712 0.08898231]
 [0.11760707 0.11961965 0.09565797 0.09418272 0.1038381  0.08564749
  0.10071903 0.09656228 0.0969794  0.08918624]
 [0.11742038 0.11976313 0.09564956 0.09404211 0.10384057 0.08588417
  0.10072319 0.0967235  0.09698033 0.08897299]]
```

Figure 11: The initial prediction of the model

Loss Functions

Loss functions are to calculate how close to a given ground truth value a set of predictions are. The losses calculated serve as the basis for backpropagation methods, thus their importance [1]. We present the cross-entropy loss of a set of predictions and also calculate the SVM Multiclass loss for illustration purposes as shown in Figure 12.

```
tensor(2.3198, grad_fn=<NllLossBackward>)
Manual cross-entropy loss: 2.3197806719932377
Reference cross-entropy loss: 2.3197805881500244
Multiclass SVM Loss:0.9117947816848755
```

Figure 12: cross-entropy loss of a set of predictions and the SVM Multiclass loss

Training

Togheter with the creation of a network, the training routine represents the most important aspect of a DL-based image classification framework [1]. In this part, we implement a training loop that predicts scores from batches of samples from the training set, and based on the losses (and gradients) calculated, updates the values of the network's parameters.

Here we use ADAM optimizer and cross-entropy loss as a loss function. Specifically, we set the learning rate = 0.001, number of epochs = 4 (the number of times the dataset is accessed). For tracking the accuracy, we could also set the number of interval that denotes the number of minibatches before one round of performance evaluation on the validation set.

We use tensorboard to show the training accuracy, training loss and validation accuracy for each interval number of minibatches as shown in Figure 13.

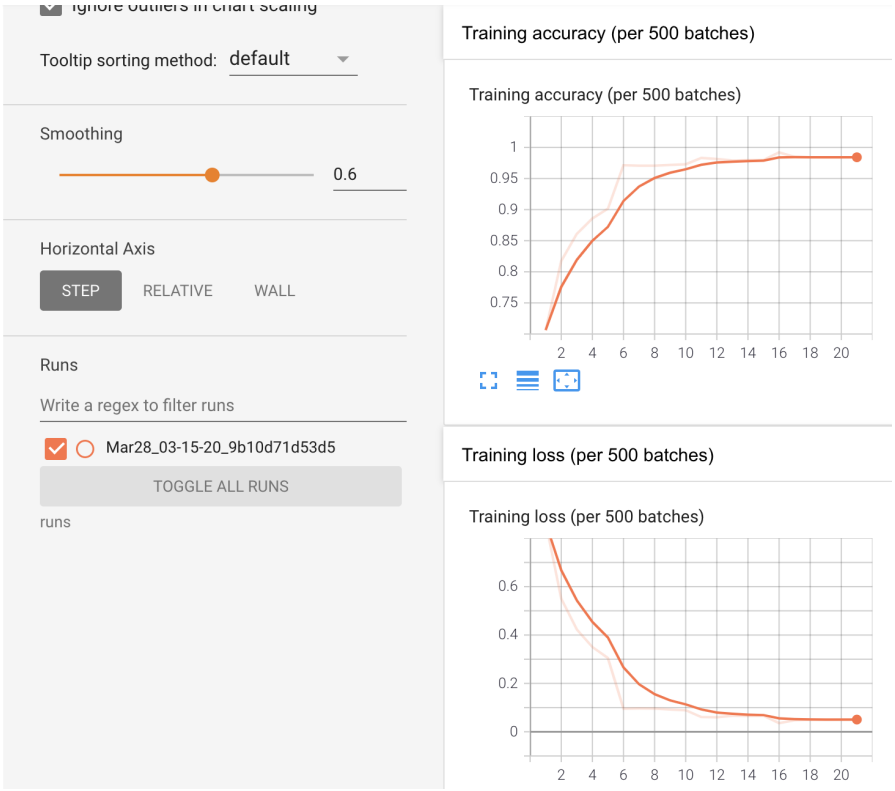


Figure 13: tensorboard

From the figure, it could be seen that the training accuracy keeps increasing up to 1 and the training loss keeps decreasing to 0, which demonstrates that there is no overfitting and underfitting in this case.

Evaluation

The tensorboard already shows that our accuracy approaches to 1. In the last part of this project, we numerically and visually present the performance of our trained model using the test set. Upon the complete

testing, we achieved the accuracy of 0.9856. We also display 160 samples with the true labels and the predicted results as shown in Figure 14. There is only one incorrect prediction, at location (9, 10),in these 160 predictions.

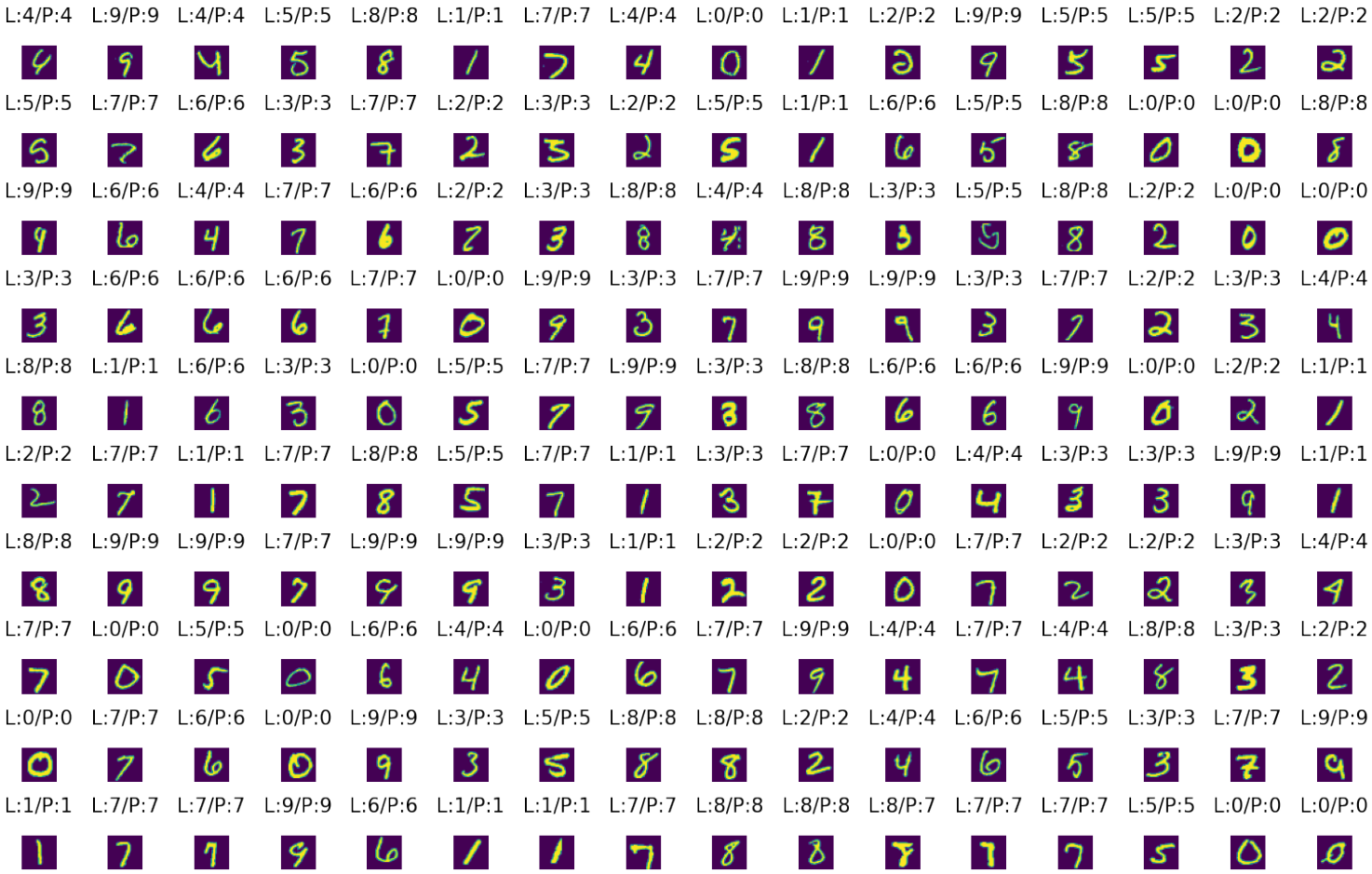


Figure 14: The initial prediction of the model

Conclusion

In this project, I introduced the image classification using different machine learning and deep learning methods. For machine learning, I primarily used SVM, Logistic Regression and Gaussian Naive Bayes based on the features extracted using Histogram of Oriented Gradients (HoG), Hue Histogram and Raw RGB for this purpose. For deep learning, I primarily used convolutional neural networks for this purpose. I achieved up to about 90% accuracy using machine learning methods and up to 98.5% accuracy using deep learning method. The results show the effectiveness of machine learning and deep learning in this difficult task, image classification.

References

[1] T. P. Marques, *Ece471: Computer vision course materials.*