The Comparison of Different Machine Learning Platform and Models for CIFAR 100

Group 2
Li Du, Jianing Wang, Wen-Chuan Chang

Professor Amir Jafari

DATS6203 Machine Learning II
December, 2018

Outline

- Introduction
- Data Description
- Learning network and algorithm
- Experimental setup
- Result
- Conclusion
- Reference

1. Introduction

In today's world, businesses can no longer turn a blind eye to the opportunity for expansion, by providing goods and services online instead of just in stores. To target specific markets and demographics, implementing the use of A.I. and big data are becoming increasing mainstream.

For example, if we can classify those pictures which have been uploaded by users to social media, then we could know the preference of users and classify the picture. On top of that, we can send different type of advertisements to different groups of users based on their preference, which would decrease the number of aimless advertisements.

This study aims to build a model to classify 100 different types of pictures. To solve this problem, we plan to use different platforms (Pytorch and Tensorflow) to implement the network. The classification accuracy of these methods will be compared.

2. Data Description

CIFAR-100 dataset will be used to train and test a neural network for this project.

CIFAR-100[1] consists of 60000 32x32 color images in 100 classes, with 600 images in each class. In these 600 images, there are 500 images in the training set and 100 images in the testing set. In addition, the 100 classes were grouped into 20 superclasses. For example, an image labeled as “dolphin” also belongs to the superclass of “aquatic mammals”. This adds one more dimension to the dataset and allows for more potential analyses.

The raw data for Python based modeling are available as “pickled” files, which can be unzipped and extracted using the “pickle” package. The data archive contains images for training grouped into 5 files (i.e. batches) and images for testing grouped into 1 file. The images are provided as numpy arrays with 3072 columns and 10000 rows. Each image is “flattened” into one row with the first 1024 columns containing the red channel values, the next 1024 the green and the final 1024 the blue. The labels of these images are indicated as 10000 numbers ranging from 0 to 9. The crosswalk between the numbers and the actual label is given in a dictionary by a separate file.

Rodrigo Benenson hosts a very comprehensive compilation of the papers and works[2] that explore the best performing approaches using a variety of datasets for deep learning projects. According to his record, the best accuracy score of a deep

learning method is approximately 76% and the lowest accuracy score is about 54%. These papers provide good reference information for this project, especially the range of accuracy to expect and target at.

The proposed work plan includes training and testing a multilayer neural network (e.g. CNN) as well as exploring hyperparameters that optimizes the performance and investigating the performance from multiple perspectives (such as intra-class variations of neural network performances and superclass accuracies).

3. Learning network and algorithm

3.1 Multilayer perceptron by Pytorch

Pytorch is one of the open-source machine learning libraries running on Python and built on Torch. Pytorch has more flexibility to offer than some other open-source platform because it supports dynamic computational graphing abilities.

In this project, the type of deep learning network we used is called a Multilayer Perceptron (MLP). An MLP is a deep, artificial neural network composed of more than 1 perceptron or layer. In Figure 1, the basic components of MLP model are comprised of: input, hidden layers and output layer. Data is brought in the neuron network from the Input component, and then the input data will proceed doc product with weight vector. After that, output of current layer will come out after finishing computation in transfer function. After all the computations which are in one or more than one hidden layer complete, the output layer will make the final prediction.

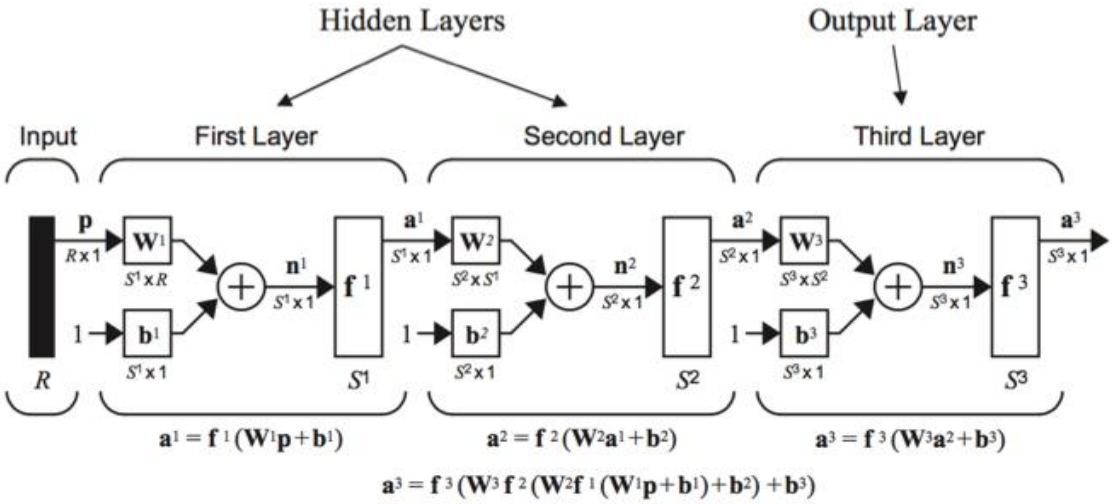


Figure 1. Structure of a typical multilayer perceptron network[1]

[1] Adopted from GWU DATS6203 Class slides (slides created by Prof. Amir Jafari).

3.2 Convolution neural network in Pytorch

The code for the computation of Convolutional Neural Network (CNN) is programmed using Pytorch, which deploys dynamic computational graphs. The dynamic computational graph defines the order of computations that are required to be performed. The Pytorch uses Object Oriented Programming feature to set up the framework for deep learning.

CNN typically consists of convolutional layers, pooling layers, fully connected layers and normalization layers. Convolutional layers utilize kernels (weights) to identify elemental features in the input with Shared Weights. The method of Shared Weight is considered as local connected neural nets to extract information from the input imagines. To sum up, a convolution layer then takes a set of feature maps as an input, and produces another set of feature maps as an output. Then, the pooling layers are keys to ensure that the subsequent layers of the CNN are able to pick up larger-scale detail than just edges and curves. However, in this project, the pooling layer connected after the second convolution layer greatly reduce the number of output resulting in insufficient input information.

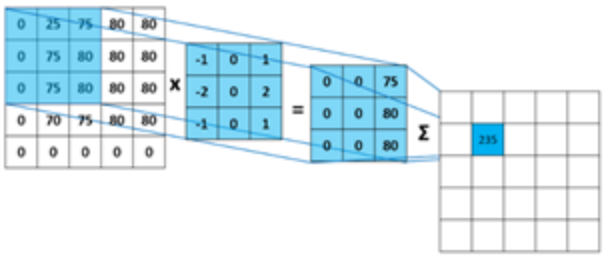


Figure 2. a step in the Convolution Process

Initially, CNN with two convolutional layers connected with pooling layer and two fully connected layer with hidden size=120 and 100 separately is applied into the datasets CIFAR100. In the condition of ReLU as transfer function, Cross-entropy loss function as performance index, SGD as optimization function, the model reach 16% accuracy and the training model takes 191s. Subsequently, to improve the performance at accuracy, the changes of kernel size, stride, and pooling layer has investigated in this report.

Figure 3. CNN Model by Pytorch

3.3 Transfer learning in TensorFlow (Keras API)

One of the major challenges for image classification, especially involving complex and large datasets, is the high computational cost. Such demands for computational resources are usually beyond the capability any standard computers. In addition, new ideas and new architectures are driving the progress for better network performance and higher accuracy. Fortunately, there are research groups and/or companies that have taken great efforts to develop and improve neural networks with the advantage of having access to high-performance facilities and talented team of researchers and scientist. As a result, these trained models are now accessible to the public for many individual projects.

In this project, we used the pre-trained deep neural network “Inception 5h” (Inception V1), or popularly known as “GoogLeNet”. It originated from the work done by a group of scientists with Google as a submission to the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC 2014)[3]. Inception module had more than one filters at the same level and then concatenated them before stacking all the layers up. This is particularly helpful in dealing with images having the key features occupying varying portion of the images. The entire net has 22 layers (27, including the pooling layers).

Figure 4[4] is a simple demonstration of the structure of the InceptionV1 network.

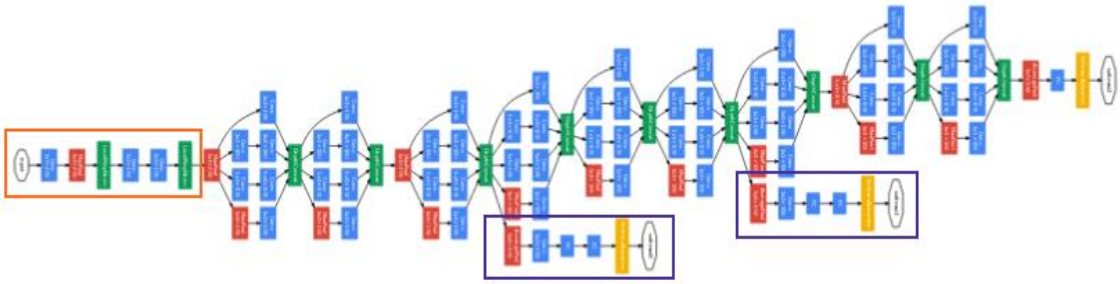


Figure 4. Structure of the InceptionV1 network

4. Experimental setup

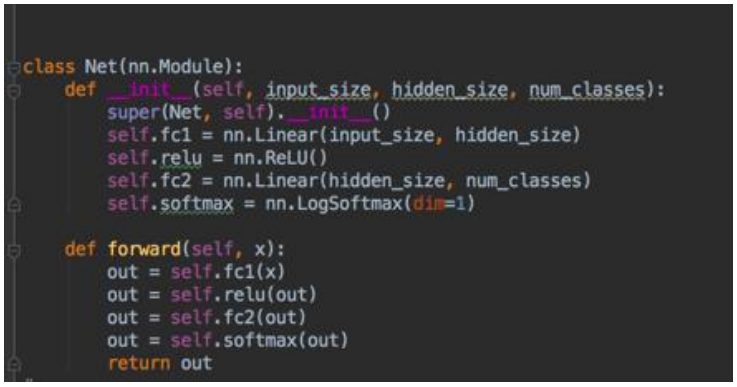
4.1 Multilayer perceptron in Pytorch

To start building the MLP model in PyTorch, we need to decide the initial parameters of the model. These parameters are: input size, batch size, and neuron size (Figure 2). In CIFAR-100 dataset, all of the inputs are color images, meaning that each of them will have three color channels. The number of output layers should be equal to the number of classes. Hidden size presents the number of neurons. Also, Mini batch has been picked in this study instead of batch or stochastic gradient descent (SGD), because it is faster than the stochastic method and will achieve a higher accuracy than the batch method.

```
input_size = 3 * 32 * 32
hidden_size = 50
num_classes = 100
num_epochs = 20
batch_size = 100
learning_rate = 0.001
```

Figure 5. initial set up for the MLP model

Second, we initiated the frame of the network (Figure 6). Relu and Softmax transfer functions have been chosen in the hidden layer and output layer respectively. On one hand, the Relu function is sparse and can reduce the likelihood of vanishing gradient. On the other hand, Softmax function can transfer the data to the range of 0-1, which is suitable for multi-classification.



```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.softmax(out)
        return out
```

Figure 6. structure of the MLP

There are four parameters have been tested in this project: number of layers, number of neurons, types of optimizer and batch size.

As shown in Table 1, with increasing the number of layers, accuracy declined Theoretically, the network with more layers is more complex network, so the network needs more training time to get a better result, However, even though we doubled the number of epochs, the accuracy is still lower than two-layer network. It may get a higher accuracy if we keep adding epoch number, but considering cost from training time, we decided to use two-layers network. After trying several optimizer tests, we decided to use SGD which has highest accuracy under same iteration (Table 2).

	Layer	Total parameters	Accuracy
Test	2	317200	23 %
Test1	4a	316856	21%; 22% (double epochs)
Test2	7	317400	10%

Table 1 Accuracy with different layers under same total parameters

Optimizer	SGD	Adam	Adagrad	RMSprop	Adadelta
Accuracy	22%	21%	18%	9%	9%

Table 2 Accuracy with different optimizers

Small batch size needs more time to converge, and it takes less time to converge in bigger batch size.

If the memory capacity is bigger enough, we can choose large size of mini batch, but if the batch size is too big, it will increase the cost to get a higher accuracy. With same number of iterations, increasing batch size will decrease the training time, theoretically. Based on the result show in Table 3, when batch size is larger

than 200, training time stops declining. What's more, larger batch size takes longer time to have a higher accuracy. As a result, 50 or 200 will be considered as the final batch size in the network based on the consequence in Table 3, and 500 will be the hidden size (Table 4).

Batch size	50	100	200	500	1000	2000
Accuracy	23.84%	22.41	19%	15.73	12%	5%
Time	280	265	243	245	248	248

Table 3 Different batch size with accuracy and training time

Hidden size	100	200	500	800
Accuracy	19.45	20.2	21	21
Time	253	250	258	268

Table 4 Accuracy with different number of neurons

If learning rate is too small, it will be hard to converge in the network. Because the step size for updates of weight and bias are tiny. In another word, it will take longer time to get a better performance. However, it doesn't mean that the bigger the learning is, the quicker to get a high performance. The reason is that if learning rate is too big, the update will not be stable, also it will bypass the minimum point (light blue line in Figure 7).

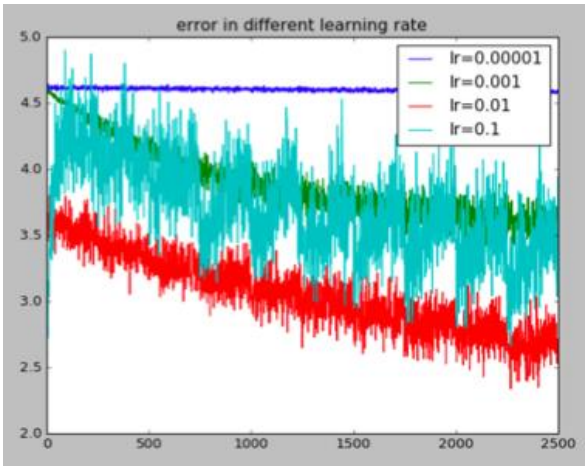


Figure 7. comparison of the loss using different learning rates

4.2 Convolution neural network in Pytorch

The initial model, CV_1,is introduced with two convolution layers and two fully connected layers. The accuracy of CV_1 is only 16%. The first parameter to adjust the initial model is increasing the number of kernels. 5 times kennels is adopted as CV-2 model. The number of kennels in the first convolution layer increase from 6 to 30, and 30 to 80 in the second convolution layer. As a result, the accuracy increases from 16% to 22%, and the training time increases from 191s to 242s. Compared to MLP, CNN costs more training time in writing number into computer's memory.
The second parameter to adjust model is changing kernel size. Kernel size of 3, 5, 8 has been used in this project. The models of CV_2, CV_3, CV_4 adopt different Kernel sizes. However, there is no linear correlation between different kernel size, and kernel size =5 has the best performance at accuracy among the models of CV_2, CV_3, CV_4.

		CV_1	CV_2	CV_3	CV_4
	batch size	200	200	200	100
	epoch	20	20	20	40
Conv1	nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5)	v			
	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=3)			v	
	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5)		v		
	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=8)				v
pool1	nn.MaxPool2d(kernel_size=2, stride=2)	v	v	v	v
Conv2	nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)	v			
	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=3)			v	
	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=5)		v		
	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=8)				v
pool2	nn.MaxPool2d(kernel_size=2, stride=2)	v	v	v	v
ip	nn.Linear(**, 120)	v	v	v	v
	nn.Linear(**, 100)	v	v	v	v
	Accuracy	16%	22%	20%	17%
	training time(s)	191	242	230	229

Figure 8. Exploration of different combinations of model hyperparameters

Moreover, the model of CV_5 used larger stride. Because larger stride will reduce the number of inputs into the next layer. The number of inputs has down to 80*5*5 after first convolution and pooling layer. Then, increasing stride is not suitable to use in the second convolution layer. Besides, to keep the inputs at the minimum number of inputs, 80*1*1, at the fully connected layer, the pooling at the second convolution layer should not also included. As expected, the performance at accuracy drop to 12% due to a smaller number of outputs from feature maps.

From the experiment of model of CV_5, it implies that the original model has potential risk of insufficient number of inputs from feature maps. Therefore, the model of removing second pooling layer has been proposed to increase the number of outputs from feature maps. In the model of CV_6, without second pooling layer, the number of inputs into first fully connected layer increased to 80*10*10. And, the accuracy of CV_6 has also increased to 28%.

		CV_2	CV_5	CV_6
	batch size	200	200	200
	epoch	20	20	20
Conv1	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5)	v		v
	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5,stride=3)		v	
pool1	nn.MaxPool2d(kernel_size=2, stride=2)	v	v	v
Conv2	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=5)	v		v
	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=5,stride=1)		v	
pool2	nn.MaxPool2d(kernel_size=2, stride=2)	v		
ip	nn.Linear(**, 120)	v	v	v
	nn.Linear(**, 100)	v	v	v
	Accuracy	22%	12%	28%
	training time(s)	242	186	246

Figure 9. Exploration of different combinations of model hyperparameters

Finally, CV_7 combing the advantage of CV_6 and increasing neurons at fully connected layers reaches the best performance at accuracy of 37%. Because the more number of weights has been used, the iteration and epoch times should also add to the model CV_7.

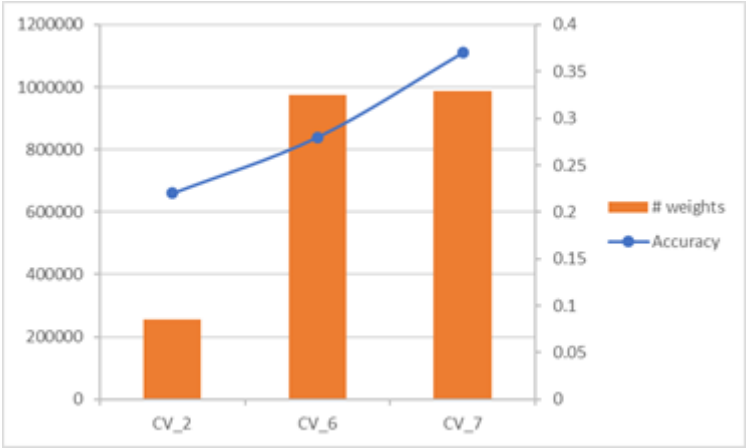


Figure 10. Accuracy and the number of weights of CV_2, CV_6,CV_7

		CV_2	CV_6	CV_7
	batch size	200	200	100
	epoch	20	20	40
Conv1	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5)	v	v	v
pool1	nn.MaxPool2d(kernel_size=2, stride=2)	v	v	v
Conv2	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=5)	v	v	v
pool2	nn.MaxPool2d(kernel_size=2, stride=2)	v		
ip	nn.Linear(**, 120)	v	v	v
	nn.Linear(**, 110)			v
	nn.Linear(**, 100)	v	v	v
	Accuracy	22%	28%	37%
	training time(s)	242	246	500

Figure 11. Exploration of different combinations of model hyperparameters

4.3 Transfer learning in TensorFlow

The pre-trained InceptionV1 nets can be downloaded from Google' developers' tool deposit and easily imported into tensorflow's framework. The imported model worked seamlessly with the tensorflow keras API, which made it easy to extract tensors from any particular layer.

The idea is to first run the CIFAR100 images through this pre-trained network and export the tensors from a pooling layer before the fully connected layers. Then, these tensors are subsequently fed to a fully connected layer with dropouts followed by an activation layer using softmax. The first step took about 1.5 hours to finish, but it only needs to be executed once.

5. Result

5.1 MLP in Pytorch

According to the discussion above, we summarized several combinations which are shown in Table 5.

Table 5. Summary of the performance and corresponding hyperparameter settings

Accuracy	Learning rate	Batch size	epochs
20%	0.01	50	50
24%	0.01	200	50
25%	0.001	50	50
22%	0.001	200	50

After adjusting several parameters compared above, the highest accuracy we got in MLP was 27%, and accuracy in superclasses was around 27% also (Figure 12).

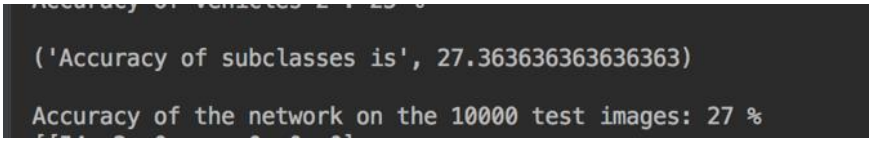


Figure 12. Output from Pycharm showing the final accuracy

Since the accuracy of superclasses and total classes are same basically, so we plot the confusion matrix for superclasses instead of entire classes (Figure 13). It didn't show any spots with saturated color outside of diagonal, which means

there is no a specific misclassification.

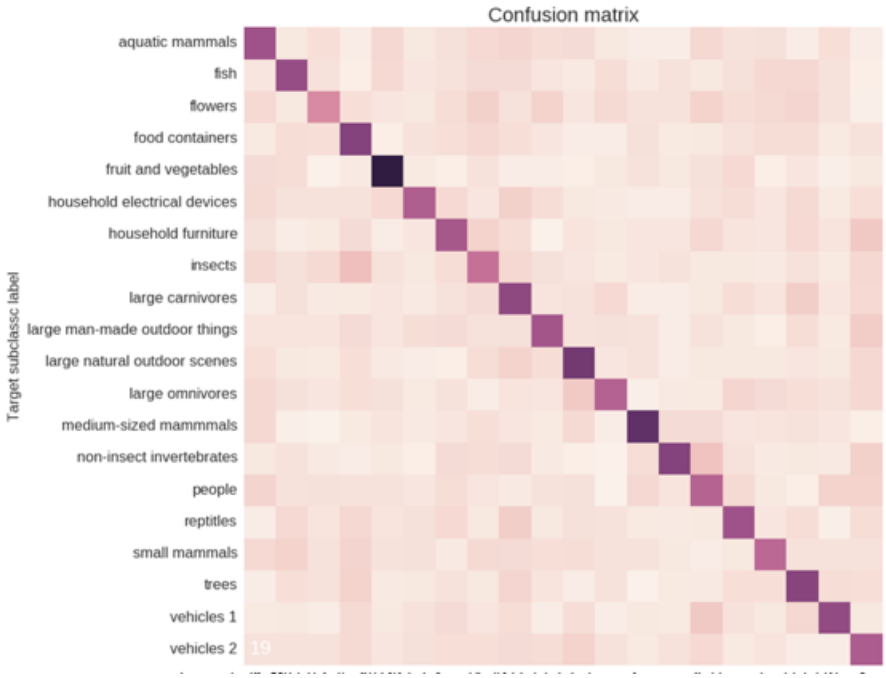


Figure 13. Confusion matrix of the results from MLP

As shown in Figure 14, the best performance happened in class fox, mountain, and spider with accuracies around 73%, 70%, 65% respectively. Three of the top 10 classes belong to fruit and vegetables subclass, and half of them belong to different mammals' group. The finding in Figure 14 is consistent with the result in Figure 15 which the best performance is class fruit and vegetables.

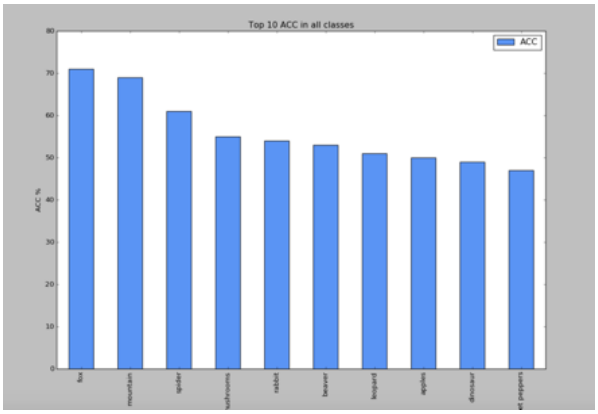


Figure 14. Top 10 Accuracy classes in total classes

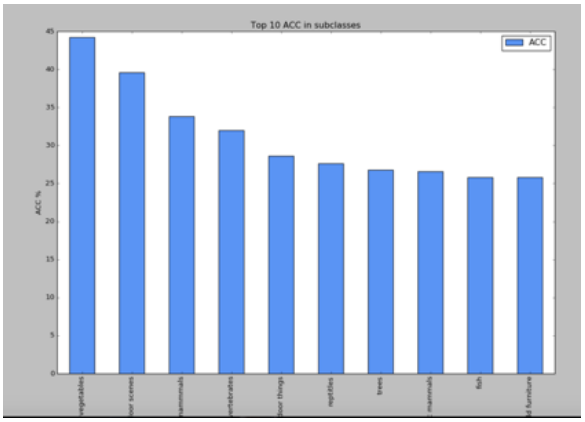


Figure 15. Top 10 Accuracy in superclasses

5.2 CNN in Pytorch

CNNs are usually applied to image data. Every image is a matrix of pixel values, and the CNN learns the features from the input images. Typically, they emerge repeatedly from the data to gain prominence. If the number of inputs extracting the information from the original picture is insufficient, the training model leads to lower performance at accuracy. To increase the number of inputs from feature maps, removing second pooling layer is proposed in this project.

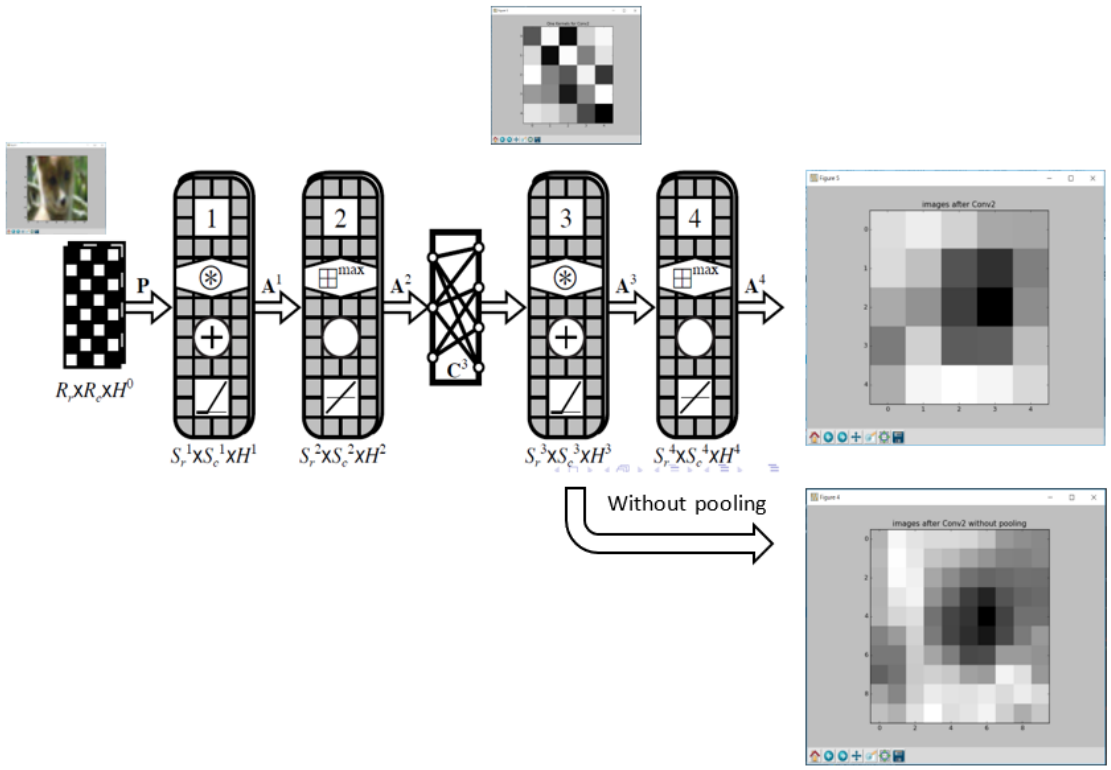


Figure 16: Output after convolution 2 with and without pooling

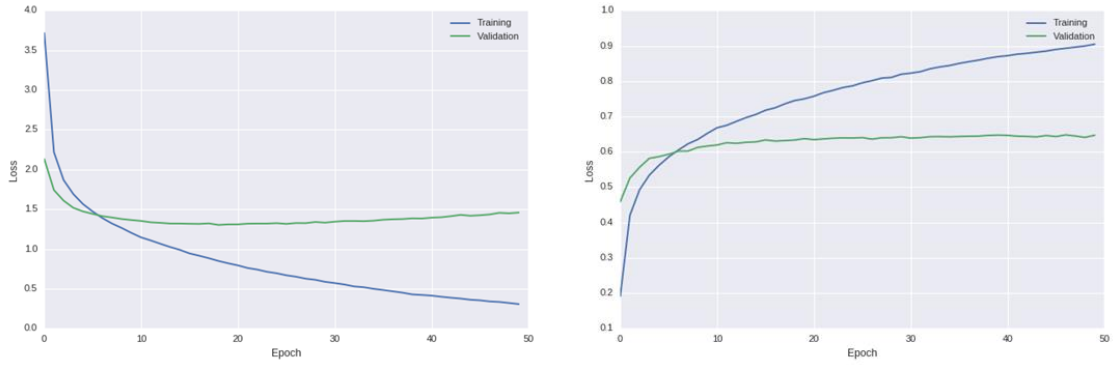
Take an example of test picture of a fox. The output after second convolution layer without pooling has more features about a fox, the input. It captures more three dimensional information about fox's facial features compared the result with pooling layer. To sum up, the CNN can efficiently apply to pattern recognition, but the parameter used in the convolution and pooling layers may resulting in insufficient inputs. Therefore, the parameters, kernel number, kernel size, stride in the convolution and pooling layers, need to be optimized to create a better CNN model.

5.3 Transfer learning in TensorFlow

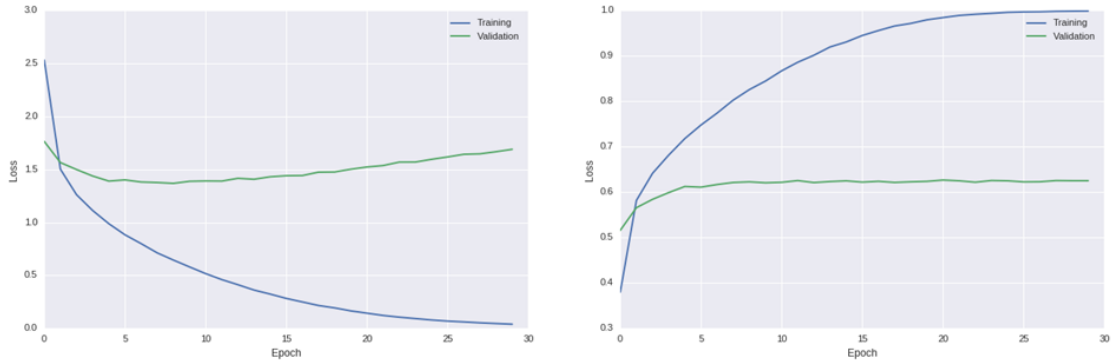
As discussed previously, the output from the convolution and pooling layers were first exported and imported to feed two fully connected layers and an output layer. The base case has 2 layers with 1024 neurons each, 1 dropout layer with 50% neurons left out and 1 more layer having softmax as the activation function. With this design, the model will predict the probability of each image being classified as each of the 100 classes. For the purpose of evaluating the accuracy, simple post-processing was done to identify the class with highest probability.

After tuning the hyperparameters of the MLP, it was found that the performance of the entire transfer learning model was dominated by the pre-trained deep nets. The below graphs demonstrate the effects of the multiple hyperparameters.

Base case (loss, accuracy vs. epoch)



Without dropout layers (loss, accuracy vs epoch)



With 128 neurons

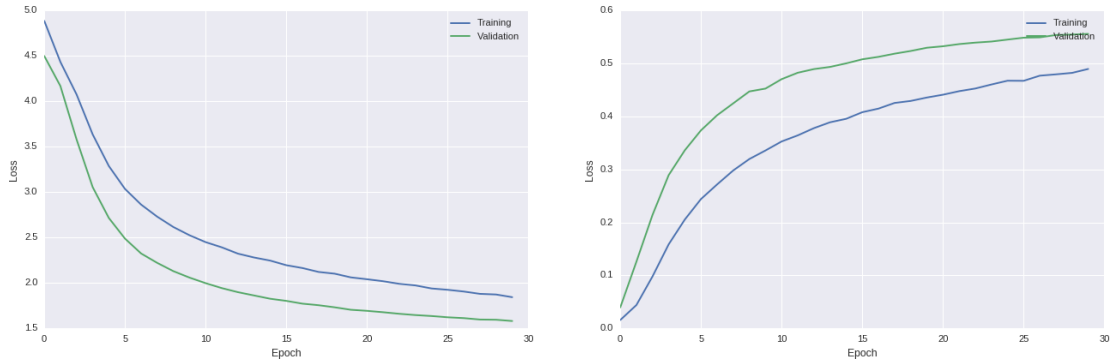


Figure 17. Effect of dropouts and number of neurons in the fully connected layers to the performance

The best performance of this transfer learning model achieves a total accuracy of approximately 66%. The relatively higher mis-classification rates represented as darker spots that are not on the diagonal line are loosely distributed and sparse, which, again, indicated no systematic misclassification. The class “motorcycle” has the highest accuracy (> 90%) whereas the class “seal” has the lowest accuracy (< 30%). By investigating several, classes that have visually similar textures and shapes are prone to misclassification (such as snake vs. worm, baby vs. girl).

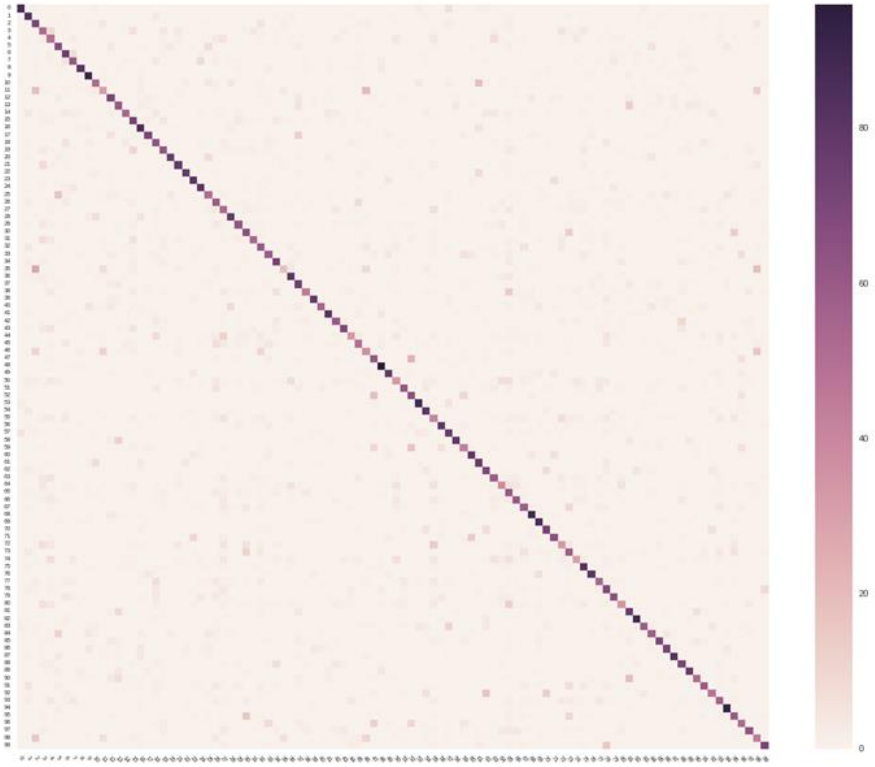


Figure XXX. Confusion matrix for the results from transfer learning

6. Conclusion

The result reveals that MLP is not a good model for CIFAR 100 as the accuracy is only 27%. However, it is suggested that this model works relatively better to classify fruit and vegetables and large natural outdoor senses whose performance are over 40%.

One of methods to approach the CIFAR100 adopts Convolutional Neural Network (CNN) programmed using Pytorch. The convolution layers in the CNN identify elemental features in the input. Therefore, the number of outputs of convolutions layers should be enough to ensure that the model extracts sufficient information from the input pictures. In the project, the comparison between models of CV_1 and CV_6 confirmed that the initial model is lack of the number of inputs. After increasing the number of neurons and training times, the final model CV_7 reaches the highest accuracy of the CNN model in this project.

Based on the exploration of transfer learning using the Inceptions model, it was found that the performance was dominated by the pre-trained model. Tuning the

structure and hyperparameters (at least lightly) of the following customized fully connected layers had marginal effects on the overall accuracy. The accuracy specific to each class differs significantly, with the best predicted class motorcycle being over 90% and the most poorly predicted class “seal” being less than 30%.

Comparing the three approaches, convolution neural network performs better than multilayer perceptrons with decent level of tuning. This has been consistent with the observations widely acknowledged by common practice. Transfer learning appears to a very powerful method for the image classification problem based on the exploration in this project. It allows the convenience of building neural network on top of what has been done nicely and previously without the need to deal with the requirements for dramatic amount computational resources. In this project, with decent effort in tuning the subsequent fully connected layers, the model was able to achieve 66% accuracy.

7. Reference

[1] Official hosting page for CIFAR-100 dataset can be found at: https://www.cs.toronto.edu/~kriz/cifar.html

[2] “What is the class of this image?” at http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d313030accessed Oct 29 2018.

[3] Szegedy, C., Liu, W., Jia Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.. 2014. Going deeper with convolutions. Publicly available at: https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf.

[4] A simple guide to the versions of the Inception Network. Available at: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202. Accessed on Dec 1, 2018