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

Du Li, Jianing(Janine) Wang, Wen-Chuan(Selina) Chang

Group 2

4^{th} Dec, 2018

CONTENT

Introduction to the dataset – CIFAR100

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

100 classes containing 600 images each.

500 training images and 100 testing images per class.

Total 50,000 training set and 10,000 test set

The 100 classes in the CIFAR-100 are grouped into 20 superclasses.

02

MLP in PyTorch

MLP in PyTorch – Initial Setup

```python
input_size = 3 * 32 * 32
hidden_size = 200
num_classes = 100
num_epochs = 20
batch_size = 100
learning_rate = 0.0001
momentum = 0.9
```

```python
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
```

MLP in PyTorch - Adjustment

	Layer	Total parameters	Accuracy
Test	2	317200	23 %
Test1	4	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

MLP in PyTorch - Adjustment

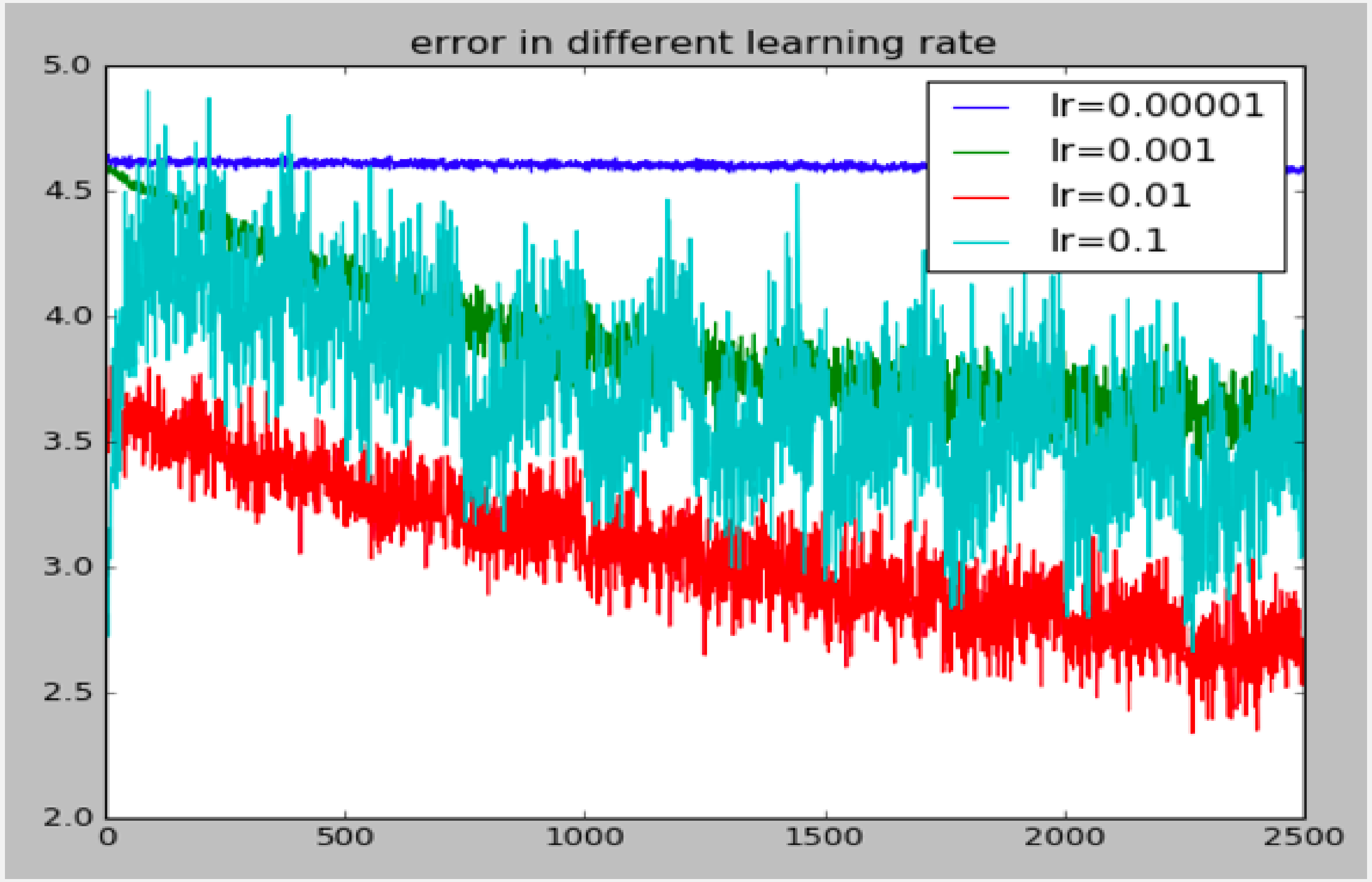
Batch size	50	100	200	500	1000	2000
Accuracy	23.84%	22.41	19%	15.73	12%	5%
Time	280	265	243	245	248	248

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

MLP in PyTorch - Adjustment



Small learning rate will be hard to converge in the network.
If it is too big, the update will not be stable, also it will bypass the minimum point.

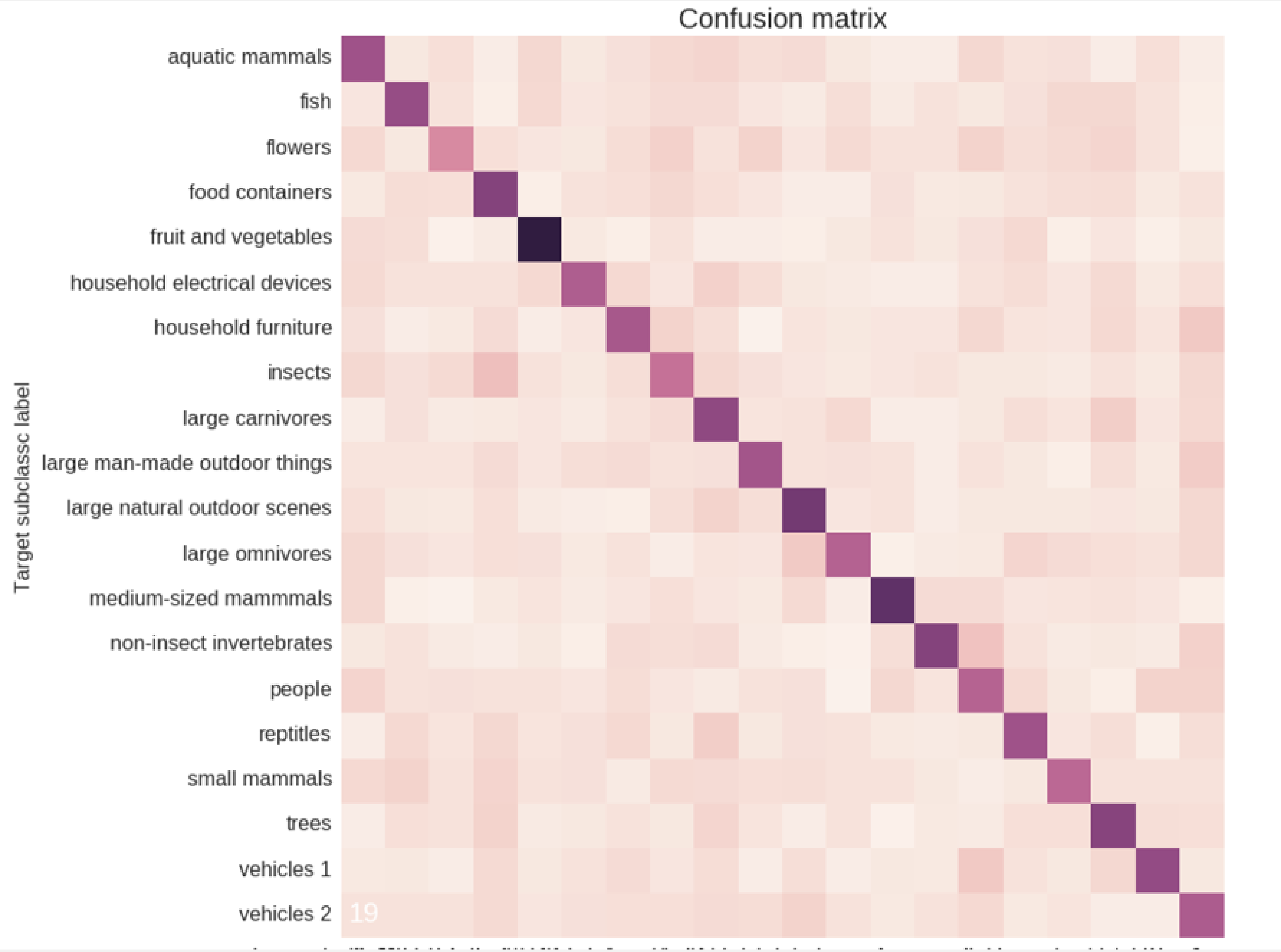
MLP in PyTorch - Result

```
input_size = 3 * 32 * 32
hidden_size = 500
num_classes = 100
num_epochs = 50
batch_size = 50
learning_rate = 0.001
momentum = 0.9
```

```
('Accuracy of subclasses is', 27.363636363636363)

Accuracy of the network on the 10000 test images: 27 %
```

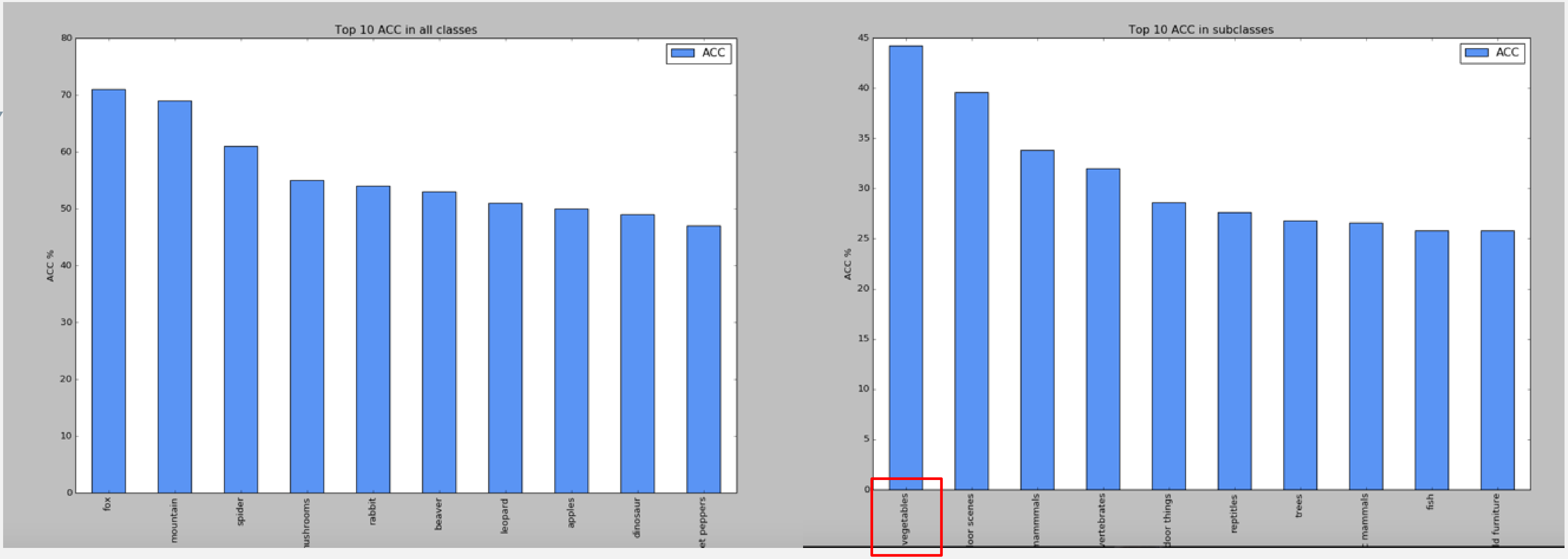
```
# ---------------------------------------------------------------

class_correct = list(0. for i in range(100))
class_total = list(0. for i in range(100))
subclass_correct=list(0. for i in range(20))
subclass_total=list(0. for i in range(20))
for data in test_loader:
    images, labels = data
    images = Variable(images.view(-1,3* 32 * 32)).cuda()
    outputs = net(images)
    _, predicted = torch.max(outputs.data, 1)
    labels = labels.cpu().numpy()
    c = (predicted.cpu().numpy() == labels)
    for i in range(batch_size):
        label = labels[i]
        if (label>=0) and (label <5):
            subclass_correct[0]+= c[i]
            subclass_total[0]+= 1

        elif (label>=5) and (label <10):
            subclass_correct[1]+= c[i]
            subclass_total[1]+= 1

        elif (label>=10) and (label <15):
            subclass_correct[2]+= c[i]
            subclass_total[2]+= 1

        elif (label>=15) and (label <20):
            subclass_correct[3]+= c[i]
            subclass_total[3]+= 1

        elif (label>=20) and (label <25):
            subclass_correct[4]+= c[i]
            subclass_total[4]+= 1

        elif (label>=25) and (label <30):
```

MLP in PyTorch - Result



It didn’t show any saturated spots outside of diagonal, which means there is no a specific misclassification.

MLP in PyTorch - Result



Fox	mountain	Spider	Mushrooms	rabbit	beaver	leopard	apple	dinosaur	Sweet pepper
medium-sized mammals	large natural outdoor scenes	non-insect invertebrates	fruit and vegetables	small mammals	aquatic mammals	large carnivores	fruit and vegetables	reptiles	fruit and vegetables

03

CNN in Pytorch

CNN in PyTorch

		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

- Increasing the number of kernels increases the accuracy.
- Kernel size =5 has the best performance in terms of accuracy.

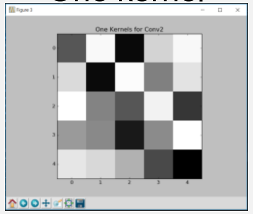
CNN in PyTorch

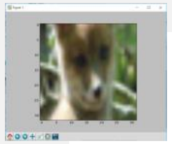
		CV_2	CV_5	CV_6
	batch size	200	200	200
	epoch	20	20	20
Conv1	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5)	v		v
	nn.Conv2d(in_channels=3, out_channels=30, kernel_size=5,stride=3)		v	
pool1	nn.MaxPool2d(kernel_size=2, stride=2)	v	v	v
Conv2	nn.Conv2d(in_channels=30, out_channels=80, kernel_size=5)	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

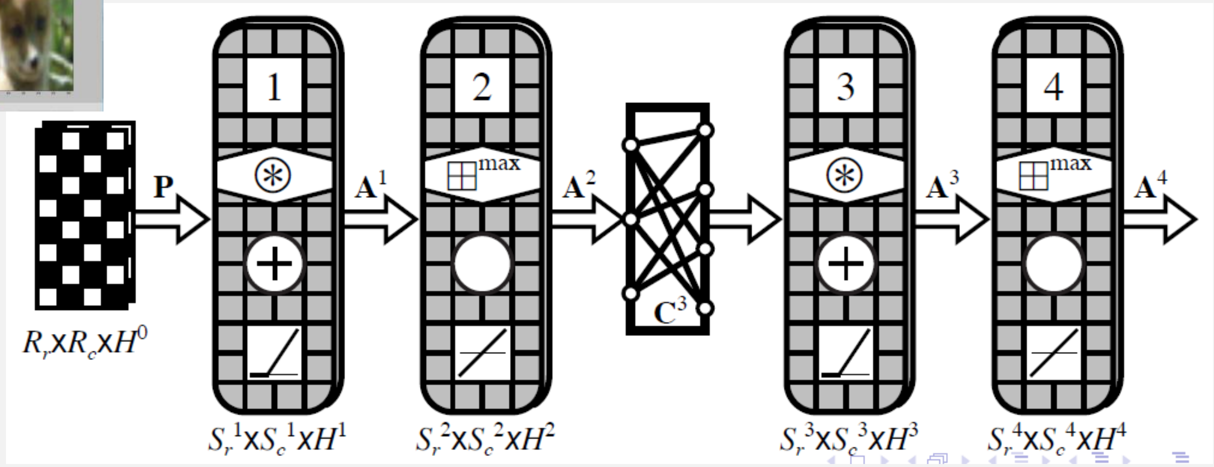
- Changing the number of stride at convolution layer decreases with accuracy because FM did not sufficiently extract the feature of the picture.
  - The size of outputs of Conv2 down to 80x1x1
  - Suggest to increase the size of outputs of Conv2 layer by removing pool2 layer
- Model CV_6 without pool2 layer has a better performance at accuracy

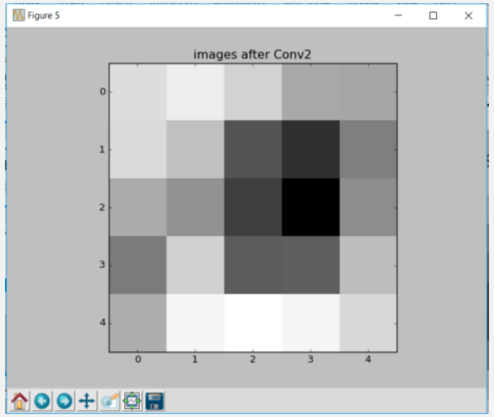
CNN in PyTorch

One kernel

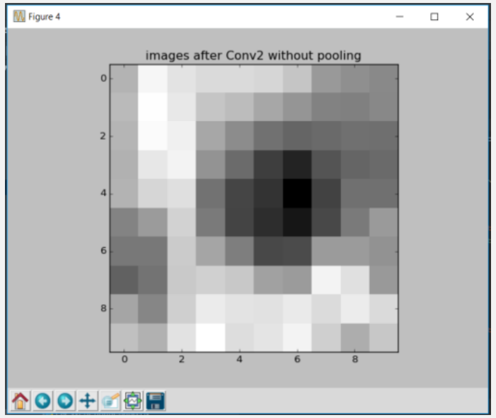








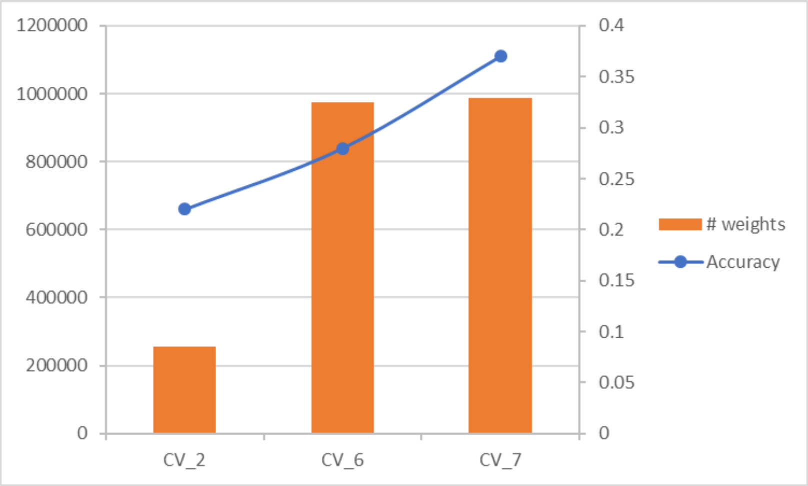
Without pooling



- Output after convolution2 without pooling has more information extracted from the original picture

CNN in PyTorch

		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



- Increasing the number of neuron at fully connected layers (ip), the numbers of epoch, and decreasing batch size increases the accuracy from 28% to 37%.

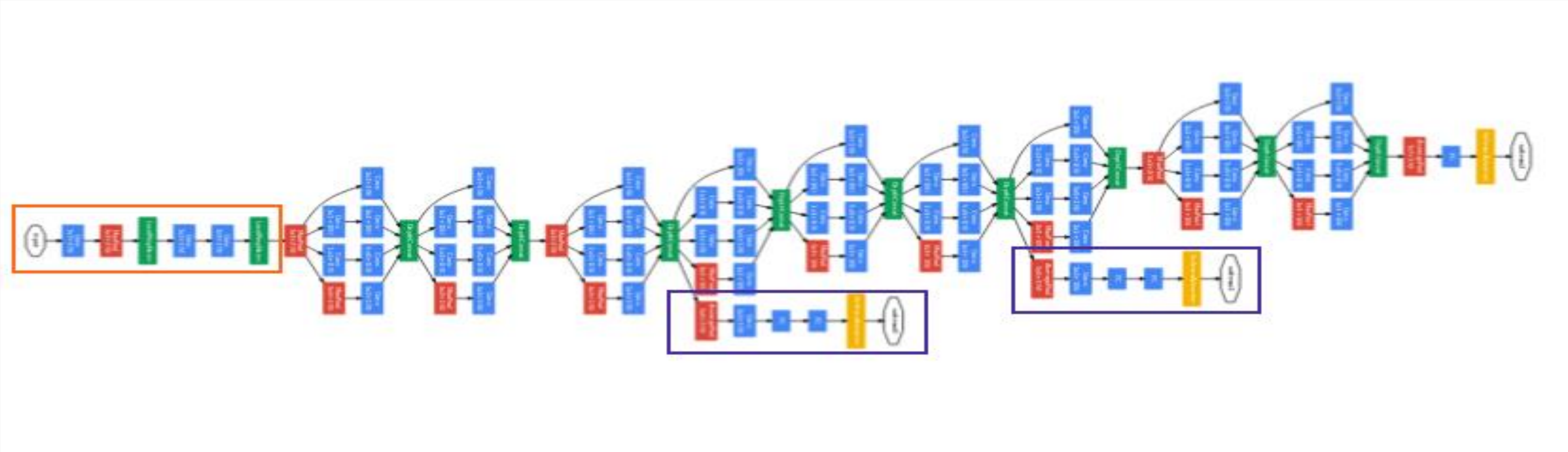
04

Transfer learning in TensorFlow

Transfer learning

- Transfer learning is to reuse a model that was trained in a prior task as a starting point for the new/current task
- GoogLeNet (InceptionV1) was used in this project as a starting point. It was originally developed as a submission to a competition by a group of scientists with Google

Inception V1

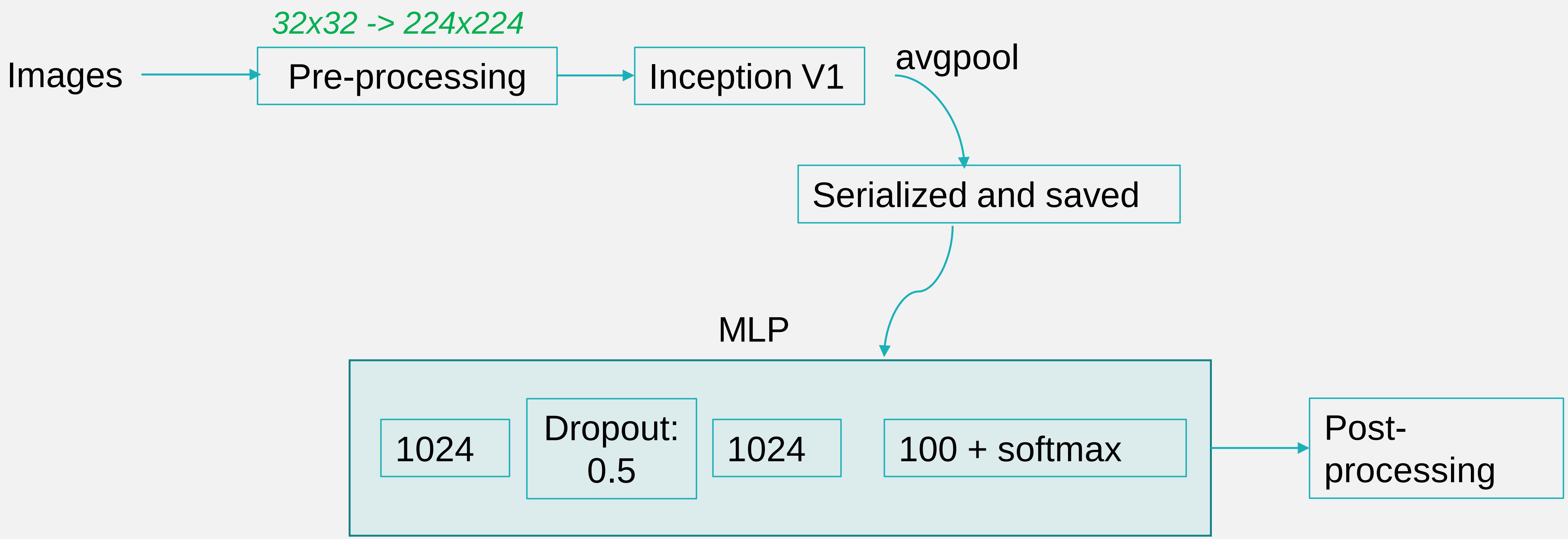


22 layers (27, including the pooling layers)

Image credit: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

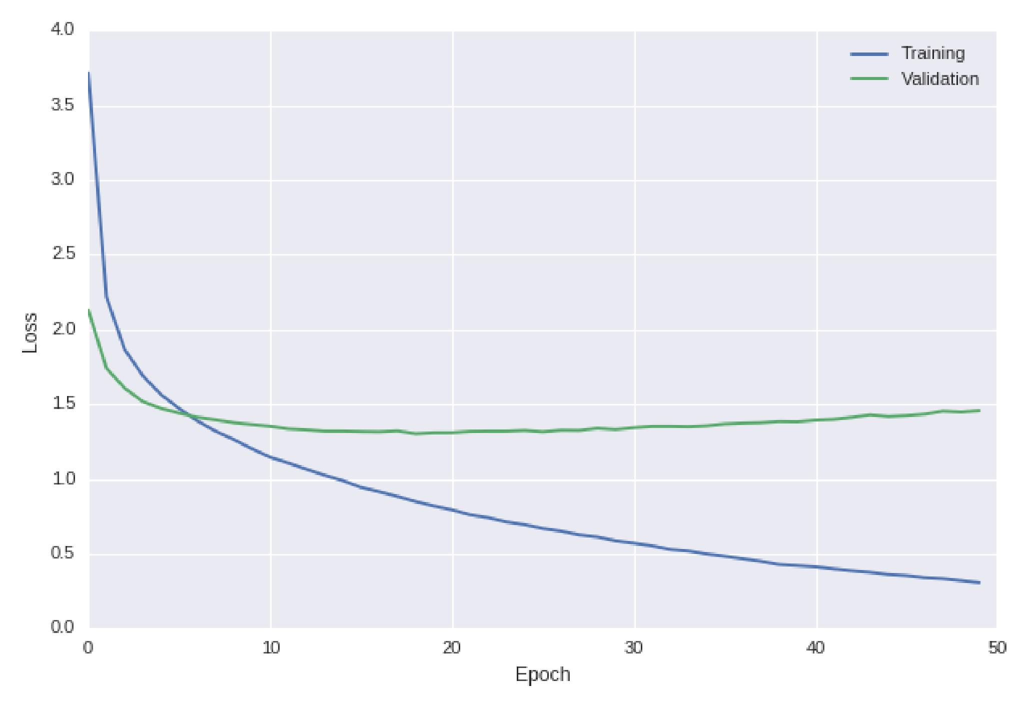
Transfer learning in TensorFlow

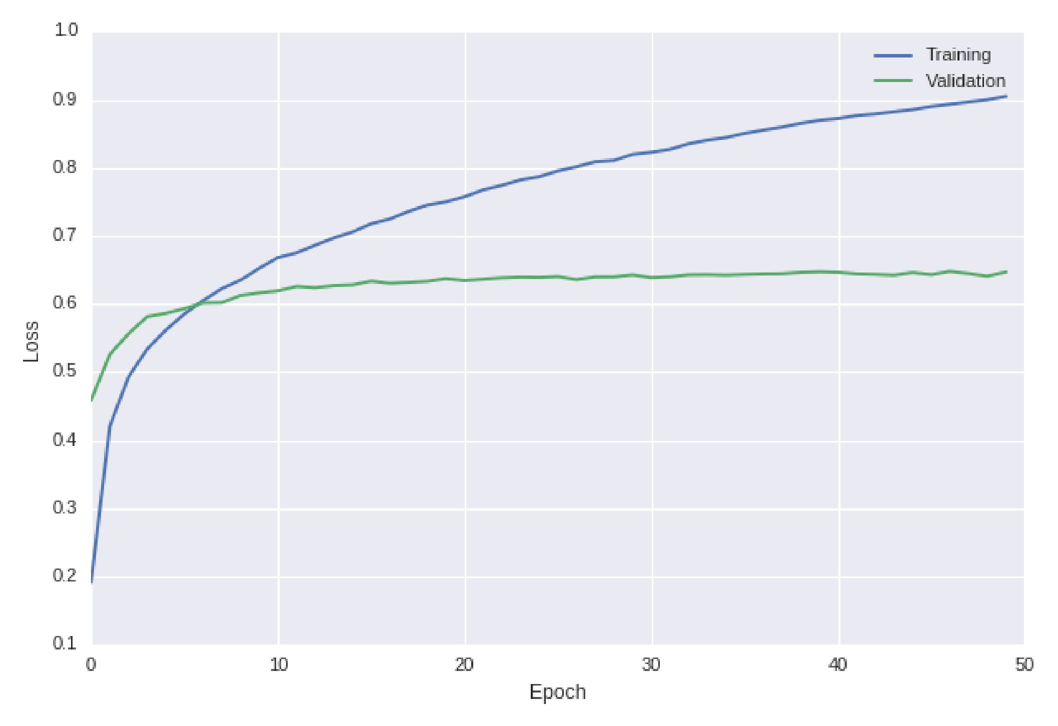
Platform: tensorflow.keras
Data ingestion: tensorflow.keras.datasets.cifar100



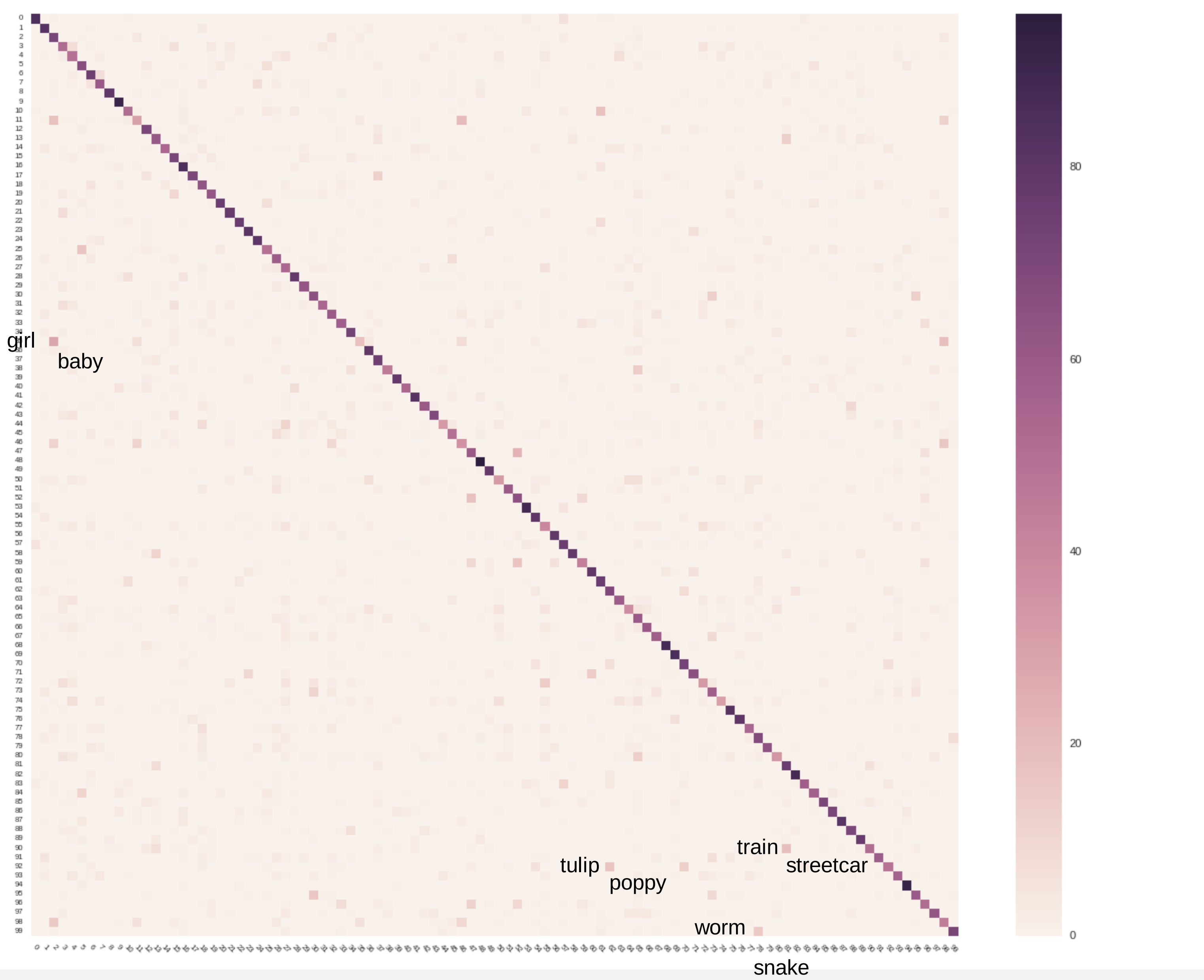
Transfer learning in TensorFlow - Performance

- Keras can conveniently set aside pre-defined portion of training data for validation purposes
- Training for more epochs will continuously lower the loss with the training set, but the loss on the validation set can be indicative of the optimal number of epochs
- The performance of this transfer learning net is dominated by the pre-trained model. MLP layers has marginal effects on the accuracy.

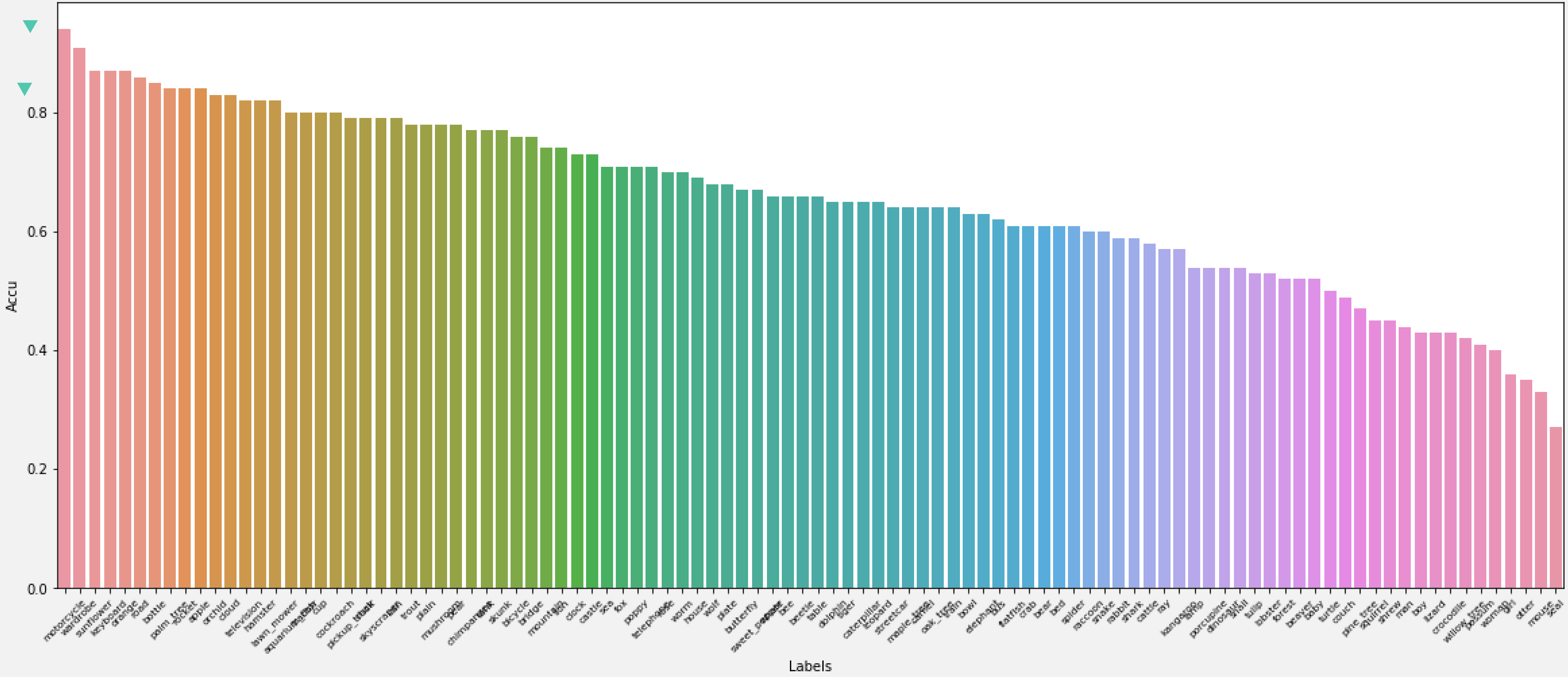




Transfer learning in TensorFlow - Performance



Transfer learning in TensorFlow - Performance



Conclusion

- MLP is not a good model for CIFAR100 as the accuracy is only 27%. However, our analysis suggests that MLP model performs relatively better to classify fruit and vegetables and large natural outdoor senses whose performance are over 40%

- The CNN model reaches its highest accuracy at 37%.
  - Control the size of output of FM is important to train the model

- Output from the Inception model + fully connected layers yields decent results (overall accuracy = 66%), but the class-specific accuracies range from 27% to 94%.

Question?

Thank you

Du Li, Jianing Wang, Wen-Chuan Chang

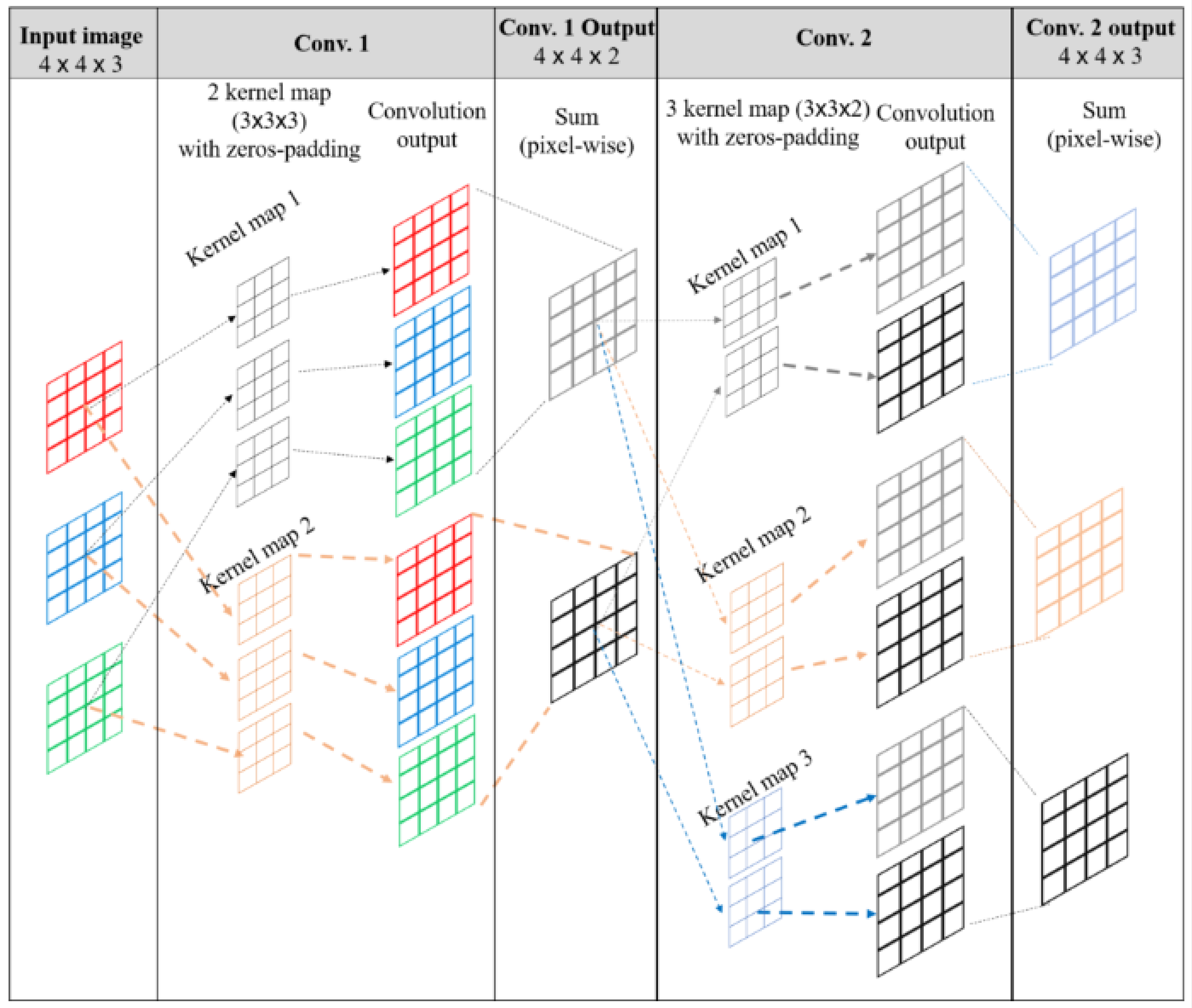
Group 2

4th Dec, 2018

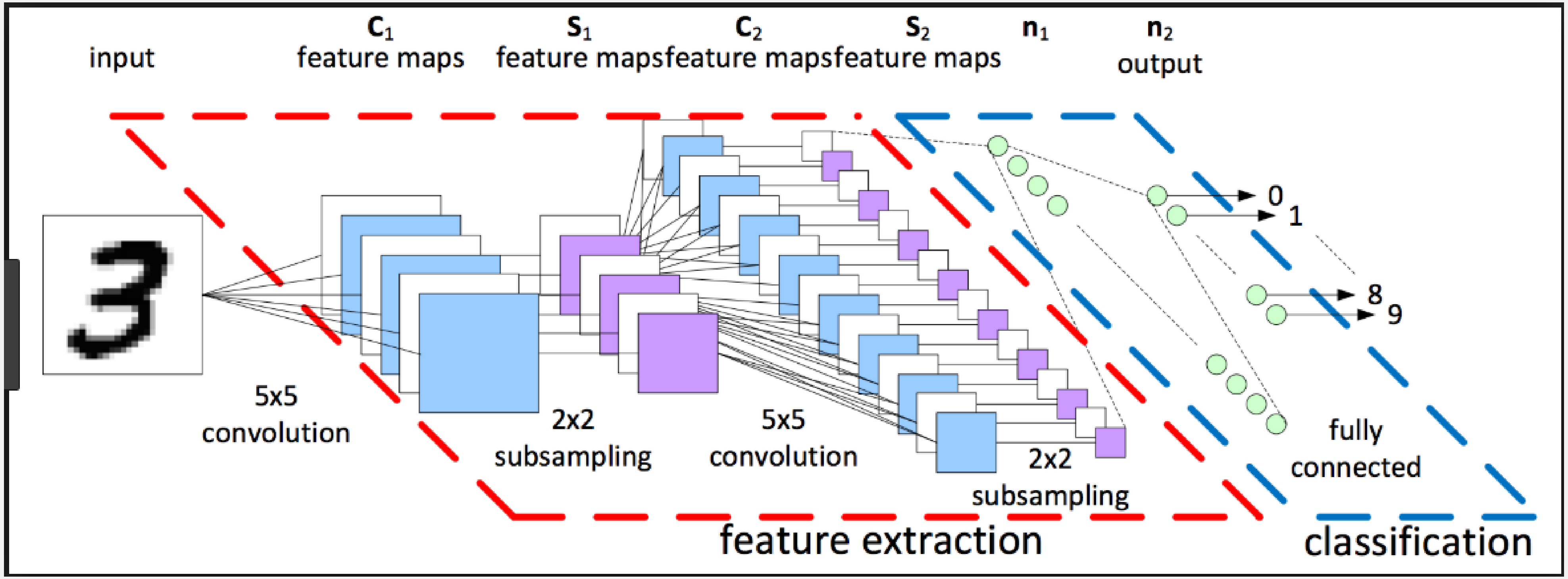
05

Back Up

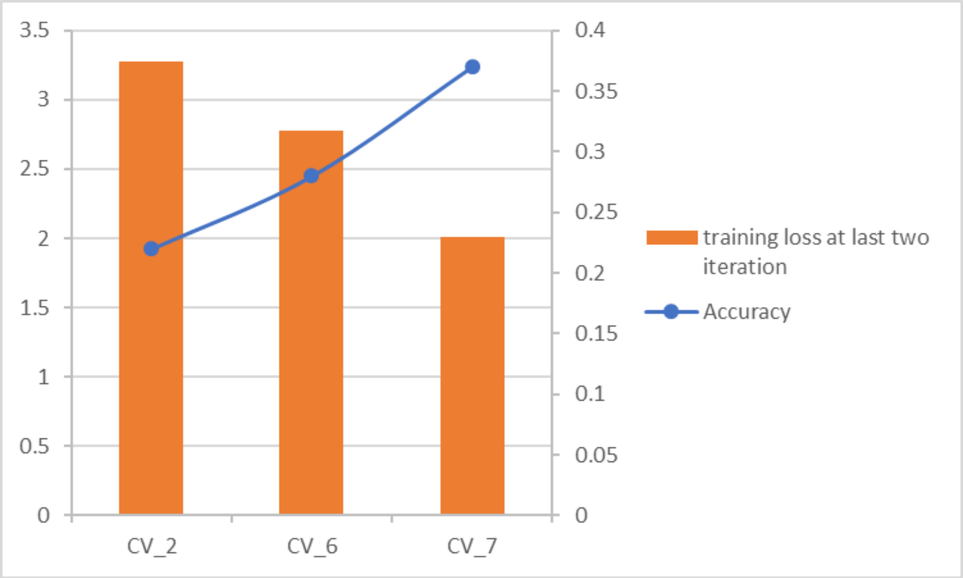
CNN Calculation Process



CNN Architecture



CNN in Pytorch



- The result of training loss have not showed that the model suffer over-fitting

Backup

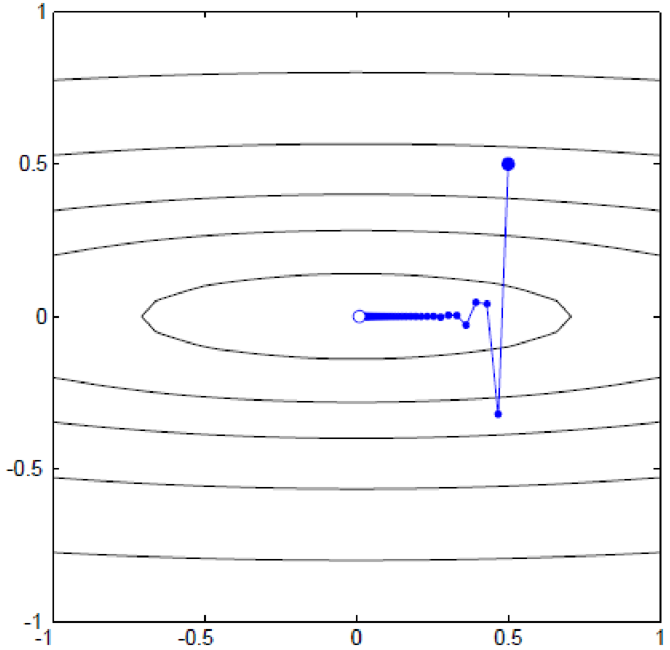


Figure P12.3 Trajectory for $\alpha = 0.041$ and $\gamma = 0.2$

Backup

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

	Layer 2 ACC
Drop = 0	20
Drop = 0.2	19.03
Drop = 0.4	17