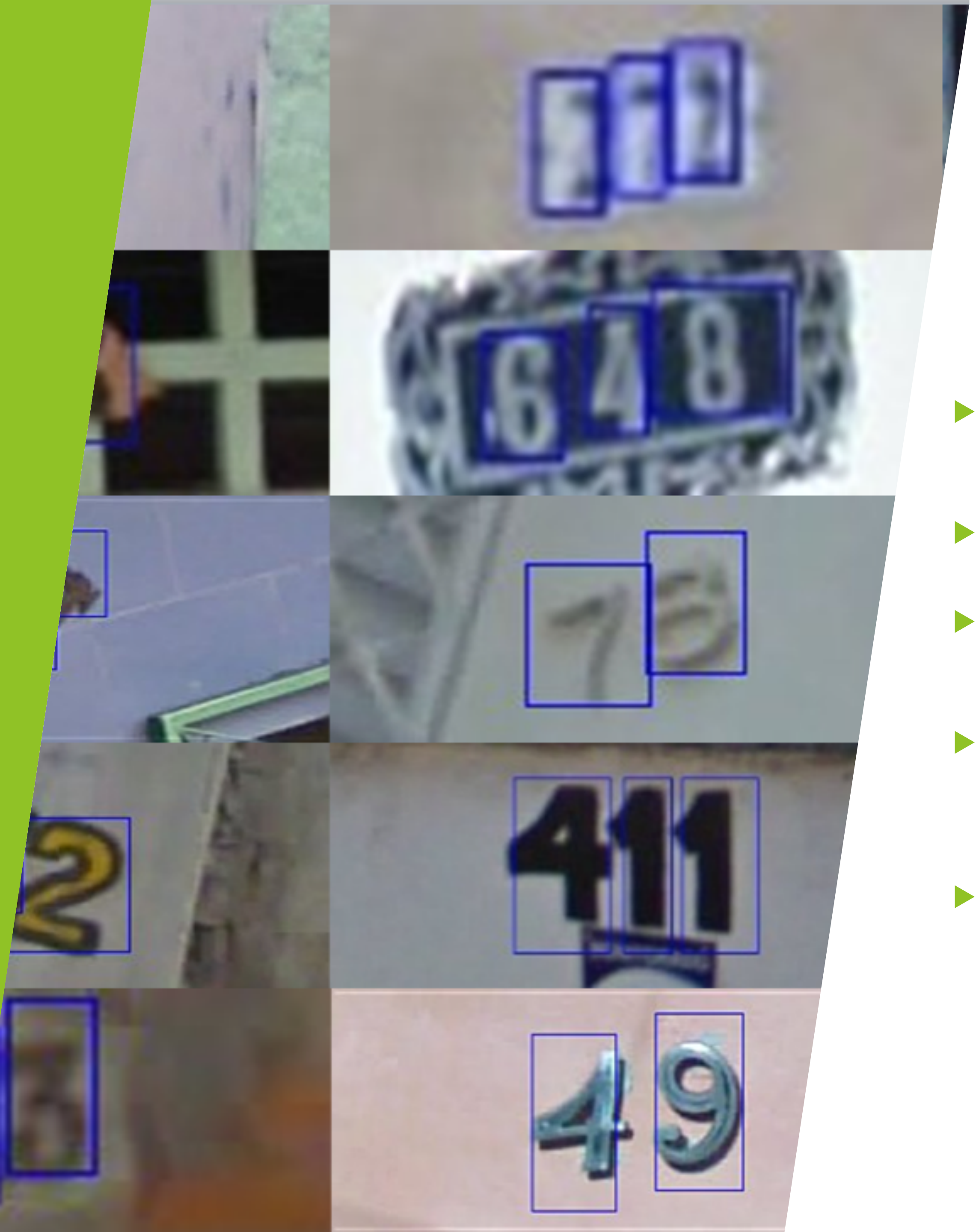
DATS 6203 Final Project Group 6

Convolutional neural network to classify handwritten digits in SVHN dataset



Description of dataset

- The Street View House Numbers (SVHN) Dataset consists of images of house numbers in Google Street View images.
- Each image is classified as 1 of 10 digits from 0 to 9.
- There are 73257 digits for training/validation and 26032 digits for testing.
- images that will be used for this project are cropped images of single characters resized to a resolution of 32-by-32 pixels.
- The dimension of each images is 32X32X3. 3 represents values of red, green, and blue color channels in a color image.

Pytorch framework

- The framework that will be used to implement CNN networks and training algorithms is the torch module in Python utilizing GPU with cuda()
- SVHN dataset is available in torchvision module: torchvision.datasets.SVHN
- CNN models are defined by specifying layers inside of a sequential container - torch.nn.Sequential()

CNN architecture

- Architecture is slightly modified from website:

https://codetolight.wordpress.com/2017/11/30/getting-started-with-pytorch-for-deep-learning-part-3-5-pytorch-sequential/

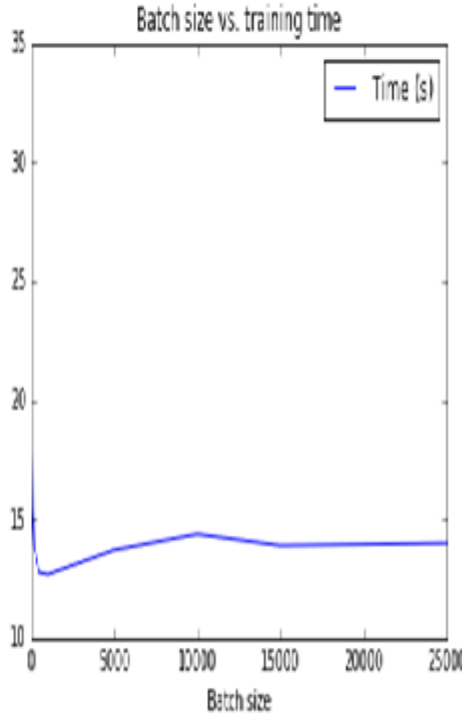
- Model has 2 convolutional layers with 20 and 50 kernels, respectively, each of which is directly followed by a max pooling layer. A 2-d dropout layer and 2 max pooling layers
- Flatten is not part of the torch module and has to be defined using .view to reshape the 2-d tensor prior to the fully connected layer
- Also need to set model.train() for training and model.eval() for testing. Dropout is only during training.

```
class Flatten(torch.nn.Module):
    def forward(self, input):
        return input.view(input.size(0), -1)

model = torch.nn.Sequential(
        torch.nn.Conv2d(3,20,5),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Conv2d(20,50,5),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Dropout2d(0.25),
        Flatten(),
        torch.nn.Linear(1250, 500),
        torch.nn.ReLU(),
        torch.nn.Linear(500, 10),
        torch.nn.LogSoftmax()
        ).cuda()
```

Selecting mini-batch size

Batch size	*Time (s)*
16	32.1282
50	18.3828
100	15.3778
200	13.7388
500	12.7581
1000	12.6773
5000	13.6990
10000	14.3668
15000	13.8882
25000	13.9889

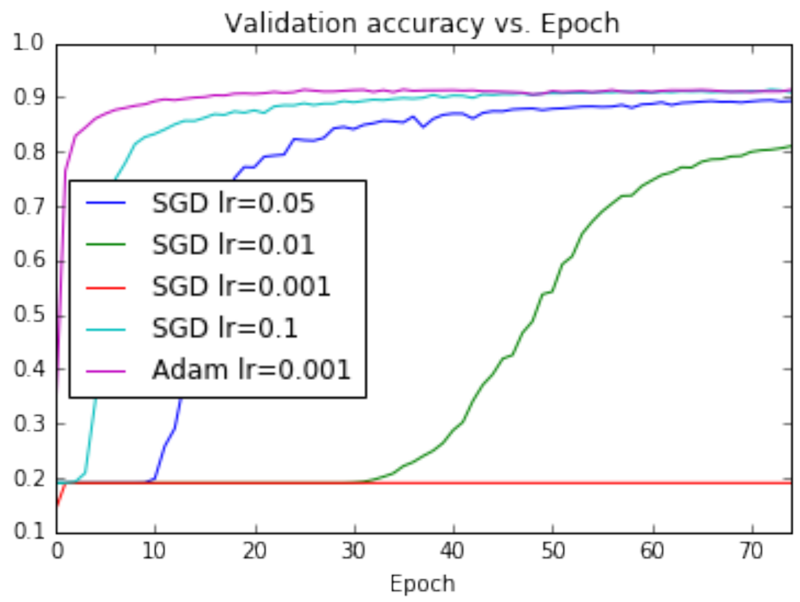


The computation times are based on 2 epochs of training using SGD.

Computation time decreases as mini-batch size increases until 1000, and then increases slightly.

Sizes larger than 30000 resulted in a memory error.

Selecting learning rate and optimizer



- Training set of 73,257 images split into 7000 validation and 66257 for training.
- Adam with learning rates of 0.1, 0.05, or 0.01 result in no improvement in the validation set accuracy
- SGD with learning rate of 0.1 performs better on validation set than 0.01, 0.05, or 0.001
- Adam with learning rate of 0.001 performs better after fewer epochs than SGD with learning rate of 0.1

Results

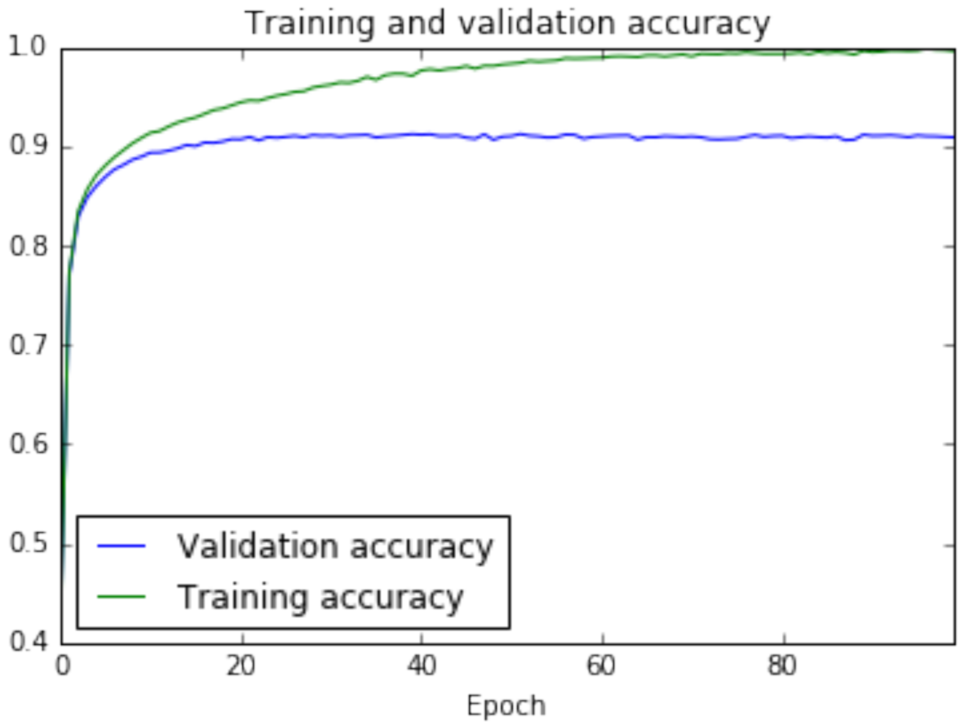
Digit	*Precision*	*Accuracy*	*F1-score*	*Support*
0	0.89	0.91	0.90	1744
1	0.92	0.94	0.93	5099
2	0.94	0.92	0.93	4149
3	0.87	0.84	0.85	2882
4	0.88	0.93	0.90	2523
5	0.91	0.90	0.90	2384
6	0.88	0.88	0.88	1977
7	0.92	0.89	0.91	2019
8	0.90	0.83	0.86	1660
9	0.79	0.89	0.84	1595
Avg/Total	0.90	0.90	0.90	26032

	Predicted									
Actual	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	1590	22	10	19	9	7	38	5	11	33
1	43	4780	47	32	105	11	15	47	7	12
2	14	50	3823	70	58	15	17	53	21	28
3	18	95	43	2407	25	77	20	12	36	149
4	16	60	29	19	2334	6	10	15	11	23
5	7	15	19	84	25	2135	54	4	7	34
6	41	16	11	27	33	44	1747	5	31	22
7	5	118	41	31	16	3	5	1791	2	7
8	22	13	10	51	29	27	73	4	1373	58
9	28	18	40	21	19	18	11	3	20	1417

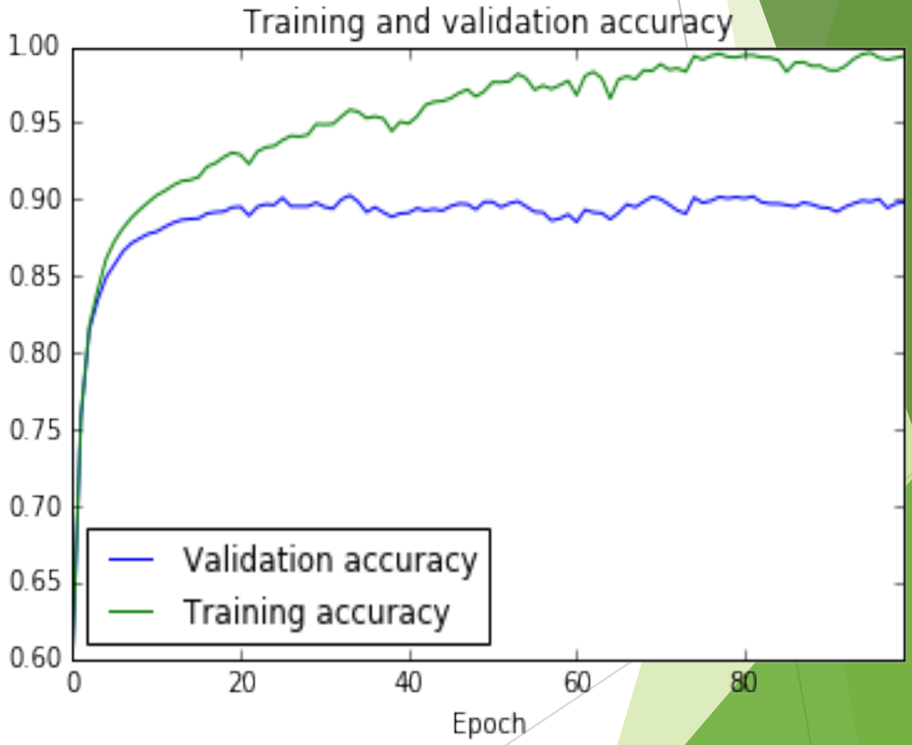
The CNN model took exactly 5 minutes to train on GPU
Overall accuracy in the test set was 89.88%.

Evaluating the effect of removing the dropout layer

With 2D dropout (89.8% accuracy)

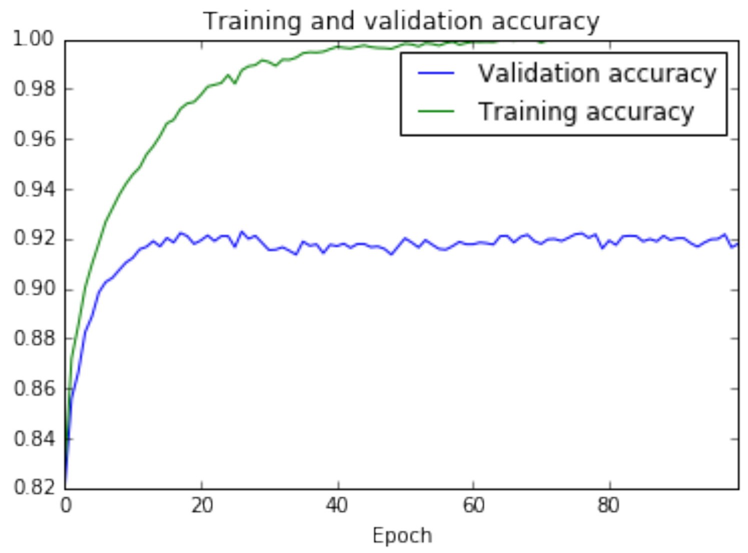


Without 2D dropout: (88.61% accuracy)



Without dropout the accuracy of prediction in both the training and validation sets fluctuates more from one training epoch to the next. The final trained model's accuracy on the test set is slightly less without dropout (88.61% vs. 89.88%)

Evaluating the effect of normalizing images

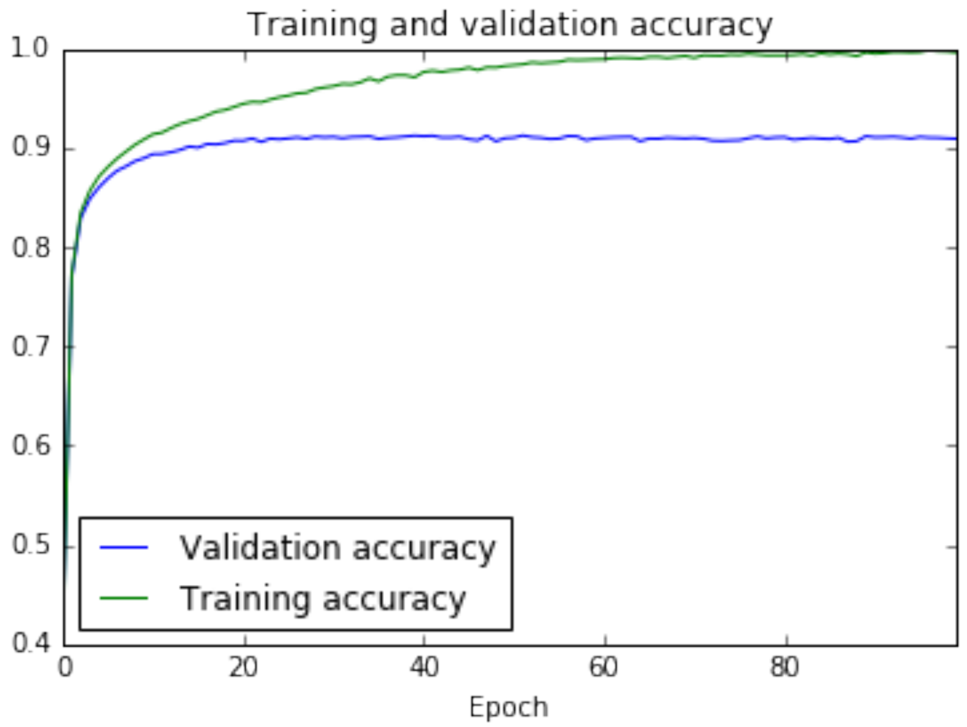


```
torchvision.transforms.Normalize((0.43768218, 0.44376934, 0.47280428), (0.1980301, 0.2010157, 0.19703591))
```

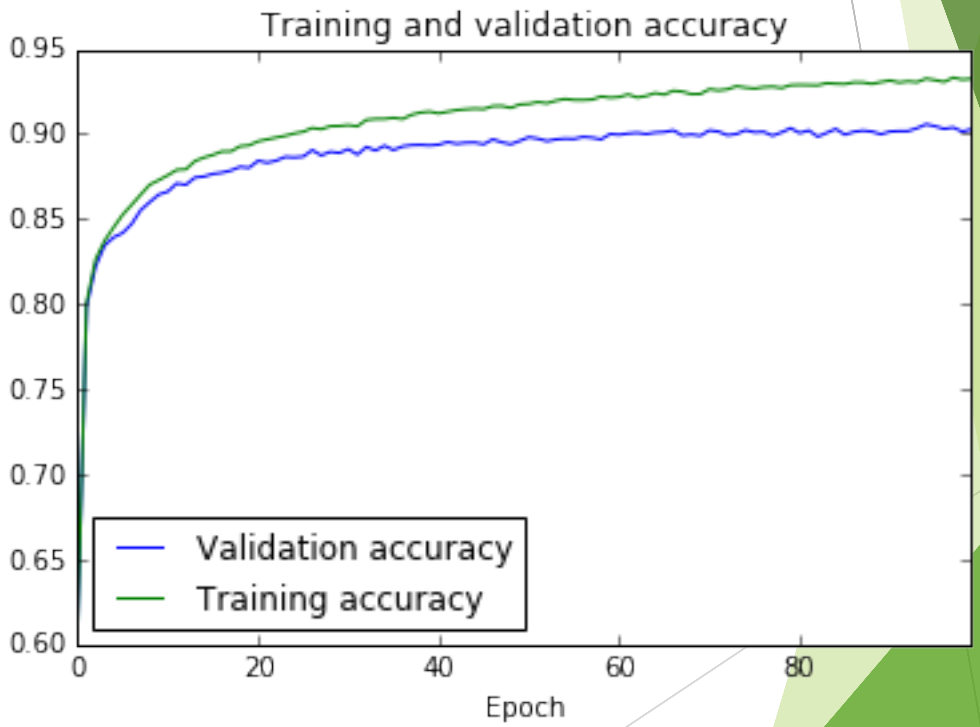
- Normalizing the images based on the mean and standard deviation in each channel in the training set increased test accuracy from 89.88% to 91.18%.

Overfitting

657,080 parameters (91.18% accuracy)



19,590 parameters (89.8% accuracy)



. Reducing the second convolution layer from 50 to 10 kernels and the fully connected layer to 50 neurons results in a model with 19,590 parameters. There is less overfitting but the test accuracy is reduced to 89.8%.

Varying the number of convolutional layers

# layers	Time (s)	Accuracy
1	71.077	0.83340
2	375.280	0.88303
3	453.275	0.90565
4	508.127	0.89705
5	538.272	0.89866

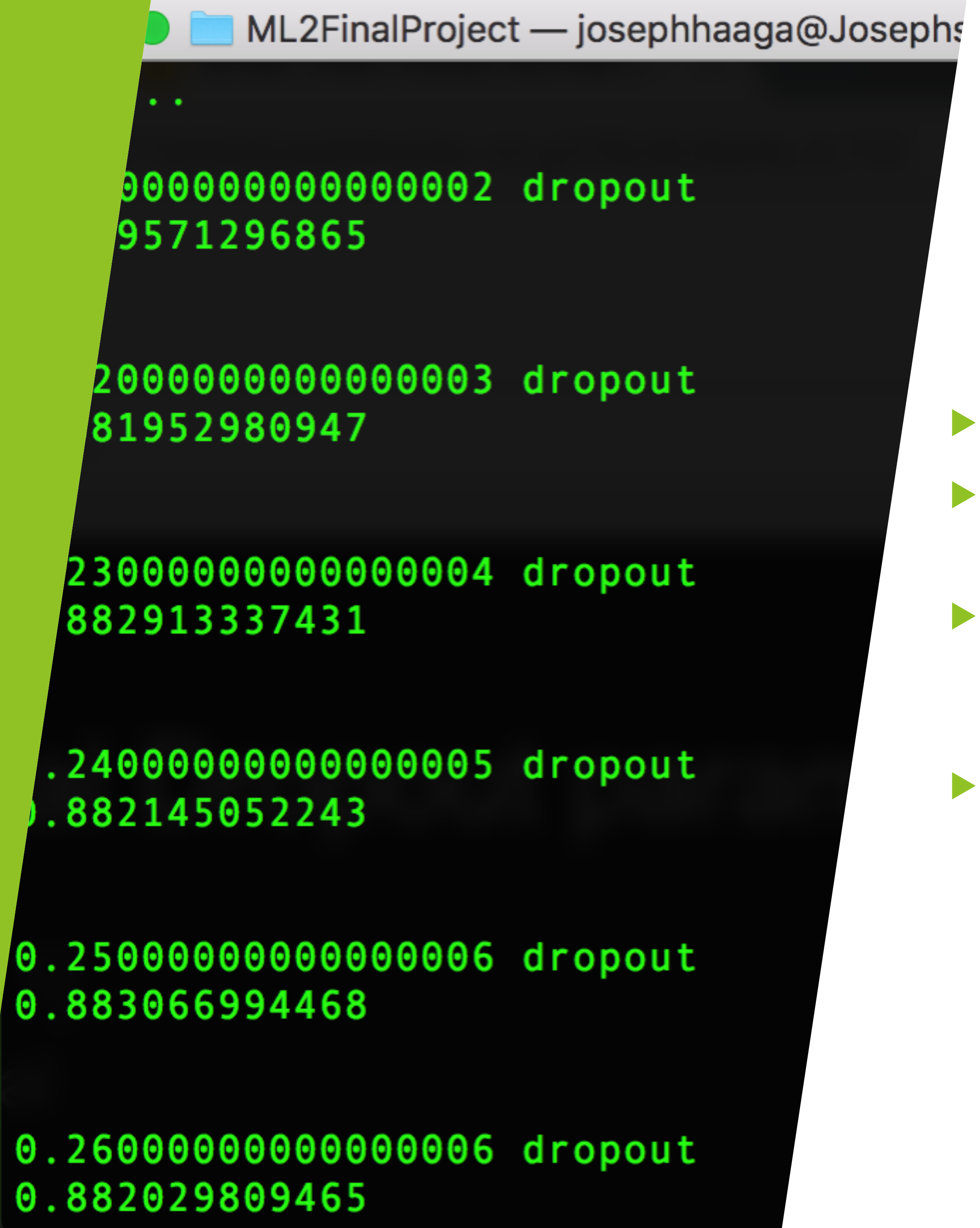
- The table is based on varying numbers of convolutional layers with 20 kernals of size 5 and stride 1 followed by a max pooling layer of size 2 with stride 1.
- The number of neurons in the fully connected layer was adjusted so that each model has roughly 1.4 million parameters to train.
- Training time increases with number of layers, but accuracy is highest for 3 layers.

Adding additional convolutional layer to CNN

- After adding 3rd convolutional layer with 50 kernals and change stride to 1 and 700 neurons in FC layer, accuracy increased to 91.59%
- Approximately 6 minutes to train model with GPU
- Images of 1 were classified most accurately (95%) while images of 0 and 8 were classified least accurately at 89%. The cross-tabulation above shows how images of each digit were classified in the test set. Both 0 and 8 were most often misclassified as 6.

	Predicted									
Actual	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	1557	21	17	22	6	7	46	10	17	41
1	27	4823	43	59	38	18	9	46	32	4
2	5	30	3932	72	14	15	13	32	26	10
3	9	43	48	2599	9	54	15	14	39	52
4	9	85	29	32	2293	12	18	13	16	16
5	7	13	17	113	14	2137	48	4	17	14
6	20	14	16	29	10	45	1778	4	49	12
7	5	99	50	24	5	2	2	1820	6	6
8	13	13	13	29	12	14	57	4	1476	29
9	23	16	52	17	9	6	11	5	27	1429

```
model = torch.nn.Sequential(
        torch.nn.Conv2d(3,20,5),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Conv2d(20,50,5),
        torch.nn.MaxPool2d(2, stride=1),
        torch.nn.ReLU(),
        torch.nn.Conv2d(50,50,5),
        torch.nn.MaxPool2d(2, stride=1),
        torch.nn.ReLU(),
        torch.nn.Dropout2d(0.25),
        Flatten(),
        torch.nn.Linear(800, 700),
        torch.nn.ReLU(),
        torch.nn.Linear(700, 10),
        torch.nn.LogSoftmax()
        ).cuda()
```



```
..

000000000000002 dropout
9571296865


2000000000000003 dropout
81952980947


23000000000000004 dropout
882913337431


.24000000000000005 dropout
).88214505224З


0.25000000000000006 dropout
0.883066994468


0.26000000000000006 dropout
0.882029809465
```

Searching for the best Dropout

- Looking for a probability [0, 1]
- Iterate thru range of potential values
- Hold model constant, while varying just the dropout layer probability
- P=0.47 performed the best

Tweaking the network

```
model = torch.nn.Sequential(
        torch.nn.Conv2d(3,20,5,padding=2),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Conv2d(20,40,5,padding=2),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Conv2d(50,40,5,padding=2),
        torch.nn.MaxPool2d(2, stride=2),
        torch.nn.ReLU(),
        torch.nn.Dropout2d(0.47),
        Flatten(),
        torch.nn.Linear(640, 320),
        torch.nn.ReLU(),
        torch.nn.Linear(320, 10),
        torch.nn.LogSoftmax()
        ).cuda()
```

- Add padding=2 to each Conv layer
- Change MaxPool stride to 2, preventing redundancies
- Update dropout layer with the ideal parameter we found previously
- Ultimately resulted in 92.356% accuracy on test set!

Summary

- We determined that the optimal way to train a CNN on the SVHN data with Pytorch and GPU is with Adam and a learning rate of 0.001 and a mini-batch size of 1000.
- In addition, normalization of images within each color channel improves prediction.
- Dropout layer improved performance
- The best CNN model has 3 convolution layers and 656,830 parameters and is able to classify 91.6% of the test images correctly.
- Additional parameters that might result in improved performance include padding in the convolution layer.

Future work

- More Conv layers
  - Different kernel configurations
- More epochs
- YOLO/SSD?
- Capsule networks

References

- Diederik P. Kingma, Jimmy Ba Adam: A Method for Stochastic Optimization. 3rd International Conference for Learning Representations, San Diego, 2015
- http://ufldl.stanford.edu/housenumbers/
- Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.
- https://codetolight.wordpress.com/2017/11/30/getting-started-with-pytorch-for-deep-learning-part-3-5-pytorch-sequential/
- pytorch.org