HW2 Cifar-10 Classification with MLP and CNN

计11 张凯文 2020080094

Project environment

```
Python 3.9.6
Pytorch
Macbook Pro Apple M1 chip
```

How does self training work

The `self.training` attribute is a built-in attribute of PyTorch's nn.Module class. Indicating whether the current model is in `training` mode or `testing` mode. The distinction between training and testing is crucial for certain layers and operations, especially dropout or batch normalization, as they behave differently depending on the mode. The differentiation between training and evaluation modes is because of the need to regularize the model during training and the subsequent difference in behavior required during inference.

`Dropout` : During training, dropout sets a fraction of the inputs to zero with a certain probability to prevent overfitting. During testing, dropout doesn't set any inputs to zero but might scale the outputs.

`Batch Normalization` : During training, batch normalization computes the mean and variance for each feature from the current mini-batch. During testing, it uses a running average of these mean and variance values computed during training.

CNN and MLP

Hyperparameters

Parameter	Value
batch_size	100

Parameter	Value
num_epochs	50
learning_rate	0.001
drop_rate	0.5

CNN

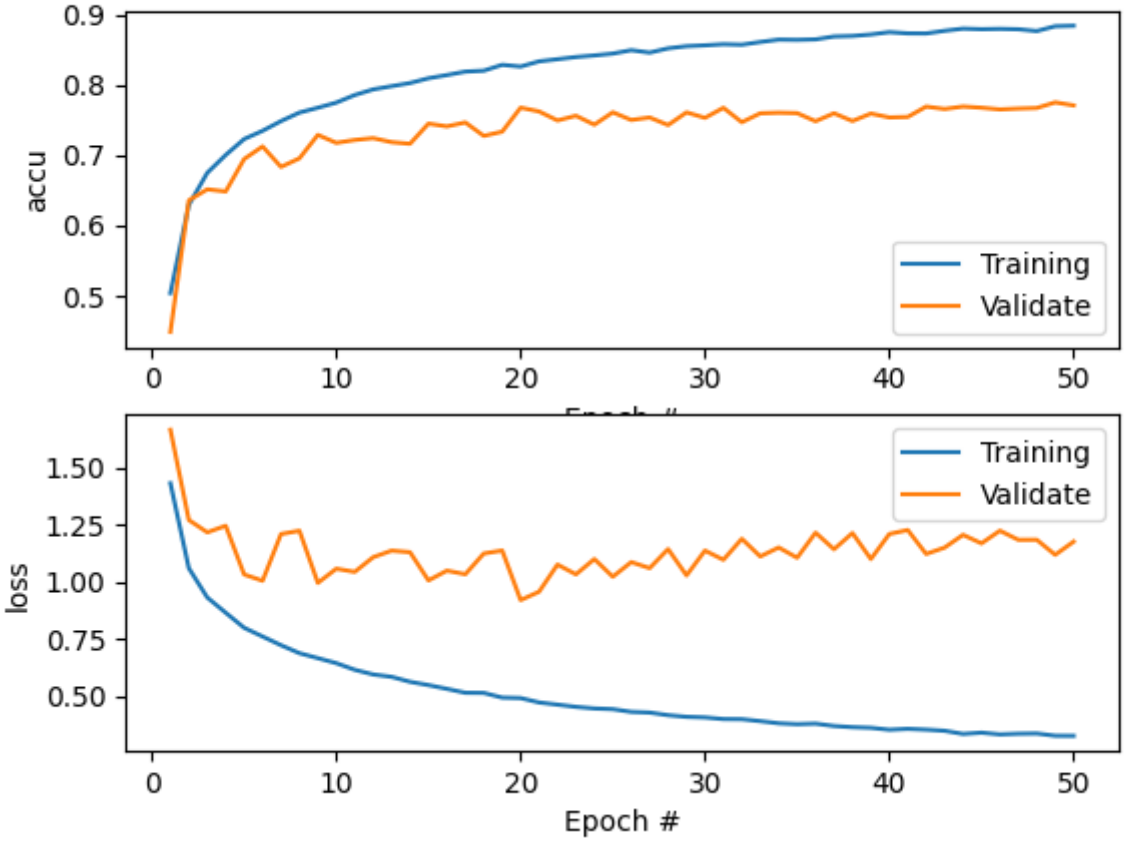
Model Structure:

```
Model(
    (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (bn1): BatchNorm2d()
    (relu1): ReLU()
    (dropout1): Dropout()
    (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1
    (bn2): BatchNorm2d()
    (relu2): ReLU()
    (dropout2): Dropout()
    (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    (fc): Linear(in_features=8192, out_features=10, bias=True)
    (loss): CrossEntropyLoss()
)
```

Results:

```
Total duration: 4871.5s
training loss:                0.32722435299307107
training accuracy:            0.8837000001966954
validation loss:              1.1768217903375626
validation accuracy:          0.7705999994277954
best validation accuracy:     0.774899999499321
test loss:                    1.1350669845938683
test accuracy:                0.7692000013589859
```

Results Graphing:



MLP

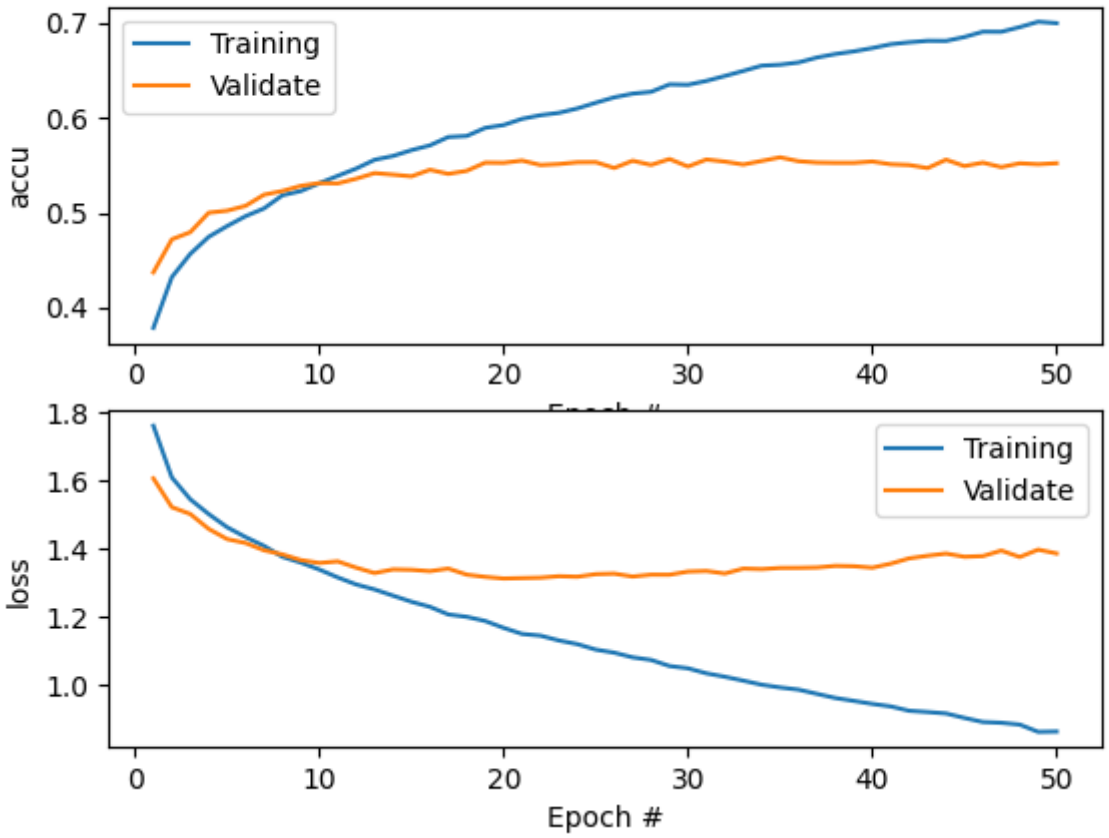
Model Structure:

```
Model(
    (fc1): Linear(in_features=3072, out_features=512, bias=True)
    (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
    (relu): ReLU()
    (dropout): Dropout(p=0.5, inplace=False)
    (fc2): Linear(in_features=512, out_features=10, bias=True)
    (loss): CrossEntropyLoss()
)
```

Results:

```
Total duration: 125s
learning rate:                  0.001
training loss:                  0.8642372807860375
training accuracy:              0.6992000035941601
validation loss:                1.3864549398422241
validation accuracy:            0.5519999960064887
best validation accuracy:       0.558199998140335
test loss:                      1.307091037631035
test accuracy:                  0.5539999967813491
```

Results Graphing:



Training loss and Validation loss differences

Data:

`Training Data` : The model is trained on this data, meaning the weights are adjusted to minimize the error for this specific set of examples.

`Validation Data` : This is a separate set of examples that the model hasn't seen during training. It's used to check the model's performance on unseen data.

Overfitting:

As the model trains, it tries to minimize the training loss. If you have a complex model (e.g., a deep neural network), it might fit the training data very well, capturing even its noise and outliers. In such cases, the model performs poorly on new, unseen data because it's too tailored to the training data. This phenomenon is called overfitting. When a model overfits, training loss will be low, but validation loss will be high.

Underfitting:

If the model is too simple to capture the underlying patterns in the data, it might perform poorly both on training and validation data. This scenario is called underfitting. Both training and validation losses will be high, but they may still be different.

How differences can help Hypertuning

Learning Rate Tuning:

A high learning rate will cause the losses to oscillate wildly, while a learning rate that's too low will cause a slow convergence. By observing how quickly or slowly training and validation losses decrease, we can adjust the learning rate accordingly.

Detecting Overfitting:

If the training loss is decreasing or very low, but the validation loss starts increasing, this would most likely mean there is overfitting. Indicating that the model is becoming too specialized to the training data and not generalizing well to new data. Using this information, we can tune dropout rate to prevent overfitting.

Final Results

`CNN`: Best validation accuracy of 77.49%

`MLP`: Best validation accuracy of 55.82%

There was a significant difference in the time it took to complete 50 epochs for `CNN` and `MLP`, this difference most likely originates from the depth of `CNN` compared to that of `MLP` as well as the complexity of the operations.

Batch Normalization

BatchNorm

Batch Normalization is a normalization technique that is implemented to improve the training of deep neural networks.

`CNN` : BatchNorm is applied to the feature maps. This means that the normalization is performed separately for each feature map, but jointly over all the locations in that feature map.

`MLP` : BatchNorm is applied to the activations of the neurons in a similar manner, but without the spatial dimensions.

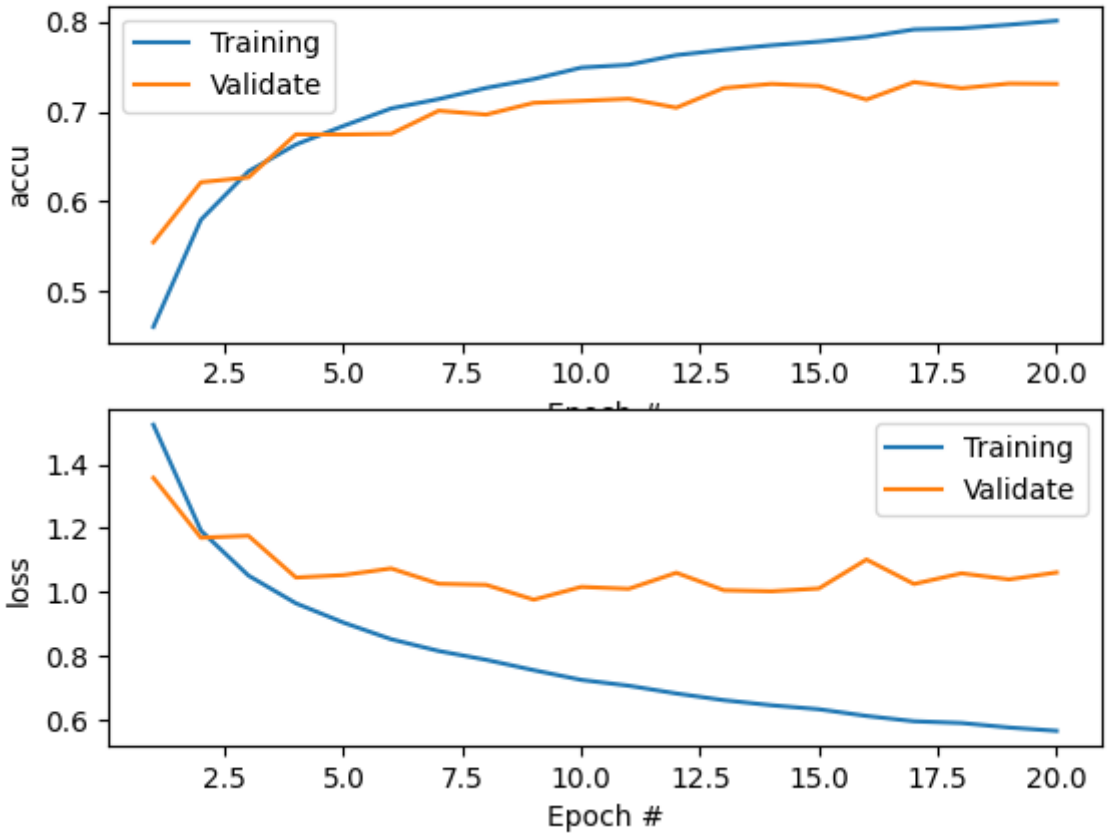
CNN without BatchNorm

Validation accuracy: 77.49% -> 73.10%, `4.39% decrease`

Results:

```
training loss:                0.5655300851166248
training accuracy:            0.8014249996840954
validation loss:              1.0613271754980087
validation accuracy:          0.7310000002384186
best validation accuracy:     0.7330000001192093
test loss:                    1.0374818515777589
test accuracy:                0.7259000021219254
```

Result Graphing:



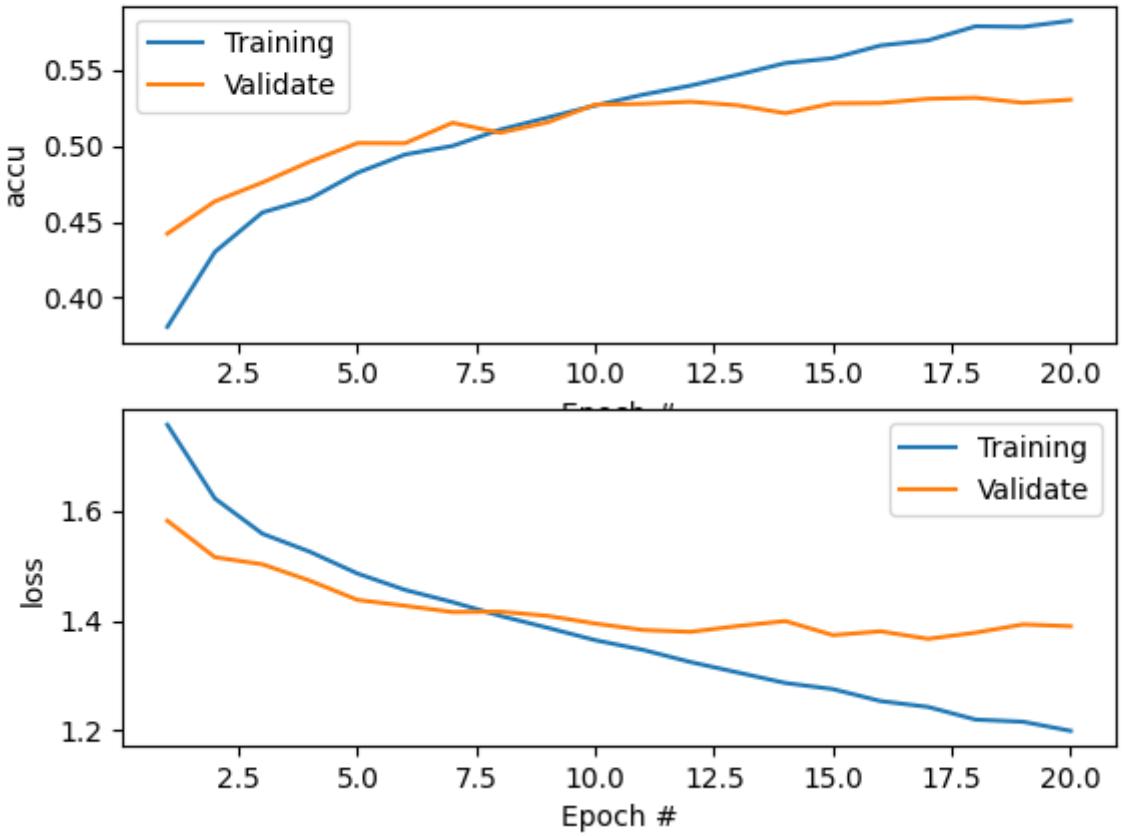
MLP without BatchNorm

Validation accuracy: 55.82% -> 53.18%, `2.64% decrease`

Results:

```
training loss:                  1.1986435125768184
training accuracy:              0.5825749968737364
validation loss:                1.389707338809967
validation accuracy:            0.5304999974370003
best validation accuracy:       0.5317999929189682
test loss:                      1.370942565202713
test accuracy:                  0.5260999953746796
```

Result Graphing:



Analysis

From the results it is significant that BatchNorm seems to have a valuable increase in accuracy for the `CNN` network, while BatchNorm does not seem to effect the accuracy of `MLP` network as much.

Dropout

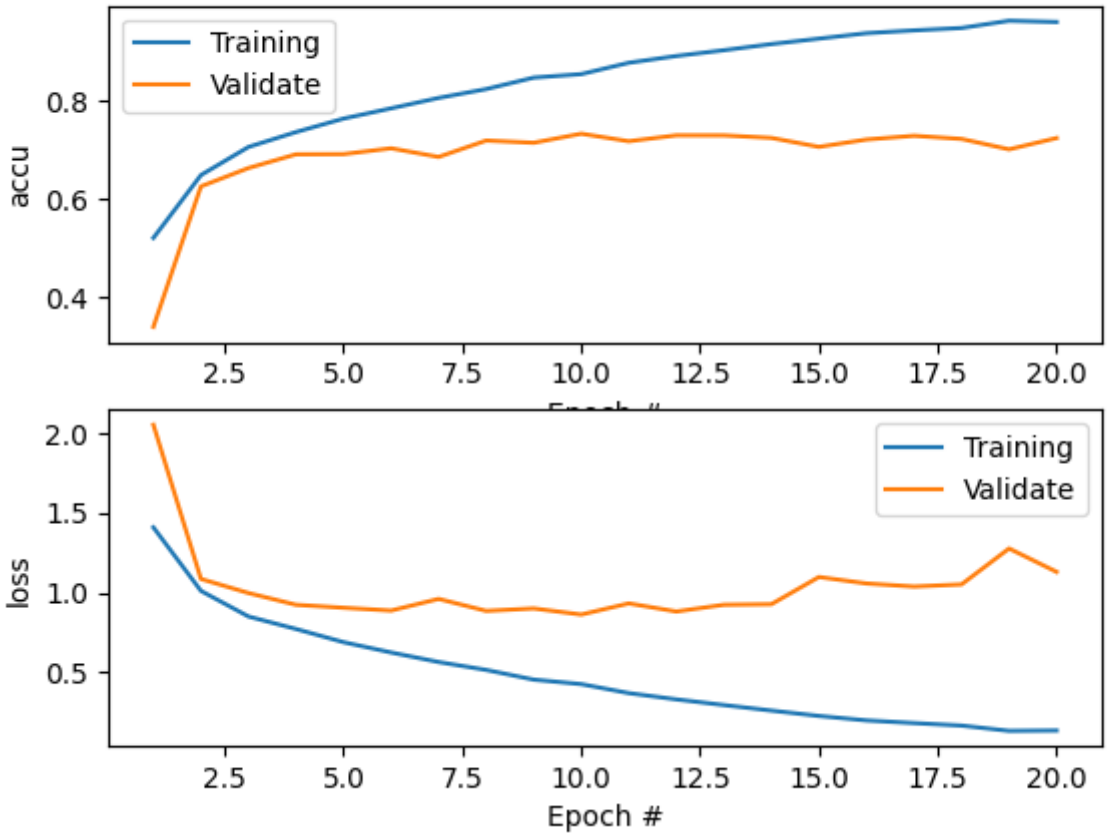
CNN without Dropout

Validation accuracy: 77.49% -> 75.26%, `2.23% decrease`

Results:

```
training loss:                  0.13074913169257343
training accuracy:              0.9590250042080879
validation loss:                1.130010039806366
validation accuracy:            0.7238000017404557
best validation accuracy:       0.7526000010967255
test loss:                      0.8678795498609543
test accuracy:                  0.7284000027179718
```

Result Graphing:



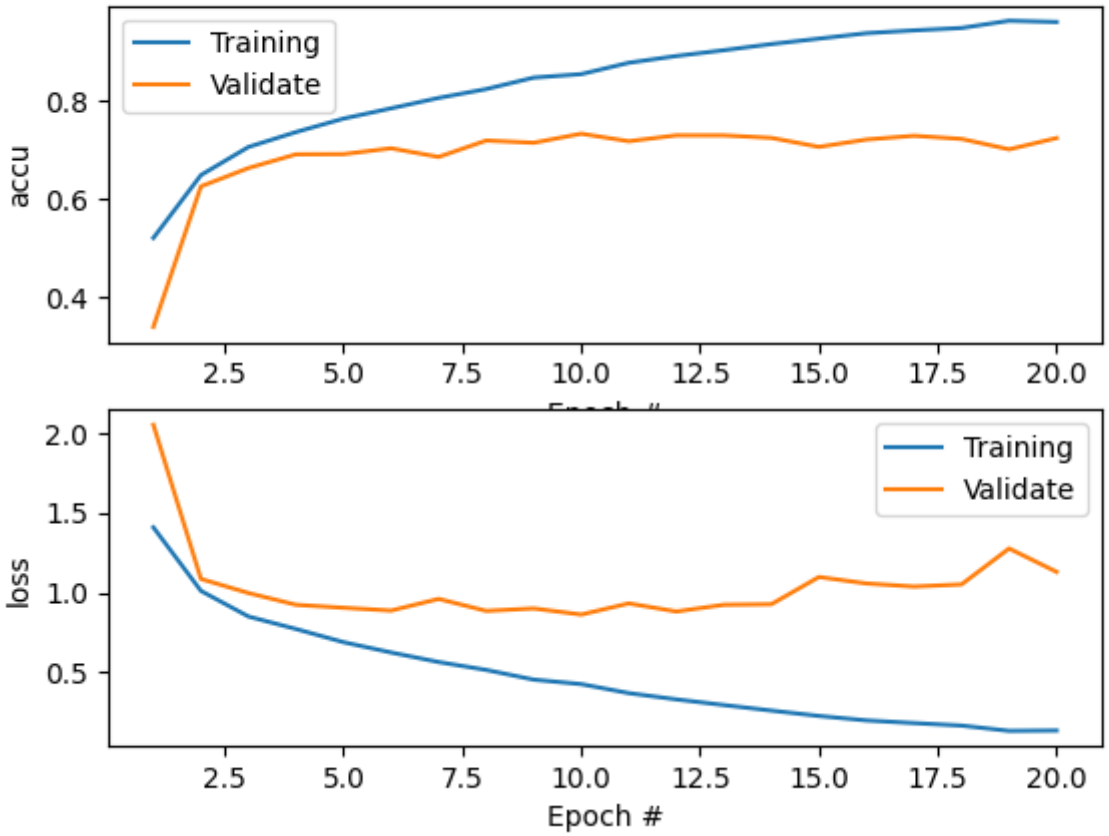
MLP without Dropout

Validation accuracy: 55.82% -> 53.97%, `1.85% decrease`

Results:

```
training loss:                  0.567880787923932
training accuracy:              0.8114999975264072
validation loss:                1.674667031764984
validation accuracy:            0.5182999962568283
best validation accuracy:       0.53969999730587
test loss:                      1.4035942482948303
test accuracy:                  0.5377999982237816
```

Result Graphing:



Analysis

Overall there does not seem to be a significant change when choosing whether or not to implement the dropout layer.

Hyperparameter tuning

All hyperparameters modified below were using the `CNN` network with all other parameters being the default values. The `epoch` size was chosen to be `20 epochs`.

batch size

Modified through changing the `batch_size` variable

batch size	validation accuracy
10	75.58%
100	78.68%
1000	67.12%
10000	50.83%

From the tuning we can observe that there is not a significant different when the batch size is relatively small. But once it exceeds 1000, there seems to be a significant decrease in accuracy. This is most likely because the batch size is using too much of the storage and decreasing the processing speed of the network.

dropout rate

Modified through changing the `drop_rate` variable

dropout rate	validation accuracy
0.0	73.45%
0.2	76.39%
0.4	72.93%
0.6	70.10%
0.8	51.74%
1.0	10.09%

From the tuning above it is clear that there is increase in accuracy when the dropout rate remains low. But once it becomes too how the network is unable to learn and therefore causes a massive decrease in accuracy.