BBM 418 - Computer Vision Laboratory

HACETTEPE UNIVERSITY
Department of Computer Engineering

Programming Assignment-3
Spring 2010-2021
Doç. Dr. Nazlı İkizler Cinbiş

Image Classification with Convolutional Neural Networks

About CNN:

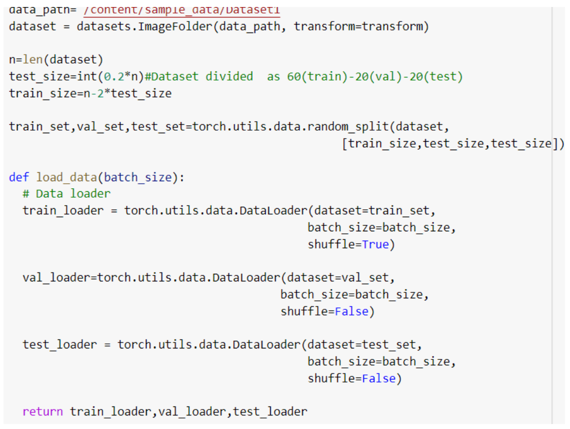
CNN is very useful if we have large dimensional feature space. Basically Cnn extract features of images, and without loosing its characteristics, squeeze it lower dimensions.

There are some edge detection filters provided. Every edge has vertical and horizontal lines in an image and convolution with stride and padding values are used.

About Pytorch:

Enables fast,flexible, efficient production. Some features are more like on numpy and pandas. Active community helps me to understand the deep structure of Pytorch.

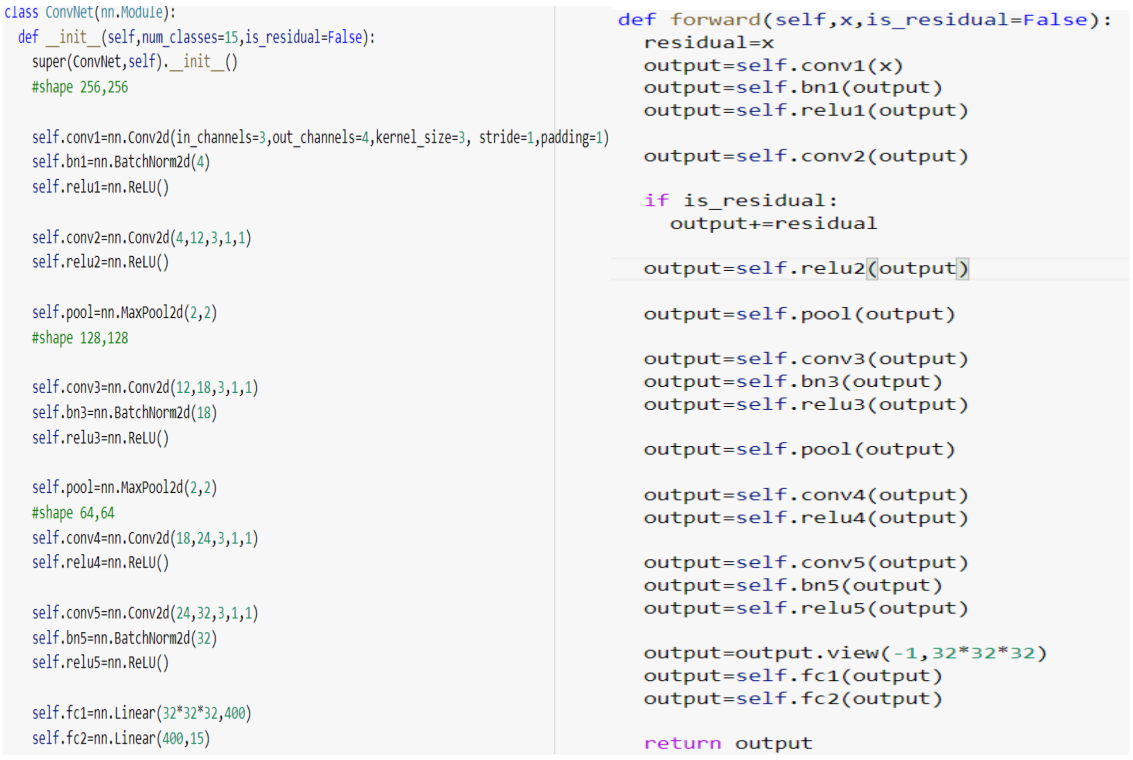
About Dataset:



```
data_path= /content/sample_data/Dataset1
dataset = datasets.ImageFolder(data_path, transform=transform)

n=len(dataset)
test_size=int(0.2*n)#Dataset divided  as 60(train)-20(val)-20(test)
train_size=n-2*test_size

train_set,val_set,test_set=torch.utils.data.random_split(dataset,
                                                          [train_size,test_size,test_size])

def load_data(batch_size):
  # Data loader
  train_loader = torch.utils.data.DataLoader(dataset=train_set,
                                             batch_size=batch_size,
                                             shuffle=True)

  val_loader=torch.utils.data.DataLoader(dataset=val_set,
                                         batch_size=batch_size,
                                         shuffle=False)

  test_loader = torch.utils.data.DataLoader(dataset=test_set,
                                            batch_size=batch_size,
                                            shuffle=False)

  return train_loader,val_loader,test_loader
```

I had used Pytroch libraries to read and store images. Firstly data path is specified. Then extract the dataset with ImageFolder with specified transforms.Compose parameters.(transform to resize 128*128, give tensor, and to take accurate values normalization was done). After whole dataset divided 60-20-20(train-val-test datasets) with random_split function. In order to load these dataset with different batch size I wrote a function that takes batch size as paramter. It is called each time when we change the hyperparameters of the Model.

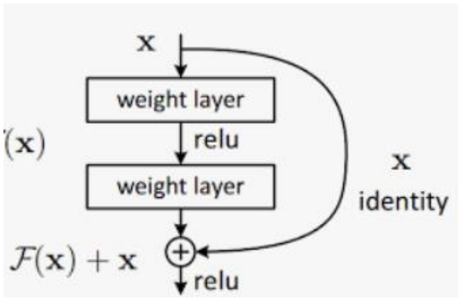
About Model parameters Part1:



```python
class ConvNet(nn.Module):
    def __init__(self,num_classes=15,is_residual=False):
        super(ConvNet,self).__init__()
        #shape 256,256

        self.conv1=nn.Conv2d(in_channels=3,out_channels=4,kernel_size=3, stride=1,padding=1)
        self.bn1=nn.BatchNorm2d(4)
        self.relu1=nn.ReLU()

        self.conv2=nn.Conv2d(4,12,3,1,1)
        self.relu2=nn.ReLU()

        self.pool=nn.MaxPool2d(2,2)
        #shape 128,128

        self.conv3=nn.Conv2d(12,18,3,1,1)
        self.bn3=nn.BatchNorm2d(18)
        self.relu3=nn.ReLU()

        self.pool=nn.MaxPool2d(2,2)
        #shape 64,64
        self.conv4=nn.Conv2d(18,24,3,1,1)
        self.relu4=nn.ReLU()

        self.conv5=nn.Conv2d(24,32,3,1,1)
        self.bn5=nn.BatchNorm2d(32)
        self.relu5=nn.ReLU()

        self.fc1=nn.Linear(32*32*32,400)
        self.fc2=nn.Linear(400,15)
```

```python
def forward(self,x,is_residual=False):
  residual=x
  output=self.conv1(x)
  output=self.bn1(output)
  output=self.relu1(output)

  output=self.conv2(output)

  if is_residual:
    output+=residual

  output=self.relu2(output)

  output=self.pool(output)

  output=self.conv3(output)
  output=self.bn3(output)
  output=self.relu3(output)

  output=self.pool(output)

  output=self.conv4(output)
  output=self.relu4(output)

  output=self.conv5(output)
  output=self.bn5(output)
  output=self.relu5(output)

  output=output.view(-1,32*32*32)
  output=self.fc1(output)
  output=self.fc2(output)

  return output
```

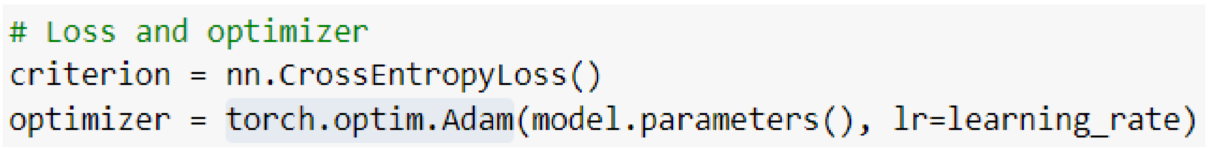
My model has 5 CNN and 2 fully connected layers. It starts with a Conv2d takes in_channels=3 (because initially we have 3 color channels as inputs). Out channel is 4 and filter is 3x3, with stride 1, padding 1. After that Batch normalization is used to improve the efficiency and improve the training speed. Then ReLU (Activation function) is applied. This is basically choosing max between zero and input of ReLU.



On the second Conv2d in_channels is 4 (out_channels of the previous Conv), out_channels is 12 and others parameters will be the same. Before taking the activation function(ReLU) we check wheter we need to consider the “RESIDUAL CONNECTION.” After ReLU activation function is applied then we take MaxPool2d with parameters 2,2 kernel and stride. We get half of its height and weight dimensions. Depth will remain the same.

It goes like that until fully connected layer.Fully connected layer takes input as 32*32*32 as the linear size of the output of last Conv2d. Then it creates 400 hidden layer size Neuron. Then second Fully connected layer gives 15 output class node.

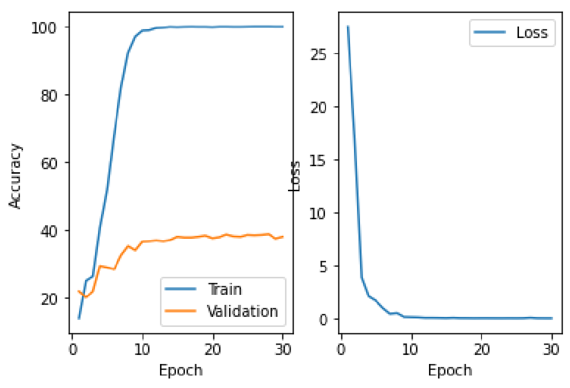
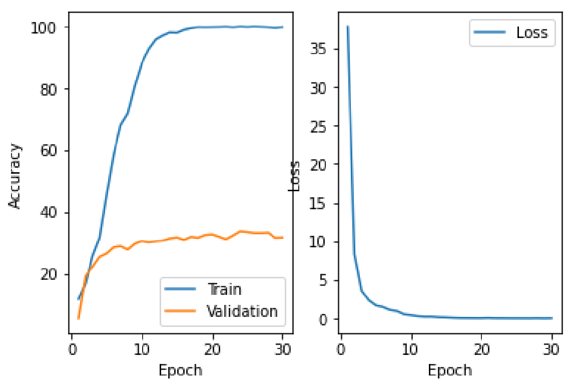
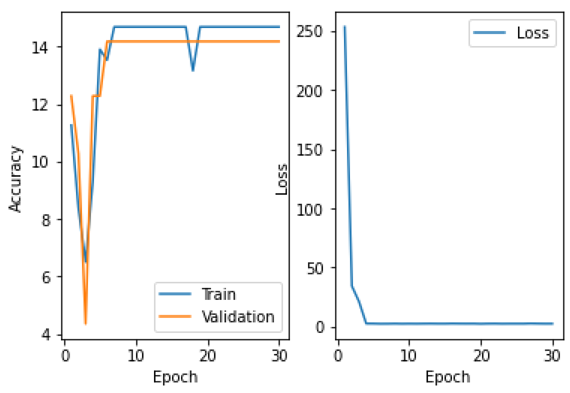
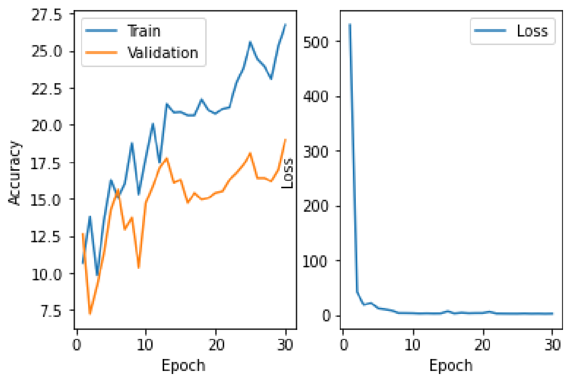
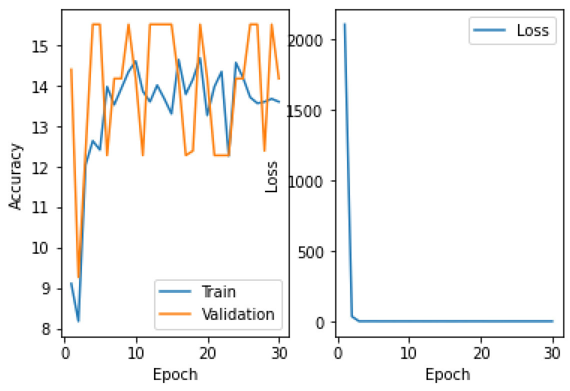
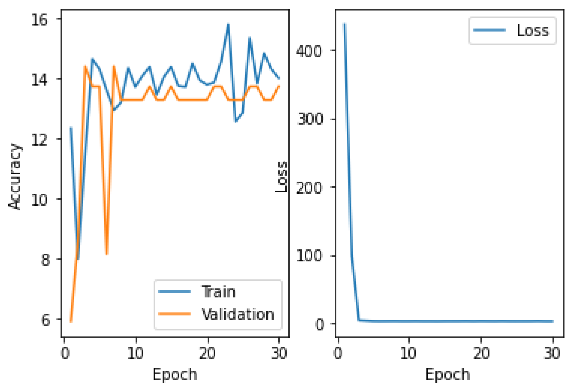
About loss function: CrossEntropyLoss is used. This criterion combines LogSoftmax and NLLLoss in one single class. It is useful when training a classification problem with *large* classes as inn our problem. About optimizer: My code implements Adam algorithm. An algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Invariant to diagonal rescaling of the gradients, and is well suited for that problem.

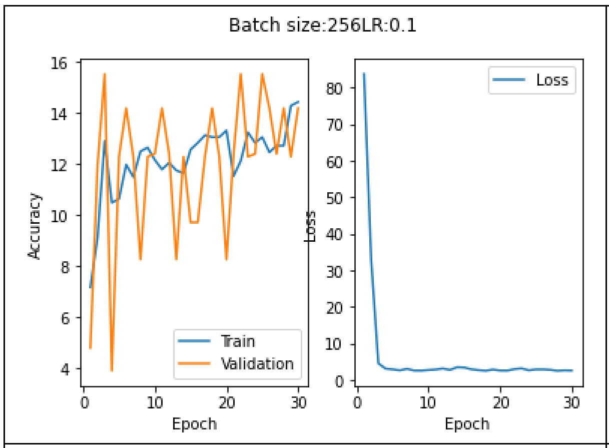


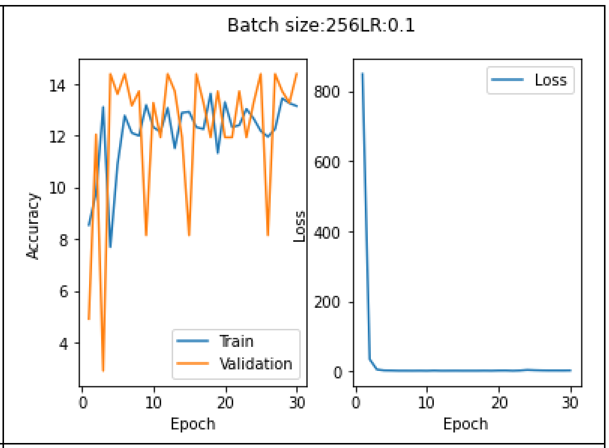
```python
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

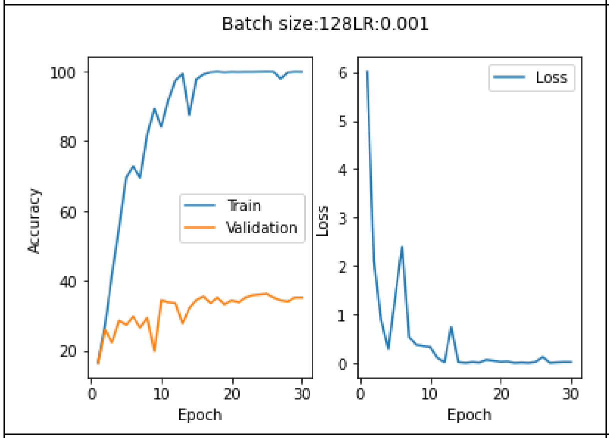
About Residual Connection: Many of them implemented but almost all them gives bad accuracy. So I only add one times before passing to first MaxPool2d. (As you can see the code above in the forward. If condition is added for residual connection)

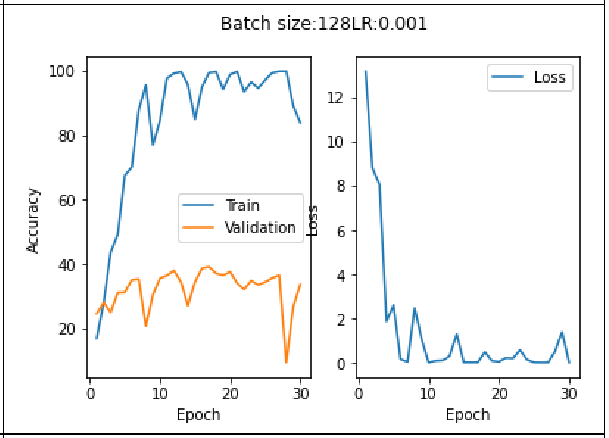
Training and Evaluating Model:

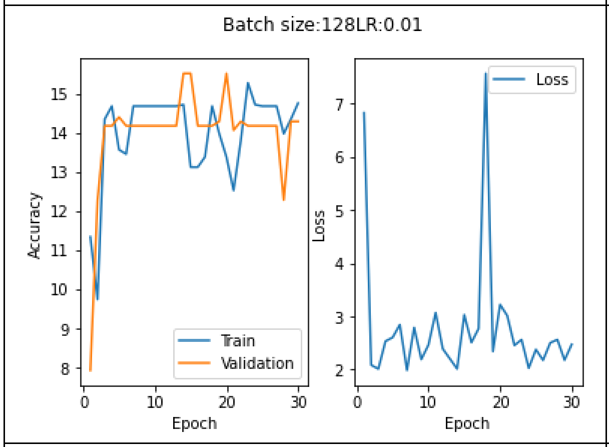
Has Residual =False	***Has Residual = True***
Batch size:256LR:0.001 	Batch size:256LR:0.001 
Batch size:256LR:0.01 	Batch size:256LR:0.01 
Batch size:256LR:0.05 	Batch size:256LR:0.05 

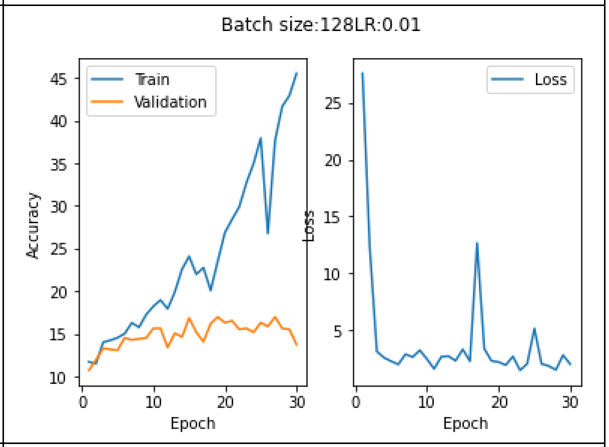

Batch size:256LR:0.1
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

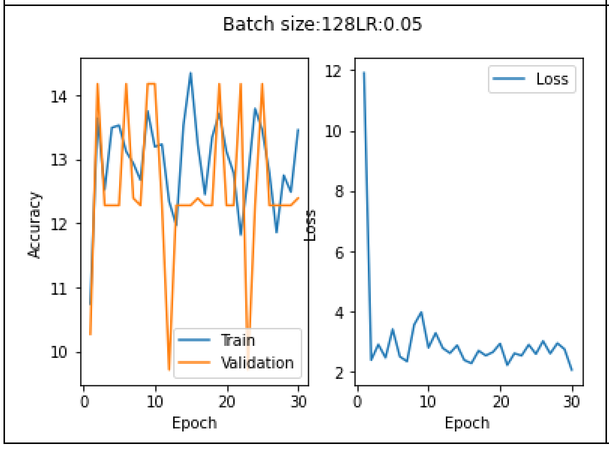

Batch size:256LR:0.1
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

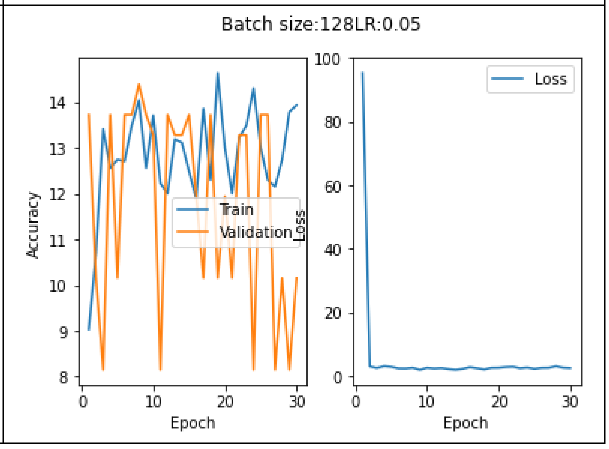

Batch size:128LR:0.001
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

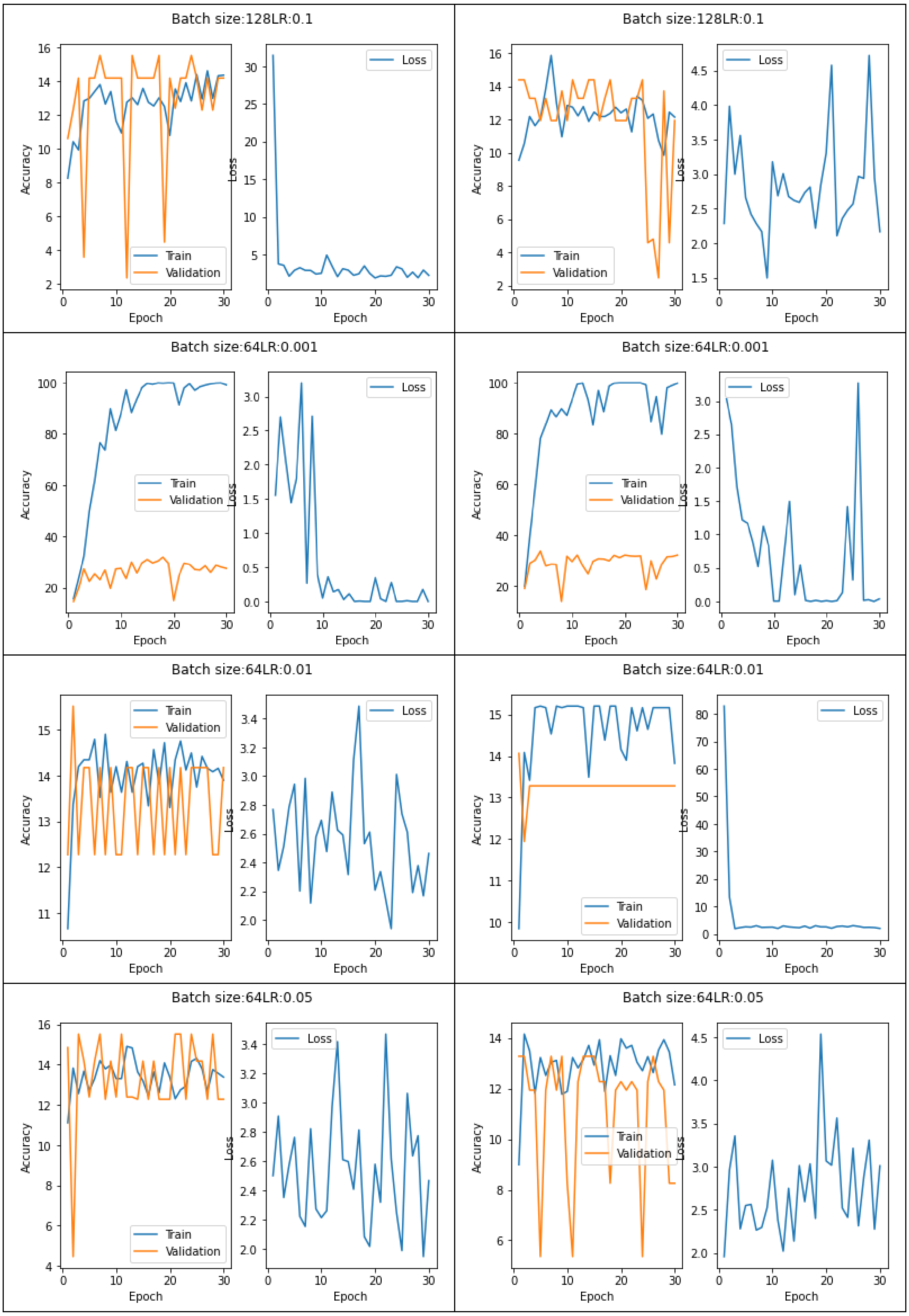

Batch size:128LR:0.001
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch


Batch size:128LR:0.01
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

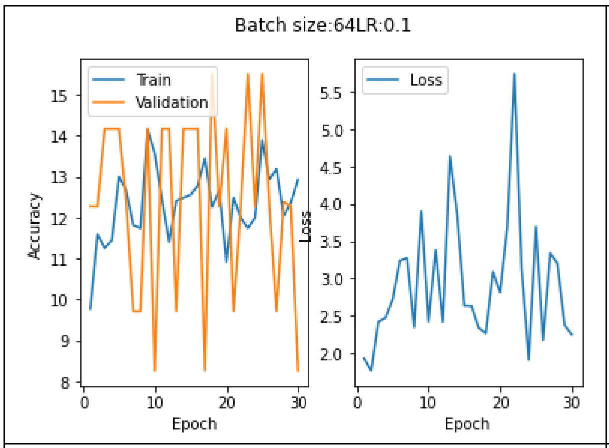

Batch size:128LR:0.01
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

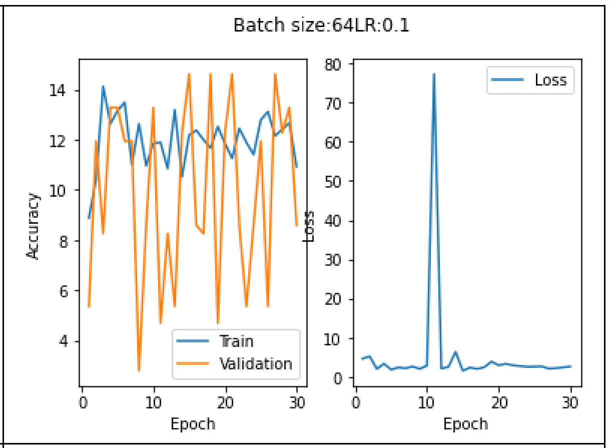

Batch size:128LR:0.05
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

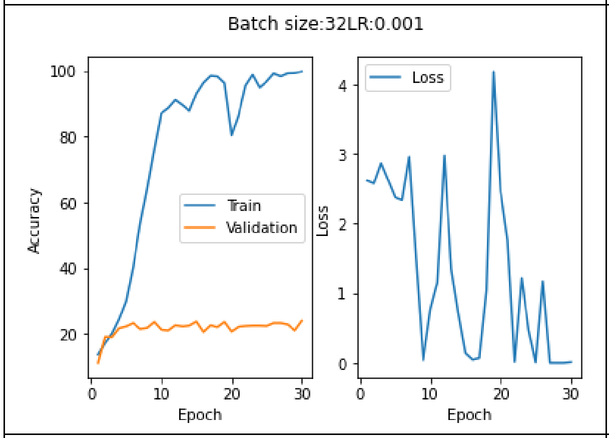

Batch size:128LR:0.05
Accuracy
Epoch
Train
Validation
Loss
Loss
Epoch

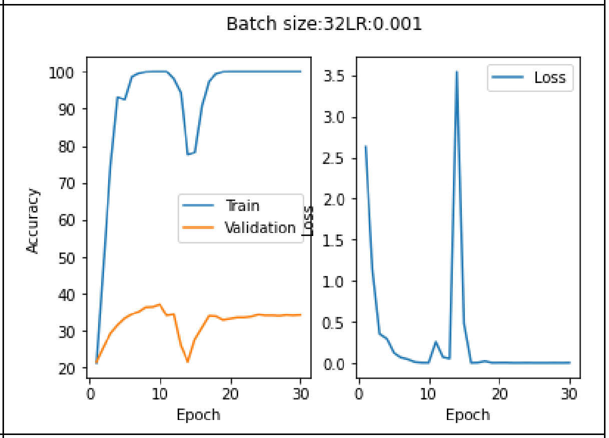


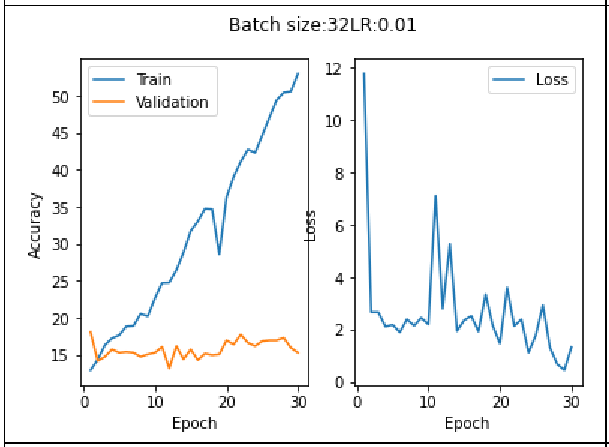
Batch size:128LR:0.1
Batch size:128LR:0.1
Batch size:64LR:0.001
Batch size:64LR:0.001
Batch size:64LR:0.01
Batch size:64LR:0.01
Batch size:64LR:0.05
Batch size:64LR:0.05
Accuracy
Loss
Epoch
Train
Validation
Loss

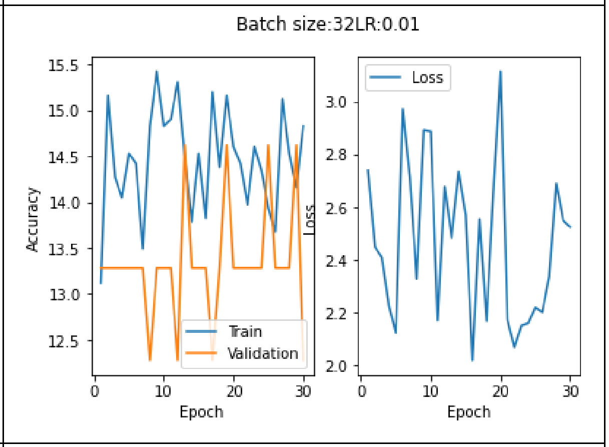

Batch size:64LR:0.1
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch

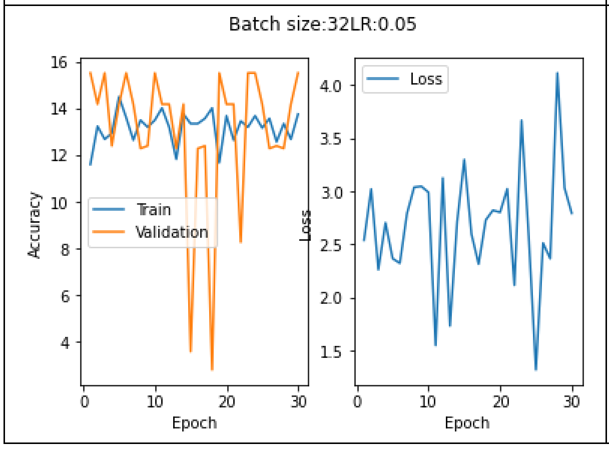

Batch size:64LR:0.1
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch

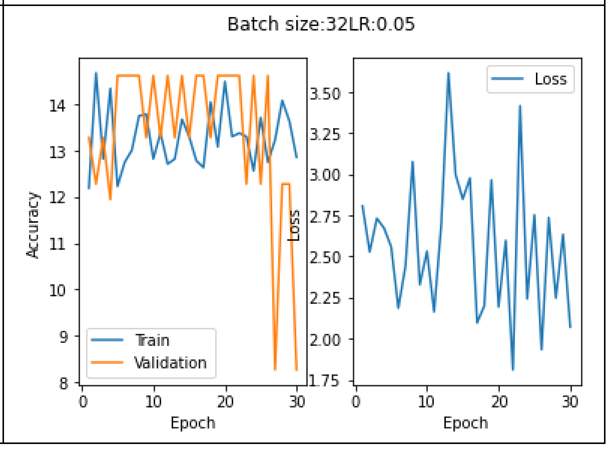

Batch size:32LR:0.001
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch

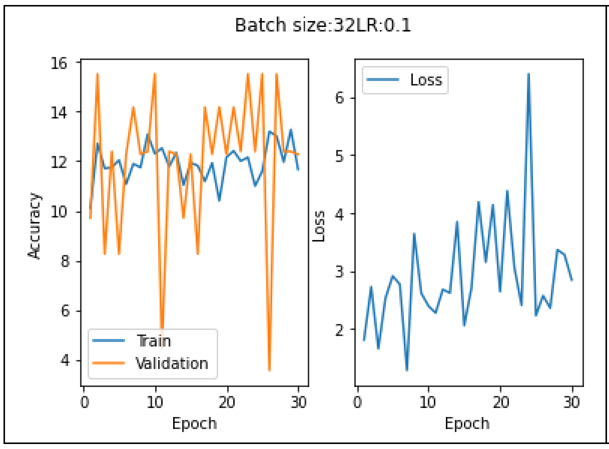

Batch size:32LR:0.001
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch

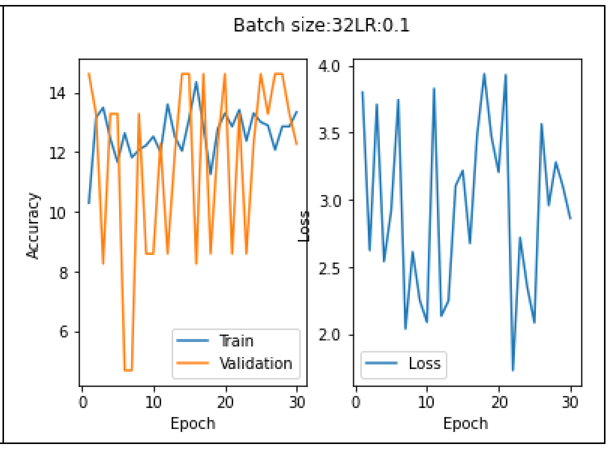

Batch size:32LR:0.01
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch


Batch size:32LR:0.01
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch


Batch size:32LR:0.05
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch

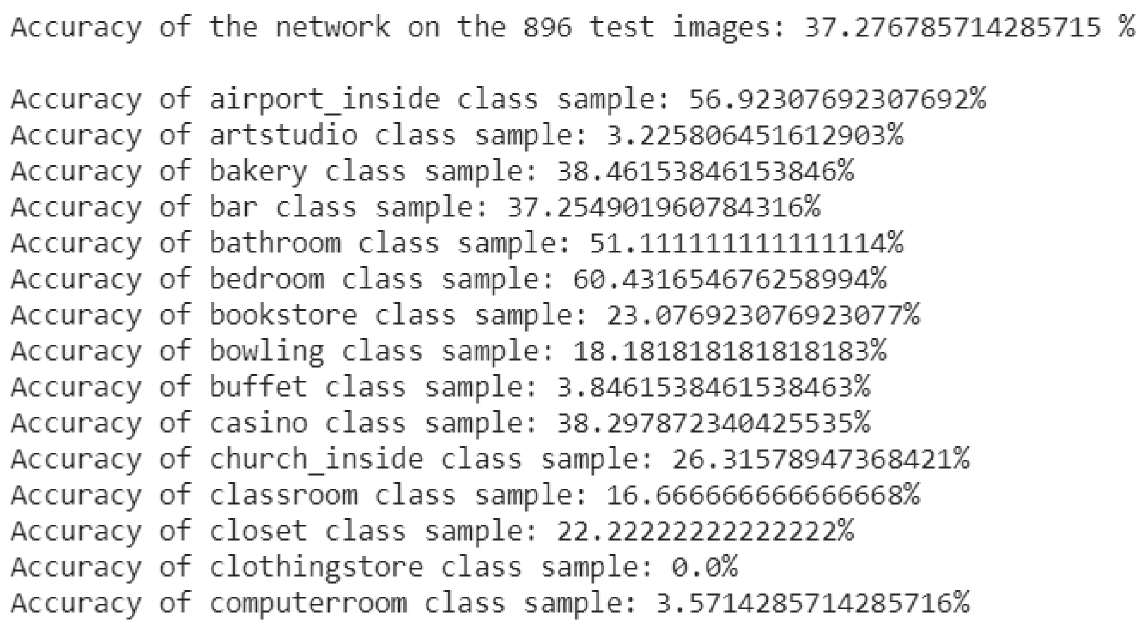

Batch size:32LR:0.05
Train
Validation
Accuracy
Epoch
Loss
Loss
Epoch





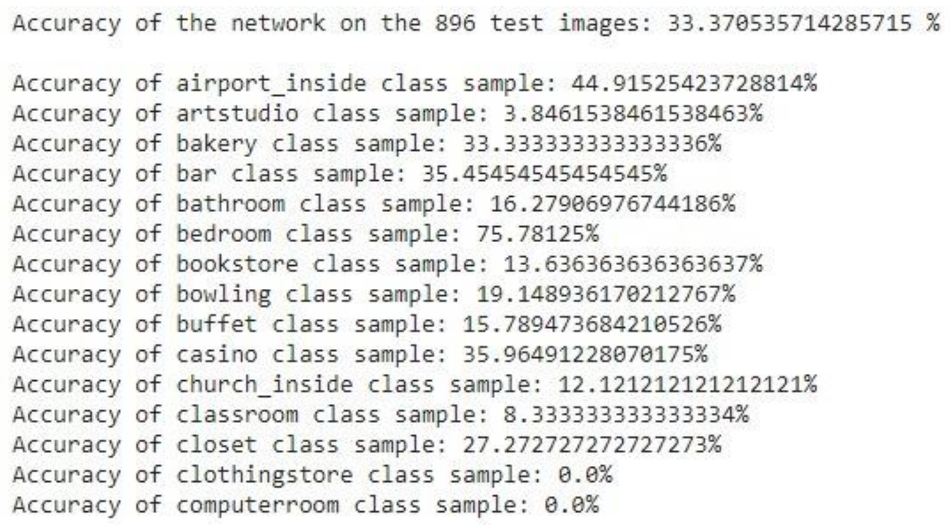
According to the plots above learning rate is much effective rather than batch size. This does not said that batch size is not important. Model has many features and choosing high learning rate makes oscilattion.

For the model **without residual connection <u>Batch size 256 and learning rate 0.001 is the best</u>**



```
Accuracy of the network on the 896 test images: 37.276785714285715 %

Accuracy of airport_inside class sample: 56.92307692307692%
Accuracy of artstudio class sample: 3.225806451612903%
Accuracy of bakery class sample: 38.46153846153846%
Accuracy of bar class sample: 37.254901960784316%
Accuracy of bathroom class sample: 51.111111111111114%
Accuracy of bedroom class sample: 60.431654676258994%
Accuracy of bookstore class sample: 23.076923076923077%
Accuracy of bowling class sample: 18.181818181818183%
Accuracy of buffet class sample: 3.8461538461538463%
Accuracy of casino class sample: 38.297872340425535%
Accuracy of church_inside class sample: 26.31578947368421%
Accuracy of classroom class sample: 16.666666666666668%
Accuracy of closet class sample: 22.22222222222222%
Accuracy of clothingstore class sample: 0.0%
Accuracy of computerroom class sample: 3.5714285714285716%
```

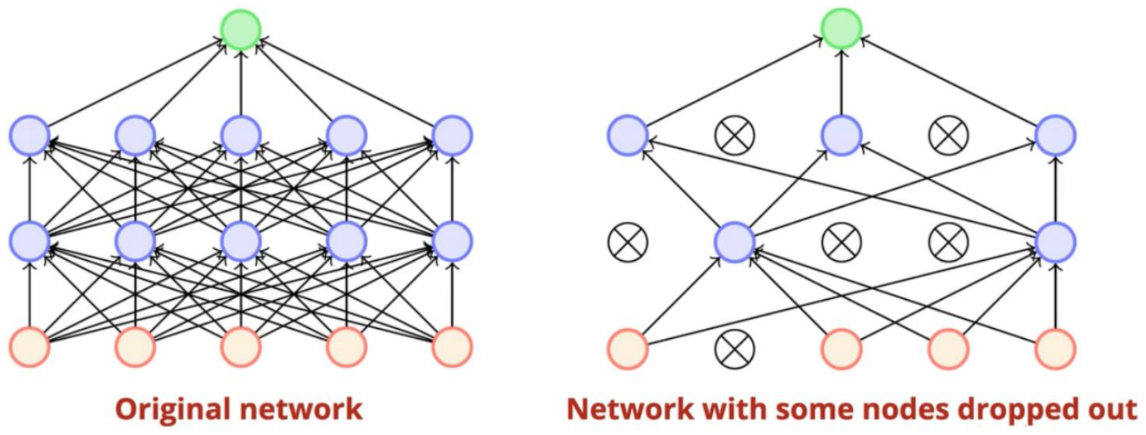
For the model **with residual connection <u>Batch size 64 and learning rate 0.001 is the best</u>**



```
Accuracy of the network on the 896 test images: 33.370535714285715 %

Accuracy of airport_inside class sample: 44.91525423728814%
Accuracy of artstudio class sample: 3.8461538461538463%
Accuracy of bakery class sample: 33.333333333333336%
Accuracy of bar class sample: 35.45454545454545%
Accuracy of bathroom class sample: 16.27906976744186%
Accuracy of bedroom class sample: 75.78125%
Accuracy of bookstore class sample: 13.636363636363637%
Accuracy of bowling class sample: 19.148936170212767%
Accuracy of buffet class sample: 15.789473684210526%
Accuracy of casino class sample: 35.96491228070175%
Accuracy of church_inside class sample: 12.121212121212121%
Accuracy of classroom class sample: 8.333333333333334%
Accuracy of closet class sample: 27.272727272727273%
Accuracy of clothingstore class sample: 0.0%
Accuracy of computerroom class sample: 0.0%
```

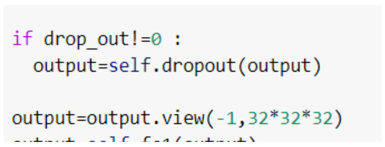
These models selected as their validation accuracy on the given plots.

About DropOut:



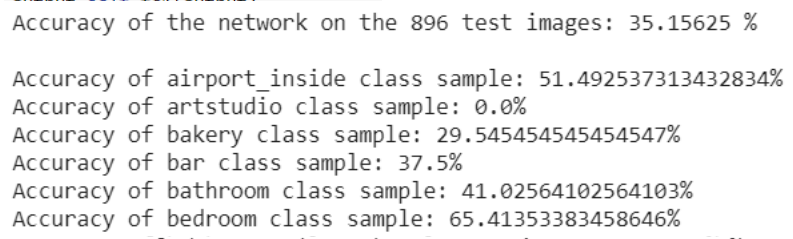
Dropout is only applied during train phase. So when prediction on test data it must be switch off. Large networks such as Neural Network are easy to overfit. And dropout is the main solution for that by ignoring some nodes and their weights. In the original paper that proposed dropout layers, by Hinton (2012), dropout (with p=0.5) was used on each of the fully connected (dense) layers before the output; it was not used on the convolutional layers. This became the most commonly used configuration. More recent research has shown some value in applying dropout also to convolutional layers, although at much lower levels: p=0.1 or 0.2. Dropout was used after the activation function of each convolutional layer: CONV->RELU->DROP.

But using many dropout is loosing many information. So it must be selected wisely. And I used only one time and it will be before fully connected layer.



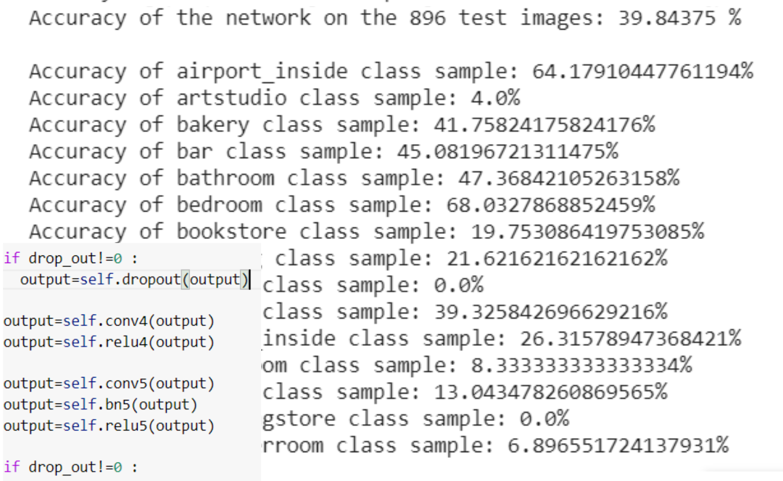
```
if drop_out!=0 :
  output=self.dropout(output)

output=output.view(-1,32*32*32)
```

On the first condition we add dropout before fully connected layer. Only ignore 20% of nodes when dropout is done.



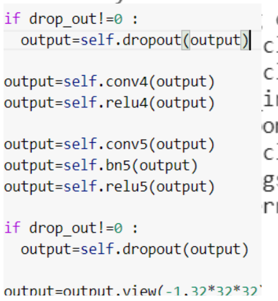
```
Accuracy of the network on the 896 test images: 35.15625 %

Accuracy of airport_inside class sample: 51.492537313432834%
Accuracy of artstudio class sample: 0.0%
Accuracy of bakery class sample: 29.545454545454547%
Accuracy of bar class sample: 37.5%
Accuracy of bathroom class sample: 41.02564102564103%
Accuracy of bedroom class sample: 65.41353383458646%
```

If we add dropout on **residual connection**: before fully connected layer our dropout increase from 33% to 35%.



```
Accuracy of the network on the 896 test images: 39.84375 %

Accuracy of airport_inside class sample: 64.17910447761194%
Accuracy of artstudio class sample: 4.0%
Accuracy of bakery class sample: 41.75824175824176%
Accuracy of bar class sample: 45.08196721311475%
Accuracy of bathroom class sample: 47.36842105263158%
Accuracy of bedroom class sample: 68.0327868852459%
Accuracy of bookstore class sample: 19.753086419753085%
 class sample: 21.62162162162162%
class sample: 0.0%
class sample: 39.325842696629216%
inside class sample: 26.31578947368421%
om class sample: 8.333333333333334%
class sample: 13.043478260869565%
gstore class sample: 0.0%
rroom class sample: 6.896551724137931%
```

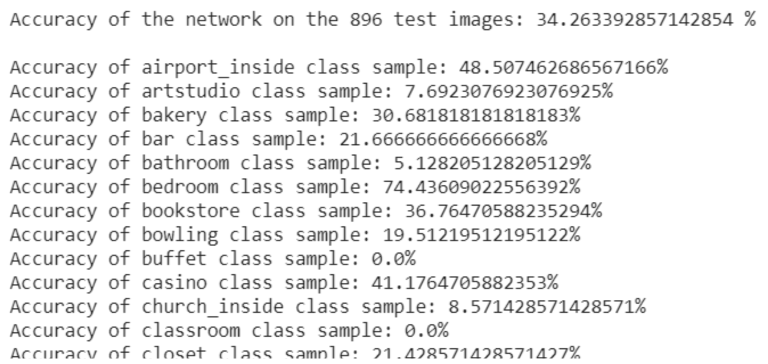
If we add 1 dropout on **without residual connection:** Accuracy with 1 dropout is better.



```
if drop_out!=0 :
  output=self.dropout(output)

output=self.conv4(output)
output=self.relu4(output)

output=self.conv5(output)
output=self.bn5(output)
output=self.relu5(output)

if drop_out!=0 :
  output=self.dropout(output)

output=output.view(-1,32*32*32)
```

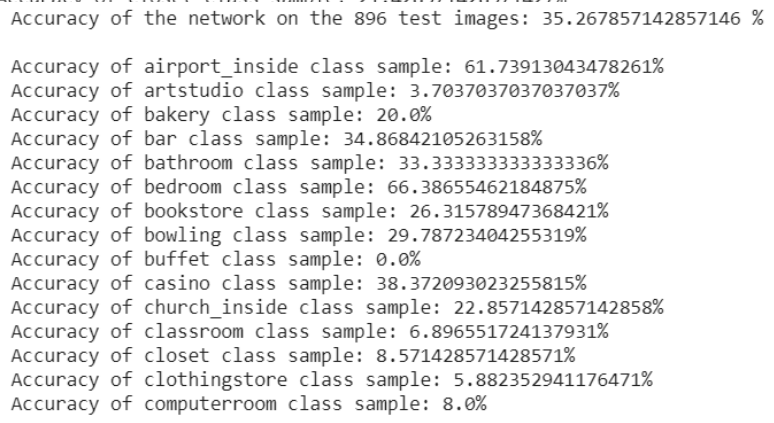
If we also add the second dropout before 4th convolutional layer we loss many information but we avoid overfit by far. And again we ignore 20% of the nodes.

According to some article using dropout before fully connected is better.



```
Accuracy of the network on the 896 test images: 34.263392857142854 %

Accuracy of airport_inside class sample: 48.507462686567166%
Accuracy of artstudio class sample: 7.6923076923076925%
Accuracy of bakery class sample: 30.681818181818183%
Accuracy of bar class sample: 21.666666666666668%
Accuracy of bathroom class sample: 5.128205128205129%
Accuracy of bedroom class sample: 74.43609022556392%
Accuracy of bookstore class sample: 36.76470588235294%
Accuracy of bowling class sample: 19.51219512195122%
Accuracy of buffet class sample: 0.0%
Accuracy of casino class sample: 41.1764705882353%
Accuracy of church_inside class sample: 8.571428571428571%
Accuracy of classroom class sample: 0.0%
Accuracy of closet class sample: 21.428571428571427%
```

If we add second dropout on **residual connection**: As you can see the accuracy using second dropout makes the model less accurate. Loss of information is much with 2 dropout, rather than 1 dropout.



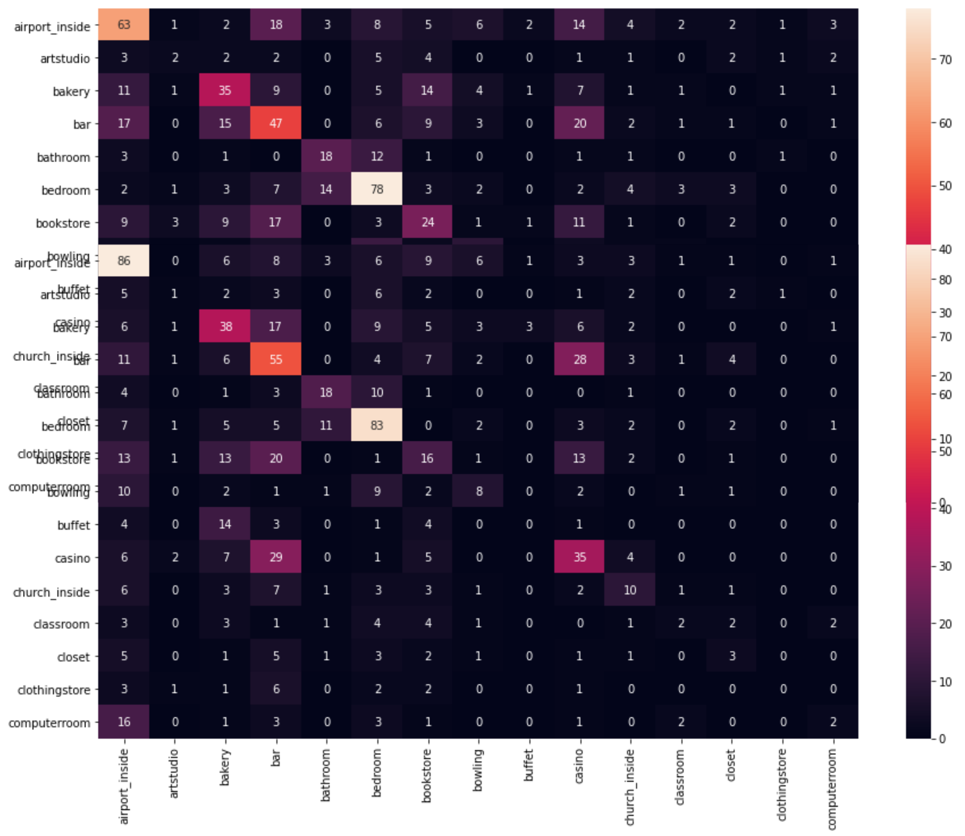
```
Accuracy of the network on the 896 test images: 35.267857142857146 %

Accuracy of airport_inside class sample: 61.73913043478261%
Accuracy of artstudio class sample: 3.7037037037037037%
Accuracy of bakery class sample: 20.0%
Accuracy of bar class sample: 34.86842105263158%
Accuracy of bathroom class sample: 33.333333333333336%
Accuracy of bedroom class sample: 66.38655462184875%
Accuracy of bookstore class sample: 26.31578947368421%
Accuracy of bowling class sample: 29.78723404255319%
Accuracy of buffet class sample: 0.0%
Accuracy of casino class sample: 38.372093023255815%
Accuracy of church_inside class sample: 22.857142857142858%
Accuracy of classroom class sample: 6.896551724137931%
Accuracy of closet class sample: 8.571428571428571%
Accuracy of clothingstore class sample: 5.882352941176471%
Accuracy of computerroom class sample: 8.0%
```

If we add second dropout on **without residual connection**: As stated before using 2 dropout gives less accurate results.

Finally dropout before fully connected layers gives better results. With 1 dropout and without residual connection gives the better results.

Confusion Matrix:

Some images are easy to predict such as airport inside this is because images in that category easy to extract features. Features are less and easy to distinguish. But in the computerroom it is not easy to distinguish. Because it has many features. Prone to misclassification. And finally the results are close to each other



Best model with 1 dropout and without residual connection.

Best model with 1 dropout and with residual connection.

Analysing:

This network is prone to overfitting. As you can see the plots above. Dropout and batch normalization is good to avoid and speed up the training phase. There are many input featuers and that means there are lots unnecessary information and easy to overfit.

Dropout improves the model generalization. Even though the training accuracy is lower than the unregularized network, the overall validation accuracy has improved. This accounts for a lower generalization error. Training with two dropout layers with a dropout probability of 25% prevents model from overfitting.

Choosing the hyperparameter is the main criteria. Batch normalization is used to improve the speed of training.

Choosing small learning rate gives better results because we have large number of features.

PART 2: Transfer Learning with CNNs

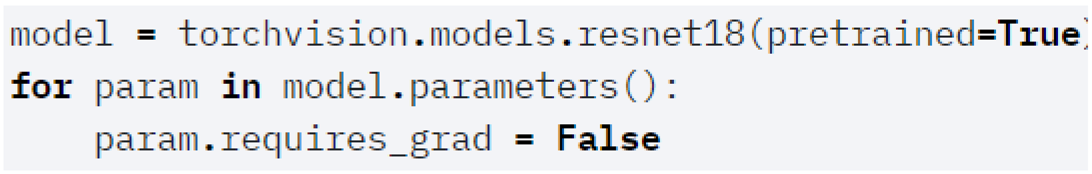
Finetuning means taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain (often e.g. images). It is used to:

1. speed up the training
2. overcome small dataset size

Transfer learning occurs when we use knowledge that was gained from solving one problem and apply it to a new but related problem.

Freezing a layer prevents its weights from being modified. This technique is often used in transfer learning, where the base model is frozen. When we do not want to update the model weights, on the backpropagation layers are avoided. For example if we freeze half of the model and then we train the model it will take half of the time (speed increasing)

Every Tensor has a flag: requires_grad that allows for fine tuning avoiding of subgraphs from gradient computation and can increase efficiency.



```
model = torchvision.models.resnet18(pretrained=True
for param in model.parameters():
    param.requires_grad = False
```

For resnet example, this loop will freeze all layers. This works only to freeze layers after they have been initially unfrozen.