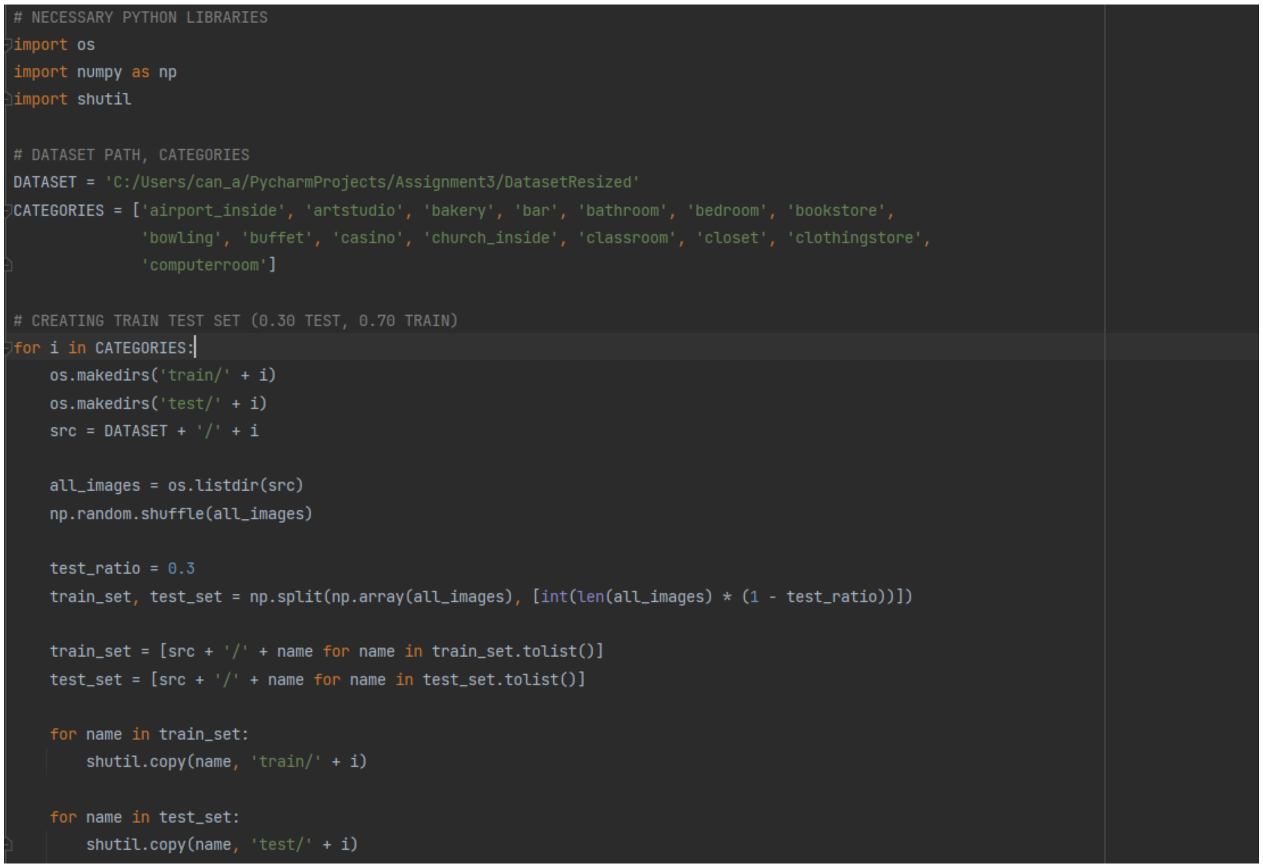
BBM418 - COMPUTER VISION LABORATORY
ASSIGNMENT 3

Sadık Can ACAR
21626843

May 5, 2021

1 Train-Validation and Test Set Split

During the train, test split phase, I first created a folder called train and test. Then I mixed all the images and split them into train and test sets at proportions of 0.7, 0.3. I used a test set to validate and evaluate my models. I wanted to create a single set for validation and testing, as it is used in many models I researched.



```python
# NECESSARY PYTHON LIBRARIES
import os
import numpy as np
import shutil

# DATASET PATH, CATEGORIES
DATASET = 'C:/Users/can_a/PycharmProjects/Assignment3/DatasetResized'
CATEGORIES = ['airport_inside', 'artstudio', 'bakery', 'bar', 'bathroom', 'bedroom', 'bookstore',
              'bowling', 'buffet', 'casino', 'church_inside', 'classroom', 'closet', 'clothingstore',
              'computerroom']

# CREATING TRAIN TEST SET (0.30 TEST, 0.70 TRAIN)
for i in CATEGORIES:
    os.makedirs('train/' + i)
    os.makedirs('test/' + i)
    src = DATASET + '/' + i

    all_images = os.listdir(src)
    np.random.shuffle(all_images)

    test_ratio = 0.3
    train_set, test_set = np.split(np.array(all_images), [int(len(all_images) * (1 - test_ratio))])

    train_set = [src + '/' + name for name in train_set.tolist()]
    test_set = [src + '/' + name for name in test_set.tolist()]

    for name in train_set:
        shutil.copy(name, 'train/' + i)

    for name in test_set:
        shutil.copy(name, 'test/' + i)
```

Figure 1: Train-Validation and Test Split

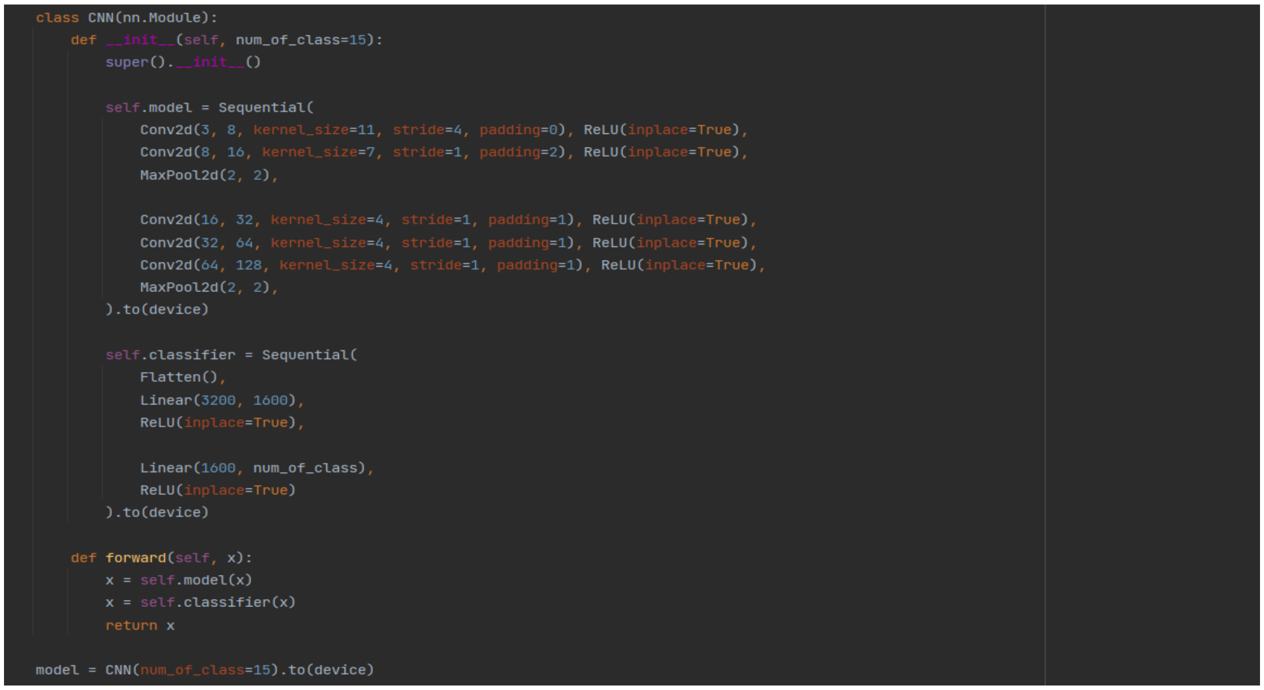
1.1 Number of Images Per Category of Train, Validation and Test Sets

	Train Set	Validation and Test Set
airport-inside	425	183
artstudio	98	42
bakery	283	122
bar	422	182
bathroom	137	60
bedroom	463	199
bookstore	266	114
bowling	149	64
buffet	77	34
casino	360	155
church-inside	125	55
classroom	79	34
closet	94	41
clothingstore	74	32
computerroom	79	35

2 Architecture

2.1 My CNN Architecture

In my Convolutional Neural Network architecture, there are 5 convolutional layers and 2 fully connected layers. The in-channel value of the first convolutional layer is 3 which corresponds to ”RGB” since the images are colored. I adjusted the out-channels as shown in the screenshot to match the image size until it comes to fully connected layers. Since the images have a size like (128, 128), the out-channel of the 5th convolutional layer should have been 128. In the first of the fully connected layers, the in-channel value was calculated with (128 * 5 * 5) and out-channel was calculated as (128 * 5 * 5) / 2 since there are two fully connected layers. The in-channel value of the second fully connected layer should be 1600 and the out-channel value should be the number of classes of the images we need to classify.

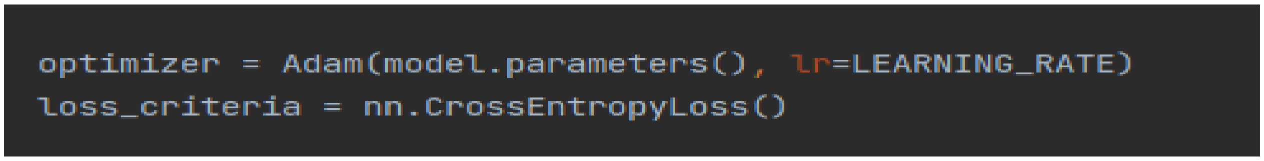


```
class CNN(nn.Module):
    def __init__(self, num_of_class=15):
        super().__init__()

        self.model = Sequential(
            Conv2d(3, 8, kernel_size=11, stride=4, padding=0), ReLU(inplace=True),
            Conv2d(8, 16, kernel_size=7, stride=1, padding=2), ReLU(inplace=True),
            MaxPool2d(2, 2),

            Conv2d(16, 32, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            Conv2d(32, 64, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            Conv2d(64, 128, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            MaxPool2d(2, 2),
        ).to(device)

        self.classifier = Sequential(
            Flatten(),
            Linear(3200, 1600),
            ReLU(inplace=True),

            Linear(1600, num_of_class),
            ReLU(inplace=True)
        ).to(device)

    def forward(self, x):
        x = self.model(x)
        x = self.classifier(x)
        return x

model = CNN(num_of_class=15).to(device)
```

Figure 2: My CNN Architecture

2.2 Activation, Loss Functions and Optimization

When activating the layers, I used the ReLU (Rectified Linear Unit) function provided by PyTorch. I have defined a variable named loss-criteria while calculating the loss in education and loss in the validation-test phase. I have synchronized this variable to a loss function (CrossEntropyLoss) provided by PyTorch. I used Adam optimizer to optimize the model. The optimizer takes the parameters of the model parameters and additionally the learning rate parameter that causes changes in accuracy rates.
Note: ReLU functions screenshots are contained above.

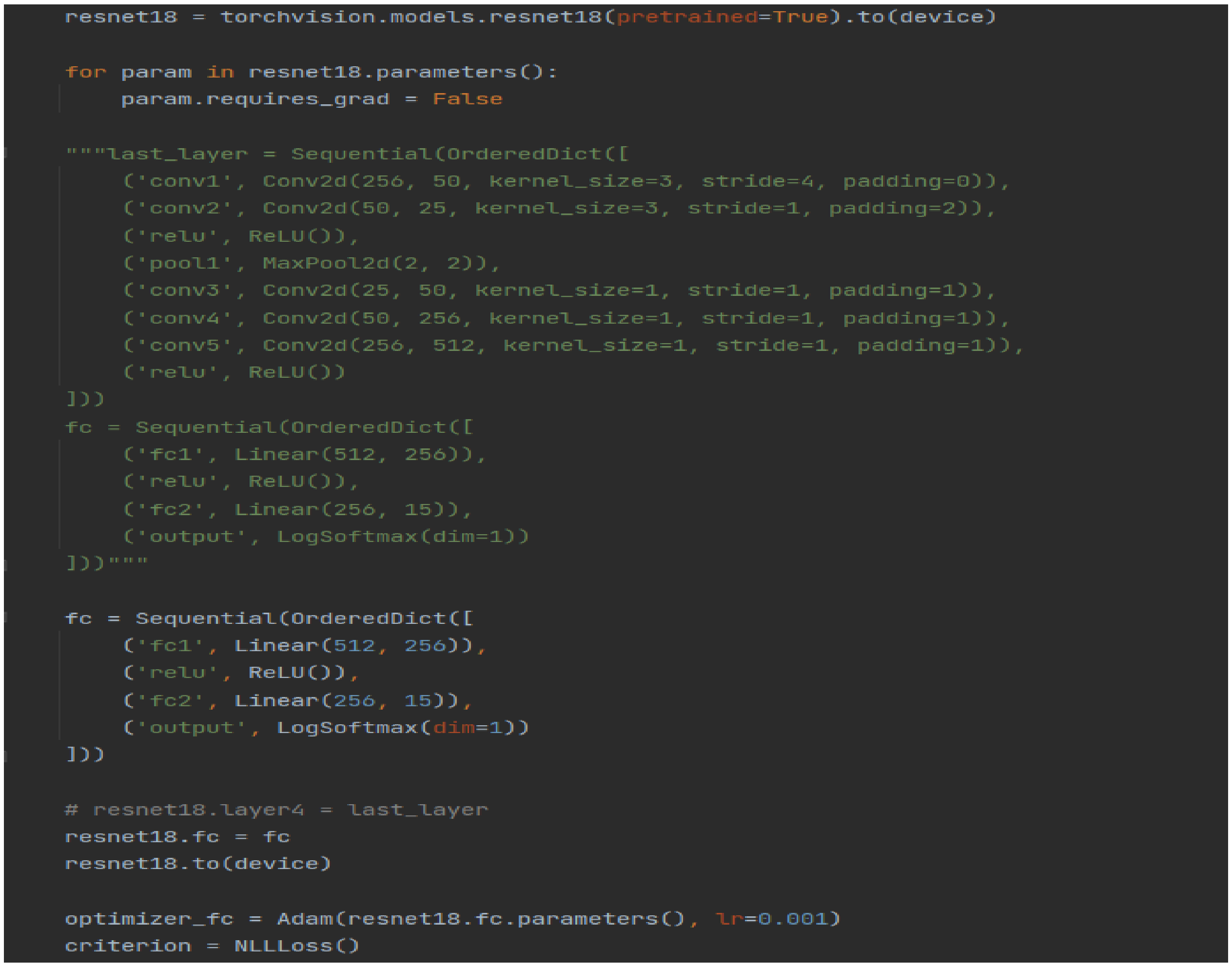


```
optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
loss_criteria = nn.CrossEntropyLoss()
```

Figure 3: Optimization - Loss Functions

2.3 ResNet18 Implementation

I installed ResNet18 from PyTorch models library to my workspace and device by giving the pretrained parameter to True. I used the codes I wrote in the screenshot to perform the transfer learning operations.

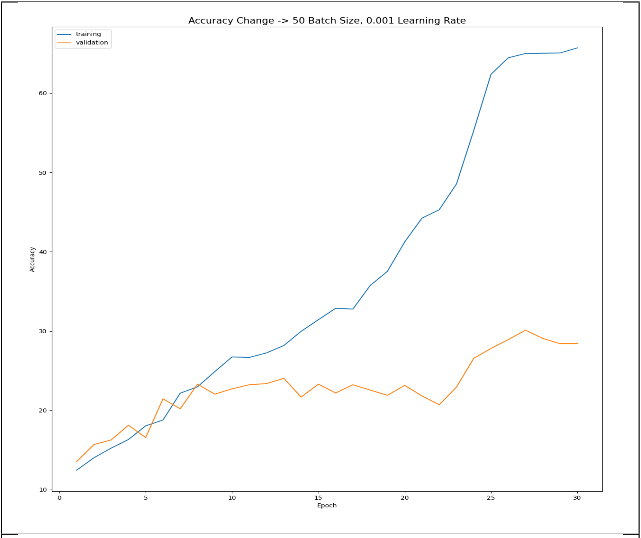


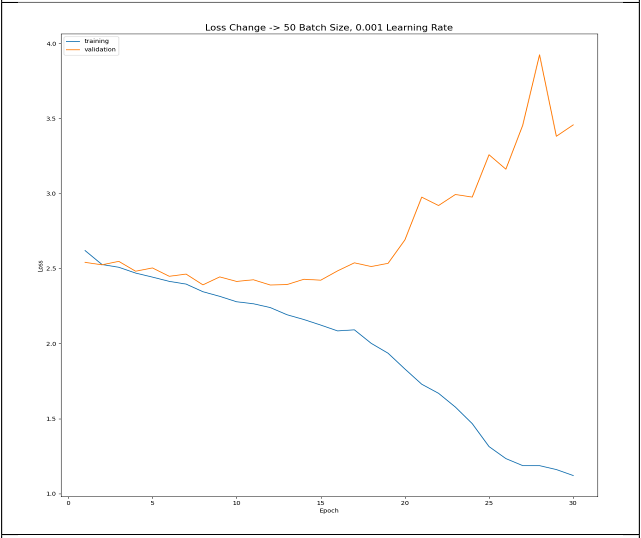
```
resnet18 = torchvision.models.resnet18(pretrained=True).to(device)

for param in resnet18.parameters():
    param.requires_grad = False

"""last_layer = Sequential(OrderedDict([
    ('conv1', Conv2d(256, 50, kernel_size=3, stride=4, padding=0)),
    ('conv2', Conv2d(50, 25, kernel_size=3, stride=1, padding=2)),
    ('relu', ReLU()),
    ('pool1', MaxPool2d(2, 2)),
    ('conv3', Conv2d(25, 50, kernel_size=1, stride=1, padding=1)),
    ('conv4', Conv2d(50, 256, kernel_size=1, stride=1, padding=1)),
    ('conv5', Conv2d(256, 512, kernel_size=1, stride=1, padding=1)),
    ('relu', ReLU())
]))
fc = Sequential(OrderedDict([
    ('fc1', Linear(512, 256)),
    ('relu', ReLU()),
    ('fc2', Linear(256, 15)),
    ('output', LogSoftmax(dim=1))
]))"""

fc = Sequential(OrderedDict([
    ('fc1', Linear(512, 256)),
    ('relu', ReLU()),
    ('fc2', Linear(256, 15)),
    ('output', LogSoftmax(dim=1))
]))

# resnet18.layer4 = last_layer
resnet18.fc = fc
resnet18.to(device)

optimizer_fc = Adam(resnet18.fc.parameters(), lr=0.001)
criterion = NLLLoss()
```

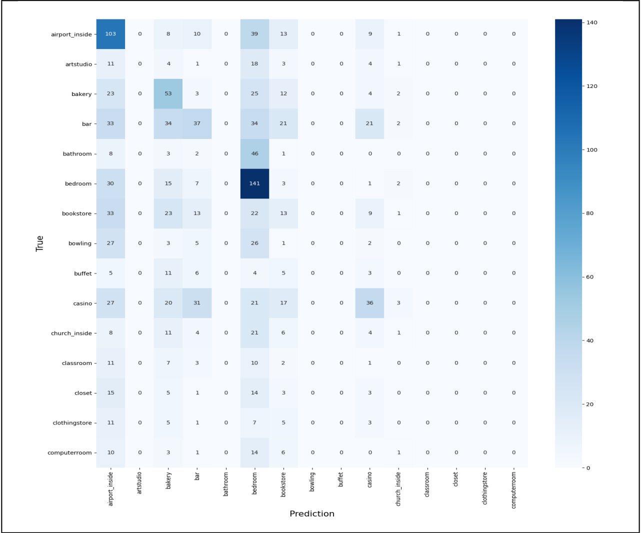
Figure 4: ResNet18

3 Training and Evaluating My Model

3.1 Accuracy - Loss Change Graphs and Confusion Matrix 50 Batch Size - 0.001 Learning Rate

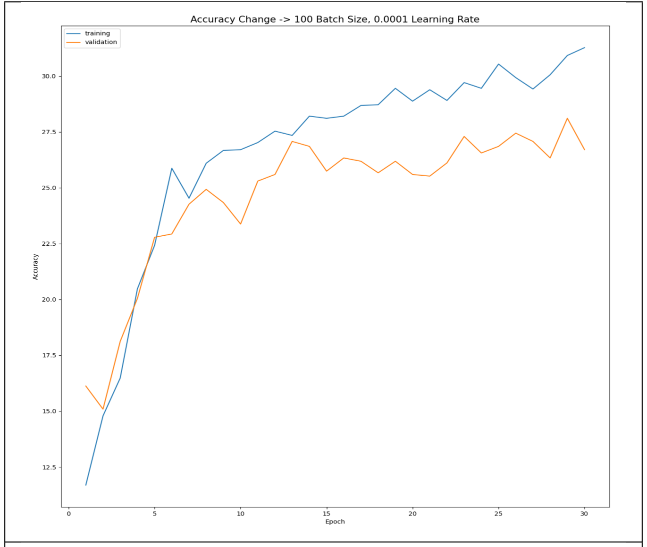


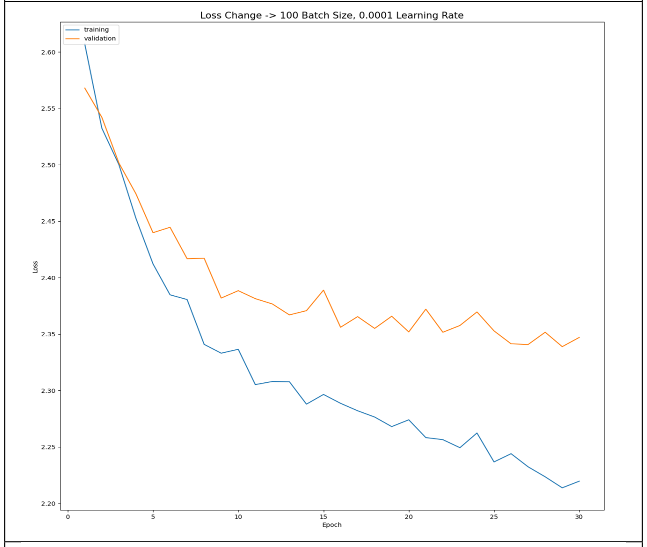


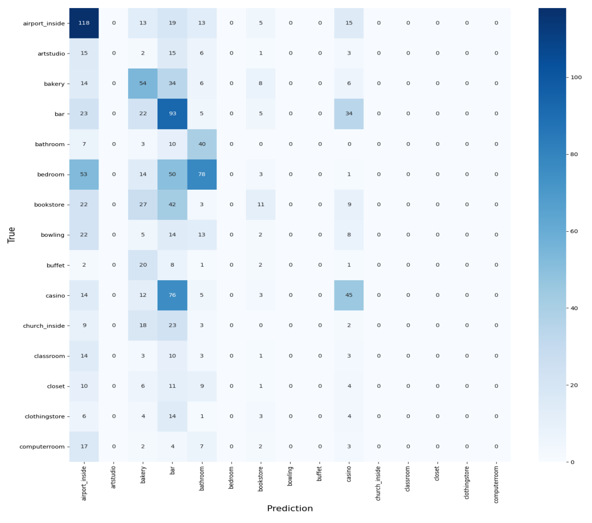


	airport_inside	artstudio	bakery	bar	bathroom	bedroom	bookstore	bowling	buffet	casino	church_inside	classroom	closet	clothingstore	computerroom
airport_inside	103	0	8	10	0	39	13	0	0	9	1	0	0	0	0
artstudio	11	0	4	1	0	18	3	0	0	4	1	0	0	0	0
bakery	23	0	53	3	0	25	12	0	0	4	2	0	0	0	0
bar	33	0	34	37	0	34	21	0	0	21	2	0	0	0	0
bathroom	8	0	3	2	0	46	1	0	0	0	0	0	0	0	0
bedroom	30	0	15	7	0	141	3	0	0	1	2	0	0	0	0
bookstore	33	0	23	13	0	22	13	0	0	9	1	0	0	0	0
bowling	27	0	3	5	0	26	1	0	0	2	0	0	0	0	0
buffet	5	0	11	6	0	4	5	0	0	3	0	0	0	0	0
casino	27	0	20	31	0	21	17	0	0	36	3	0	0	0	0
church_inside	8	0	11	4	0	21	6	0	0	4	1	0	0	0	0
classroom	11	0	7	3	0	10	2	0	0	1	0	0	0	0	0
closet	15	0	5	1	0	14	3	0	0	3	0	0	0	0	0
clothingstore	11	0	5	1	0	7	5	0	0	3	0	0	0	0	0
computerroom	10	0	3	1	0	14	6	0	0	0	1	0	0	0	0

3.2 Accuracy - Loss Change Graphs and Confusion Matrix 100 Batch Size - 0.0001 Learning Rate

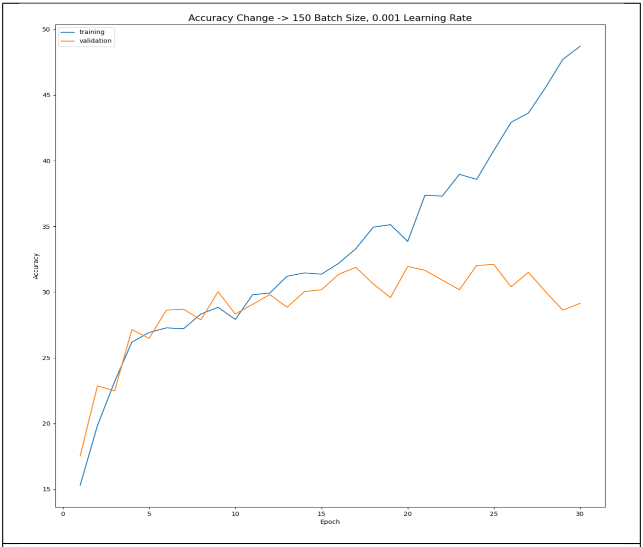


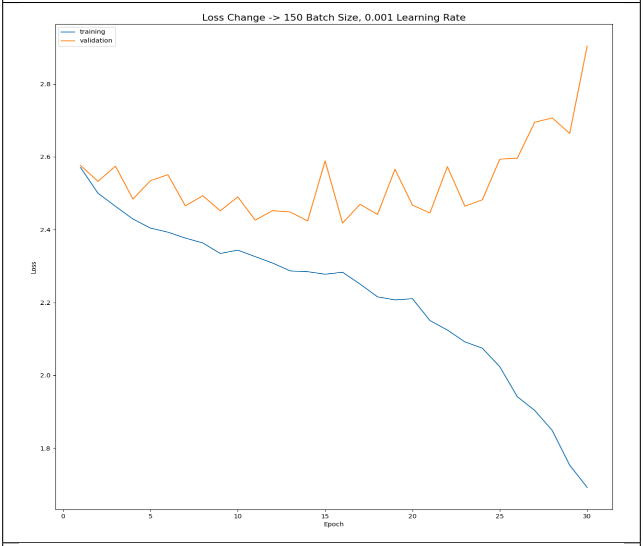


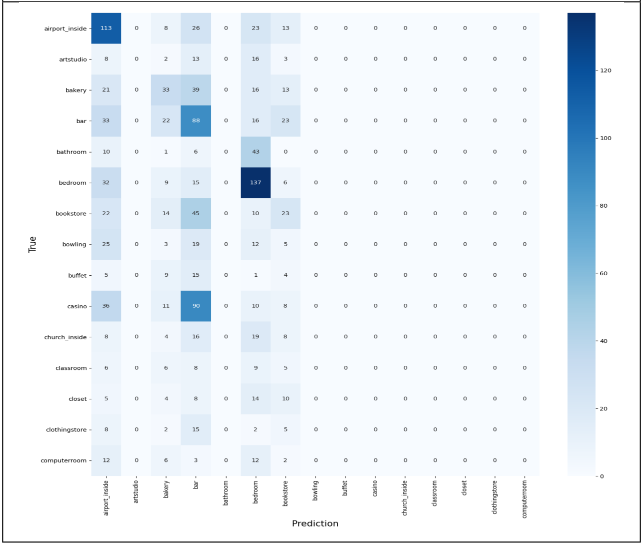


	airport_inside	artstudio	bakery	bar	bathroom	bedroom	bookstore	bowling	buffet	casino	church_inside	classroom	closet	clothingstore	computerroom
airport_inside	118	0	13	19	13	0	5	0	0	15	0	0	0	0	0
artstudio	15	0	2	15	6	0	1	0	0	3	0	0	0	0	0
bakery	14	0	54	34	6	0	8	0	0	6	0	0	0	0	0
bar	23	0	22	93	5	0	5	0	0	34	0	0	0	0	0
bathroom	7	0	3	10	40	0	0	0	0	0	0	0	0	0	0
bedroom	53	0	14	50	78	0	3	0	0	1	0	0	0	0	0
bookstore	22	0	27	42	3	0	11	0	0	9	0	0	0	0	0
bowling	22	0	5	14	13	0	2	0	0	8	0	0	0	0	0
buffet	2	0	20	8	1	0	2	0	0	1	0	0	0	0	0
casino	14	0	12	76	5	0	3	0	0	45	0	0	0	0	0
church_inside	9	0	18	23	3	0	0	0	0	2	0	0	0	0	0
classroom	14	0	3	10	3	0	1	0	0	3	0	0	0	0	0
closet	10	0	6	11	9	0	1	0	0	4	0	0	0	0	0
clothingstore	6	0	4	14	1	0	3	0	0	4	0	0	0	0	0
computerroom	17	0	2	4	7	0	2	0	0	3	0	0	0	0	0

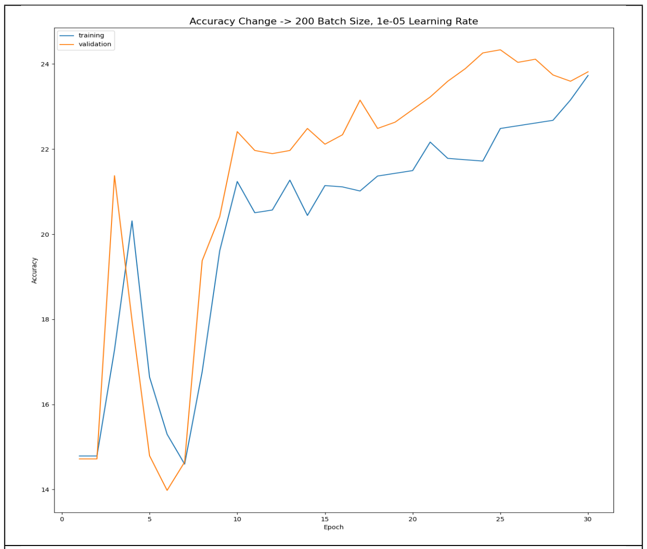
3.3 Accuracy - Loss Change Graphs and Confusion Matrix 150 Batch Size - 0.001 Learning Rate

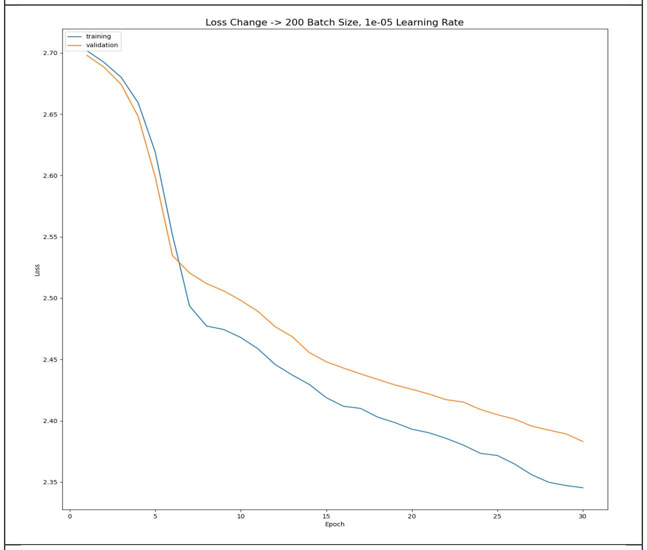


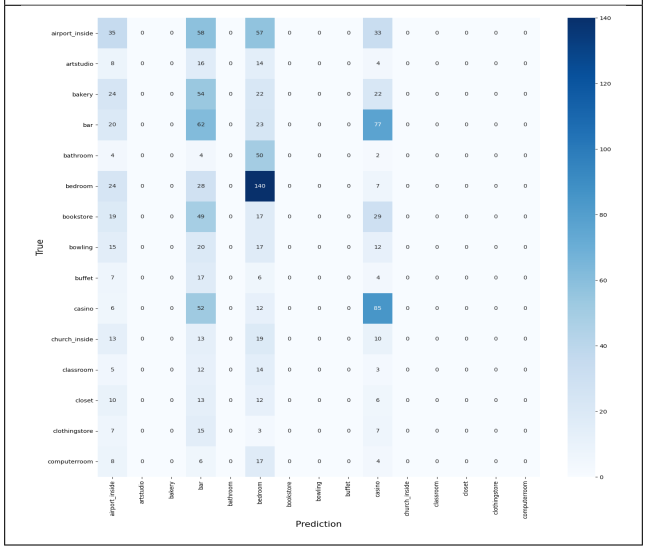




3.4 Accuracy - Loss Change Graphs and Confusion Matrix 200 Batch Size - 0.00001 Learning Rate



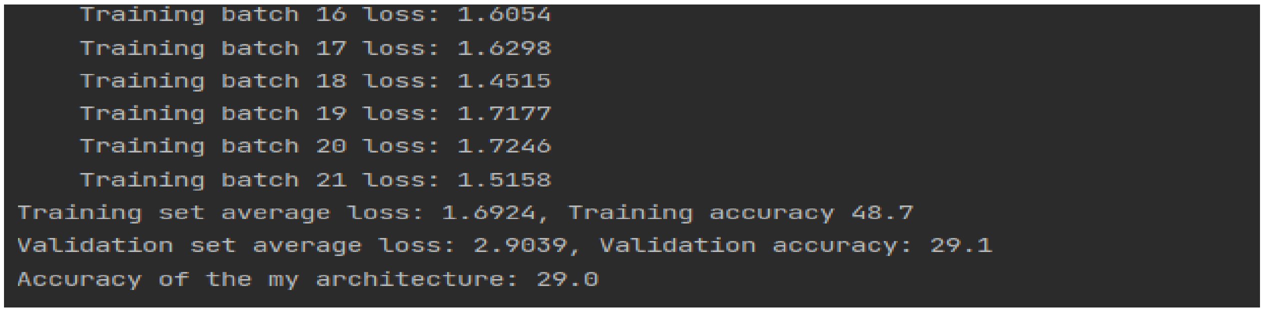




	airport_inside	artstudio	bakery	bar	bathroom	bedroom	bookstore	bowling	buffet	casino	church_inside	classroom	closet	clothingstore	computerroom
airport_inside	35	0	0	58	0	57	0	0	0	33	0	0	0	0	0
artstudio	8	0	0	16	0	14	0	0	0	4	0	0	0	0	0
bakery	24	0	0	54	0	22	0	0	0	22	0	0	0	0	0
bar	20	0	0	62	0	23	0	0	0	77	0	0	0	0	0
bathroom	4	0	0	4	0	50	0	0	0	2	0	0	0	0	0
bedroom	24	0	0	28	0	140	0	0	0	7	0	0	0	0	0
bookstore	19	0	0	49	0	17	0	0	0	29	0	0	0	0	0
bowling	15	0	0	20	0	17	0	0	0	12	0	0	0	0	0
buffet	7	0	0	17	0	6	0	0	0	4	0	0	0	0	0
casino	6	0	0	52	0	12	0	0	0	85	0	0	0	0	0
church_inside	13	0	0	13	0	19	0	0	0	10	0	0	0	0	0
classroom	5	0	0	12	0	14	0	0	0	3	0	0	0	0	0
closet	10	0	0	13	0	12	0	0	0	6	0	0	0	0	0
clothingstore	7	0	0	15	0	3	0	0	0	7	0	0	0	0	0
computerroom	8	0	0	6	0	17	0	0	0	4	0	0	0	0	0

3.5 My Best Model without Dropout

These results are derived from the only CNN architecture I have created. Batch size is 150 and learning rate is 0.001. Because the validation and test set are the same, the validation and test accuracy results are the same. In line with the results, I saw that the accuracy was high for a certain range for the learning rate, while batch size did not have much effect on the accuracy score. I think the best learning rate is 0.001. As seen in the screenshot below, the training accuracy for the 30th epoch is 48.7 and the validation and test accuracy score is 29.1. Validation and test accuracy for the whole model is 29.0.

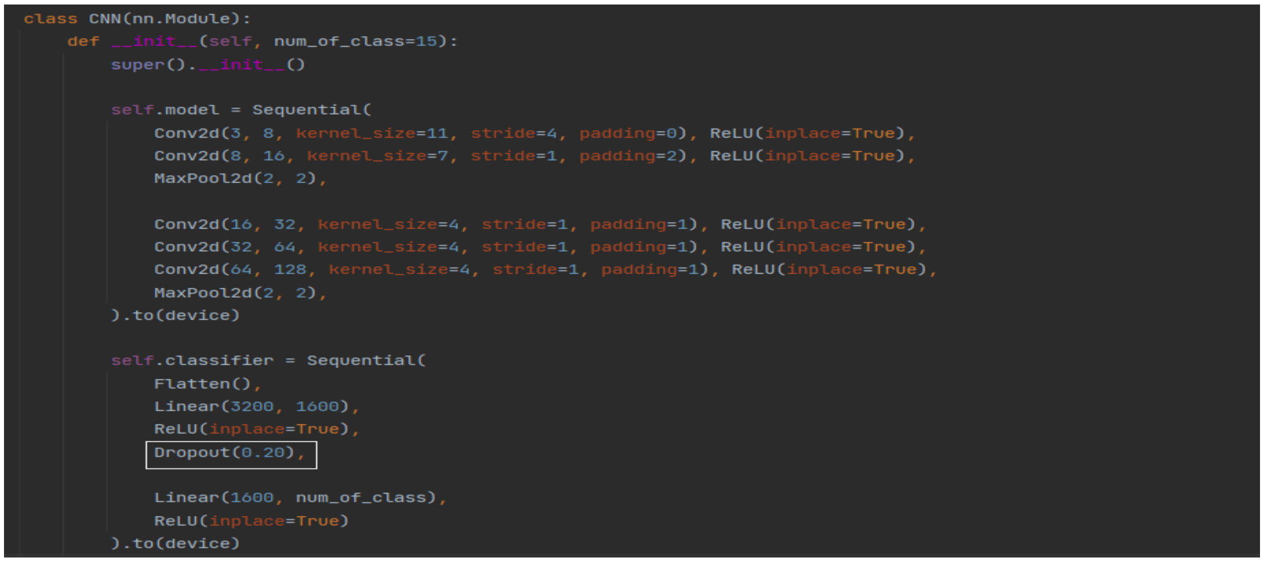


```
    Training batch 16 loss: 1.6054
    Training batch 17 loss: 1.6298
    Training batch 18 loss: 1.4515
    Training batch 19 loss: 1.7177
    Training batch 20 loss: 1.7246
    Training batch 21 loss: 1.5158
Training set average loss: 1.6924, Training accuracy 48.7
Validation set average loss: 2.9039, Validation accuracy: 29.1
Accuracy of the my architecture: 29.0
```

Figure 5: My Best Model Train-Validation and Model Accuracy

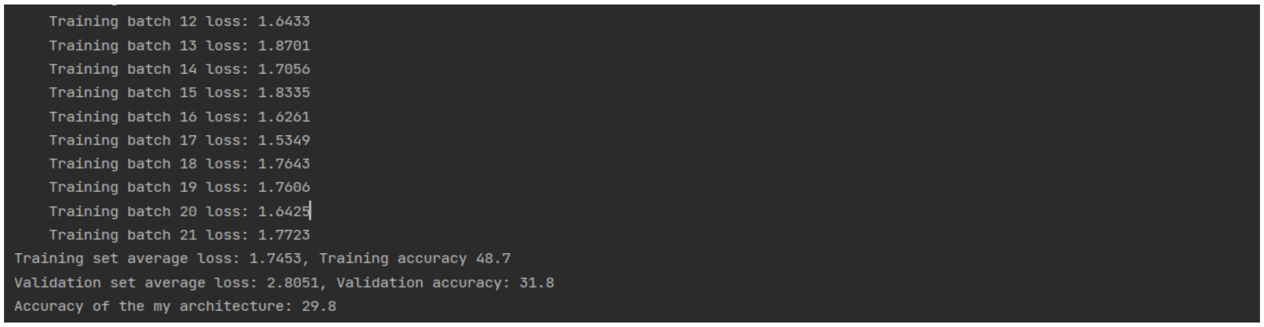
3.6 Evaluating My Model with Dropout

I added a dropout after my network's fully connected connections. Because the dropout technique is generally used after fully-connected layers. By using Dropout, the bonds in fully-connected layers are broken. Thus, nodes have less information about each other and as a natural consequence of this, nodes are less affected by each other's weight changes. For this reason, more consistent (robust) models can be created with the dropout method. At the same time, a better learning will take place as different combinations of hidden units work with each other in each layer. In this sense, when dropout is used, it can be thought that hidden layers work as ensemble like random forest. This will provide better performance for the model in terms of both time and performance. As can be seen in the performance graph of the cases with and without dropout, the performance of the model has increased by decreasing the overfitting after the dropout. The dropout method is one of the most commonly used regularization methods in deep learning methods.



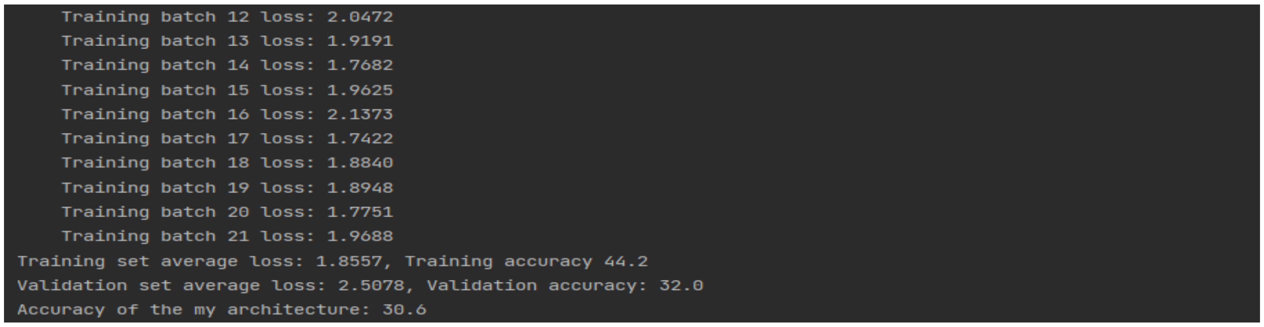
```
class CNN(nn.Module):
    def __init__(self, num_of_class=15):
        super().__init__()

        self.model = Sequential(
            Conv2d(3, 8, kernel_size=11, stride=4, padding=0), ReLU(inplace=True),
            Conv2d(8, 16, kernel_size=7, stride=1, padding=2), ReLU(inplace=True),
            MaxPool2d(2, 2),

            Conv2d(16, 32, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            Conv2d(32, 64, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            Conv2d(64, 128, kernel_size=4, stride=1, padding=1), ReLU(inplace=True),
            MaxPool2d(2, 2),
        ).to(device)

        self.classifier = Sequential(
            Flatten(),
            Linear(3200, 1600),
            ReLU(inplace=True),
            Dropout(0.20),

            Linear(1600, num_of_class),
            ReLU(inplace=True)
        ).to(device)
```

Figure 6: Integrating Dropout



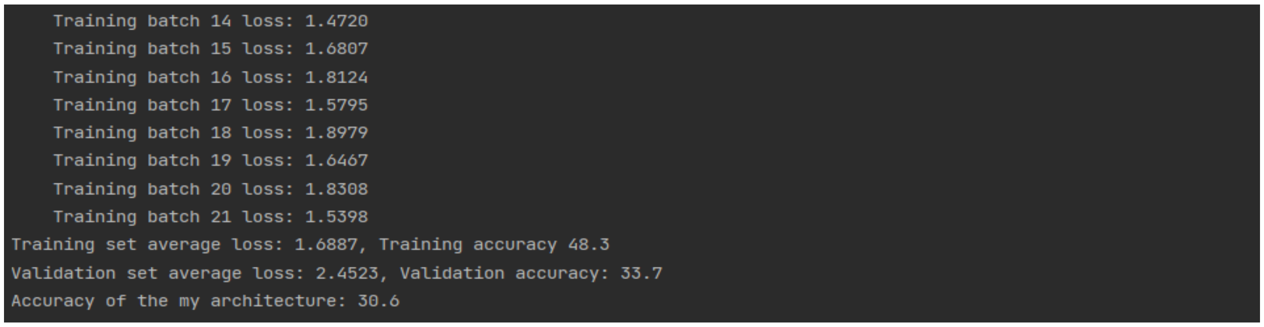
```
    Training batch 12 loss: 1.6433
    Training batch 13 loss: 1.8701
    Training batch 14 loss: 1.7056
    Training batch 15 loss: 1.8335
    Training batch 16 loss: 1.6261
    Training batch 17 loss: 1.5349
    Training batch 18 loss: 1.7643
    Training batch 19 loss: 1.7606
    Training batch 20 loss: 1.6425
    Training batch 21 loss: 1.7723
Training set average loss: 1.7453, Training accuracy 48.7
Validation set average loss: 2.8051, Validation accuracy: 31.8
Accuracy of the my architecture: 29.8
```

Figure 7: Accuracy Score - Dropout = 0.20



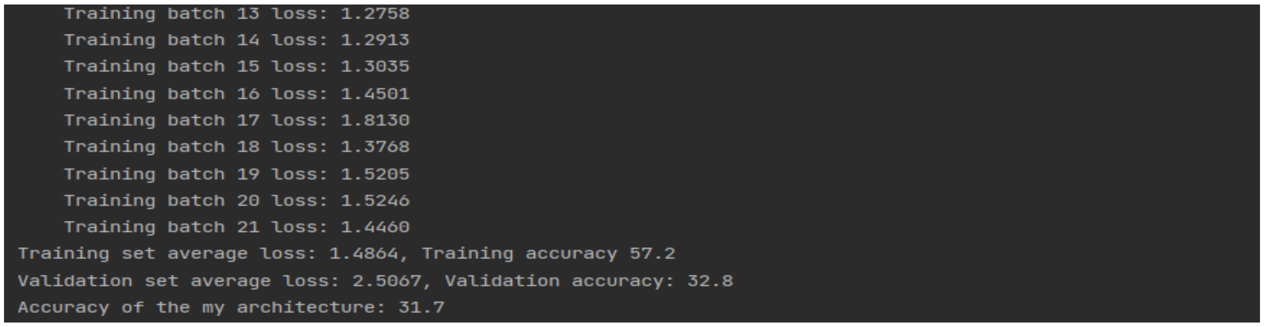
```
    Training batch 12 loss: 2.0472
    Training batch 13 loss: 1.9191
    Training batch 14 loss: 1.7682
    Training batch 15 loss: 1.9625
    Training batch 16 loss: 2.1373
    Training batch 17 loss: 1.7422
    Training batch 18 loss: 1.8840
    Training batch 19 loss: 1.8948
    Training batch 20 loss: 1.7751
    Training batch 21 loss: 1.9688
Training set average loss: 1.8557, Training accuracy 44.2
Validation set average loss: 2.5078, Validation accuracy: 32.0
Accuracy of the my architecture: 30.6
```

Figure 8: Accuracy Score - Dropout = 0.30



```
    Training batch 14 loss: 1.4720
    Training batch 15 loss: 1.6807
    Training batch 16 loss: 1.8124
    Training batch 17 loss: 1.5795
    Training batch 18 loss: 1.8979
    Training batch 19 loss: 1.6467
    Training batch 20 loss: 1.8308
    Training batch 21 loss: 1.5398
Training set average loss: 1.6887, Training accuracy 48.3
Validation set average loss: 2.4523, Validation accuracy: 33.7
Accuracy of the my architecture: 30.6
```

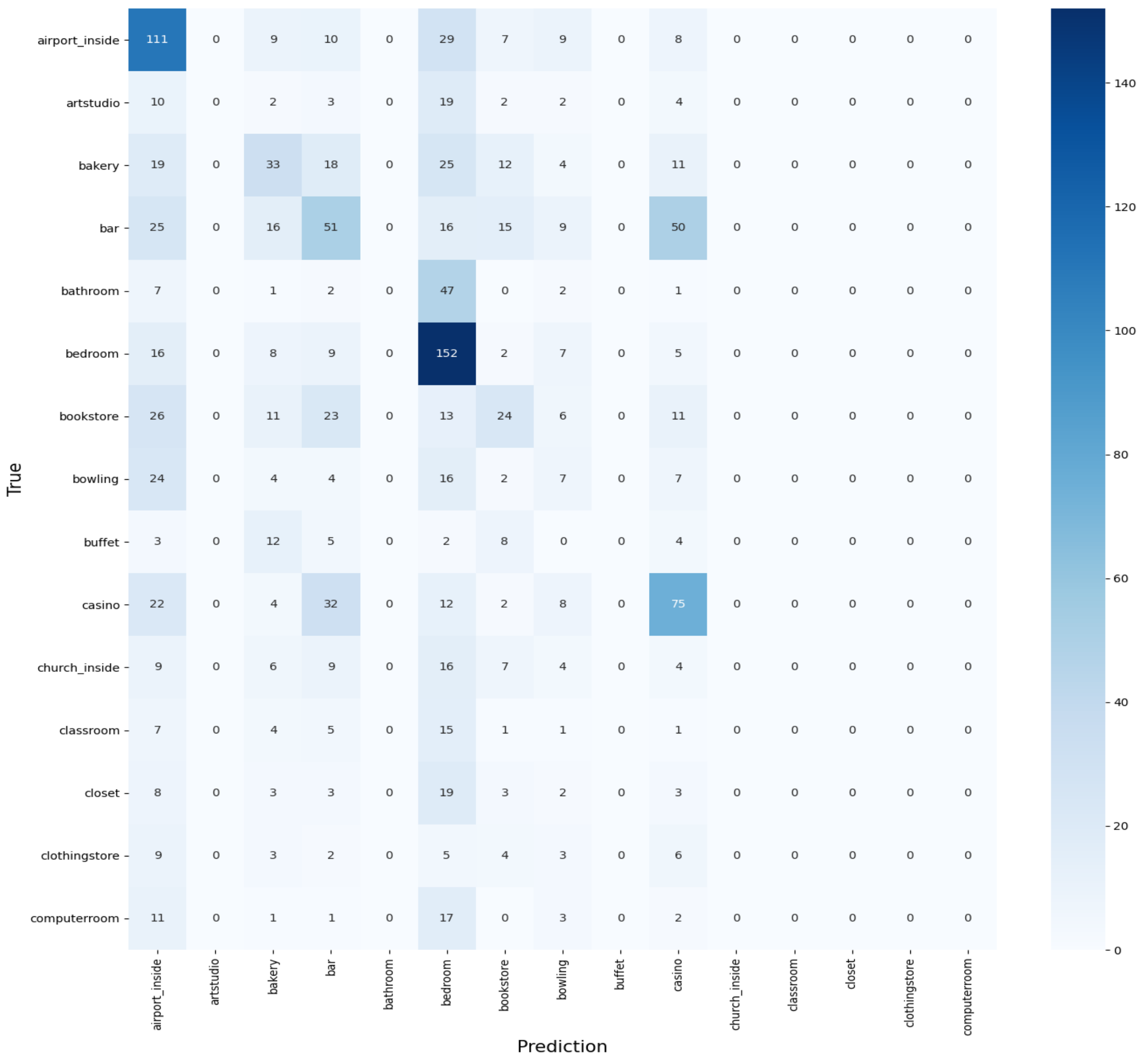
Figure 9: Accuracy Score - Dropout = 0.40



```
    Training batch 13 loss: 1.2758
    Training batch 14 loss: 1.2913
    Training batch 15 loss: 1.3035
    Training batch 16 loss: 1.4501
    Training batch 17 loss: 1.8130
    Training batch 18 loss: 1.3768
    Training batch 19 loss: 1.5205
    Training batch 20 loss: 1.5246
    Training batch 21 loss: 1.4460
Training set average loss: 1.4864, Training accuracy 57.2
Validation set average loss: 2.5067, Validation accuracy: 32.8
Accuracy of the my architecture: 31.7
```

Figure 10: Accuracy Score - Dropout = 0.50

3.7 My Best Model's Confusion Matrix



True \ Prediction	airport_inside	artstudio	bakery	bar	bathroom	bedroom	bookstore	bowling	buffet	casino	church_inside	classroom	closet	clothingstore	computerroom
airport_inside	111	0	9	10	0	29	7	9	0	8	0	0	0	0	0
artstudio	10	0	2	3	0	19	2	2	0	4	0	0	0	0	0
bakery	19	0	33	18	0	25	12	4	0	11	0	0	0	0	0
bar	25	0	16	51	0	16	15	9	0	50	0	0	0	0	0
bathroom	7	0	1	2	0	47	0	2	0	1	0	0	0	0	0
bedroom	16	0	8	9	0	152	2	7	0	5	0	0	0	0	0
bookstore	26	0	11	23	0	13	24	6	0	11	0	0	0	0	0
bowling	24	0	4	4	0	16	2	7	0	7	0	0	0	0	0
buffet	3	0	12	5	0	2	8	0	0	4	0	0	0	0	0
casino	22	0	4	32	0	12	2	8	0	75	0	0	0	0	0
church_inside	9	0	6	9	0	16	7	4	0	4	0	0	0	0	0
classroom	7	0	4	5	0	15	1	1	0	1	0	0	0	0	0
closet	8	0	3	3	0	19	3	2	0	3	0	0	0	0	0
clothingstore	9	0	3	2	0	5	4	3	0	6	0	0	0	0	0
computerroom	11	0	1	1	0	17	0	3	0	2	0	0	0	0	0

Figure 11: Best Model with Dropout (Dropout = 0.50)

3.8 Analyzing My Model

According to the results, I saw that the accuracy score was slightly lower in the model I initially set up because I did not edit and improve it. After adding the dropout layer, I observed a 2.7 percent increase in accuracy score. Accuracy scores could be higher if I had made improvements like this. Also, I kept the batch size as 150 and the learning rate as 0.001. Because he gave the highest accuracy in the non-dropout model.

4 Transfer Learning with ResNet18

4.1 What is fine-tuning?

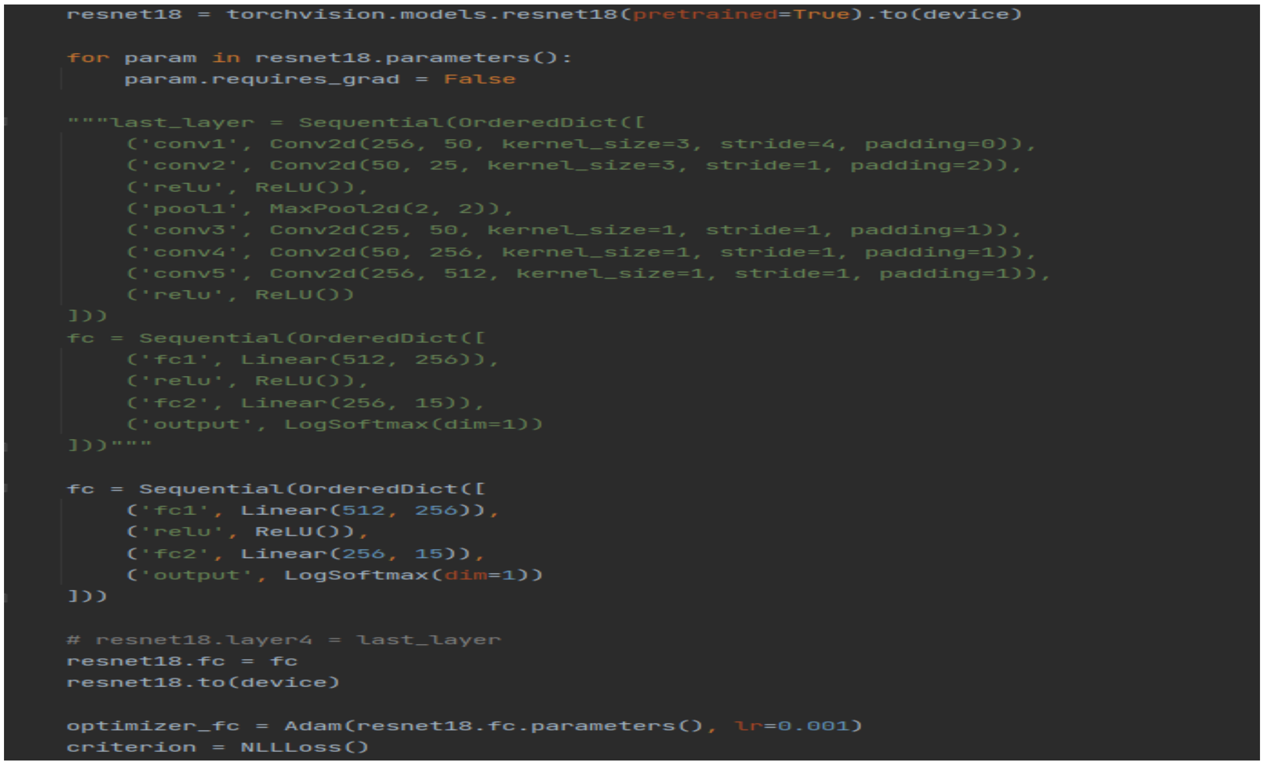
Fine-tuning deep learning algorithms will help to improve the accuracy of a new neural network model by integrating data from an existing neural network and using it as an initialization point to make the training process time and resource-efficient.

4.2 Why should we do this?

Whenever we are given the task of training a deep learning neural network, we usually think of training it from scratch. Training a neural network is a time and resource-intensive process. The neural network needs to be fed tons of data for it to actually work as intended. Gathering the data for the neural network can take long periods of time. With fine-tuning, the deep learning neural networks already have most of the data available for the new model from previous ones. Thus, a lot of time and resources are saved when fine-tuning deep learning models is carried out.
Fine-tuning deep learning models prove useful in training new models. It eases the process by being time-efficient. A huge amount of data is imported from previous models. Hence it can help save a lot of time. It also provides a ton of data for the new model, therefore making the new model much more reliable. Fine-tuning can help push the envelope of deep-learning as developing new algorithms will become much more simplified and time-efficient.

4.3 Why do we freeze the rest and train only FC layers?

We may require to add or remove certain layers depending upon the similarities of the two models. Once we have added or removed layers depending upon the data required, we must then freeze the layers in the new model. Freezing a layer means the layer does not need any modification to the data contained in them, henceforth. The weights for these layers don't update when we train the new model on the new data for the new task.



```
resnet18 = torchvision.models.resnet18(pretrained=True).to(device)

for param in resnet18.parameters():
    param.requires_grad = False

"""last_layer = Sequential(OrderedDict([
    ('conv1', Conv2d(256, 50, kernel_size=3, stride=4, padding=0)),
    ('conv2', Conv2d(50, 25, kernel_size=3, stride=1, padding=2)),
    ('relu', ReLU()),
    ('pool1', MaxPool2d(2, 2)),
    ('conv3', Conv2d(25, 50, kernel_size=1, stride=1, padding=1)),
    ('conv4', Conv2d(50, 256, kernel_size=1, stride=1, padding=1)),
    ('conv5', Conv2d(256, 512, kernel_size=1, stride=1, padding=1)),
    ('relu', ReLU())
]))
fc = Sequential(OrderedDict([
    ('fc1', Linear(512, 256)),
    ('relu', ReLU()),
    ('fc2', Linear(256, 15)),
    ('output', LogSoftmax(dim=1))
]))"""

fc = Sequential(OrderedDict([
    ('fc1', Linear(512, 256)),
    ('relu', ReLU()),
    ('fc2', Linear(256, 15)),
    ('output', LogSoftmax(dim=1))
]))

# resnet18.layer4 = last_layer
resnet18.fc = fc
resnet18.to(device)

optimizer_fc = Adam(resnet18.fc.parameters(), lr=0.001)
criterion = NLLLoss()
```

Figure 12: ResNet18

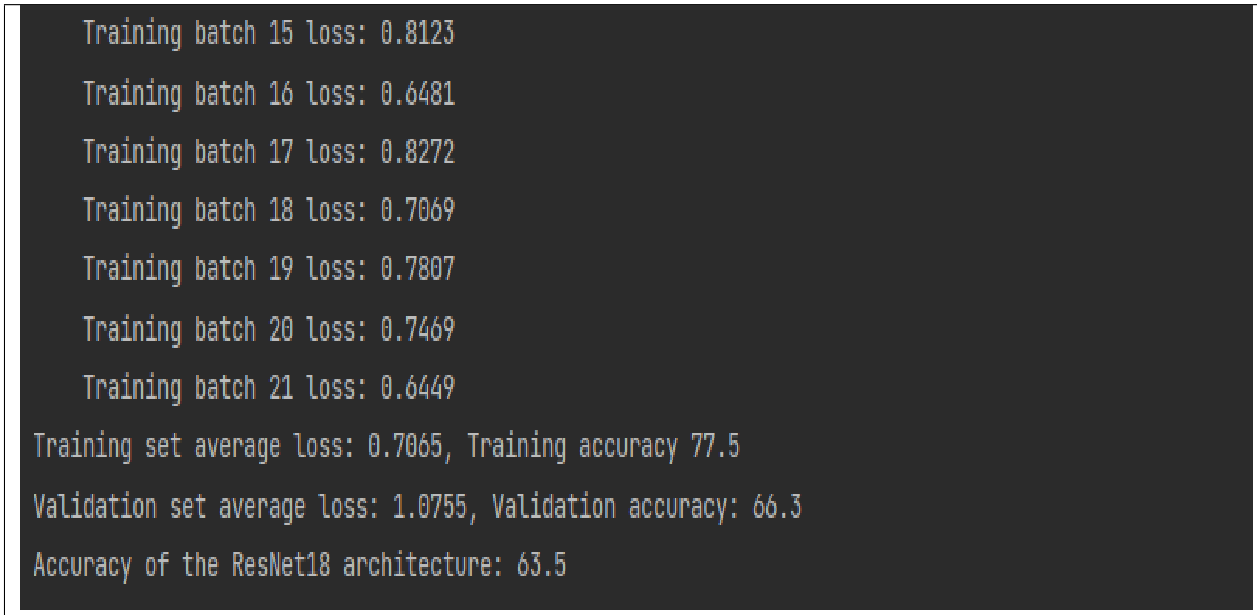
The screenshot above shows the implementation of the ResNet18 model. After the model is downloaded from the library, layers other than the fully connected layer are frozen.

for param in resnet18.parameters():
param.requires-grad = False

Then we create our own fully connected layer and apply it to ResNet. If we want to write the last convolutional layer ourselves, we can apply it to layer4 of ResNet.

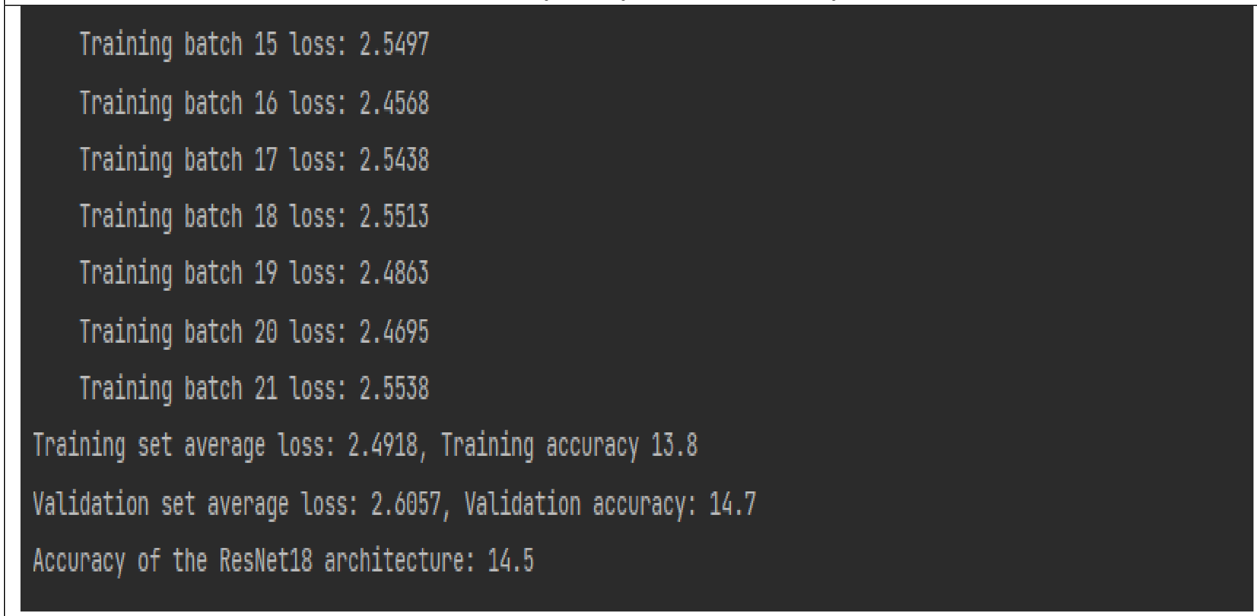
4.4 Evaluating Transfer Learning Model

4.5 Accuracy Scores



```
    Training batch 15 loss: 0.8123
    Training batch 16 loss: 0.6481
    Training batch 17 loss: 0.8272
    Training batch 18 loss: 0.7069
    Training batch 19 loss: 0.7807
    Training batch 20 loss: 0.7469
    Training batch 21 loss: 0.6449
Training set average loss: 0.7065, Training accuracy 77.5
Validation set average loss: 1.0755, Validation accuracy: 66.3
Accuracy of the ResNet18 architecture: 63.5
```

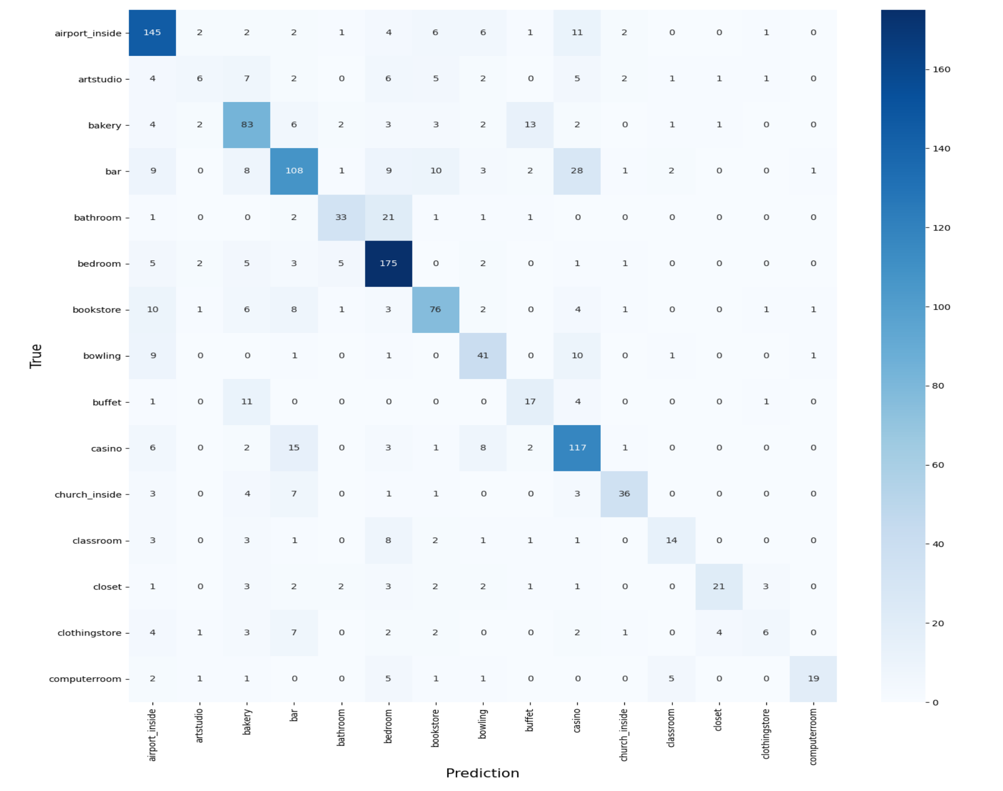
Trained only Fully Connected Layers



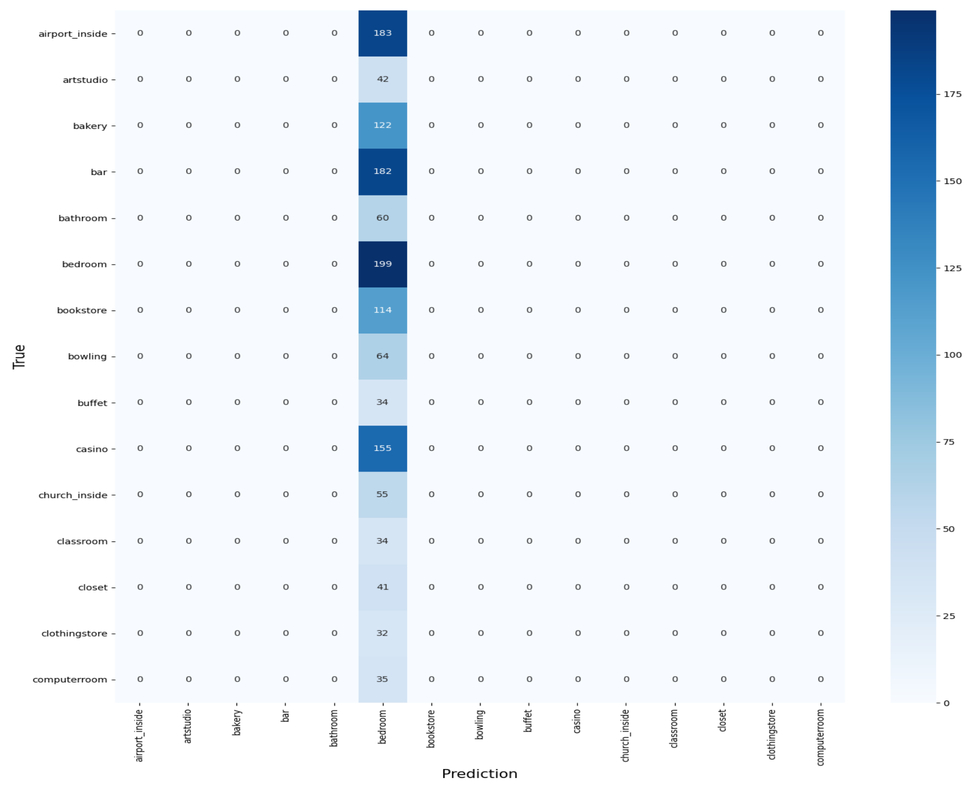
```
    Training batch 15 loss: 2.5497
    Training batch 16 loss: 2.4568
    Training batch 17 loss: 2.5438
    Training batch 18 loss: 2.5513
    Training batch 19 loss: 2.4863
    Training batch 20 loss: 2.4695
    Training batch 21 loss: 2.5538
Training set average loss: 2.4918, Training accuracy 13.8
Validation set average loss: 2.6057, Validation accuracy: 14.7
Accuracy of the ResNet18 architecture: 14.5
```

Trained Fully Connected Layers and Last Convolutional Layer

4.6 Confusion Matrices



Trained only Fully Connected Layers



Trained Fully Connected Layers and Last Convolutional Layer

5 Analyzing Results

When we look at all the results, we can see that the accuracy rate of the CNN architecture I designed is not very good. I observe that I cannot design my architecture well because I do not know the improvement methods very well. I also observed that Batch size does not have much effect on accuracy, while learning rate gives good results in certain ranges. There is an increase in accuracy scores after dropout layers are added. Of course, it was when I applied my fully connected layer to the model ResNet18 where I achieved the highest accuracy scores. On the contrary, when I changed the last convolutional layer and the fully connected layer of ResNet18, the accuracy score dropped to about 15 percent. We have determined that we can get very good results if we adjust the fully connected layer according to our class number and the in-out channel of that model on pre-trained models.