DL ASSIGNMENT 3 PART1

[DRIVE LINK FOR SAVED MODELS], [COLAB LINK FOR THE CODE FILE]

A. Implement a CNN architecture with

a. block1 followed by FC layers, and a softmax layer
b. block1, 2 followed by FCs, and a softmax layer
c. block 1,2, 3, followed by FCs and a softmax layer

```
CNN(
  (block1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (block2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (block3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=512, out_features=128, bias=True)
    (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

- For part a, consider block1 only
- For part b, consider block1 and block2 only.
- For part c, consider the whole architecture.

B. For all the three architectures apply the Tanh or ReLU activation function on all layers.

- ReLU was chosen.

C. Implement Dropout and use

a. After convolutional layers
   Dropouts were tried after every block of convolution layers, but Batch Normalization gave better results.
b. Between FC layers
   Dropouts have been applied after all the fully connected layers except the output layer.

DL ASSIGNMENT 3 PART1

1. Visualize 10 random images from each class.

- Dolphin

```
Class 0:
```

- Flower

```
Class 1:
```

- Keyboard

```
Class 2:
```

- Tree

```
Class 3:
```

- Building

```
Class 4:
```

- House

```
Class 5:
```

DL ASSIGNMENT 3 PART1

- Forest

Class 6:

- Bike

Class 7:

- Tree

Class 8:

- Fishing

Class 9:

2. Analyze the accuracy and loss while adding block 1, block 2, and block 3 with mentioned non-linearities.

- Using just block1 gives a not very bad model.
- Using block1 and block2, the model starts overfitting so we added BatchNorm and L2 norms.
- Using all 3 blocks, we got an improved model hence we selected this architecture.

DL ASSIGNMENT 3 PART1

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/block_1_only.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=84, Train-Loss: 0.3532, Val-Loss: 0.5595, Train-Acc: 87.44 %, Val-Acc: 81.07 %
```

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/block_1_and_2.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=76, Train-Loss: 0.2198, Val-Loss: 0.5322, Train-Acc: 92.49 %, Val-Acc: 82.00 %
```

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/block_1_2_and_3.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Trai
```

```
Epochs=60, Train-Loss: 0.2010, Val-Loss: 0.5346, Train-Acc: 92.89 %, Val-Acc: 83.07 %
```

DL ASSIGNMENT 3 PART1

3. Analyze the accuracy and loss while changing the dropout probability. Report the results obtained on CNN with all three blocks. Try at least 3 different dropout probabilities.

- All the dropout probabilities were set to 0.15, 0.25, and 0.50. Results have been shown below.
- We used the 3 block architecture that was found to be the best from the above analysis.
- From the following analysis, we found that **dropout=0.5** gave the best results and this will be our **best model** for the subsequent analysis.

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/dropout_0_15.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=42, Train-Loss: 0.1866, Val-Loss: 0.5679, Train-Acc: 93.11 %, Val-Acc: 81.20 %
```

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/dropout_0_25.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=53, Train-Loss: 0.1560, Val-Loss: 0.5363, Train-Acc: 94.59 %, Val-Acc: 82.40 %
```

DL ASSIGNMENT 3 PART1

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/dropout_0_50.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Trai
```

```
Epochs=57, Train-Loss: 0.2125, Val-Loss: 0.4951, Train-Acc: 92.59 %, Val-Acc: 84.13 %
```

4. Report the best accuracy with model architecture and detailed analysis of choosing specific hyperparameters, different training techniques, and data augmentation used (if any).

The best model: test accuracy = 84.13%, training accuracy = 92.59%.

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/best_model.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=57, Train-Loss: 0.2125, Val-Loss: 0.4951, Train-Acc: 92.59 %, Val-Acc: 84.13 %
```

The Best Model's Architecture:

DL ASSIGNMENT 3 PART1

```
CNN(
  (block1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (block2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (block3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=512, out_features=128, bias=True)
    (5): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

Hyperparameters:

- Batch-Size=64
- Epochs=100
- L2 Regularization, lambda = 1e-5

Training Techniques:

- Adam Optimizer with default learning rate=0.001 and L2 regularization of 1e-5.

```
optimizer = torch.optim.Adam(model.parameters(), weight_decay=0.00001)
```

Data Augmentation:

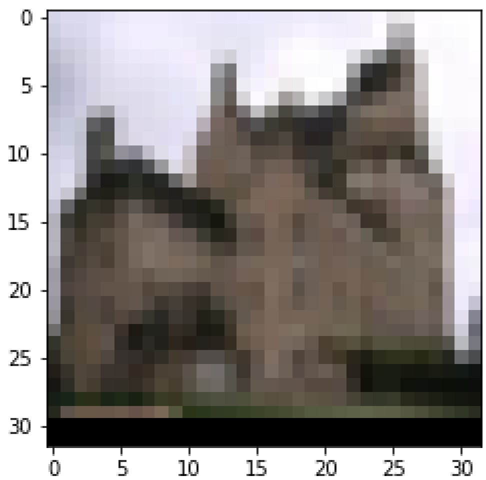
DL ASSIGNMENT 3 PART1

```
train_transform = T.Compose([
  T.ToPILImage(),
  T.RandomHorizontalFlip(),
  T.RandomCrop(32, padding = 4),
  T.ToTensor(),
])
```

- RandomHorizontalFlip()
  - creating additional samples by horizontally flipping original samples.
- RandomCrop(32, padding = 4)
  - creating additional samples by first padding the original samples and then taking a random crop of the same size as the original image sample.

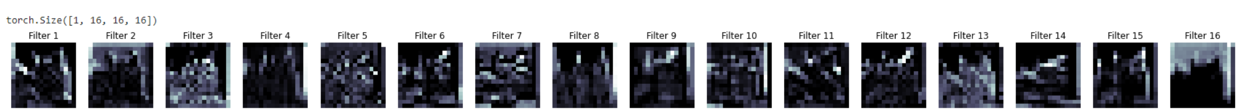
5. Visualize some convolutional filters and feature maps obtained after each block.

```
Input Image
Class: 4
```

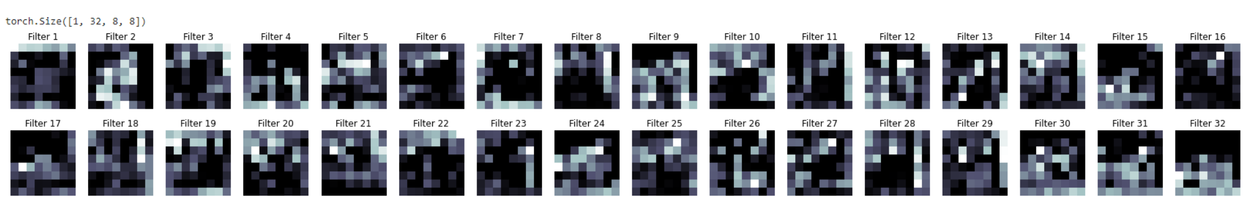


Feature Maps

- After Block1

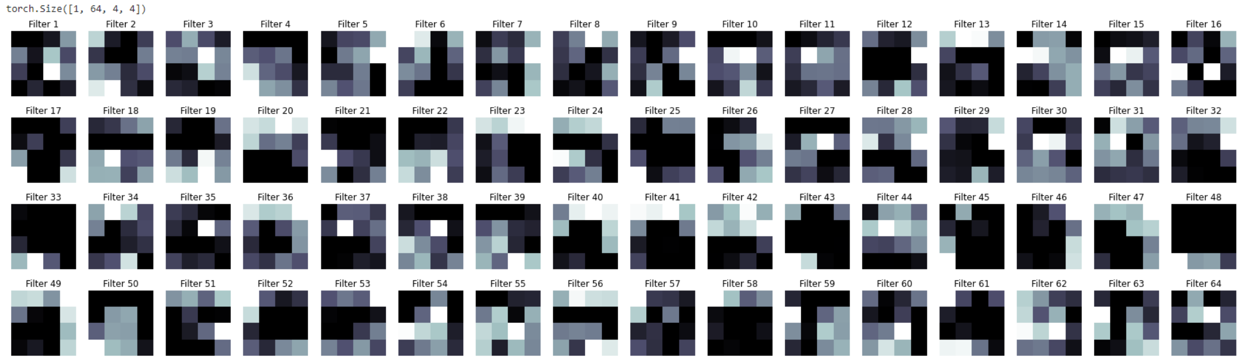


- After Block2



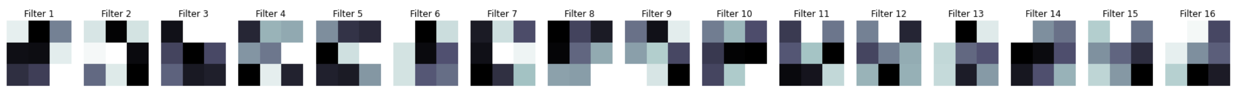
- After Block3

DL ASSIGNMENT 3 PART1

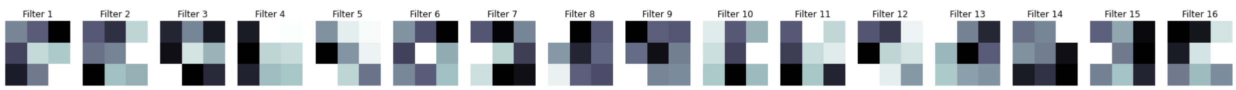


Convolution Filters

- Block1 - convX1

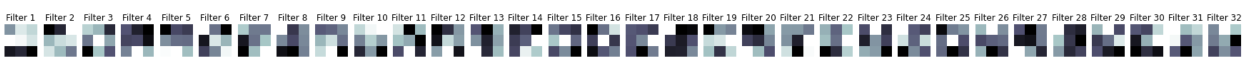


- Block1 - convX2



- Block2 - convY1

- Block2 - convY2



- Block3 - convZ1

- Block3 - convZ2

6. Analyze the results of the best model when all the activation functions are removed. Justify the performance drop.

- Train accuracy dropped from 92 to 84 while test accuracy dropped from 84 to 80.
- When we drop the activation functions, indirectly we are switching to the linear activation function.
- We already know that linear activation functions render the hidden layers useless since the hypothesis generated can be expressed as just a linear function of the inputs as we saw in one of the questions asked in the mid-sem examination.
- Without Activations Model Evaluations

DL ASSIGNMENT 3 PART1

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P1_Models/best_model_no_acti.pt"

checkpoint = torch.load(PATH)

model = CNN().cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Trai
```

```
Epochs=53, Train-Loss: 0.4378, Val-Loss: 0.5902, Train-Acc: 84.85 %, Val-Acc: 80.53 %
```

7. [Bonus] During the demo you are given labels of test data (format will be the same as training data), you have to evaluate the test accuracy of your best model.
https://colab.research.google.com/drive/1r4TOgR0X2woRziqrib6hGBZ6AE9jhosu?usp=sharing

References:

- https://www.youtube.com/playlist?list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
- https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html