CSCI 5922 -Neural Networks and Deep Learning
Lab - 2 Solutions

Gowri Shankar Raju Kurapati
Student ID 110568555

September 26, 2022

1 Influence of Regularization

1.1 Dataset & Hyperparameters Used

- CIFAR 10 Dataset
  - Source : From *torchvision.datasets*
  - The CIFAR-10 dataset used consists of 50000 32x32 color images in 10 classes, with 5000 images per class.
  - Since these are color images, each image has the tensor dimension of 3 * 32 * 32 where the first dimension is RGB colors.
  - The 10 classes (labels) are $airplane(0), automobile(1), bird(2), cat(3), deer(4), dog(5), frog(6), horse(7), ship(8), truck(9)$
- Hyperparameters
  - training/validation/test split is 70 : 15 : 15
  - The input size to the neural network is 3 * 32 * 32 image.
  - As stated above, the output layer consists of 10 neurons to predict 10 classes of the CIFAR 10 dataset.
  - The learning rate is set to 0.001 with a batch size of 64 and is held constant for the experiment.
  - The models are trained for 10 epochs.
- Methods:
  - All (Eight) neural networks are Convolution Neural Networks with a series of convolution blocks. Each convolution block consists of a
    * conv2d layer with max pool
    * followed by a conv2d layer, max pool and batchNorm (if it is opted as regularizer)
  - Two CNN architectures are used
    * Two Convolution blocks followed by three fully connected Layers

        * Three Convolution blocks followed by three fully connected Layers.
        * Convolution Block has been described in the previous bullet point.
    - I have considered three regularizers
        * Batch Normalization
        * Drop out at Fully Connected Layers
        * Training with Data Augmentation.
    - Loss function used for the experiment is the Cross Entropy.
    - Adam Optimizer with a learning rate of 0.001 is used.
    - I have iterated the experiment with either of two regularizers (Batch Norm & Drop Out) and no regularizer with two architectures defined above and on the training dataset specified(Without any augmentation) resulting in six models.
        * The models are named Lab2_P1 _Regularizer_<RegularizerUsed/NoRegularizer>_<2/3>LayerCNN to capture the number of the convolution blocks and regularizer used.
    - I have also iterated with no regularizer but training the models by augmenting the training dataset during training on two architectures resulting in two different models.
        * The models are named as Lab2_P1 _Regularizer _DataAugmentation_<2/3>LayerCNN.
        * The Augmentation used is a Random Horizontal Flip followed by random cropping of the image with different paddings.
    - All the models are trained on Google Collab using the single GPU instance.

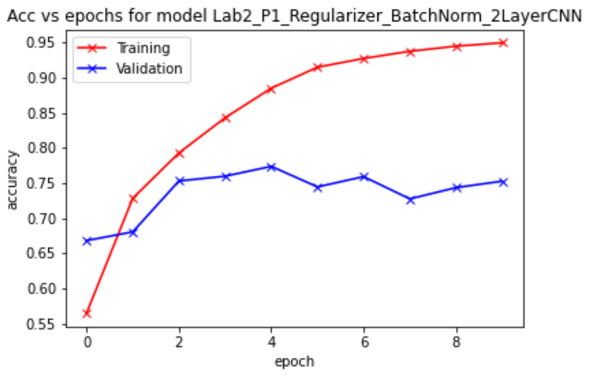
1.2 Results & Analysis

Reporting **Training Times** in seconds for the Eight CNN Models for comparison

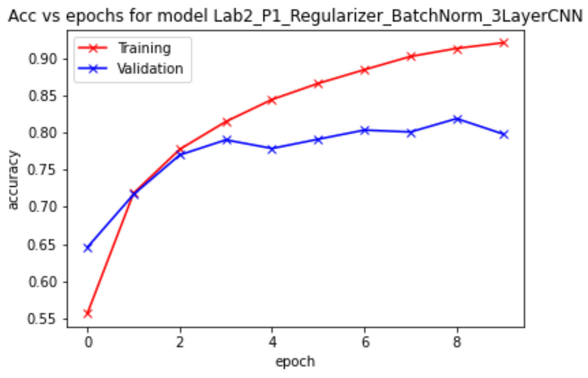
CNN Block Layers / Regularizer	Batch Norm	Drop Out	Data Augmentation	No Regularizer
2	106.56	103	136.60	109.15
3	115.19	108.6	147.88	118.6

- From the table above, we could infer that,
    - [**3 Layer Networks and DataAugmentation Take more Time to Train**] When the 2 Layer CNN training times are compared with three layers CNN of the same regularizer, the time has increased significantly with an average difference of 9 seconds. Since the number of epochs is constant for all the models, the 3-layer network has to update significantly more model parameters compared to the corresponding 2 Layer.
      Also, among all the different variations of regularizer used, training with data augmentation takes significantly more time. This can be because of the augmentation taking place during the batch training. The trainLoader being used to generate the batch of images for the training is augmented at the runtime of the train loader for the batch. This processing time is included during the training and hence a significant increase.

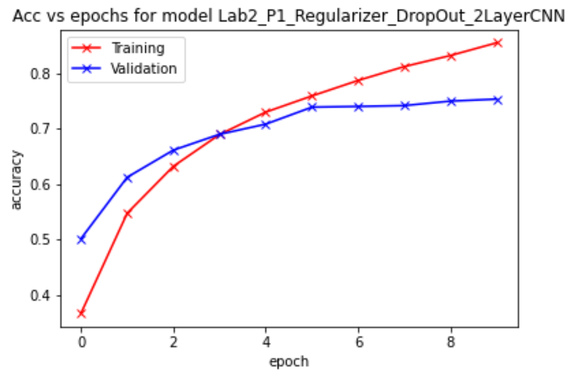
Reporting **Final Validation Accuracies** and **Learning Curves** for the Eight CNN Models for comparison



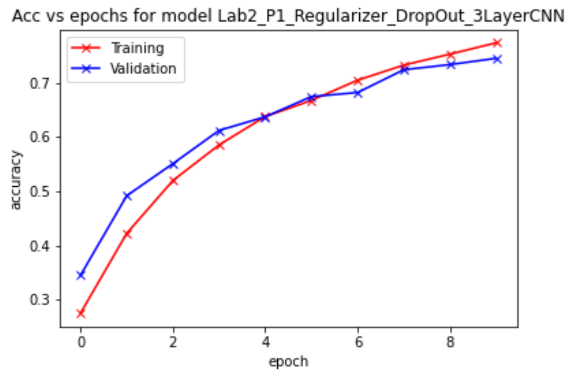
(a) Batch Norm - 2 Layer CNN



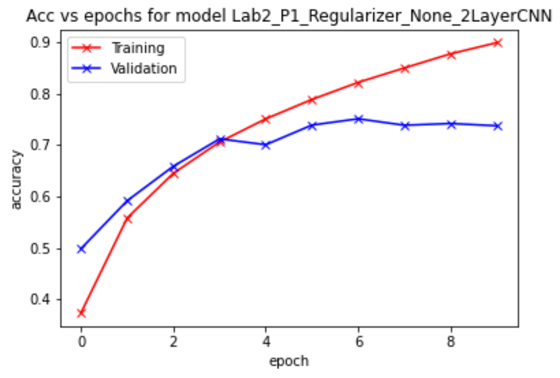
(b) Batch Norm - 3 Layer CNN



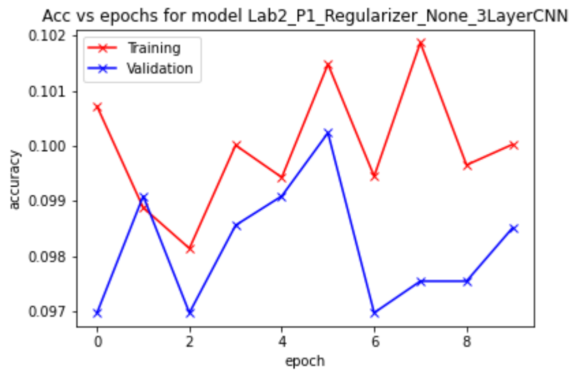
(a) Drop Out - 2 Layer CNN



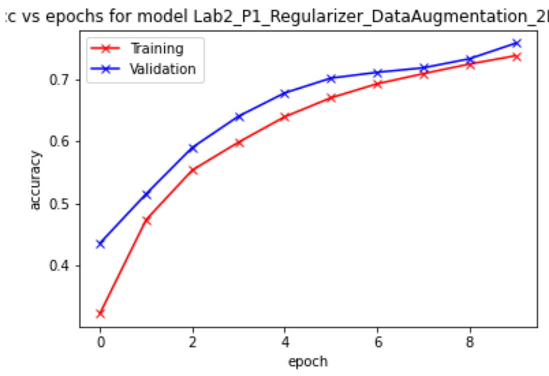
(b) Drop Out - 3 Layer CNN



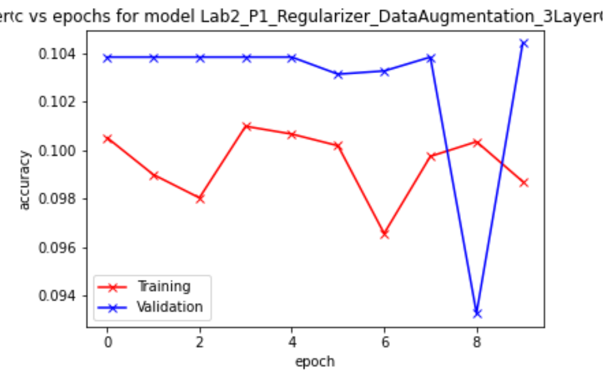
(a) No Regularizer - 2 Layer CNN



(b) No Regularizer - 3 Layer CNN



(a) Data Augmentation - 2 Layer CNN



(b) Data Augmentation - 3 Layer CNN

CNN Block Layers / Regularizer	Batch Norm	Drop Out	Data Augmentation	No Regularizer
2	73.97	75.49	**76.23**	74.08
3	**79.07**	75.08	9.94	9.76

- From the above table we can infer that
  - [**2 Layer CNN - Data Augmentation works Better**] When no regularizer is used, we see from the corresponding learning curve that there is a little overfitting as the validation accuracy gets stagnant but training accuracy increased. Though using batch normalization doesn't significantly increase the performance, it overfits more. It may be because the model is not able to learn any new features. This assumption is true because once we use Data Augmentation to train thereby making the model more robust to different angles of the image, we see that overfitting is gone (from the corresponding learning curve) and accuracy is increased by nearly 2.5 %.
  - [**3 Layer CNN**] We clearly observe that the from the learning curves of Data Augmentation 3 Layer and No Regularizer, the accuracies are very low and there is no pattern to observe. This is expected as we are trying to learn a lot of features (model parameters compared to 2Layer CNN) in just 10 epochs. Since no regularizer is used and on top of it we are also augmenting data with an intention of learning robust features but not giving enough epochs for the model to train as converging takes more time, and a lot of epochs are needed to make proper inferences. Batch Normalization gives more accuracy as normalizing the features at every convolution layer block will ease the updates to move faster in the loss space, thereby decreasing the number of epochs for convergence.
  - [**Comparing 2 Layer - 3 Layer CNN for a regularizer**] Since we have established that number of epochs wasn't sufficient for 3 Layer Data Augmentation and NO regularizer, I am omitting the comparison for the corresponding 2 Layer network. We can then observe that the overfitting decreased when the number of layers is increased (so are the accuracies) for the network using the same regularizer. This can be explained as the model capacity has been increased to accommodate more features to learn.
- Thus, the top performing model is 3 Layer CNN with Batch Normalization (Lab2_P1_Regularizer_BatchNorm _3LayerCNN). The Test Accuracy achieved when it is trained on both training split and validation split is **80.81%** .

2 Interpreting CNN Representations

2.1 Network Used And Interpretations used

3 Layer CNN with Batch Normalization (Lab2_P1_Regularizer_BatchNorm _3LayerCNN)

- We use the best model from question-1 to interpret its representations.

[**Interpretation Used**] Retrieving images with similar feature representations. The Feature representations are taken on two spots.

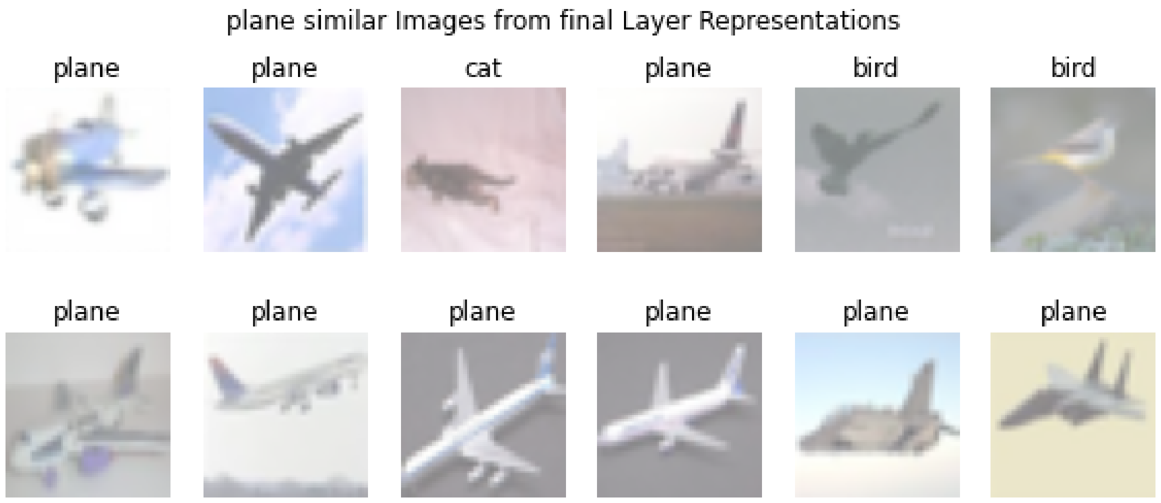
- At the end of the second Convolution Layer Block, when flattened gives you 8192 length vector.
- At the output Layer (at the end of the last fully connected Layer), when flattened has 10 length feature vector

[**Method Used**] For an image in each class, I have taken the representation vector and did a cosine similarity with all the images of the test Dataset and displayed the top 10 results with the highest similarity. The intuition behind this is that at a certain level, images having certain features will be nearer to each other in a feature space. By looking at the images, we can understand what features are more prominent at the end of each layer.

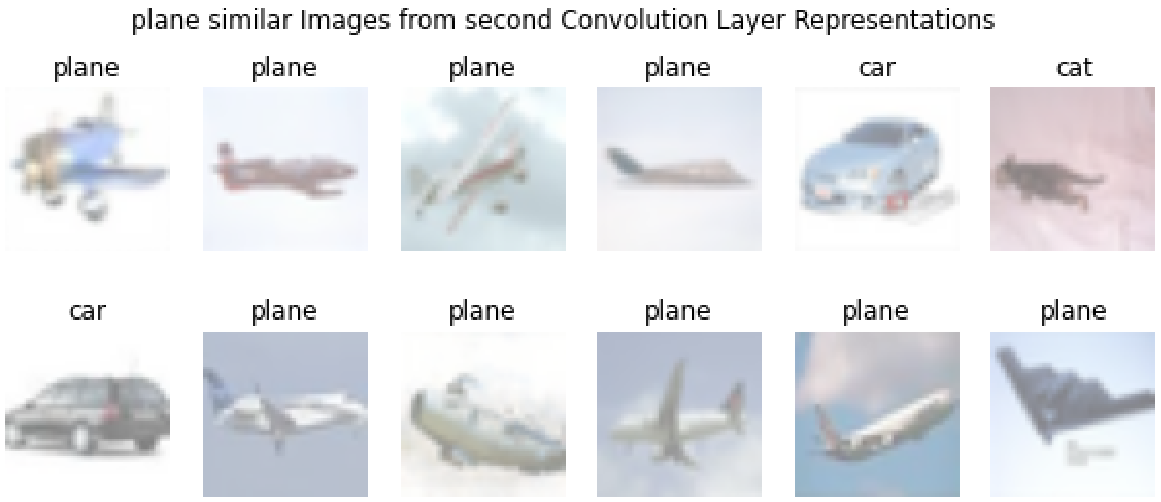
2.2 Results & Analysis

- [**PLANE SIMILAR IMAGES**] For instance, if we retrieve similar images of a plane (*figure -1 (a) & (b)*), we can see that when convolution layer 2 feature representations are used (*figure-1(b)*) the similar images are mostly planes stating that the classification is being done with good accuracy by this layer. But we can also see observe that there are images of cars and cat. Since these are the top 10 similar images, the high-level feature might be an image with a sharp nose, as we can see the car image is like protruding nose tip, and also for the cars, the front portion is like a plane's tip/nose.
  Also, when the same level comparison is made with representations of the end layer, we can see all the images are plane except the same cat image we talked about earlier. Here, we don't see any cars as the model would have learned what a plane should be like at this point (As this is the last layer) but it still misclassifies the cat and a bird as planes. The model should be seeing a tip/nose, wings, and a tail. As both the cat image and bird image have all three, the model might be misclassifying it.

Figure 1: Similar Images for the "first Image of first row" [PLANE] in each section



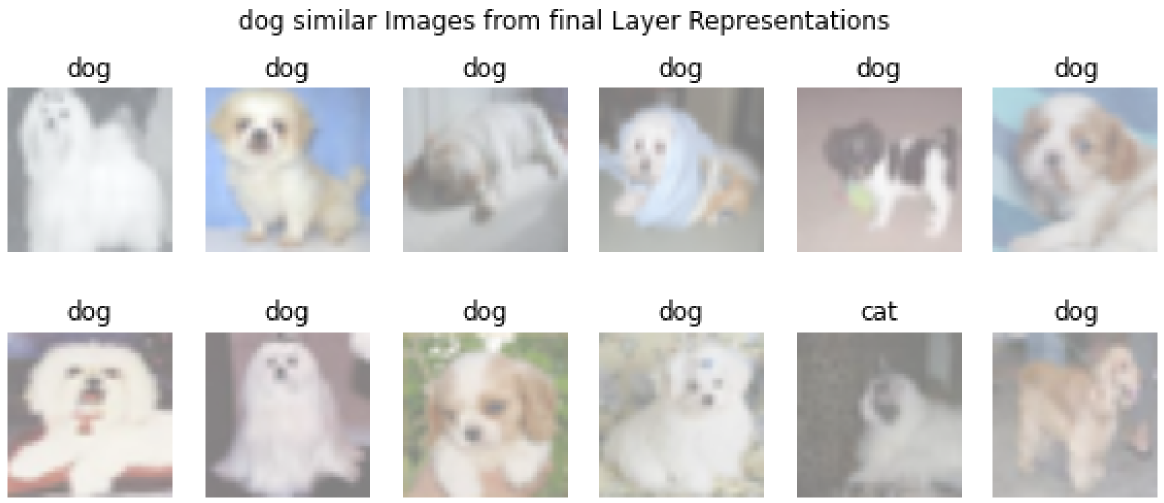
(a) Plane - End of Fully Connected Layer Representation


plane similar Images from second Convolution Layer Representations
plane
plane
plane
plane
car
cat
car
plane
plane
plane
plane
plane

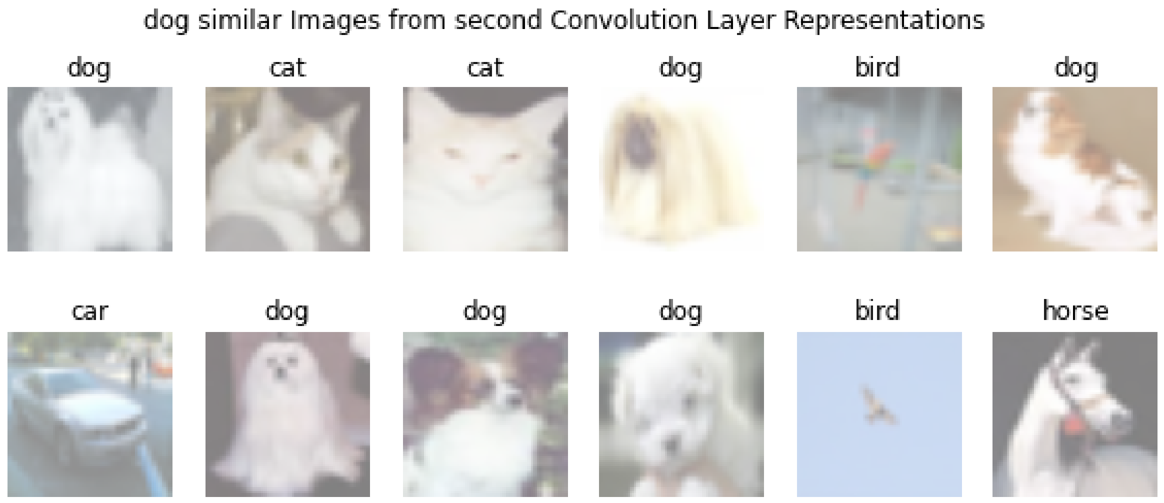
(b) Plane - End of Second Convolution Layer Block Representation

- [**DOGS SIMILAR IMAGES**] Taking another example for Dogs (*figure - 2 (a) & (b)*), we can observe that for similar images retrieved from the second convolution Layer input, we can see a couple of cat images, a horse, a bird, and a car. Maybe the high-level feature is the face with two eyes and a muzzle. That is why even the horse image has a very high similarity at this stage in the network. // On the contrary, when the images from the final layer representations are compared, we can see all of them as dogs except one cat image. Model might be learning that dog has two eyes, a round face and NO protruding ears (which might be the differentiating factor between other images), and that is why horse image and cat images which had protruding ears and high similarity before have not appeared in the top similarity results at the end of this layer.

Figure 2: Similar Images for the "first Image of first row" [DOG] in each section



(a) Dog - End of Fully Connected Layer Representation

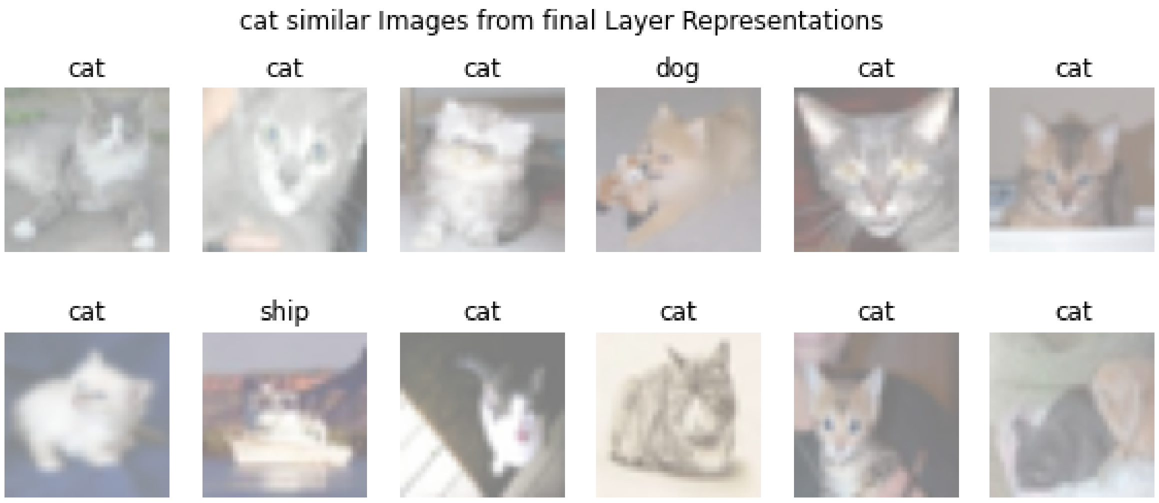

dog similar Images from second Convolution Layer Representations
dog
cat
cat
dog
bird
dog
car
dog
dog
dog
bird
horse

(b) Dog - End of Second Convolution Layer Block Representation

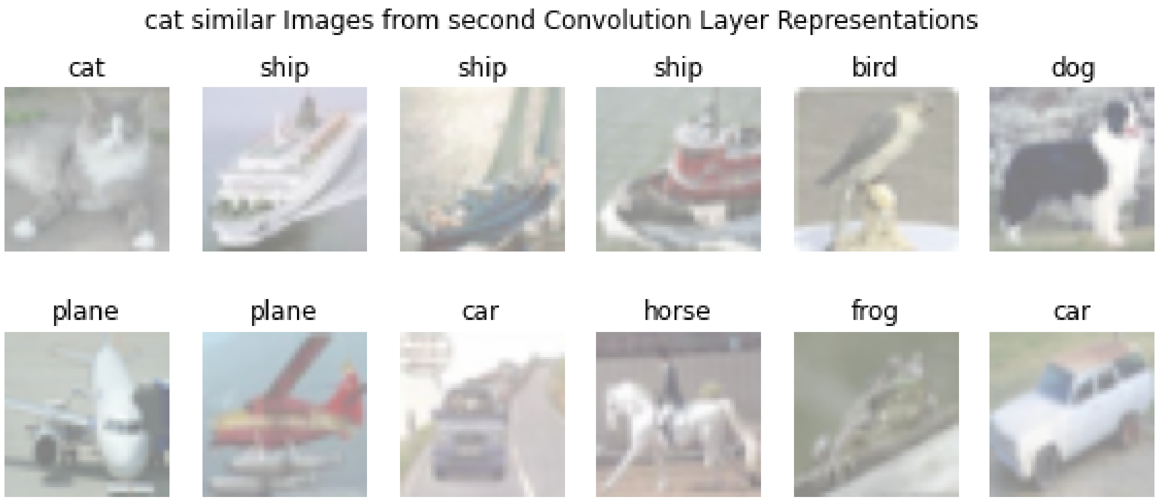
RESULT From the above examples representations, we can confidently say that the network starts with learning low-level features such as eyes, muzzle, noses, ears (whether protruding or not), etc and use these low-level features to map them to high-level features and use those to predict the final class.

Adding similar images for all the other classes. This gives more ideas on differentiating features between the classes.

Figure 3: Similar Images for the ”first Image of first row” [CAT] in each section

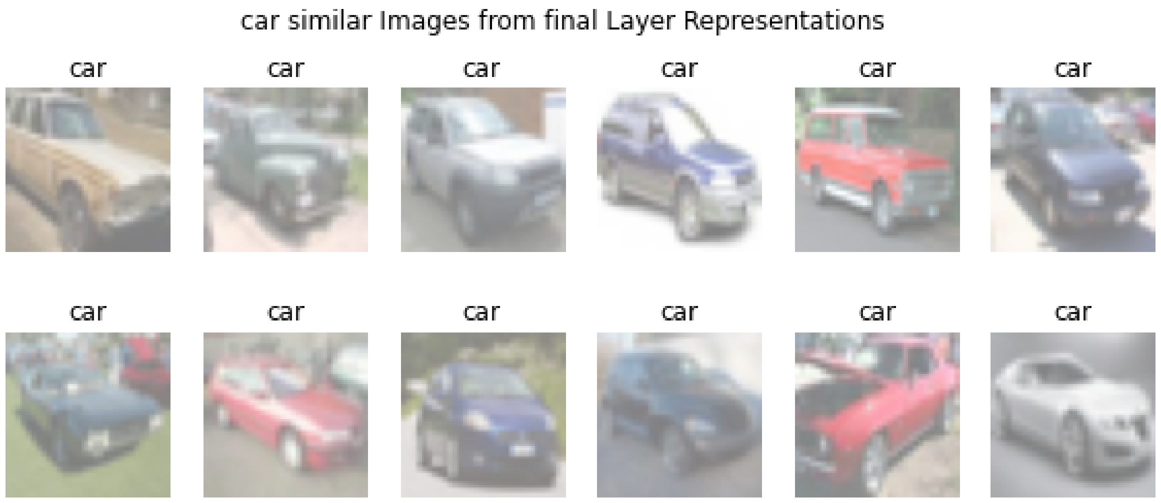


(a) Cat - End of Fully Connected Layer Representation

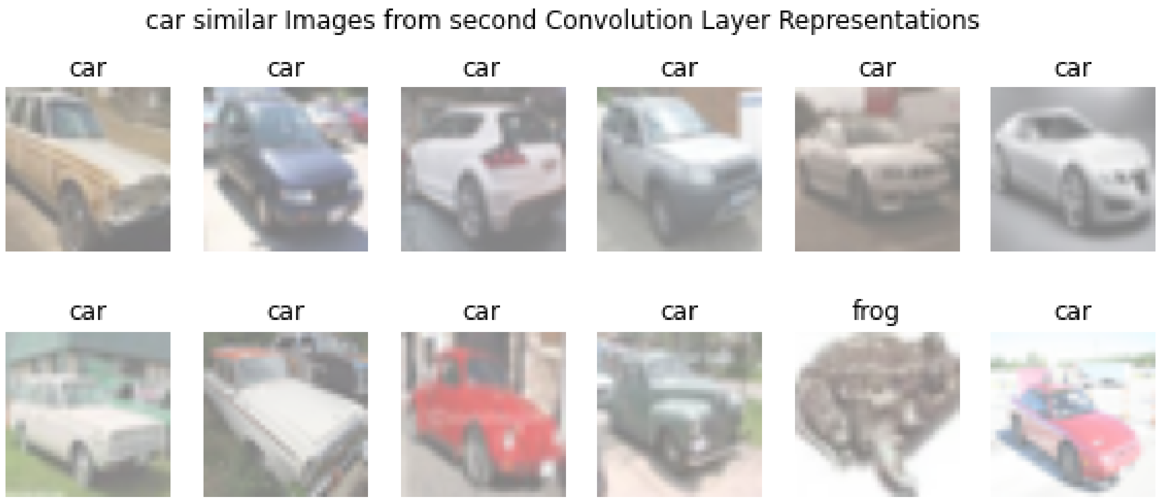

cat similar Images from second Convolution Layer Representations
cat
ship
ship
ship
bird
dog
plane
plane
car
horse
frog
car

(b) Cat - End of Second Convolution Layer Block Representation

Figure 4: Similar Images for the ”first Image of first row” [CAR] in each section

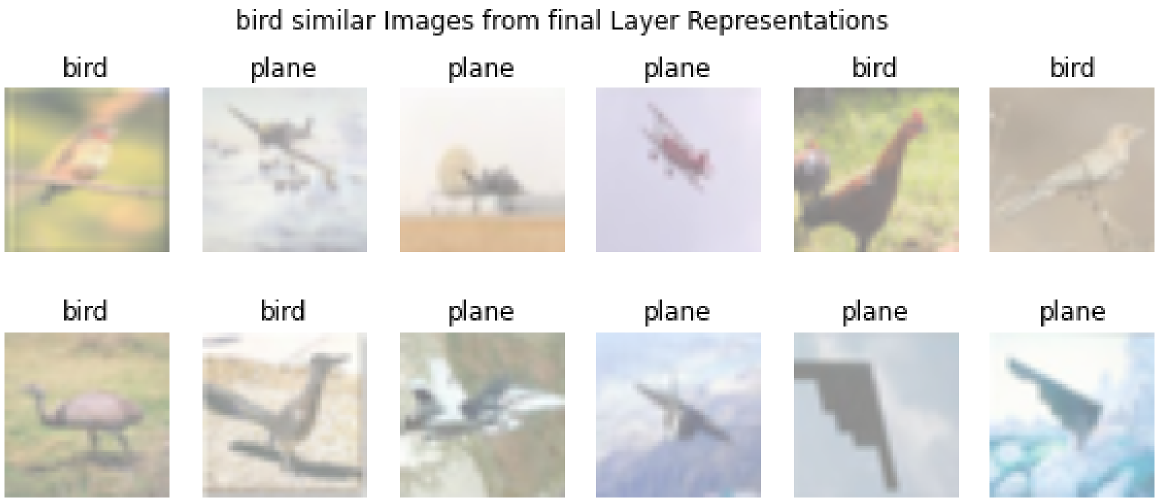


(a) Car - End of Fully Connected Layer Representation

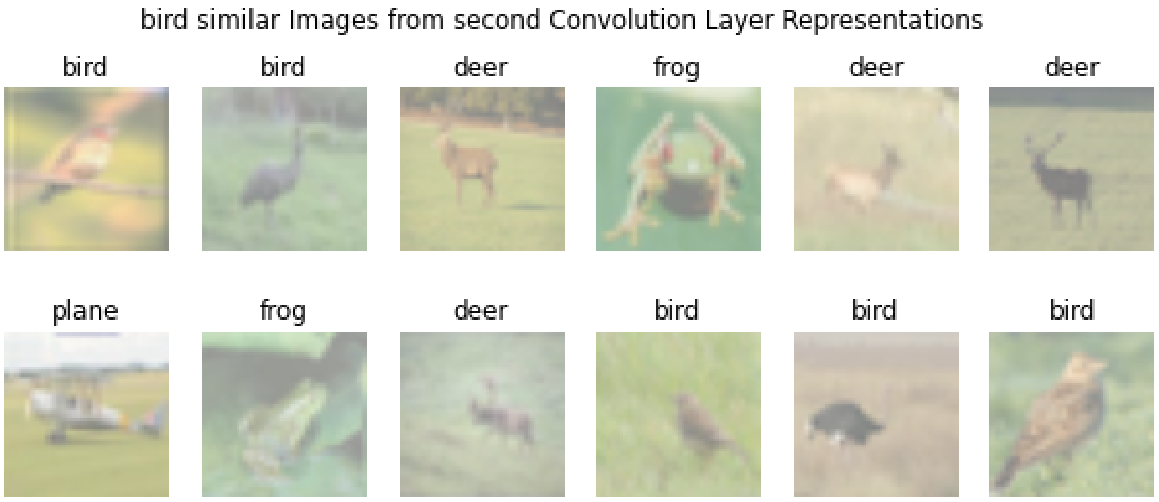

car similar Images from second Convolution Layer Representations
car
car
car
car
car
car
car
car
car
car
frog
car

(b) Car - End of Second Convolution Layer Block Representation

Figure 5: Similar Images for the "first Image of first row" [BIRD] in each section

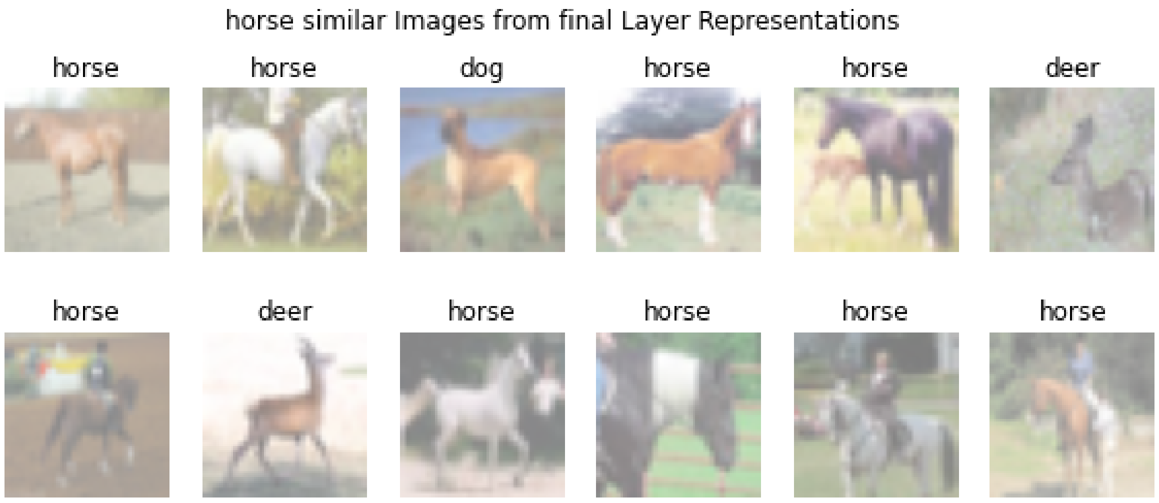


(a) Bird - End of Fully Connected Layer Representation

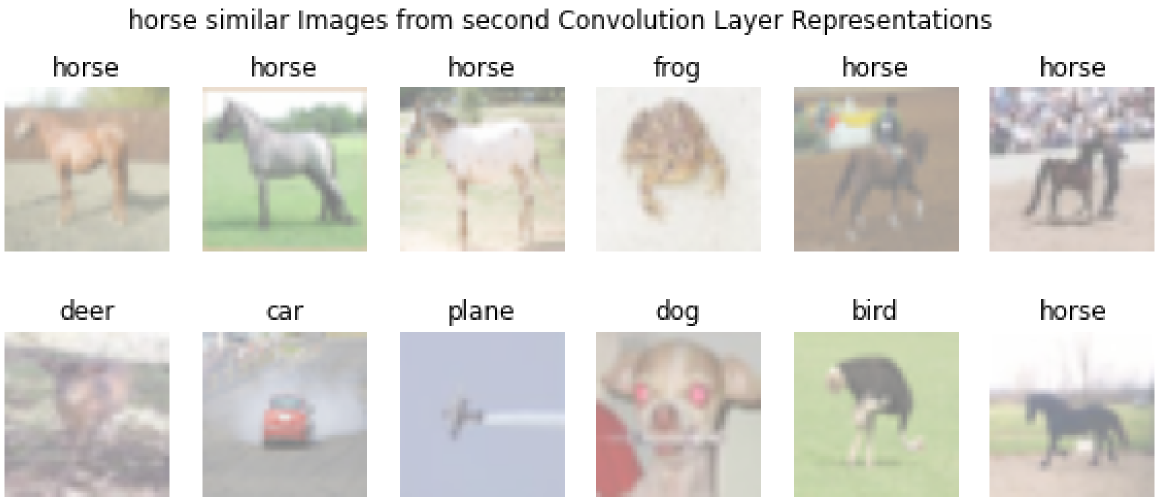

bird similar Images from second Convolution Layer Representations
bird
bird
deer
frog
deer
deer
plane
frog
deer
bird
bird
bird

(b) Bird - End of Second Convolution Layer Block Representation

Figure 6: Similar Images for the "first Image of first row" [HORSE] in each section

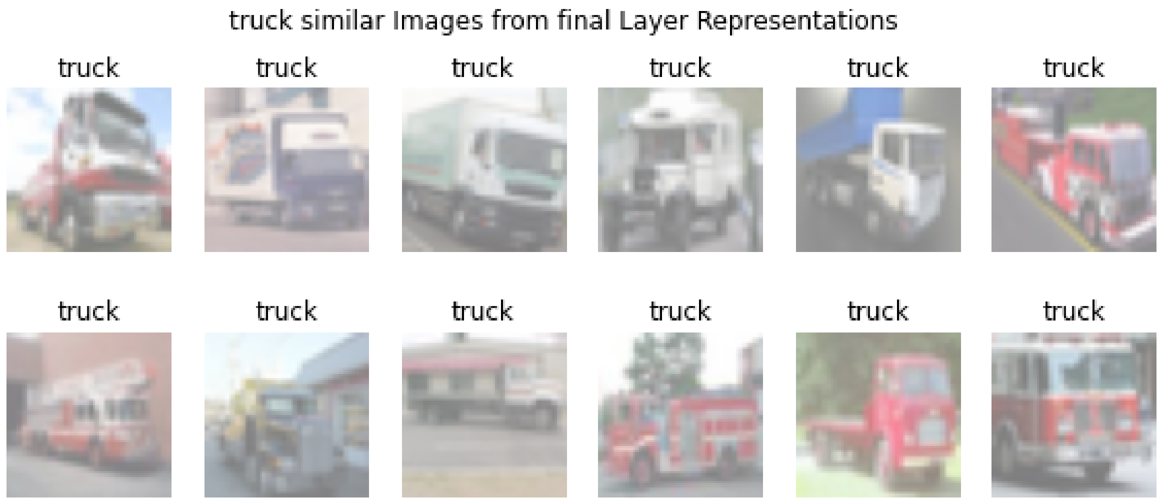


(a) Horse - End of Fully Connected Layer Representation

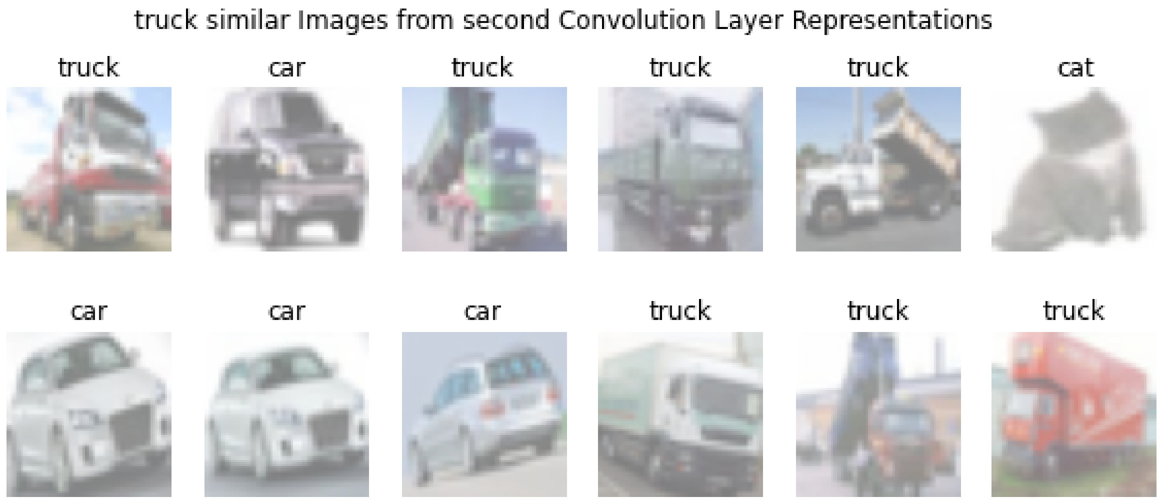

horse similar Images from second Convolution Layer Representations
horse
horse
horse
frog
horse
horse
deer
car
plane
dog
bird
horse

(b) Horse - End of Second Convolution Layer Block Representation

Figure 7: Similar Images for the ”first Image of first row” [TRUCK] in each section

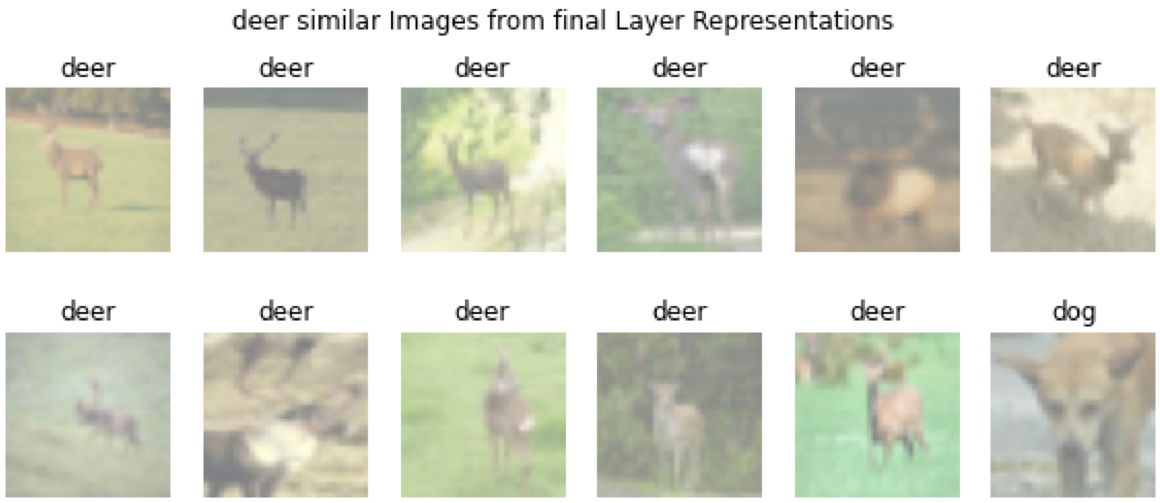


(a) Truck - End of Fully Connected Layer Representation

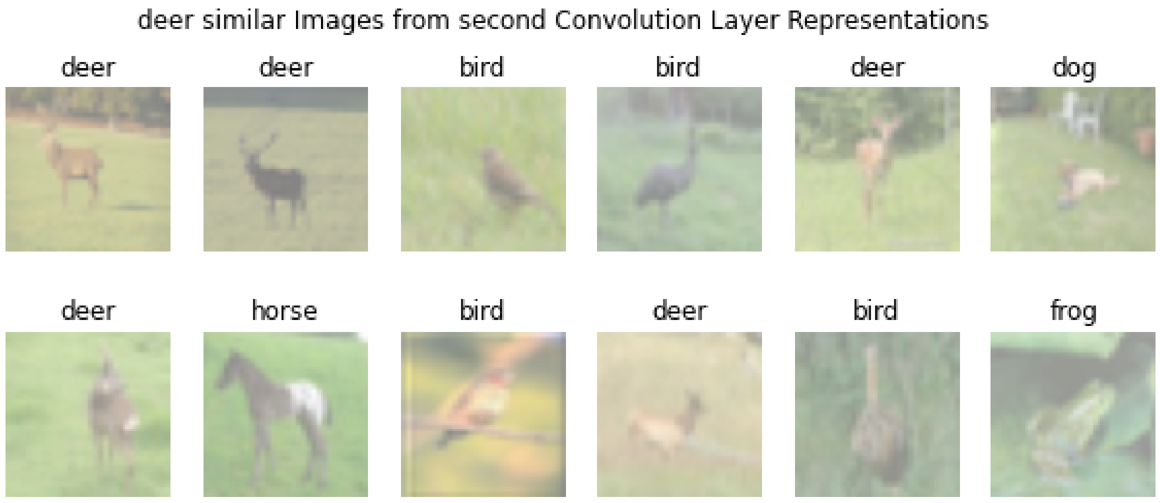

truck similar Images from second Convolution Layer Representations
truck
car
truck
truck
truck
cat
car
car
car
truck
truck
truck

(b) Truck - End of Second Convolution Layer Block Representation

Figure 8: Similar Images for the "first Image of first row" [DEER] in each section

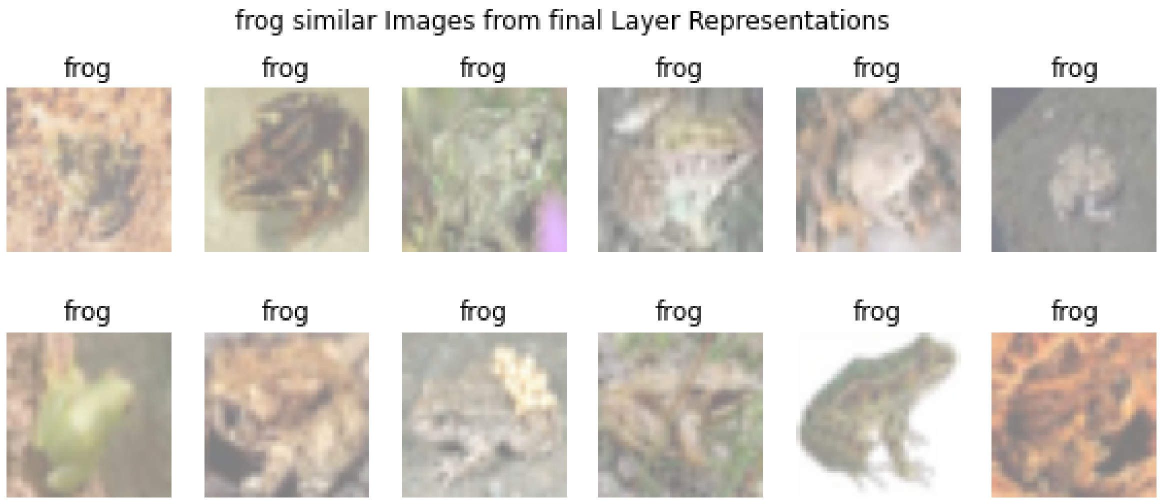


(a) Deer - End of Fully Connected Layer Representation

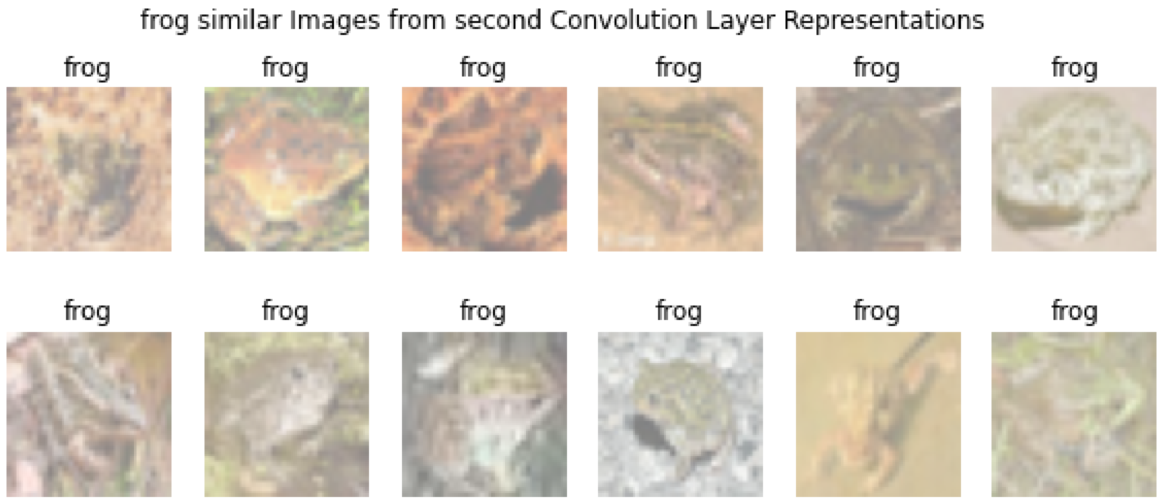

deer similar Images from second Convolution Layer Representations
deer
deer
bird
bird
deer
dog
deer
horse
bird
deer
bird
frog

(b) Deer - End of Second Convolution Layer Block Representation

Figure 9: Similar Images for the "first Image of first row" [FROG] in each section

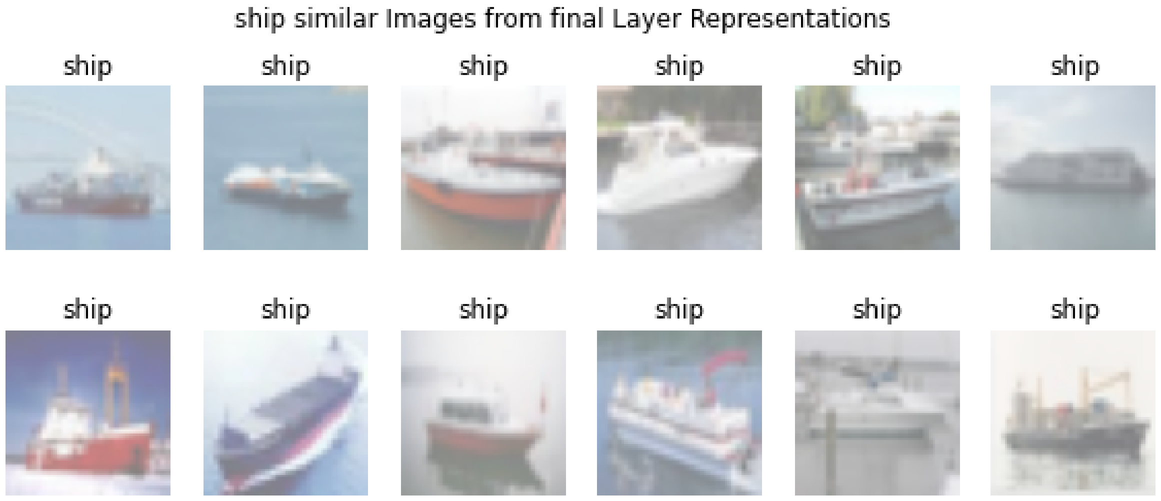


(a) Frog - End of Fully Connected Layer Representation

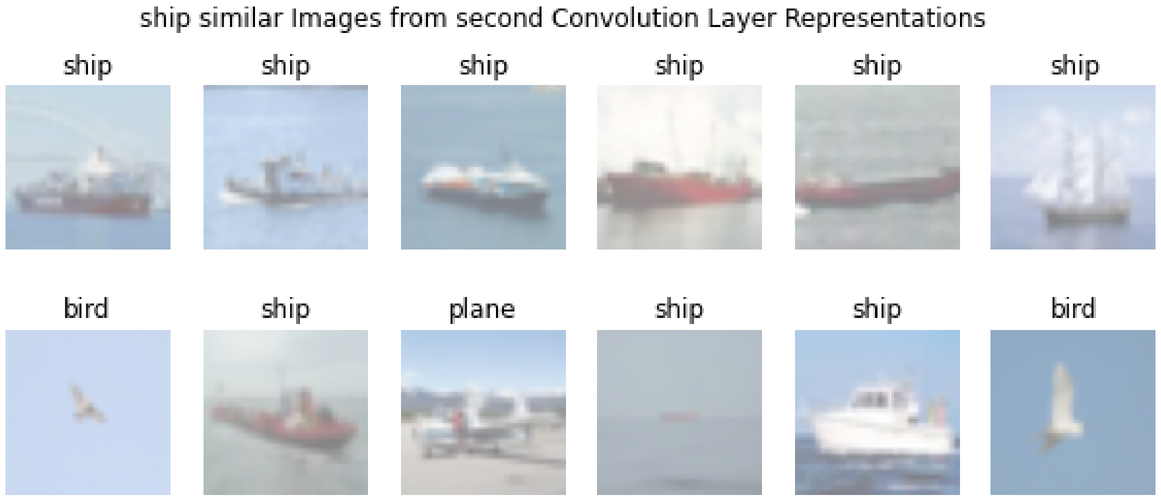

frog similar Images from second Convolution Layer Representations
frog
frog
frog
frog
frog
frog
frog
frog
frog
frog
frog
frog

(b) Frog - End of Second Convolution Layer Block Representation

Figure 10: Similar Images for the ”first Image of first row” [SHIP] in each section



(a) Ship - End of Fully Connected Layer Representation


ship similar Images from second Convolution Layer Representations
ship
ship
ship
ship
ship
ship
bird
ship
plane
ship
ship
bird

(b) Ship - End of Second Convolution Layer Block Representation

3 CODE

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision.utils as utils
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import numpy as np
import time
```

```
#Using GPU if exists
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
device
```

```
device(type='cuda')
```

```
# Load Data
CIFAR10_dataset = datasets.CIFAR10(root='dataset/CIFAR10',
                                         train=True,
                                         transform=transforms.ToTensor(), download=True)
```

```
transform_images = transforms.Compose(
               [transforms.RandomHorizontalFlip(),
                transforms.RandomCrop(size=32, padding=[0, 2, 3, 4]),
                transforms.ToTensor()])
CIFAR10_dataset_augmented = datasets.CIFAR10(root='dataset/CIFAR10',
                                         train=True,
                                         transform=transform_images, download=True)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
CIFAR10_dataset
```

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: dataset/CIFAR10
    Split: Train
    StandardTransform
Transform: ToTensor()
```

```
CIFAR10_dataset_augmented
```

```
 Dataset CIFAR10
     Number of datapoints: 50000
     Root location: dataset/CIFAR10
     Split: Train
     StandardTransform
 Transform: Compose(
                RandomHorizontalFlip(p=0.5)
                RandomCrop(size=(32, 32), padding=[0, 2, 3, 4])
                ToTensor()
            )
```

```
@torch.no_grad()
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

@torch.no_grad()
def evaluate(model, data_loader, testOrValidation):
    model.eval()
    outputs = [model.testOrValidation_step(batch, testOrValidation) for batch in data_load
    return model.at_testOrValidation_epoch_end(outputs,testOrValidation)

class Utils(nn.Module):
    def training_step(self, batch):
        images, true_labels = batch
        images = images.to(device=device)
        true_labels = true_labels.to(device=device)
        pred_labels = self(images)
        loss = F.cross_entropy(pred_labels, true_labels)
        accur = accuracy(pred_labels,true_labels)
        return loss,accur

    def testOrValidation_step(self, batch, testOrValidation):
        images, true_labels = batch
        images = images.to(device=device)
        true_labels = true_labels.to(device=device)
        pred_labels = self(images)
        loss = F.cross_entropy(pred_labels, true_labels)
        accur= accuracy(pred_labels, true_labels)
        return {testOrValidation +'_loss': loss.detach(), testOrValidation+'_accuracy': ac

    def at_testOrValidation_epoch_end(self, outputs, testOrValidation):
        batch_losses = [x[testOrValidation+'_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()
        batch_accuracies = [x[testOrValidation+'_accuracy'] for x in outputs]
        epoch_accuracy = torch.stack(batch_accuracies).mean()
        return {testOrValidation +'_loss': epoch_loss.item(), testOrValidation+'_accuracy'

    def at_epoch_end(self, epoch, result):
        print("Epoch :",epoch + 1)
        print(f'Train Accuracy:{result["train_accuracy"]*100:.2f}% Validation Accuracy:{re
        print(f'Train Loss:{result["train_loss"]:.4f} Validation Loss:{result["validation_
def conv_block_1(in_f, out_f, enable_BN, enable_dropOut):
    return nn.Sequential(
        nn.Conv2d(in_f, out_f, kernel_size=3, stride=1, padding=1),
        nn.ReLU()
    )


def conv_block_2(in_f, out_f, enable_BN, enable_dropOut):
    if(enable_BN):
        return  nn.Sequential(
            conv_block_1(in_f, out_f, enable_BN, enable_dropOut),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(out_f)
        )
    else :
        return  nn.Sequential(
            conv_block_1(in_f, out_f, enable_BN, enable_dropOut),
            nn.MaxPool2d(2, 2)
        )


def conv_block_3(in1_f, out1_f,in2_f, out2_f, enable_BN, enable_dropOut):
        return  nn.Sequential(
            conv_block_1(in1_f, out1_f, enable_BN, enable_dropOut),
            conv_block_2(in2_f, out2_f, enable_BN, enable_dropOut)
        )

def linear_block(in_f, out_f, enable_dropOut, activation_function):
    if(enable_dropOut):
        return nn.Sequential(
             nn.Linear(in_f, out_f),
             nn.Dropout(0.2),
             activation_function
            )
    else:
        return nn.Sequential(
             nn.Linear(in_f, out_f),
             activation_function
            )

class Cifar10CnnModel_2Layer(Utils):
  def __init__(self, enable_BN, enable_dropOut):
        super().__init__()
        self.ConvLayer1 = conv_block_3(3, 32, 32, 64, enable_BN, enable_dropOut)
        self.ConvLayer2 = conv_block_3(64, 128, 128, 128, enable_BN, enable_dropOut) # out
        self.flatten =  nn.Flatten()
        self.Linear1 = linear_block(128 *8*8, 1024,enable_dropOut,nn.ReLU())
        self.Linear2 = linear_block(1024, 512, enable_dropOut,nn.ReLU())
        self.Linear3 = nn.Linear(512, 10)

  def forward(self, xb):
        x = self.ConvLayer1(xb)
        x = self.ConvLayer2(x)
        x = self.flatten(x)
        x = self.Linear1(x)
        x = self.Linear2(x)
        x = self.Linear3(x)
        return x

  def outputConv1(self, xb):
         x = self.ConvLayer1(xb)
         return x

  def outputConv2(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.flatten(x)
         return x



class Cifar10CnnModel_3Layer(Utils):
    def __init__(self, enable_BN, enable_dropOut):
        super().__init__()
        self.ConvLayer1 = conv_block_3(3, 32, 32, 64, enable_BN, enable_dropOut)
        self.ConvLayer2 = conv_block_3(64, 128, 128, 128, enable_BN, enable_dropOut) # out
        self.ConvLayer3 = conv_block_3(128, 256, 256, 256, enable_BN, enable_dropOut)
        self.flatten =  nn.Flatten()
        self.Linear1 = linear_block(256 *4*4, 1024,enable_dropOut,nn.ReLU())
        self.Linear2 = linear_block(1024, 512, enable_dropOut,nn.ReLU())
        self.Linear3 = nn.Linear(512, 10)

    def forward(self, xb):
        x = self.ConvLayer1(xb)
        x = self.ConvLayer2(x)
        x = self.ConvLayer3(x)
        x = self.flatten(x)
        x = self.Linear1(x)
        x = self.Linear2(x)
        x = self.Linear3(x)
        return x

    def outputConv1(self, xb):
         x = self.ConvLayer1(xb)
         return x

    def outputConv2(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.flatten(x)
         return x

    def outputConv3(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.ConvLayer3(x)
         x = self.flatten(x)
         return x
```

```
def split_dataset(dataset, ratio):
    subsetALength = (int)( len(dataset) * ratio)
    subsetA, subsetB = torch.utils.data.random_split(dataset,
                        [subsetALength, len(dataset)-subsetALength])
    subsetALoader = DataLoader(dataset=subsetA, batch_size=batch_size, shuffle=True)
    subsetBLoader =  DataLoader(dataset=subsetB, batch_size=batch_size, shuffle=True)
    return(subsetALoader,subsetBLoader,subsetA,subsetB)

def trainNetwork(model, train_loader, validation_loader,epochs,learning_rate):
    t1 = time.time()
    best_valid = None
    history = []
    optimizer = torch.optim.Adam(model.parameters(), learning_rate,weight_decay=0.0005)
    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        train_accuracy = []
        for batch in tqdm(train_loader):
            loss,accu = model.training_step(batch)
            train_losses.append(loss)
            train_accuracy.append(accu)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation
        result = evaluate(model, validation_loader,"validation")
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['train_accuracy'] = torch.stack(train_accuracy).mean().item()
        model.at_epoch_end(epoch, result)
        if(best_valid == None or best_valid<result['validation_accuracy']):
            best_valid=result['validation_accuracy']
            torch.save(model.state_dict(), 'Lab2_P1'+ model.name +'.pth')
        history.append(result)
    t2 = time.time()
    timeToTrain = t2 - t1
    print(f'TIme to train {model.name} : {timeToTrain}')
    return (history, timeToTrain)


def plot_accuracies(history, model_name):
    Validation_accuracies = [x['validation_accuracy'] for x in history]
    Training_Accuracies = [x['train_accuracy'] for x in history]
    plt.figure()
    plt.plot(Training_Accuracies, '-rx')
    plt.plot(Validation_accuracies, '-bx')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend(['Training', 'Validation'])
    plt.title('Acc vs epochs for model ' + model_name);
    plt.savefig(f'{model_name}_acc.png')
    plt.close()

def plot_losses(history,model_name):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['validation_loss'] for x in history]
    plt.figure()
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs epochs '+ model_name);
    plt.savefig(f'{model_name}_loss.png')
    plt.close()



#Splitting Dataset to Training & Test
tr = 0.7
batch_size = 64
trainLoader, testValidationLoader, trainDataset, testValidationDataset = split_dataset(CIF
testLoader, validationLoader, testDataset, validationDataset = split_dataset(testValidatio

#Splitting Dataset to Training & Test
tr = 0.7
batch_size = 64
trainLoaderAugmented, testValidationLoaderAugmented, trainDatasetAugmented, testValidation
testLoaderAugmented, validationLoaderAugmented, testDatasetAugmentedt, validationDatasetAu


#Question 1

# Hyperparameters
tr = 0.7
learning_rate = 0.001
batch_size = 64
epochs = 10

models = []

## No Regularizer

model = Cifar10CnnModel_2Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_None_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_None_"+"3LayerCNN"
models.append(model)


##Batch Norm
model = Cifar10CnnModel_2Layer(True, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_BatchNorm_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(True, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_BatchNorm_"+"3LayerCNN"
models.append(model)
```

```
## Drop Out
model = Cifar10CnnModel_2Layer(False, True).to(device)
model.name = "Lab2_P1_" +"Regularizer_DropOut_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(False, True).to(device)
model.name = "Lab2_P1_" +"Regularizer_DropOut_"+"3LayerCNN"
models.append(model)
```

```
testAccuracies = []
trainingTimes = []
for model in models:
    print(model.name)
    (listAccuracies, trainingTime) = trainNetwork(model,trainLoader, validationLoader,epoc
    trainingTimes.append(trainingTime)
    plot_accuracies(listAccuracies,model.name)
    plot_losses(listAccuracies,model.name)
    result = evaluate(model, testLoader, "test")
    testAccuracies.append(result)
```

```
Epoch : 7
Train Accuracy:78.40% Validation Accuracy:73.44%
Train Loss:0.6195 Validation Loss:0.7843
100%|██████████| 547/547 [00:09<00:00, 60.26it/s]
Epoch : 8
Train Accuracy:81.08% Validation Accuracy:74.11%
Train Loss:0.5350 Validation Loss:0.7785
100%|██████████| 547/547 [00:09<00:00, 60.02it/s]
Epoch : 9
Train Accuracy:83.83% Validation Accuracy:73.97%
Train Loss:0.4638 Validation Loss:0.7657
100%|██████████| 547/547 [00:09<00:00, 60.12it/s]
Epoch : 10
Train Accuracy:85.55% Validation Accuracy:75.49%
Train Loss:0.4105 Validation Loss:0.7356
TIme to train Lab2_P1_Regularizer_DropOut_2LayerCNN : 104.29744410514832
Lab2_P1_Regularizer_DropOut_3LayerCNN
100%|██████████| 547/547 [00:09<00:00, 55.09it/s]
Epoch : 1
Train Accuracy:9.73% Validation Accuracy:9.79%
Train Loss:2.3031 Validation Loss:2.3026
100%|██████████| 547/547 [00:09<00:00, 55.72it/s]
Epoch : 2
Train Accuracy:9.75% Validation Accuracy:10.49%
Train Loss:2.3028 Validation Loss:2.3027
100%|██████████| 547/547 [00:09<00:00, 55.92it/s]
Epoch : 3
Train Accuracy:9.86% Validation Accuracy:10.43%
Train Loss:2.3028 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 56.62it/s]
Epoch : 4
```

```
Train Accuracy:9.87% Validation Accuracy:9.70%
Train Loss:2.3027 Validation Loss:2.3029
100%|██████████| 547/547 [00:09<00:00, 57.06it/s]
Epoch : 5
Train Accuracy:9.78% Validation Accuracy:9.59%
Train Loss:2.3027 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.79it/s]
Epoch : 6
Train Accuracy:10.05% Validation Accuracy:9.76%
Train Loss:2.3028 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 57.14it/s]
Epoch : 7
Train Accuracy:9.95% Validation Accuracy:9.73%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 56.79it/s]
Epoch : 8
Train Accuracy:9.92% Validation Accuracy:9.42%
Train Loss:2.3028 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.55it/s]
Epoch : 9
Train Accuracy:10.06% Validation Accuracy:9.42%
Train Loss:2.3028 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.52it/s]
Epoch : 10
Train Accuracy:10.01% Validation Accuracy:9.76%
Train Loss:2.3028 Validation Loss:2.3030
TIme to train Lab2_P1_Regularizer_DropOut_3LayerCNN : 110.35315823554993
```

```
## No Regularizer
modelsAugmented = []
model = Cifar10CnnModel_2Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_DataAugmentation_"+"2LayerCNN"
modelsAugmented.append(model)

model = Cifar10CnnModel_3Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_DataAugmentation_"+"3LayerCNN"
modelsAugmented.append(model)

for model in modelsAugmented:
    print(model.name)
    (listAccuracies, trainingTime) = trainNetwork(model,trainLoaderAugmented, validationLo
    trainingTimes.append(trainingTime)
    plot_accuracies(listAccuracies,model.name)
    plot_losses(listAccuracies,model.name)
    result = evaluate(model, testLoaderAugmented, "test")
    testAccuracies.append(result)
```

```
Epoch : 7
Train Accuracy:71.19% Validation Accuracy:74.22%
Train Loss:0.8112 Validation Loss:0.7325
100%|██████████| 547/547 [00:12<00:00, 43.49it/s]
Epoch : 8
Train Accuracy:72.65% Validation Accuracy:74.78%
Train Loss:0.7707 Validation Loss:0.7231
100%|██████████| 547/547 [00:12<00:00, 43.55it/s]
Epoch : 9
Train Accuracy:73.39% Validation Accuracy:75.54%
Train Loss:0.7502 Validation Loss:0.6911
```

```
100%|██████████| 547/547 [00:12<00:00, 43.23it/s]
Epoch : 10
Train Accuracy:74.23% Validation Accuracy:76.23%
Train Loss:0.7285 Validation Loss:0.6630
TIme to train Lab2_P1_Regularizer_DataAugmentation_2LayerCNN : 139.46855878829956
Lab2_P1_Regularizer_DataAugmentation_3LayerCNN
100%|██████████| 547/547 [00:13<00:00, 40.57it/s]
Epoch : 1
Train Accuracy:10.15% Validation Accuracy:9.77%
Train Loss:2.3030 Validation Loss:2.3031
100%|██████████| 547/547 [00:13<00:00, 40.17it/s]
Epoch : 2
Train Accuracy:10.15% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3030
100%|██████████| 547/547 [00:13<00:00, 40.57it/s]
Epoch : 3
Train Accuracy:10.29% Validation Accuracy:9.83%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.54it/s]
Epoch : 4
Train Accuracy:10.14% Validation Accuracy:9.83%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.76it/s]
Epoch : 5
Train Accuracy:10.29% Validation Accuracy:9.89%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.72it/s]
Epoch : 6
Train Accuracy:10.29% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.65it/s]
Epoch : 7
Train Accuracy:10.29% Validation Accuracy:9.89%
Train Loss:2.3027 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.52it/s]
Epoch : 8
Train Accuracy:10.12% Validation Accuracy:9.89%
Train Loss:2.3028 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.91it/s]
Epoch : 9
Train Accuracy:10.18% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3031
100%|██████████| 547/547 [00:13<00:00, 40.03it/s]
Epoch : 10
Train Accuracy:10.19% Validation Accuracy:9.94%
Train Loss:2.3028 Validation Loss:2.3027
TIme to train Lab2_P1_Regularizer_DataAugmentation_3LayerCNN : 148.44200730323792
```

```
testAccuracies
```

```
[{'test_loss': 0.8836011290550232, 'test_accuracy': 0.7393185496330261},
 {'test_loss': 0.7560953497886658, 'test_accuracy': 0.7452330589294434},
 {'test_loss': 1.0305886268615723, 'test_accuracy': 0.741525411605835},
 {'test_loss': 0.7129078507423401, 'test_accuracy': 0.7845162153244019},
 {'test_loss': 0.7489970326423645, 'test_accuracy': 0.754016637802124},
 {'test_loss': 2.3027384281158447, 'test_accuracy': 0.09706038236618042},
 {'test_loss': 0.6891036033630371, 'test_accuracy': 0.7572386860847473},
 {'test_loss': 2.302833080291748, 'test_accuracy': 0.09666313230991364}]
```

```
for model in models:
  print(model.name)
```

```
Lab2_P1_Regularizer_None_2LayerCNN
Lab2_P1_Regularizer_None_3LayerCNN
Lab2_P1_Regularizer_BatchNorm_2LayerCNN
Lab2_P1_Regularizer_BatchNorm_3LayerCNN
Lab2_P1_Regularizer_DropOut_2LayerCNN
Lab2_P1_Regularizer_DropOut_3LayerCNN
```

```
finalModel = models[3]
```

```
finalModel.name
```

```
'Lab2_P1_Regularizer_BatchNorm_3LayerCNN'
```

```
finalModel
```

```
Cifar10CnnModel_3Layer(
  (ConvLayer1): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer2): Sequential(
    (0): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer3): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
```

```
        (1): Sequential(
          (0): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU()
          )
          (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
          (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
        )
      )
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (Linear1): Sequential(
        (0): Linear(in_features=4096, out_features=1024, bias=True)
        (1): ReLU()
      )
      (Linear2): Sequential(
        (0): Linear(in_features=1024, out_features=512, bias=True)
        (1): ReLU()
```

```
modelq2 = Cifar10CnnModel_3Layer(True, False)
print(modelq2)
modelq2.name = 'modelq2'
print(modelq2.name)
```

```
    Cifar10CnnModel_3Layer(
      (ConvLayer1): Sequential(
        (0): Sequential(
          (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): ReLU()
        )
        (1): Sequential(
          (0): Sequential(
            (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU()
          )
          (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
          (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
        )
      )
      (ConvLayer2): Sequential(
        (0): Sequential(
          (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): ReLU()
        )
        (1): Sequential(
          (0): Sequential(
            (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU()
          )
          (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
          (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
        )
      )
      (ConvLayer3): Sequential(
        (0): Sequential(
          (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): ReLU()
        )
        (1): Sequential(
          (0): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU()
          )
          (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stat
        )
      )
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (Linear1): Sequential(
        (0): Linear(in_features=4096, out_features=1024, bias=True)
        (1): ReLU()
      )
      (Linear2): Sequential(
        (0): Linear(in_features=1024, out_features=512, bias=True)
        (1): ReLU()
      )
      (Linear3): Linear(in_features=512, out_features=10, bias=True)
    )
    modelq2
```

```
modelq2.to(device=device)
```

```
Cifar10CnnModel_3Layer(
  (ConvLayer1): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer2): Sequential(
    (0): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer3): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (Linear1): Sequential(
    (0): Linear(in_features=4096, out_features=1024, bias=True)
    (1): ReLU()
  )
  (Linear2): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): ReLU()
  )
```

```
tr = 0.85
batch_size = 64
trainLoader, testLoader, trainDataset, testDataset = split_dataset(CIFAR10_dataset, tr)
print(modelq2.name)
(listAccuraciesq2, trainingTimeq2) = trainNetwork(modelq2,trainLoader, testLoader,epochs,l
plot_accuracies(listAccuraciesq2,modelq2.name)
plot_losses(listAccuraciesq2,modelq2.name)
resultq2 = evaluate(modelq2, testLoader, "test")
```

```
modelq2
100%|██████████| 665/665 [00:11<00:00, 56.09it/s]
Epoch : 1
Train Accuracy:90.28% Validation Accuracy:88.87%
Train Loss:0.2876 Validation Loss:0.3318
100%|██████████| 665/665 [00:11<00:00, 55.98it/s]
Epoch : 2
Train Accuracy:91.76% Validation Accuracy:88.32%
Train Loss:0.2375 Validation Loss:0.3449
100%|██████████| 665/665 [00:11<00:00, 55.72it/s]
Epoch : 3
Train Accuracy:93.00% Validation Accuracy:86.04%
Train Loss:0.2017 Validation Loss:0.4331
100%|██████████| 665/665 [00:11<00:00, 55.89it/s]
Epoch : 4
Train Accuracy:93.20% Validation Accuracy:85.28%
Train Loss:0.1959 Validation Loss:0.4619
100%|██████████| 665/665 [00:11<00:00, 55.64it/s]
Epoch : 5
Train Accuracy:93.59% Validation Accuracy:84.20%
Train Loss:0.1845 Validation Loss:0.5067
```

```
    100%|██████████| 665/665 [00:12<00:00, 55.24it/s]
    Epoch : 6
    Train Accuracy:93.80% Validation Accuracy:84.71%
    Train Loss:0.1787 Validation Loss:0.4919
    100%|██████████| 665/665 [00:12<00:00, 54.89it/s]
    Epoch : 7
    Train Accuracy:94.20% Validation Accuracy:84.29%
    Train Loss:0.1659 Validation Loss:0.5337
    100%|██████████| 665/665 [00:12<00:00, 55.08it/s]
    Epoch : 8
    Train Accuracy:94.91% Validation Accuracy:85.17%
    Train Loss:0.1430 Validation Loss:0.5012
    100%|██████████| 665/665 [00:12<00:00, 55.37it/s]
    Epoch : 9
    Train Accuracy:95.01% Validation Accuracy:84.38%
    Train Loss:0.1443 Validation Loss:0.5296
    100%|██████████| 665/665 [00:12<00:00, 55.07it/s]
    Epoch : 10
    Train Accuracy:94.94% Validation Accuracy:81.55%
    Train Loss:0.1444 Validation Loss:0.6436
    TIme to train modelq2 : 133.0724036693573
```

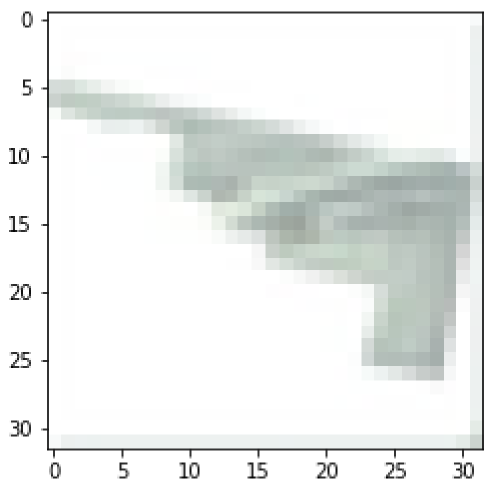
```
resultq2
```

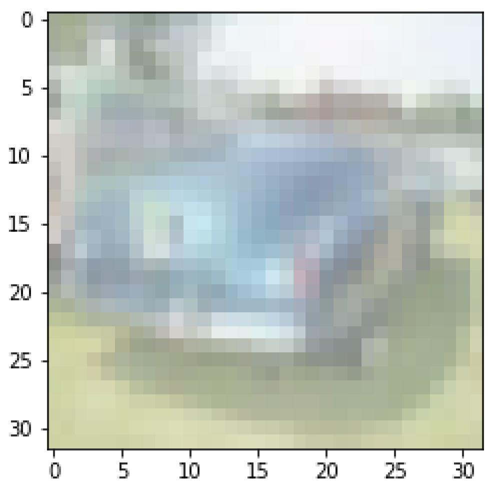
```
    {'test_loss': 0.6509191989898682, 'test_accuracy': 0.8138241767883301}
```

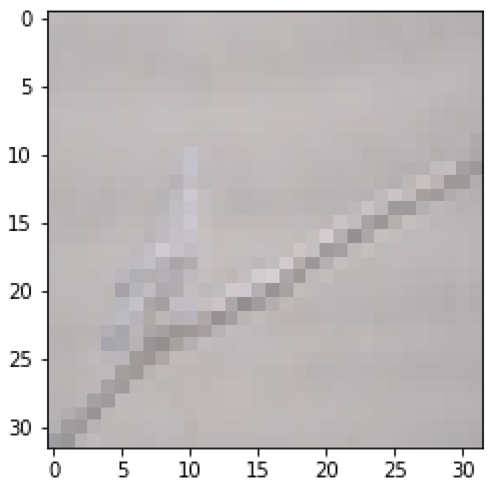
```
def imshow(img):
  img = img / 2 + 0.5   # unnormalize
  npimg = img.numpy()   # convert from tensor
  plt.imshow(np.transpose(npimg, (1, 2, 0)))
  plt.show()

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog','frog', 'horse', 'ship', 'truck']

#   # get first 100 training images
# dataiter = iter(trainLoader)
# n = nn.Softmax(dim=1);
# imgs, lbls = dataiter.next()

# a = torch.Tensor(50000, 3, 32, 32)
# for i in range(1000):  # show just the frogs
#     imshow(utils.make_grid(imgs[i]))
#     print(classes[torch.argmax(n(modelq2.forward(imgs[i].reshape(1,3,32,32).to(device=de
```

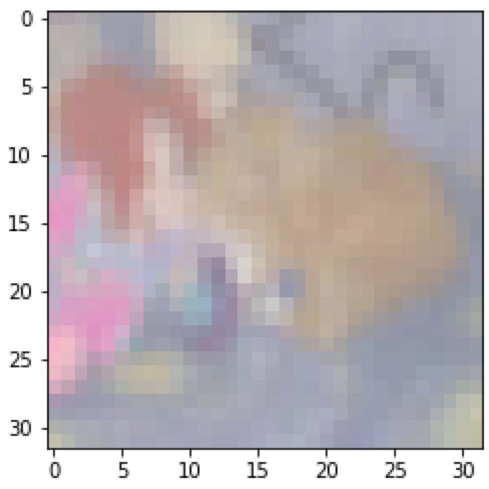
```
l = []
a = torch.Tensor(15000, 3, 32, 32)
for i, (image, label) in enumerate(testDataset):
        a[i, :, :, :] = image
        l.append(label)
```

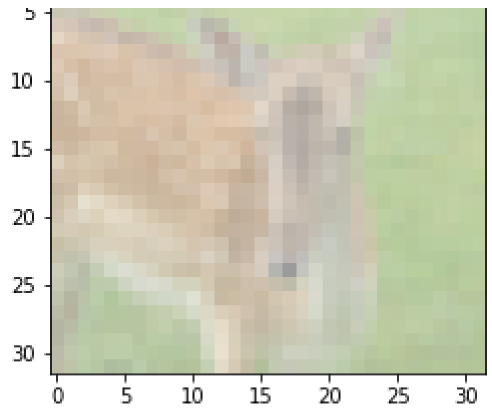
```
finalList = []
forwardEmbeddings = []
Conv2Embeddings = []
for i in enumerate(classes):
  finalList.append([])
  forwardEmbeddings.append([])
  Conv2Embeddings.append([])

for i in range(7500):
    finalList[l[i]].append(a[i, :, : , :])



for i in range(len(finalList)):
  for j in range(1):
    imshow(utils.make_grid(finalList[i][j]))
```

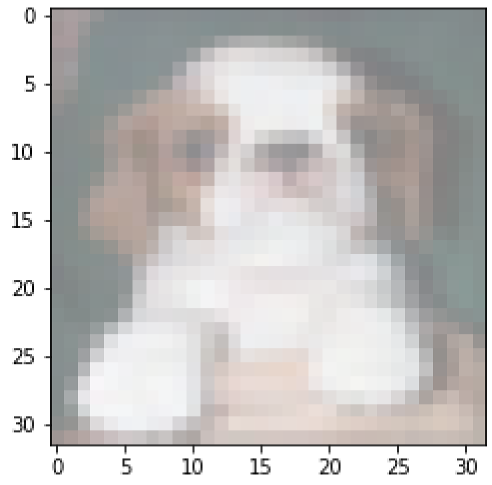


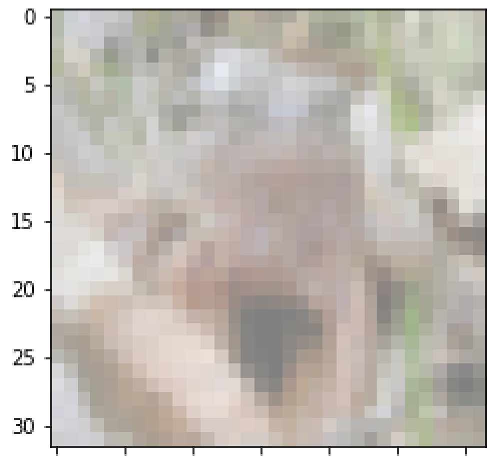












```
forwardEmbeddingsCov = []
conv2dEmbeddingsConv = []
for i in range(len(classes)):
  forwardEmbeddingsCov.append([])
  conv2dEmbeddingsConv.append([])
```

```
modelq2.eval()
with torch.no_grad():
  for i in range(len(finalList)):
      for j in range(len(finalList[i])):
        forwardEmbeddingsCov[i].append(modelq2.forward(finalList[i][j].reshape(1,3,32,32).
        conv2dEmbeddingsConv[i].append(modelq2.outputConv2(finalList[i][j].reshape(1,3,32,
```

```
def plot_forward(m):
  f, axarr = plt.subplots(2,6,figsize=(10,4))
  k = 0
  for i in range(2):
    for j in range(6):
      img = utils.make_grid(finalList[simiL_forward[k][1]][simiL_forward[k][2]])
      img = img / 2 + 0.5   # unnormalize
      npimg = img.numpy()
      #print(npimg.shape)  # convert from tensor
      axarr[i,j].imshow(np.transpose(npimg, (1, 2, 0)))
      axarr[i,j].set_title(f'{classes[simiL_forward[k][1]]}')
      axarr[i,j].axis('off')
      k = k+1

  f.suptitle(f'{classes[m]} similar Images from final Layer Representations')
  plt.savefig(f'{classes[m]}1.png', bbox_inches='tight')


def plot_conv2d(m) :
  f, axarr = plt.subplots(2,6,figsize=(10,4))
  k = 0
  for i in range(2):
    for j in range(6):
      img = utils.make_grid(finalList[simiL_convolution[k][1]][simiL_convolution[k][2]])
      img = img / 2 + 0.5   # unnormalize
      npimg = img.numpy()
      #print(npimg.shape)  # convert from tensor
      axarr[i,j].imshow(np.transpose(npimg, (1, 2, 0)))
      axarr[i,j].set_title(f'{classes[simiL_convolution[k][1]]}')
      axarr[i,j].axis('off')
      k = k+1

  f.suptitle(f'{classes[m]} similar Images from second Convolution Layer Representations')
  plt.savefig(f'{classes[m]}'+'2.png', bbox_inches='tight')




for m in range(10):
  simiL_forward = []
  simiL_convolution = []
  cos = nn.CosineSimilarity(dim=0, eps=1e-6)
  for i in range(10) :
    for j in range(100):
      simiL_forward.append((cos(forwardEmbeddingsCov[m][0].squeeze(),forwardEmbeddingsCov[
      simiL_convolution.append((cos(conv2dEmbeddingsConv[m][0].squeeze(),conv2dEmbeddingsC
  simiL_forward.sort(key = lambda x: x[0], reverse=True)
  simiL_convolution.sort(key = lambda x: x[0], reverse=True)
  plot_forward(m)
  plot_conv2d(m)
```