CSCI 677 ADVANCED COMPUTER VISION
HOMEWORK 5

ROHAN KARNAWAT
2102695224

TABLE OF CONTENTS - (Page No. in [.])

Section 1: Description

A LeNet5 style deep convolution neural network with custom parameters has been implemented. Parameters such as kernel sizes, number of hidden layers, etc have been modified from the standard LeNet5 specifications. Model shown below:

```
FlixNet(
  (conv_block_1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (dense_model): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

1.1: Implementation Points

Multiple experiments were run to get best results. The following points gave the best testing accuracy and avoided training overfitting.

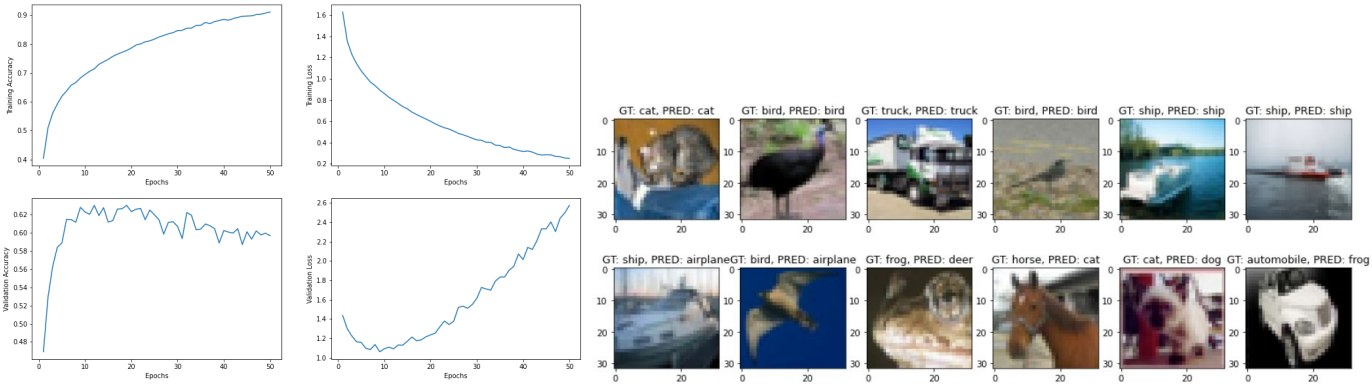
- Environment Settings: **GPU: RTX2070, CUDA: 11.0, Pytorch: 1.7.0, Python: 3.6.9**
- Dataset: **CIFAR-10**
- Batch Size: **64**
- Learning Rate: **0.001**
- Epochs: **30**
- Train-Val Split: **90%-10%**
- Optimizer: **ADAM**
- Augmentations: *Random Crop, Horizontal Flip, Vertical Flip, Rotate up to 15^o, Gaussian Conv (k=5)*
- Testing: *Multi-Crop done* - 3 different sizes of crops are taken, along with original image. Label assigned is the majority (mode). If all labels are different, label from the original image is assigned.
- Types of Layers used: *Conv2D, ReLU, MaxPool2D, AvgPool2D, BatchNorm2D, Linear (FC)*

1.2: Code

The complete code is presented as a python notebook along with the report. Source listing is described below in Section 4.
The model is frozen at the one for which best validation accuracy os achieved

Section 2: Results

2.1 LeNet5



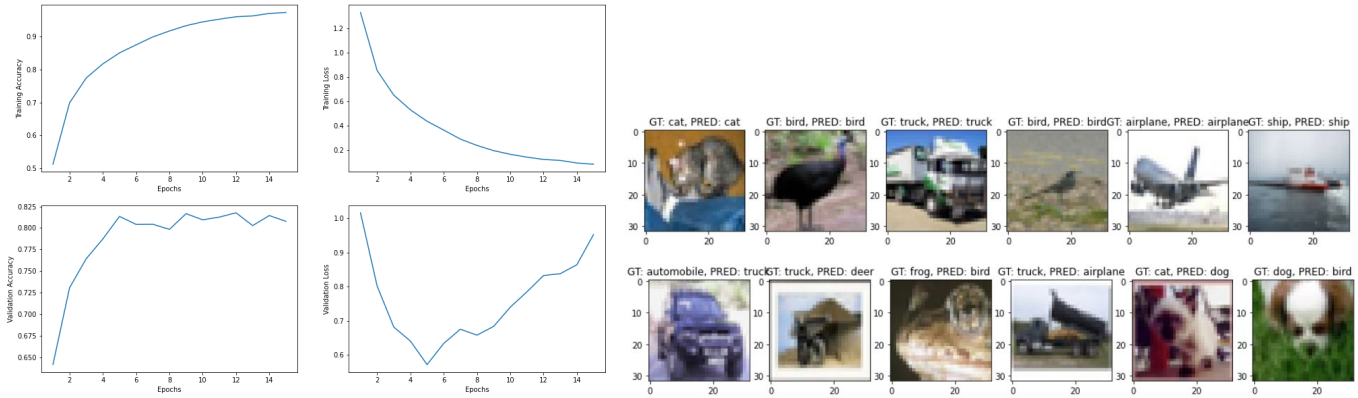
Here, the training accuracy goes very high to upto 97% and validation accuracy and validation loss become worse as the training continues. This is a sign of overfitting.
Testing Accuracy was : 59.89252%. Some Example cases have been shown, the first row denotes Correct predictions, second row denotes wrong examples.

Confusion Matrix:

	airpl	autom	bird	cat	deer	dog	frog	horse	ship	truck
airplane	657	19	51	30	23	14	26	19	103	58
automobile	56	674	11	16	9	13	18	8	69	126
bird	82	4	438	86	128	79	78	61	29	15
cat	32	12	72	390	106	208	78	67	14	21
deer	32	5	64	64	603	55	69	90	15	3
dog	17	8	58	195	76	495	39	85	13	14
frog	16	3	42	87	69	45	700	16	14	8
horse	18	4	20	64	100	97	13	648	10	26
ship	94	28	22	29	12	21	21	4	737	32
truck	45	101	11	41	13	29	24	27	57	652

Per Class Acc: 65.7 67.4 43.8 39 60 49.5 70 64.8 73.7 65.2
(In %)

2.2 My Network (FlixNet) without Augmentation, simple testing



Here, the training accuracy goes very high (95+) after a few epochs with validation accuracy fluctuating around 81% and validation loss increasing. There is some sign of overfitting here.

Testing Accuracy was 81.44905%. Some Example cases have been shown, the first row denotes Correct predictions, second row denotes wrong examples.

Confusion Matrix:

	airpl	autom	bird	cat	deer	dog	frog	horse	ship	truck
airplane	900	5	29	13	5	4	1	4	34	5
automobile	15	916	4	4	2	1	1	1	11	45
bird	57	3	762	41	40	45	31	11	6	4
cat	30	10	66	637	48	119	35	31	12	12
deer	26	1	55	65	769	16	24	38	4	2
dog	17	6	49	120	32	719	14	28	4	11
frog	10	8	36	50	18	5	855	4	9	5
horse	16	5	20	30	37	29	4	852	3	4
ship	55	20	11	5	5	1	6	2	882	13
truck	47	58	2	10	1	1	2	9	17	853
Per Class Acc: (In %)	90	91.6	76.2	63.7	76.9	71.9	85.5	85.2	88.2	85.3

The confusion matrix shows that most of the shapes/objects are learned well. There is some similarity between cats and dogs which is shown in the confusion matrix. They probably look similar, with only the pointed ears as a distinguishing factor. There are some examples where cats may look like dogs and vice versa.

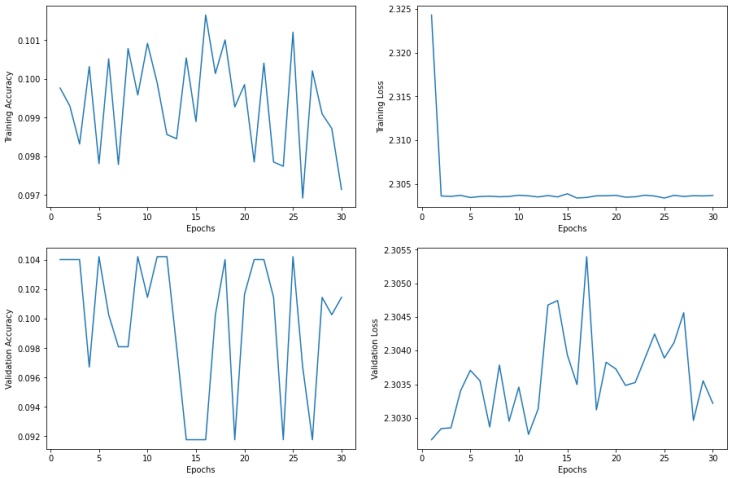
2.2.1 Varying Batch Size

Batch Size	Testing Acc	Comments
16	77.32%	Overfitting observed during training
32	80.68%	-
64	81.45%	Best
128	-	GPU Out of Memory

2.2.2 Varying Learning Rate

Learning Rate	Testing Acc	Comments
0.01	9.95%	There was no semblance of learning observed. The gradients were clearly overshooting the solutions. The LR is too high
0.001	81.45%	Best
0.0001	80.66%	Similar, but not the highest
0.00001	68.98%	The LR is too low, and the validation and training were observed to be slow as well. Even after 30 epochs, the accuracies went only up to 70%.

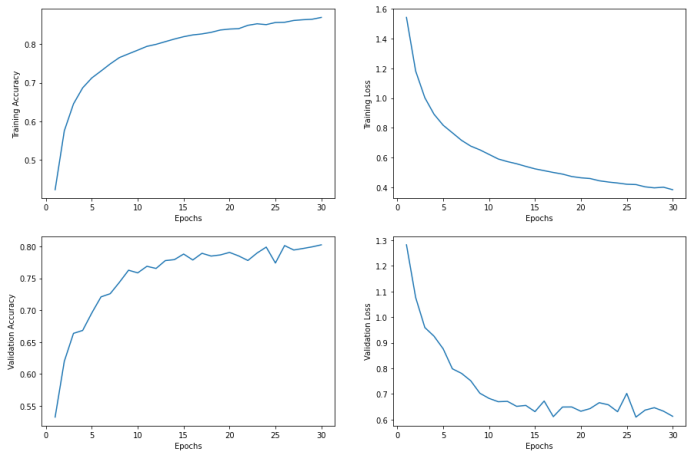
For Learning Rate - 0.01:

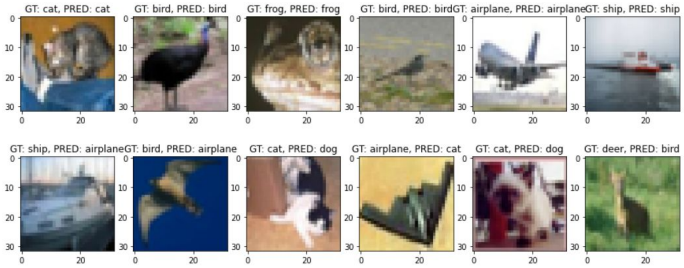


	airpl	autom	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0	0	0	0	0	0	0	0	0	1000
automobile	0	0	0	0	0	0	0	0	0	1000
bird	0	0	0	0	0	0	0	0	0	1000
cat	0	0	0	0	0	0	0	0	0	1000
deer	0	0	0	0	0	0	0	0	0	1000
dog	0	0	0	0	0	0	0	0	0	1000
frog	0	0	0	0	0	0	0	0	0	1000
horse	0	0	0	0	0	0	0	0	0	1000
ship	0	0	0	0	0	0	0	0	0	1000
truck	0	0	0	0	0	0	0	0	0	1000

Everything was classified as a truck. There was no learning, the erratic nature of training and validation can be seen from the graphs. We can conclude that this learning rate is too high.

2.3 My Network (FlixNet) with Data Augmentation



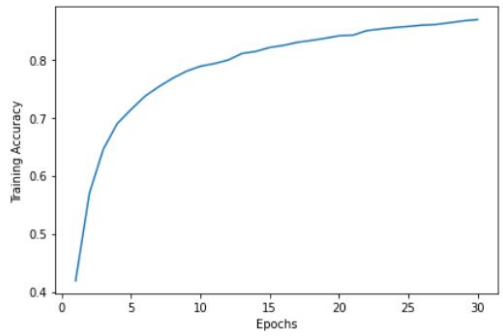


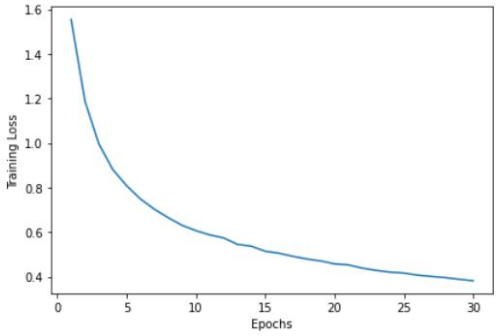
Here, it is observed that the validation loss does not increase, and the gap between training and validation accuracy is lower than previous cases. This means that the overfitting problem has been solved. A small increase in testing accuracy is also observed.
Testing Accuracy was 83.80772%. Some Example cases have been shown, the first row denotes Correct predictions, second row denotes wrong examples.

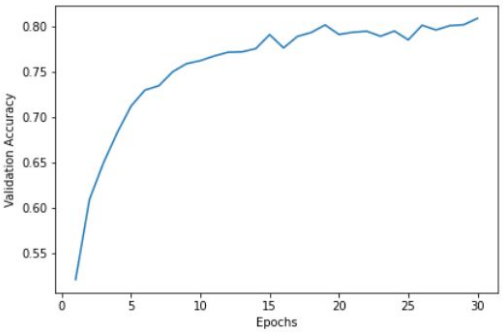
	airpl	autom	bird	cat	deer	dog	frog	horse	ship	truck
airplane	882	6	22	9	16	6	1	12	31	15
automobile	11	916	3	2	0	1	2	4	15	46
bird	48	3	739	57	55	37	28	27	5	1
cat	16	5	39	679	40	145	36	27	9	4
deer	4	1	36	35	840	21	18	41	3	1
dog	8	4	17	123	25	769	10	38	3	3
frog	4	1	38	38	17	10	883	7	1	1
horse	14	0	6	41	24	27	2	883	0	3
ship	55	14	10	8	2	2	3	4	881	21
truck	23	41	3	6	1	4	3	8	4	907

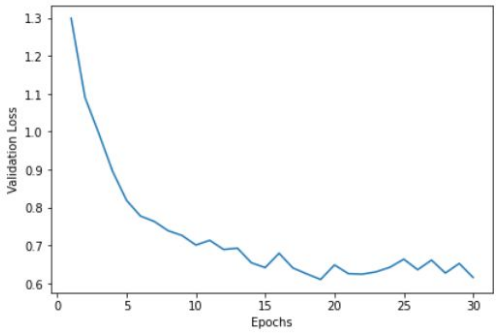
Per Class Acc: 88.2 91.6 73.9 67.9 84 76.9 88.3 88.3 88.1 90.7
(In %)

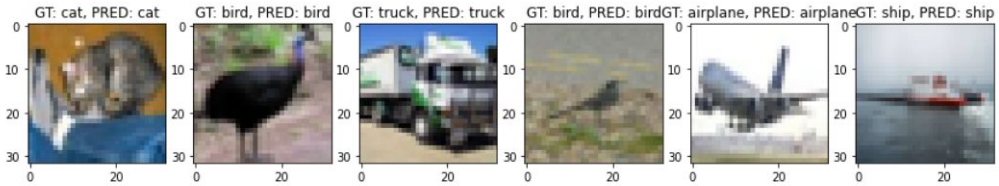
2.4 My Network (FlixNet) with Data Augmentation and MultiCrop testing

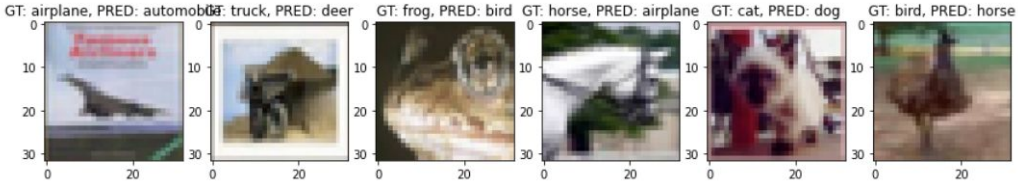












	airpl	autom	bird	cat	deer	dog	frog	horse	ship	truck
airplane	890	10	24	6	14	0	5	13	24	14
automobile	8	940	4	3	1	0	3	0	5	36
bird	47	2	791	34	56	28	24	11	6	1
cat	11	3	64	681	54	111	42	15	6	13
deer	9	1	48	34	850	14	17	25	2	0
dog	8	4	23	125	34	760	14	27	0	5
frog	7	0	30	23	26	15	888	5	5	1
horse	9	1	20	33	51	24	2	860	0	0
ship	79	20	9	3	5	1	3	2	860	18
truck	24	51	9	6	4	4	4	2	8	888

Per Class Acc:	89	94	79.1	68.1	85	76	88.8	86	86	88.8

(In %)

Best Results were observed when at test time I took multiple crops of the image and picked the most frequently occurring label across all crops for an image.

Testing Accuracy: 84.22667% ⇒ Best taken over multiple runs, it fluctuates around 84.1%

⇒ Testing Accuracy = 84.1 ± 0.15%

Section 3: Conclusions

- We always observe a high confusion between cat and dog categories. This is probably because of similarity in structure and shape. Higher quality images, (in resolution and how there are taken) which are not angling away may help.
- Best learning rate was chosen to be 0.001. 0.01 was high (and erratic) and 0.00001 was low (and slow)
- The accuracy can be improved by using even deeper networks, such as ResNet or InceptionNet.
- The accuracy can also be improved by applying more regularization techniques like Dropout layers
- It was observed that with data augmentation, and multicrop testing, the accuracy increased by a few percent (1% increase means 100 more images were classified correctly at inference). The increase was from 81.45% to 84.23%

Section 4: Source Listing

IMPORTS

```
import os
import sys
import numpy as np
from copy import deepcopy

from PIL import Image
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix as confusion

import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
from torchvision.datasets import CIFAR10
from torch.utils.data.dataloader import DataLoader
```

PRE-PROCESSING AND LOADING DATASET

```
def get_mean_std():
      train_transform =
torchvision.transforms.Compose([torchvision.transforms.ToTensor()])
      train_set = CIFAR10(root='dataset/', train=True, download=True,
transform=train_transform)
      mean = tuple(train_set.data.mean(axis=(0,1,2))/255)
      std = tuple(train_set.data.std(axis=(0,1,2))/255)
      return mean,std
# Training and Validation Split (10%)
```

```
mean, std = get_mean_std()
composed_tranforms_train = torchvision.transforms.Compose(
      [
        torchvision.transforms.RandomCrop(32),
        torchvision.transforms.RandomHorizontalFlip(0.2),
        torchvision.transforms.RandomVerticalFlip(0.2),
        torchvision.transforms.RandomRotation(15),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean, std)
      ]
)
composed_tranforms = torchvision.transforms.Compose(
      [
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean, std)
      ]
)

train_val = CIFAR10(root='dataset/', download=True,
transform=composed_tranforms_train)
source_labels = train_val.classes
torch.manual_seed(24)
train, val = torch.utils.data.random_split(train_val, [45000, 5000])
# Test Split
test  = CIFAR10(root='dataset/', train=False, transform=composed_tranforms)


print(train_val)
print(test)
print([{i:j} for i,j in enumerate(source_labels)])


kwargs = {
      'batch_size':64,
      'shuffle':True,
      'pin_memory':True,
      'num_workers':1
}
train_loader = DataLoader(train,**kwargs)
kwargs['shuffle'] = False
val_loader   = DataLoader(val,  **kwargs)
test_loader  = DataLoader(test, **kwargs)
```

CHOOSING DEVICE

```
is_gpu = torch.cuda.is_available()
device = torch.device('cpu')
if is_gpu:
      device = torch.device('cuda')


print("Device to be used: {}".format(device))
```

UTILITY FUNCTIONS

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

def train_step(op, labels):
      return F.cross_entropy(op, labels)

def val_step(op, labels):
      return {
      'val_loss': F.cross_entropy(op, labels).detach(),
      'val_acc': accuracy(op, labels)
      }

def de_normalize(tensor):
      mean, std = get_mean_std()
      for t, m, s in zip(tensor,mean,std):
            t.mul_(s).add_(m)
      return tensor

def display(good_examples):
      L = len(good_examples)
      fig = plt.figure(figsize=(15,5))
      for i in range(L):
            plt.subplot(1,L,i+1)
            img = good_examples[i]['image']
            img = np.transpose(img, (1,2,0))
            plt.imshow(img)
            plt.title("GT: {}, PRED: {}".format(
                  num2word[int(good_examples[i]['gt'])],
                  num2word[good_examples[i]['pred']]
            ))
```

```
def get_samples(x,y):
      ans = [-1, -1]
      for i in range(x.shape[0]):
            if x[i]!=y[i]:
                  ans[0] = i
                  break
      for i in range(x.shape[0]):
            if x[i]==y[i]:
                  ans[1] = i
                  break
      return ans


def get_multi_crops(data0):
      crops = [data0]
      data1 = deepcopy(data0)
      data2 = deepcopy(data0)

      data1 = data1[:,:,3:29,3:29]
      data1 = F.interpolate(data1, size=(32,32), mode='bilinear')

      data2 = data2[:,:,5:27,5:27]
      data2 = F.interpolate(data2, size=(32,32), mode='bilinear')
      crops.append(data1)
      crops.append(data2)
      return crops
```

MODEL SPECIFICATION:

```
class FlixNet(nn.Module):
      def __init__(self):
      super(FlixNet, self).__init__()
      self.conv_block_1 = nn.Sequential(
            nn.Conv2d(3, 32, 3, 1, 1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 64, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
      )
            self.conv_block_2 = nn.Sequential(
            nn.Conv2d(64, 64, 5, 1, 2),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 128, 5, 1, 2),
            nn.ReLU(),
```

```
            nn.MaxPool2d(2,2)
      )
      self.conv_block_3 = nn.Sequential(
                        nn.Conv2d(128,128,5),
            nn.ReLU(),
            nn.AvgPool2d(2,2)
      )

            self.dense_model = nn.Sequential(
                        nn.Linear(512, 256),
            nn.ReL      U(),
            nn.Linear(256, 128),
                  nn.ReLU(),
                        nn.Linear(128, 10)
      )

      def forward(self, databatch):
      y = self.conv_block_1(databatch)
      y = self.conv_block_2(y)
      y = self.conv_block_3(y)
      y = torch.flatten(y, 1)
      y = self.dense_model(y)
            return y


def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
      val_history = []
      train_history = []
      best_val = -float('inf')
      optimizer = opt_func(model.parameters(), lr)
      for epoch in range(epochs):
      # Training Phase
      tr = []
      tra = []
            for chunk in train_loader:
                  data, labels = chunk
            data = data.to(device)
            labels = labels.to(device)
            op = model(data)
            loss = train_step(op, labels)
            acc = accuracy(op, labels)
            tr.append(float(loss))
            tra.append(float(acc))
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
```

```
      tr_loss = np.array(tr).mean()
      tr_acc  = np.array(tra).mean()
      train_history.append({'tr_loss': tr_loss, 'tr_acc': tr_acc})

      # Validation phase
            outputs = []
      for chunk in val_loader:
            data, labels = chunk
            data = data.to(device)
            labels = labels.to(device)
            op = model(data)
            outputs.append(val_step(op, labels))

      loss = torch.stack([it['val_loss'] for it in outputs]).mean()
      acc = torch.stack([it['val_acc'] for it in outputs]).mean()
      res = {'val_loss': loss.item(), 'val_acc': acc.item()}
      if float(res['val_acc']) > best_val and epoch>7:
            torch.save(model.state_dict(), 'best.pth')

      print("Epoch [{}], tr_acc: {:.4f}, val_acc: {:.4f}".\
            format(epoch, tr_acc, res['val_acc']))
            val_history.append(res)

      return train_history, val_history
```

```
flixnet = FlixNet()
net = flixnet.to(device)
print(net)
epochs = 30
tr_info, val_info = fit(epochs, 0.001, net, train_loader, val_loader)
```

PLOTTING TRAINING CURVES

```
tr_loss, tr_acc, val_loss, val_acc = [], [], [], []
x_axis = range(1, epochs+1)
for info in tr_info:
      tr_loss.append(info['tr_loss'])
      tr_acc.append(info['tr_acc'])

for info in val_info:
      val_loss.append(info['val_loss'])
      val_acc.append(info['val_acc'])

fig = plt.figure(figsize=(15, 10))
```

```
plt.subplot(2, 2, 1)
plt.plot(x_axis, tr_acc)
plt.xlabel("Epochs")
plt.ylabel("Training Accuracy")

plt.subplot(2, 2, 2)
plt.plot(x_axis, tr_loss)
plt.xlabel("Epochs")
plt.ylabel("Training Loss")

plt.subplot(2, 2, 3)
plt.plot(x_axis, val_acc)
plt.xlabel("Epochs")
plt.ylabel("Validation Accuracy")

plt.subplot(2, 2, 4)
plt.plot(x_axis, val_loss)
plt.xlabel("Epochs")
plt.ylabel("Validation Loss")

plt.show()
```

TESTING

```
test_model = FlixNet()
test_model.load_state_dict(torch.load('best.pth'))
test_model.eval()
test_model.to(device)
y_pred = []
y_gt = []
acc = []
bad_examples = []
good_examples = []
for i, chunk in enumerate(test_loader):
      data, labels = chunk
      batches = get_multi_crops(data)
      labels_g = labels.to(device)
      tenses = []
      for data_chunk in batches:
      data_g = data_chunk.to(device)
            y = test_model(data_g)
      tenses.append(y)
      tensors = torch.stack((tenses[0], tenses[0], tenses[1], tenses[2]))
```

```
      y = torch.mode(tensors, dim=0).values

_, pred = torch.max(y, dim=1)
      y_pred.append(pred.cpu().tolist())
      y_gt.append(labels.tolist())
      acc.append(accuracy(y, labels_g))
      if i % 30 == 0:
            if is_gpu:
                  pred = pred.cpu()
      bad, good = get_samples(np.array(pred), np.array(labels))
            if bad is not -1:
                  bad_examples.append({'image':de_normalize(data[bad]),\
                                       'gt':labels[bad], 'pred':int(pred[bad])})
                  if good is not -1:
            good_examples.append({'image':de_normalize(data[good]),\
 'gt':labels[good], 'pred':int(pred[good])})

test_accuracy = np.array(acc).mean()
print("Testing Accuracy is: {}%".format(round(100*test_accuracy, 5)))

source_labels = train_val.classes
num2word = {}
for i,j in enumerate(source_labels):
      num2word[i] = j
display(good_examples)
display(bad_examples)
```

CONFUSION MATRIX AND PER CLASS ACCURACY

```
Y_PRED = np.array([])
for lis in y_pred:
      Y_PRED = np.hstack((Y_PRED,np.array(lis)))
Y_GT = np.array([])
for lis in y_gt:
      Y_GT = np.hstack((Y_GT,np.array(lis)))
conf = confusion(Y_GT, Y_PRED)

print("--------------------------------------------------------------------------------")
print('|', end='')
print('{:10}'.format(" "), end=' ')
for i in range(10):
      print(' {:5}'.format(source_labels[i][:5]), end='')

print('|')
for i, row in enumerate(conf):
      print('|'+'{:10}'.format(source_labels[i]), end=' ')
```

```
      for val in row:
      print('{:5}'.format(val), end=' ')
      print('|')
print("-------------------------------------------------------------------------------")
print("Per Class Accuracy", conf.diagonal()/10.0)
print("  Testing Accuracy is: {}%".format(round(100*test_accuracy, 5)))
print("-------------------------------------------------------------------------------")
```

MODEL NETWORK VISUALIZATION

```
x = torch.zeros(1, 3, 32, 32, dtype=torch.float, requires_grad=False)
out = test_model(x.cuda())
graph = make_dot(out)
```