Training a ConvNet PyTorch ¶

In this notebook, you'll learn how to use the powerful PyTorch framework to specify a conv net architecture and train it on the human action recognition dataset.

In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import DataLoader,sampler,Dataset
import torchvision.datasets as dset
import torchvision.transforms as T
import timeit
from PIL import Image
import os
import numpy as np
import scipy.io
```

What's this PyTorch business?

- When using a framework like PyTorch or TensorFlow you can harness the power of the GPU for your own custom neural network architectures without having to write CUDA code directly.
- this notebook will walk you through much of what you need to do to train models using pytorch. if you want to learn more or need further clarification on topics that aren't fully explained here, here are 2 good Pytorch tutorials. 1): http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html) 2)http://pytorch.org/tutorials/beginner/pytorch_with_examples.html (http://pytorch.org/tutorials/beginner/pytorch_with_examples.html)
- It's not necessary to have a GPU for this homework, using a GPU can make your code run faster.

Load Datasets

In this part, we will load the action recognition dataset for the neural network. In order to load data from our custom dataset, we need to write a custom Dataloader. If you put q3_2_data.mat, /valClips,/trainClips,/testClips under the folder of ./data/ , you do not need to change anything in this part.

First, load the labels of the dataset, you should write your path of the q3_2_data.mat file.

In [2]:

```
label_mat=scipy.io.loadmat('./data/q3_2_data.mat')
label_train=label_mat['trLb']
print(len(label_train))
label_val=label_mat['valLb']
print(len(label_val))
```

```
7770
2230
```

Dataset class

torch.utils.data.Dataset is an abstract class representing a dataset. The custom dataset should inherit Dataset and override the following methods:

```
__len__ so that len(dataset) returns the size of the dataset.
__getitem__ to support the indexing such that dataset[i] can be used to get ith
 sample
```

Let’s create a dataset class for our action recognition dataset. We will read images in **getitem**. This is memory efficient because all the images are not stored in the memory at once but read as required.

Sample of our dataset will be a dict {'image':image,'img_path':img_path,'Label':Label}. Our datset will take an optional argument transform so that any required processing can be applied on the sample.

In [8]:

```python
class ActionDataset(Dataset):
    """Action dataset."""

    def __init__(self,  root_dir,labels=[], transform=None):
        """
        Args:
            root_dir (string): Directory with all the images.
            labels(list): labels if images.
            transform (callable, optional): Optional transform to be applied on a sampl
e.
        """
        self.root_dir = root_dir
        self.transform = transform
        self.length=len(os.listdir(self.root_dir))
        self.labels=labels
    def __len__(self):
        return self.length*3

    def __getitem__(self, idx):

        folder=int(idx/3)+1
        imidx=idx%3+1
        folder=format(folder,'05d')
        imgname=str(imidx)+'.jpg'
        img_path = os.path.join(self.root_dir,
                                folder,imgname)
        image = Image.open(img_path)
        if len(self.labels)!=0:
            Label=self.labels[int(idx/3)][0]-1
        if self.transform:
            image = self.transform(image)
        if len(self.labels)!=0:
            sample={'image':image,'img_path':img_path,'Label':Label}
        else:
            sample={'image':image,'img_path':img_path}
        return sample
```

Iterating over the dataset by a for loop.

In [8]:

```
image_dataset=ActionDataset(root_dir='./data/trainClips/',\
                            labels=label_train,transform=T.ToTensor())

#iterating though the dataset
for i in range(10):
    sample=image_dataset[i]
    print(sample['image'].shape)
    print(sample['Label'])
    print(sample['img_path'])
```

```
torch.Size([3, 64, 64])
0.0
./data/trainClips/00001\1.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00001\2.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00001\3.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00002\1.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00002\2.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00002\3.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00003\1.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00003\2.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00003\3.jpg
torch.Size([3, 64, 64])
0.0
./data/trainClips/00004\1.jpg
```

We can iterate over the created dataset with a 'for' loop as before. However, we are losing a lot of features by using a simple for loop to iterate over the data. In particular, we are missing out on:

- Batching the data
- Shuffling the data
- Load the data in parallel using multiprocessing workers.

torch.utils.data.DataLoader is an iterator which provides all these features.

In [11]:

```
image_dataloader = DataLoader(image_dataset, batch_size=4,
                        shuffle=True, num_workers=0)


for i,sample in enumerate(image_dataloader):
    sample['image']=sample['image'].cuda()
    print(i,sample['image'].shape,sample['img_path'],sample['Label'])
    if i>20:
        break
```

```
0 torch.Size([4, 3, 64, 64]) ['./data/trainClips/04715\\2.jpg', './data/tr
ainClips/06997\\2.jpg', './data/trainClips/01324\\1.jpg', './data/trainCli
ps/04755\\2.jpg'] tensor([5., 8., 1., 5.])
1 torch.Size([4, 3, 64, 64]) ['./data/trainClips/04687\\1.jpg', './data/tr
ainClips/04728\\2.jpg', './data/trainClips/02008\\3.jpg', './data/trainCli
ps/05357\\2.jpg'] tensor([5., 5., 2., 6.])
2 torch.Size([4, 3, 64, 64]) ['./data/trainClips/04602\\2.jpg', './data/tr
ainClips/02145\\1.jpg', './data/trainClips/01136\\1.jpg', './data/trainCli
ps/06170\\2.jpg'] tensor([5., 2., 1., 7.])
3 torch.Size([4, 3, 64, 64]) ['./data/trainClips/02396\\3.jpg', './data/tr
ainClips/00154\\2.jpg', './data/trainClips/05192\\3.jpg', './data/trainCli
ps/05440\\1.jpg'] tensor([2., 0., 6., 6.])
4 torch.Size([4, 3, 64, 64]) ['./data/trainClips/01493\\3.jpg', './data/tr
ainClips/04980\\3.jpg', './data/trainClips/01414\\1.jpg', './data/trainCli
ps/07203\\3.jpg'] tensor([1., 6., 1., 9.])
5 torch.Size([4, 3, 64, 64]) ['./data/trainClips/05372\\1.jpg', './data/tr
ainClips/01556\\3.jpg', './data/trainClips/00555\\2.jpg', './data/trainCli
ps/05992\\2.jpg'] tensor([6., 1., 0., 7.])
6 torch.Size([4, 3, 64, 64]) ['./data/trainClips/07514\\1.jpg', './data/tr
ainClips/01989\\2.jpg', './data/trainClips/01161\\3.jpg', './data/trainCli
ps/00471\\1.jpg'] tensor([9., 2., 1., 0.])
7 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03672\\2.jpg', './data/tr
ainClips/02909\\2.jpg', './data/trainClips/05152\\2.jpg', './data/trainCli
ps/02687\\1.jpg'] tensor([4., 3., 6., 3.])
8 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03993\\2.jpg', './data/tr
ainClips/02993\\2.jpg', './data/trainClips/04092\\1.jpg', './data/trainCli
ps/00582\\1.jpg'] tensor([4., 3., 4., 0.])
9 torch.Size([4, 3, 64, 64]) ['./data/trainClips/02361\\3.jpg', './data/tr
ainClips/03858\\1.jpg', './data/trainClips/00979\\3.jpg', './data/trainCli
ps/03214\\1.jpg'] tensor([2., 4., 1., 3.])
10 torch.Size([4, 3, 64, 64]) ['./data/trainClips/04841\\2.jpg', './data/t
rainClips/00284\\3.jpg', './data/trainClips/07454\\2.jpg', './data/trainCl
ips/07243\\3.jpg'] tensor([5., 0., 9., 9.])
11 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03691\\1.jpg', './data/t
rainClips/04670\\3.jpg', './data/trainClips/01173\\1.jpg', './data/trainCl
ips/06656\\3.jpg'] tensor([4., 5., 1., 8.])
12 torch.Size([4, 3, 64, 64]) ['./data/trainClips/07564\\3.jpg', './data/t
rainClips/06166\\1.jpg', './data/trainClips/02619\\3.jpg', './data/trainCl
ips/06199\\1.jpg'] tensor([9., 7., 2., 7.])
13 torch.Size([4, 3, 64, 64]) ['./data/trainClips/06423\\2.jpg', './data/t
rainClips/00081\\2.jpg', './data/trainClips/01357\\1.jpg', './data/trainCl
ips/03861\\1.jpg'] tensor([7., 0., 1., 4.])
14 torch.Size([4, 3, 64, 64]) ['./data/trainClips/01225\\2.jpg', './data/t
rainClips/06596\\2.jpg', './data/trainClips/05968\\3.jpg', './data/trainCl
ips/05778\\2.jpg'] tensor([1., 8., 7., 7.])
15 torch.Size([4, 3, 64, 64]) ['./data/trainClips/05743\\1.jpg', './data/t
rainClips/00704\\3.jpg', './data/trainClips/03370\\1.jpg', './data/trainCl
ips/01950\\1.jpg'] tensor([7., 0., 3., 2.])
16 torch.Size([4, 3, 64, 64]) ['./data/trainClips/01314\\2.jpg', './data/t
rainClips/02781\\2.jpg', './data/trainClips/02821\\2.jpg', './data/trainCl
ips/01724\\2.jpg'] tensor([1., 3., 3., 1.])
17 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03425\\3.jpg', './data/t
rainClips/03036\\2.jpg', './data/trainClips/01583\\2.jpg', './data/trainCl
ips/02192\\1.jpg'] tensor([3., 3., 1., 2.])
18 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03238\\1.jpg', './data/t
rainClips/06332\\2.jpg', './data/trainClips/03852\\3.jpg', './data/trainCl
ips/05103\\2.jpg'] tensor([3., 7., 4., 6.])
19 torch.Size([4, 3, 64, 64]) ['./data/trainClips/03213\\3.jpg', './data/t
rainClips/07633\\3.jpg', './data/trainClips/01976\\2.jpg', './data/trainCl
ips/00143\\2.jpg'] tensor([3., 9., 2., 0.])
20 torch.Size([4, 3, 64, 64]) ['./data/trainClips/07431\\1.jpg', './data/t
rainClips/01025\\3.jpg', './data/trainClips/01580\\3.jpg', './data/trainCl
ips/05785\\2.jpg'] tensor([9., 1., 1., 7.])
21 torch.Size([4, 3, 64, 64]) ['./data/trainClips/06630\\3.jpg', './data/t
rainClips/02956\\2.jpg', './data/trainClips/04798\\3.jpg', './data/trainCl
ips/02164\\2.jpg'] tensor([8., 3., 5., 2.])
```

Dataloaders for the training, validationg and testing set.

In [105]:

```
image_dataset_train=ActionDataset(root_dir='./data/trainClips/',labels=label_train,tran
sform=T.ToTensor())

image_dataloader_train = DataLoader(image_dataset_train, batch_size=32,
                        shuffle=True, num_workers=0)
image_dataset_val=ActionDataset(root_dir='./data/valClips/',labels=label_val,transform=
T.ToTensor())

image_dataloader_val = DataLoader(image_dataset_val, batch_size=32,
                        shuffle=False, num_workers=0)
image_dataset_test=ActionDataset(root_dir='./data/testClips/',labels=[],transform=T.ToT
ensor())

image_dataloader_test = DataLoader(image_dataset_test, batch_size=32,
                        shuffle=False, num_workers=0)
```

In [106]:

```
dtype = torch.FloatTensor # the CPU datatype
# Constant to control how frequently we print train loss
print_every = 100
# This is a little utility that we'll use to reset the model
# if we want to re-initialize all our parameters
def reset(m):
    if hasattr(m, 'reset_parameters'):
        m.reset_parameters()
```

Example Model

Some assorted tidbits

Let's start by looking at a simple model. First, note that PyTorch operates on Tensors, which are n-dimensional arrays functionally analogous to numpy's ndarrays, with the additional feature that they can be used for computations on GPUs.

We'll provide you with a Flatten function, which we explain here. Remember that our image data (and more relevantly, our intermediate feature maps) are initially N x C x H x W, where:

- N is the number of datapoints
- C is the number of image channels.
- H is the height of the intermediate feature map in pixels
- W is the height of the intermediate feature map in pixels

This is the right way to represent the data when we are doing something like a 2D convolution, that needs spatial understanding of where the intermediate features are relative to each other. When we input data into fully connected affine layers, however, we want each datapoint to be represented by a single vector -- it's no longer useful to segregate the different channels, rows, and columns of the data. So, we use a "Flatten" operation to collapse the C x H x W values per representation into a single long vector. The Flatten function below first reads in the N, C, H, and W values from a given batch of data, and then returns a "view" of that data. "View" is analogous to numpy's "reshape" method: it reshapes x's dimensions to be N x ??, where ?? is allowed to be anything (in this case, it will be C x H x W, but we don't need to specify that explicitly).

In [107]:

```
class Flatten(nn.Module):
    def forward(self, x):
        N, C, H, W = x.size() # read in N, C, H, W
        return x.view(N, -1)  # "flatten" the C * H * W values into a single vector per
image
```

The example model itself

The first step to training your own model is defining its architecture.

Here's an example of a convolutional neural network defined in PyTorch -- try to understand what each line is doing, remembering that each layer is composed upon the previous layer. We haven't trained anything yet - that'll come next - for now, we want you to understand how everything gets set up. nn.Sequential is a container which applies each layer one after the other.

In this example, you see 2D convolutional layers (Conv2d), ReLU activations, and fully-connected layers (Linear). You also see the Cross-Entropy loss function, and the Adam optimizer being used.

Make sure you understand why the parameters of the Linear layer are 10092 and 10.

In [108]:

```
# Here's where we define the architecture of the model...
simple_model = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=7, stride=2),
                nn.ReLU(inplace=True),
                Flatten(), # see above for explanation
                nn.Linear(10092, 10), # affine layer
              )

# Set the type of all data in this model to be FloatTensor
simple_model.type(dtype)

loss_fn = nn.CrossEntropyLoss().type(dtype)
optimizer = optim.Adam(simple_model.parameters(), lr=1e-2) # lr sets the learning rate
 of the optimizer
```

PyTorch supports many other layer types, loss functions, and optimizers - you will experiment with these next. Here's the official API documentation for these (if any of the parameters used above were unclear, this resource will also be helpful).

- Layers: http://pytorch.org/docs/nn.html (http://pytorch.org/docs/nn.html)
- Activations: http://pytorch.org/docs/nn.html#non-linear-activations (http://pytorch.org/docs/nn.html#non-linear-activations)
- Loss functions: http://pytorch.org/docs/nn.html#loss-functions (http://pytorch.org/docs/nn.html#loss-functions)
- Optimizers: http://pytorch.org/docs/optim.html#algorithms (http://pytorch.org/docs/optim.html#algorithms)

Training a specific model

In this section, we're going to specify a model for you to construct. The goal here isn't to get good performance (that'll be next), but instead to get comfortable with understanding the PyTorch documentation and configuring your own model.

Using the code provided above as guidance, and using the following PyTorch documentation, specify a model with the following architecture:

- 7x7 Convolutional Layer with 8 filters and stride of 1
- ReLU Activation Layer
- 2x2 Max Pooling layer with a stride of 2
- 7x7 Convolutional Layer with 16 filters and stride of 1
- ReLU Activation Layer
- 2x2 Max Pooling layer with a stride of 2
- Flatten the feature map
- ReLU Activation Layer
- Affine layer to map input units to 10 outputs, you need to figure out the input size here.

In [109]:

```
fixed_model_base = nn.Sequential(
    #########1st TODO  (5 points)###################
    nn.Conv2d(3, 8, kernel_size=7, stride=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(8, 16, kernel_size=7, stride=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    Flatten(),
    nn.ReLU(),
    nn.Linear(16*11*11, 10)
    ####################################
            )
fixed_model = fixed_model_base.type(dtype)
```

To make sure you're doing the right thing, use the following tool to check the dimensionality of your output (it should be 32 x 10, since our batches have size 32 and the output of the final affine layer should be 10, corresponding to our 10 classes):

In [110]:

```
## Now we're going to feed a random batch into the model you defined and make sure the
 output is the right size
x = torch.randn(32, 3, 64, 64).type(dtype)
x_var = Variable(x.type(dtype)) # Construct a PyTorch Variable out of your input data
ans = fixed_model(x_var)        # Feed it through the model!

# Check to make sure what comes out of your model
# is the right dimensionality... this should be True
# if you've done everything correctly
print(np.array(ans.size()))
np.array_equal(np.array(ans.size()), np.array([32, 10]))
```

```
[32 10]
```

Out[110]:

```
True
```

Train the model.

Now that you've seen how to define a model and do a single forward pass of some data through it, let's walk through how you'd actually train one whole epoch over your training data (using the fixed_model_base we provided above).

Make sure you understand how each PyTorch function used below corresponds to what you implemented in your custom neural network implementation.

Note that because we are not resetting the weights anywhere below, if you run the cell multiple times, you are effectively training multiple epochs (so your performance should improve).

First, set up an RMSprop optimizer (using a 1e-4 learning rate) and a cross-entropy loss function:

In [111]:

```
################ 2nd TODO  (5 points)##################
optimizer = torch.optim.RMSprop(fixed_model.parameters(), lr=1e-4)
loss_fn = nn.CrossEntropyLoss()
```

In [36]:

```
# This sets the model in "training" mode.
# This is relevant for some layers that may have different behavior
# in training mode vs testing mode, such as Dropout and BatchNorm.
fixed_model.train()

# Load one batch at a time.
for t, sample in enumerate(image_dataloader_train):
    x_var = Variable(sample['image'])
    #print(type(x_var.data))
    #print(x_var.shape)
    y_var = Variable(sample['Label']).long()

    # This is the forward pass: predict the scores for each class, for each x in the ba
tch.
    scores = fixed_model(x_var)

    # Use the correct y values and the predicted y values to compute the loss.
    loss = loss_fn(scores, y_var)

    if (t + 1) % print_every == 0:
        print('t = %d, loss = %.4f' % (t + 1, loss.item()))

    # Zero out all of the gradients for the variables which the optimizer will update.
    optimizer.zero_grad()

    # This is the backwards pass: compute the gradient of the loss with respect to each
    # parameter of the model.
    loss.backward()

    # Actually update the parameters of the model using the gradients computed by the b
ackwards pass.
    optimizer.step()
```

```
t = 100, loss = 1.7901
t = 200, loss = 1.4626
t = 300, loss = 1.4024
t = 400, loss = 1.4161

---------------------------------------------------------------------------
-
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-36-ce429b32aec7> in <module>
      5 
      6 # Load one batch at a time.
----> 7 for t, sample in enumerate(image_dataloader_train):
      8     x_var = Variable(sample['image'])
      9     #print(type(x_var.data))

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\utils\data\dataloader.py in __next__(self)
    344     def __next__(self):
    345         index = self._next_index()  # may raise StopIteration
--> 346         data = self._dataset_fetcher.fetch(index)  # may raise Sto
pIteration
    347         if self._pin_memory:
    348             data = _utils.pin_memory.pin_memory(data)

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\utils\data\_utils\fetch.py in fetch(self, possibly_batched_index)
     42     def fetch(self, possibly_batched_index):
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_
index]
     45         else:
     46             data = self.dataset[possibly_batched_index]

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\utils\data\_utils\fetch.py in <listcomp>(.0)
     42     def fetch(self, possibly_batched_index):
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_
index]
     45         else:
     46             data = self.dataset[possibly_batched_index]

<ipython-input-7-c1173a00001a> in __getitem__(self, idx)
     25         img_path = os.path.join(self.root_dir,
     26                                 folder,imgname)
---> 27         image = Image.open(img_path)
     28         if len(self.labels)!=0:
     29             Label=self.labels[int(idx/3)][0]-1

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\PI
L\Image.py in open(fp, mode)
   2777         exclusive_fp = True
   2778 
-> 2779     prefix = fp.read(16)
   2780 
   2781     preinit()

KeyboardInterrupt: 
```

Now you've seen how the training process works in PyTorch. To save you writing boilerplate code, we're providing the following helper functions to help you train for multiple epochs and check the accuracy of your model:

In [115]:

```
def train(model, loss_fn, optimizer, dataloader, num_epochs = 1):
    for epoch in range(num_epochs):
        print('Starting epoch %d / %d' % (epoch + 1, num_epochs))
        model.train()
        for t, sample in enumerate(dataloader):
            x_var = Variable(sample['image'])
            y_var = Variable(sample['Label'].long())

            scores = model(x_var)

            loss = loss_fn(scores, y_var)
            if (t + 1) % print_every == 0:
                print('t = %d, loss = %.4f' % (t + 1, loss.item()))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

def check_accuracy(model, loader):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    for t, sample in enumerate(loader):
        x_var = Variable(sample['image'])
        y_var = sample['Label']
        #y_var=y_var.cpu()
        scores = model(x_var)
        _, preds = scores.data.max(1)#scores.data.cpu().max(1)
        #print(preds)
        #print(y_var)
        num_correct += (preds.numpy() == y_var.numpy()).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
```

Check the accuracy of the model.

Let's see the train and check_accuracy code in action -- feel free to use these methods when evaluating the models you develop below.

You should get a training loss of around 1.0-1.2, and a validation accuracy of around 50-60%. As mentioned above, if you re-run the cells, you'll be training more epochs, so your performance will improve past these numbers.

But don't worry about getting these numbers better -- this was just practice before you tackle designing your own model.

In [116]:

```
torch.random.manual_seed(12345)
fixed_model.cpu()
fixed_model.apply(reset)
fixed_model.train()
train(fixed_model, loss_fn, optimizer,image_dataloader_train, num_epochs=1)
check_accuracy(fixed_model, image_dataloader_train)# check accuracy on the training set
```

```
Starting epoch 1 / 1
t = 100, loss = 2.2570
t = 200, loss = 2.0664
t = 300, loss = 1.8087
t = 400, loss = 1.5577
t = 500, loss = 1.4869
t = 600, loss = 1.3653
t = 700, loss = 1.3690
Got 12940 / 23310 correct (55.51)
```

Don't forget the validation set!

And note that you can use the check_accuracy function to evaluate on the validation set, by passing **image_dataloader_val** as the second argument to check_accuracy. The accuracy on validation set is arround 40-50%.

In [117]:

```
check_accuracy(fixed_model, image_dataloader_val)#check accuracy on the validation set
```

```
Got 3142 / 6690 correct (46.97)
```

Train a better model for action recognition!

Now it's your job to experiment with architectures, hyperparameters, loss functions, and optimizers to train a model that achieves better accuracy on the action recognition **validation** set. You can use the check_accuracy and train functions from above.

Things you should try:

- **Filter size**: Above we used 7x7; this makes pretty pictures but smaller filters may be more efficient
- **Number of filters**: Do more or fewer do better?
- **Pooling vs Strided Convolution**: Do you use max pooling or just stride convolutions?
- **Batch normalization**: Try adding spatial batch normalization after convolution layers and vanilla batch normalization after affine layers. Do your networks train faster?
- **Network architecture**: The network above has two layers of trainable parameters. Can you do better with a deep network? Good architectures to try include:
  - [conv-relu-pool]xN -> [affine]xM -> [softmax or SVM]
  - [conv-relu-conv-relu-pool]xN -> [affine]xM -> [softmax or SVM]
  - [batchnorm-relu-conv]xN -> [affine]xM -> [softmax or SVM]
- **Global Average Pooling**: Instead of flattening and then having multiple affine layers, perform convolutions until your image gets small (7x7 or so) and then perform an average pooling operation to get to a 1x1 image picture (1, 1 , Filter#), which is then reshaped into a (Filter#) vector. This is used in Google's Inception Network (https://arxiv.org/abs/1512.00567) (See Table 1 for their architecture).
- **Regularization**: Add l2 weight regularization, or perhaps use Dropout.

Tips for training

For each network architecture that you try, you should tune the learning rate and regularization strength. When doing this there are a couple important things to keep in mind:

- If the parameters are working well, you should see improvement within a few hundred iterations
- Remember the coarse-to-fine approach for hyperparameter tuning: start by testing a large range of hyperparameters for just a few training iterations to find the combinations of parameters that are working at all.
- Once you have found some sets of parameters that seem to work, search more finely around these parameters. You may need to train for more epochs.
- You should use the validation set for hyperparameter search, and save your test set for evaluating your architecture on the best parameters as selected by the validation set.

Going above and beyond

If you are feeling adventurous there are many other features you can implement to try and improve your performance. You are **not required** to implement any of these; however they would be good things to try.

- Alternative update steps: For the assignment we implemented SGD+momentum, RMSprop, and Adam; you could try alternatives like AdaGrad or AdaDelta.
- Alternative activation functions such as leaky ReLU, parametric ReLU, ELU, or MaxOut.
- Model ensembles
- Data augmentation
- New Architectures
  - ResNets (https://arxiv.org/abs/1512.03385) where the input from the previous layer is added to the output.
  - DenseNets (https://arxiv.org/abs/1608.06993) where inputs into previous layers are concatenated together.
  - This blog has an in-depth overview (https://chatbotslife.com/resnets-highwaynets-and-densenets-oh-my-9bb15918ee32)

If you do decide to implement something extra, clearly describe it in the "Extra Credit Description" cell below.

What we expect

At the very least, you should be able to train a ConvNet that gets at least 55% accuracy on the validation set. This is just a lower bound - if you are careful it should be possible to get accuracies much higher than that! Extra credit points will be awarded for particularly high-scoring models or unique approaches.

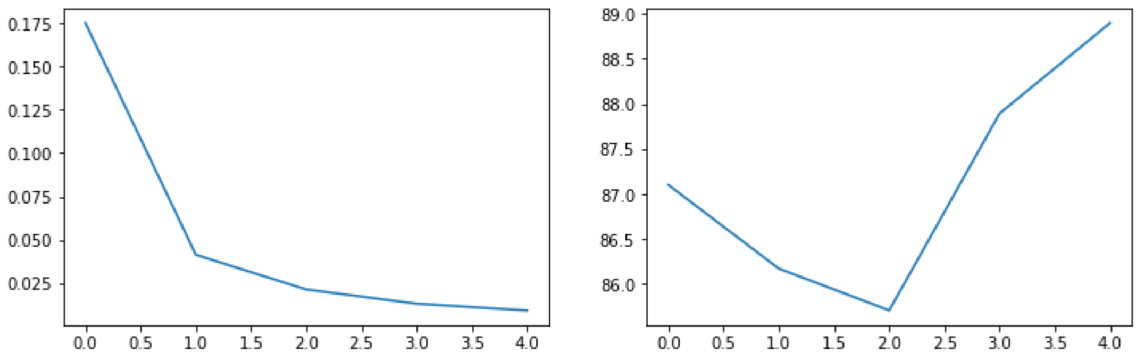
You should use the space below to experiment and train your network.

In [57]:

```
dtype = torch.cuda.FloatTensor
def check_accuracy_2d(model, loader, pretrained=None):
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    dtype = torch.cuda.FloatTensor
    for t, sample in enumerate(loader):
        if t+1 % 1100==0:
            print('wait')
        x_var = Variable(sample['image'].type(dtype))
        y_var = sample['Label'].type(dtype)
        model = model.type(dtype)
        y_var=y_var.cpu()
        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)
        _, preds = scores.data.cpu().max(1)
        num_correct += (preds.numpy() == y_var.numpy()).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
    return acc * 100

train_tf = T.Compose([
    T.Resize(224),
#     T.RandomOrder([
#         T.RandomVerticalFlip(p=0.8),
#         T.RandomRotation(45),
#         T.RandomPerspective()
#     ]),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

val_tf = T.Compose([
    T.Resize(224),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

image_dataset_train=ActionDataset(root_dir='./data/trainClips/',labels=label_train,tran
sform=train_tf)

image_dataloader_train = DataLoader(image_dataset_train, batch_size=32,
                        shuffle=True, num_workers=0)
image_dataset_val=ActionDataset(root_dir='./data/valClips/',labels=label_val,transform=
train_tf)

image_dataloader_val = DataLoader(image_dataset_val, batch_size=32,
                        shuffle=False, num_workers=0)
```

In [58]:

```
###########3rd TODO (6 points, must submit the results to Kaggle) ##############
# Train your model here, and make sure the output of this cell is the accuracy of your
 best model on the
# train, val, and test sets. Here's some code to get you started. The output of this ce
ll should be the training
# and validation accuracy on your best model (measured by validation accuracy).
%matplotlib inline
import torchvision.models as models
from IPython.display import clear_output
import matplotlib.pyplot as plt

pretrained = models.resnet18(pretrained=True)
pretrained.eval()
model = nn.Sequential(
    nn.Linear(1000, 1000),
    nn.ReLU(inplace=True),
    nn.Linear(1000, 10)
)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=1e-4)

# train(fixed_model_base, loss_fn, optimizer,image_dataloader_train, num_epochs=1)

# training
num_epochs = 5
train_losses = []
val_losses = []
val_accs = []
best_acc = 60
PATH = 'saved_models/2d_{}.pt'
for epoch in range(num_epochs):
    losses = []
    dtype = torch.cuda.FloatTensor
    for t, sample in enumerate(image_dataloader_train):
        model.train()
        x_var = sample['image'].type(dtype)
        y_var = sample['Label'].type(dtype).long()

        model = model.type(dtype)
        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)

        loss = loss_fn(scores.cpu(), y_var.cpu())
        if (t + 1) % 100 == 0:
            print('t = %d, loss = %.4f' % (t + 1, loss.data.item()))
        losses.append(loss.data.item())
        optimizer.zero_grad()
        loss.backward()
#          torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
        optimizer.step()
    train_losses.append(np.mean(np.array(losses)))

    del x_var, y_var, scores
    acc = check_accuracy_2d(model, image_dataloader_val, pretrained=pretrained)

    val_losses.append(np.mean(np.array(losses)))
    val_accs.append(acc)
    if epoch > 0:
        clear_output(True)
        plt.figure(figsize=(12, 8))
        plt.subplot(221)
        plt.plot(train_losses)
        plt.subplot(222)
        plt.plot(val_accs)
        plt.show()
        print("Epoch", epoch+1, "=> ", "Val acc:", acc)
    if acc > best_acc:
        best_acc = acc
        torch.save(model, PATH.format(str(acc)[:5]))

fixed_model = torch.load(PATH.format(str(best_acc)[:5]))
best_acc = check_accuracy_2d(fixed_model, image_dataloader_val, pretrained = pretrained
)
```



```
Epoch 5 =>  Val acc: 88.89387144992527
Got 5947 / 6690 correct (88.89)
```

Describe what you did

In the cell below you should write an explanation of what you did, any additional features that you implemented, and any visualizations or graphs that you make in the process of training and evaluating your network.

Tell us here!

#####4th TODO (4 points)

I have run experiments for this part in different notebook named '**image.ipynb**'. Please have a look at it for rest of the description.

Shorter Version

1)Baseline model(the given one): Val acc: 73.44, kaggle score:65.88

2)Pretrained Resnet18 (without augmentation) Val acc: 82, Kaggle score: 84.335

3)Pretrained Resnet18 (with augmentation) + 2 trained layers from above. Val acc: 79.19

4)Pretrained Resnet18 (with augmentation) + 2 trainable layers from scratch. Val acc: 79.52

5)Pretrained Resnet101 (without augmentation) Val acc: 86.30, kaggle score: **85.898**

Testing the model and submit on Kaggle

Testing the model on the testing set and save the results as a .csv file. Please submitted the results.csv file generated by predict_on_test() to Kaggle(https://www.kaggle.com/c/cse512springhw3 (https://www.kaggle.com/c/cse512springhw3)) to see how well your network performs on the test set.

#5th TODO (submit the result to Kaggle, the highest 3 entries get extra 10 points)

In [60]:

```
def predict_on_test(model, loader):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    results=open('results.csv','w')
    count=0
    results.write('Id'+','+'Class'+'\n')
    for t, sample in enumerate(loader):
        x_var = Variable(sample['image'])
        scores = model(x_var)
        _, preds = scores.data.max(1)
        for i in range(len(preds)):
            results.write(str(count)+','+str(preds[i])+'\n')
            count+=1
    results.close()
    return count

count=predict_on_test(fixed_model, image_dataloader_test)
print(count)
```

```
---------------------------------------------------------------------------
-
RuntimeError                              Traceback (most recent call las
t)
<ipython-input-60-463f9c917780> in <module>
     22     return count
     23 
---> 24 count=predict_on_test(fixed_model, image_dataloader_test)
     25 print(count)

<ipython-input-60-463f9c917780> in predict_on_test(model, loader)
     14     for t, sample in enumerate(loader):
     15         x_var = Variable(sample['image'])
---> 16         scores = model(x_var)
     17         _, preds = scores.data.max(1)
     18         for i in range(len(preds)):

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\nn\modules\module.py in __call__(self, *input, **kwargs)
    539             result = self._slow_forward(*input, **kwargs)
    540         else:
--> 541             result = self.forward(*input, **kwargs)
    542         for hook in self._forward_hooks.values():
    543             hook_result = hook(self, input, result)

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\nn\modules\container.py in forward(self, input)
     90     def forward(self, input):
     91         for module in self._modules.values():
---> 92             input = module(input)
     93         return input
     94 

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\nn\modules\module.py in __call__(self, *input, **kwargs)
    539             result = self._slow_forward(*input, **kwargs)
    540         else:
--> 541             result = self.forward(*input, **kwargs)
    542         for hook in self._forward_hooks.values():
    543             hook_result = hook(self, input, result)

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\nn\modules\linear.py in forward(self, input)
     85 
     86     def forward(self, input):
---> 87         return F.linear(input, self.weight, self.bias)
     88 
     89     def extra_repr(self):

c:\users\karan\appdata\local\programs\python\python36\lib\site-packages\to
rch\nn\functional.py in linear(input, weight, bias)
   1370         ret = torch.addmm(bias, input, weight.t())
   1371     else:
-> 1372         output = input.matmul(weight.t())
   1373         if bias is not None:
   1374             output += bias

RuntimeError: Expected object of device type cuda but got device type cpu
 for argument #1 'self' in call to _th_mm
```

My Implemented Function

In [99]:

```
def predict_on_test2d(model, loader, pretrained = None):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    dtype = torch.cuda.FloatTensor
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    results=open('results.csv','w')
    count=0
    results.write('Id'+','+'Class'+'\n')
    for t, sample in enumerate(loader):
        x_var = Variable(sample['image']).type(dtype)
        model = model.type(dtype)
        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)
        _, preds = scores.data.max(1)
        for i in range(len(preds)):
            results.write(str(count)+','+str(preds[i].item())+'\n')
            count+=1
    results.close()
    return count

count=predict_on_test2d(fixed_model, image_dataloader_test, pretrained=pretrained)
print(count)
```

```
9810
```

GPU! (This part is optional, 0 points)

If you have access to GPU, you can make the code run on GPU, it would be much faster.

Now, we're going to switch the dtype of the model and our data to the GPU-friendly tensors, and see what happens... everything is the same, except we are casting our model and input tensors as this new dtype instead of the old one.

If this returns false, or otherwise fails in a not-graceful way (i.e., with some error message), you may not have an NVIDIA GPU available on your machine.

In [0]:

```
# Verify that CUDA is properly configured and you have a GPU available

torch.cuda.is_available()
```

In [0]:

```
import copy
gpu_dtype = torch.cuda.FloatTensor

fixed_model_gpu = copy.deepcopy(fixed_model_base)#.type(gpu_dtype)
fixed_model_gpu.cuda()
x_gpu = torch.randn(4, 3, 64, 64).cuda()#.type(gpu_dtype)
x_var_gpu = Variable(x_gpu)#type(gpu_dtype)) # Construct a PyTorch Variable out of your
input data
ans = fixed_model_gpu(x_var_gpu)        # Feed it through the model!

# Check to make sure what comes out of your model
# is the right dimensionality... this should be True
# if you've done everything correctly
np.array_equal(np.array(ans.size()), np.array([4, 10]))
```

Run the following cell to evaluate the performance of the forward pass running on the CPU:

In [0]:

```
%%timeit
ans = fixed_model(x_var)
```

... and now the GPU:

In [0]:

```
%%timeit
torch.cuda.synchronize() # Make sure there are no pending GPU computations
ans = fixed_model_gpu(x_var_gpu)        # Feed it through the model!
torch.cuda.synchronize() # Make sure there are no pending GPU computations
```

You should observe that even a simple forward pass like this is significantly faster on the GPU. So for the rest of the assignment (and when you go train your models in assignment 3 and your project!), you should use the GPU datatype for your model and your tensors: as a reminder that is *torch.cuda.FloatTensor* (in our notebook here as *gpu_dtype*)

Let's make the loss function and training variables to GPU friendly format by '.cuda()'

In [0]:

```
loss_fn = nn.CrossEntropyLoss().cuda()
optimizer = optim.RMSprop(fixed_model_gpu.parameters(), lr=1e-4)
```

In [0]:

```
def train(model, loss_fn, optimizer, dataloader, num_epochs = 1):
    for epoch in range(num_epochs):
        print('Starting epoch %d / %d' % (epoch + 1, num_epochs))
        model.train()
        for t, sample in enumerate(dataloader):
            x_var = Variable(sample['image'].cuda())
            y_var = Variable(sample['Label'].cuda().long())

            scores = model(x_var)

            loss = loss_fn(scores, y_var)
            if (t + 1) % print_every == 0:
                print('t = %d, loss = %.4f' % (t + 1, loss.data[0]))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

def check_accuracy(model, loader):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    for t, sample in enumerate(loader):
        x_var = Variable(sample['image'].cuda())
        y_var = sample['Label'].cuda()
        y_var=y_var.cpu()
        scores = model(x_var)
        _, preds = scores.data.cpu().max(1)
        #print(preds)
        #print(y_var)
        num_correct += (preds.numpy() == y_var.numpy()).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
```

Run on GPU!

In [0]:

```
torch.cuda.random.manual_seed(12345)

fixed_model_gpu.apply(reset)
fixed_model_gpu.train()
train(fixed_model_gpu, loss_fn, optimizer,image_dataloader_train, num_epochs=1)
check_accuracy(fixed_model_gpu, image_dataloader_train)# check accuracy on the training
set
```

3D Convolution on video clips (15 points+10 extra points)

3D convolution is for videos, it has one more dimension than 2d convolution. You can find the document for 3D convolution here http://pytorch.org/docs/master/nn.html#torch.nn.Conv3dIn (http://pytorch.org/docs/master/nn.html#torch.nn.Conv3dIn). In our dataset, each clip is a video of 3 frames. Lets classify the each clip rather than each image using 3D convolution. We offer the data loader, the train_3d and check_accuracy

In [3]:

```
class ActionClipDataset(Dataset):
    """Action Landmarks dataset."""

    def __init__(self,  root_dir,labels=[], transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """

        self.root_dir = root_dir
        self.transform = transform
        self.length=len(os.listdir(self.root_dir))
        self.labels=labels

    def __len__(self):
        return self.length

    def __getitem__(self, idx):

        folder=idx+1
        folder=format(folder,'05d')
        clip=[]
        if len(self.labels)!=0:
            Label=self.labels[idx][0]-1
        for i in range(3):
            imidx=i+1
            imgname=str(imidx)+'.jpg'
            img_path = os.path.join(self.root_dir,
                                    folder,imgname)
            image = Image.open(img_path)
            image=np.array(image)
            clip.append(image)
        if self.transform:
            clip=np.asarray(clip)
            clip=np.transpose(clip, (0,3,1,2))
            clip = torch.from_numpy(np.asarray(clip))
        if len(self.labels)!=0:
            sample={'clip':clip,'Label':Label,'folder':folder}
        else:
            sample={'clip':clip,'folder':folder}
        return sample

clip_dataset=ActionClipDataset(root_dir='./data/trainClips/',\
                               labels=label_train,transform=T.ToTensor())#/home/tqvinh/
Study/CSE512/cse512-s18/hw2data/trainClips/
for i in range(10):
    sample=clip_dataset[i]
    print(sample['clip'].shape)
    print(sample['Label'])
    print(sample['folder'])
```

```
torch.Size([3, 3, 64, 64])
0.0
00001
torch.Size([3, 3, 64, 64])
0.0
00002
torch.Size([3, 3, 64, 64])
0.0
00003
torch.Size([3, 3, 64, 64])
0.0
00004
torch.Size([3, 3, 64, 64])
0.0
00005
torch.Size([3, 3, 64, 64])
0.0
00006
torch.Size([3, 3, 64, 64])
0.0
00007
torch.Size([3, 3, 64, 64])
0.0
00008
torch.Size([3, 3, 64, 64])
0.0
00009
torch.Size([3, 3, 64, 64])
0.0
00010
```

In [4]:

```
clip_dataloader = DataLoader(clip_dataset, batch_size=4,
                        shuffle=True, num_workers=0)


for i,sample in enumerate(clip_dataloader):
    print(i,sample['clip'].shape,sample['folder'],sample['Label'])
    if i>20:
        break
```

```
0 torch.Size([4, 3, 3, 64, 64]) ['00366', '05452', '07511', '02337'] tenso
r([0., 6., 9., 2.])
1 torch.Size([4, 3, 3, 64, 64]) ['03160', '00351', '02188', '02669'] tenso
r([3., 0., 2., 2.])
2 torch.Size([4, 3, 3, 64, 64]) ['03272', '06322', '06785', '00676'] tenso
r([3., 7., 8., 0.])
3 torch.Size([4, 3, 3, 64, 64]) ['02461', '01237', '03099', '06111'] tenso
r([2., 1., 3., 7.])
4 torch.Size([4, 3, 3, 64, 64]) ['06781', '04129', '01642', '03866'] tenso
r([8., 4., 1., 4.])
5 torch.Size([4, 3, 3, 64, 64]) ['00215', '00253', '06110', '02886'] tenso
r([0., 0., 7., 3.])
6 torch.Size([4, 3, 3, 64, 64]) ['07169', '02253', '02633', '07367'] tenso
r([9., 2., 2., 9.])
7 torch.Size([4, 3, 3, 64, 64]) ['06284', '07463', '03700', '01931'] tenso
r([7., 9., 4., 2.])
8 torch.Size([4, 3, 3, 64, 64]) ['06107', '03446', '00926', '06027'] tenso
r([7., 3., 1., 7.])
9 torch.Size([4, 3, 3, 64, 64]) ['00113', '06039', '05723', '00902'] tenso
r([0., 7., 7., 0.])
10 torch.Size([4, 3, 3, 64, 64]) ['00496', '06244', '03938', '01214'] tens
or([0., 7., 4., 1.])
11 torch.Size([4, 3, 3, 64, 64]) ['03353', '04298', '05116', '02291'] tens
or([3., 5., 6., 2.])
12 torch.Size([4, 3, 3, 64, 64]) ['06764', '00377', '06691', '04681'] tens
or([8., 0., 8., 5.])
13 torch.Size([4, 3, 3, 64, 64]) ['02426', '00809', '05842', '03244'] tens
or([2., 0., 7., 3.])
14 torch.Size([4, 3, 3, 64, 64]) ['00711', '04061', '06010', '03861'] tens
or([0., 4., 7., 4.])
15 torch.Size([4, 3, 3, 64, 64]) ['00559', '06134', '04873', '04927'] tens
or([0., 7., 5., 5.])
16 torch.Size([4, 3, 3, 64, 64]) ['02344', '06609', '06294', '01964'] tens
or([2., 8., 7., 2.])
17 torch.Size([4, 3, 3, 64, 64]) ['01813', '03759', '00821', '03234'] tens
or([1., 4., 0., 3.])
18 torch.Size([4, 3, 3, 64, 64]) ['05505', '06277', '00763', '00424'] tens
or([6., 7., 0., 0.])
19 torch.Size([4, 3, 3, 64, 64]) ['05768', '02795', '00355', '00178'] tens
or([7., 3., 0., 0.])
20 torch.Size([4, 3, 3, 64, 64]) ['03437', '01794', '04209', '00163'] tens
or([3., 1., 5., 0.])
21 torch.Size([4, 3, 3, 64, 64]) ['06908', '06133', '04654', '04321'] tens
or([8., 7., 5., 5.])
```

In [5]:

```
clip_dataset_train=ActionClipDataset(root_dir='./data/trainClips/',labels=label_train,t
ransform=T.ToTensor())

clip_dataloader_train = DataLoader(clip_dataset_train, batch_size=16,
                        shuffle=True, num_workers=0)
clip_dataset_val=ActionClipDataset(root_dir='./data/valClips/',labels=label_val,transfo
rm=T.ToTensor())

clip_dataloader_val = DataLoader(clip_dataset_val, batch_size=16,
                        shuffle=True, num_workers=0)
clip_dataset_test=ActionClipDataset(root_dir='./data/testClips/',labels=[],transform=T.
ToTensor())

clip_dataloader_test = DataLoader(clip_dataset_test, batch_size=16,
                        shuffle=False, num_workers=0)
```

Write the Flatten for 3d covolution feature maps.

In [6]:

```
class Flatten3d(nn.Module):
    def forward(self, x):
        ###############6th TODO (3 points)##################
        return x.view(x.size()[0], -1)
```

Design a network using 3D convolution on videos for video classification.

In [4]:

```
fixed_model_3d = nn.Sequential( # You fill this in!
    ###############7th TODO (8 points)#########################

    nn.Conv3d(3, 16, kernel_size=(2, 3, 3), padding=1),
    nn.BatchNorm3d(16),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(1, 2, 2), padding=1),

    nn.Conv3d(16, 32, kernel_size=(2, 3, 3), padding=1),
    nn.BatchNorm3d(32),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(1, 2, 2), padding=1),

    nn.Conv3d(32, 64, kernel_size=(2, 3, 3), padding=0),
    nn.BatchNorm3d(64),
    nn.ReLU(),

    nn.Conv3d(64, 64, kernel_size=(2, 3, 3), padding=1),
    nn.BatchNorm3d(64),
    nn.ReLU(),
    nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(1, 2, 2), padding=1),

    Flatten3d(),
    nn.ReLU(),
    nn.BatchNorm1d(32768),
    nn.Linear(32768, 10)
)

fixed_model_3d = fixed_model_3d.type(dtype)
x = torch.randn(32,3, 3, 64, 64).type(dtype)
x_var = Variable(x).type(dtype) # Construct a PyTorch Variable out of your input data
ans = fixed_model_3d(x_var)
np.array_equal(np.array(ans.size()), np.array([32, 10]))
```

Describe what you did (4 points)

In the cell below you should write an explanation of what you did, any additional features that you implemented, and any visualizations or graphs that you make in the process of training and evaluating your network.

8th TODO Tell us here:

Major part of 'what I did' for this section is explained in different notebook file named '**video.ipynb**'. Please have a look at it for rest of description.

Short Version

1)Basic model: Val acc: 70.49 Kaggle score: 66.055

2)Basic model (with augmentation): Val acc: 70.26 Kaggle score: 67.380

2)Pretrained Resnet18-3d (without augmentation) Val acc: 86 Kaggle Score: **81.039**

3)Pretrained Resnet18-3d (with augmentation) Val acc: 85.11 Kaggle Score: 79.306

4)Pretrained Resnet18-3d (with augmentation, minor changes in parameters) Val acc: 85.20 Kaggle Score: 80.937

In [68]:

```
loss_fn = nn.CrossEntropyLoss().type(dtype)
optimizer = optim.RMSprop(fixed_model_3d.parameters(), lr=1e-4)
```

In [7]:

```
def train_3d(model, loss_fn, optimizer,dataloader,num_epochs = 1):
    for epoch in range(num_epochs):
        print('Starting epoch %d / %d' % (epoch + 1, num_epochs))
        model.train()
        for t, sample in enumerate(dataloader):
            x_var = Variable(sample['clip'].type(dtype))
            y_var = Variable(sample['Label'].type(dtype).long())

            scores = model(x_var)

            loss = loss_fn(scores, y_var)
            if (t + 1) % print_every == 0:
                print('t = %d, loss = %.4f' % (t + 1, loss.data[0]))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

def check_accuracy_3d(model, loader):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    for t, sample in enumerate(loader):
        x_var = Variable(sample['clip'].type(dtype))
        y_var = sample['Label'].type(dtype)
        y_var=y_var.cpu()
        scores = model(x_var)
        _, preds = scores.data.cpu().max(1)
        #print(preds)
        #print(y_var)
        num_correct += (preds.numpy() == y_var.numpy()).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
```

In [0]:

```
torch.cuda.random.manual_seed(12345)
fixed_model_3d.apply(reset)
fixed_model_3d.train()
train_3d(fixed_model_3d, loss_fn, optimizer,clip_dataloader_train, num_epochs=1)
fixed_model_3d.eval()
check_accuracy_3d(fixed_model_3d, clip_dataloader_val)
```

My Implementation

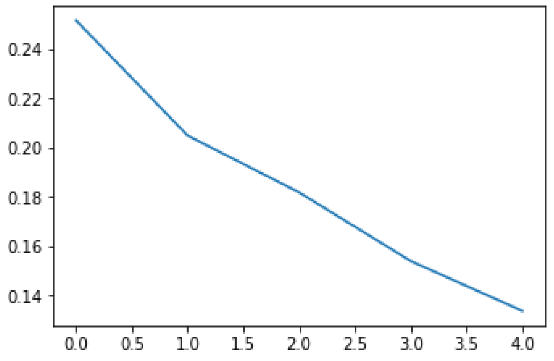
In [86]:

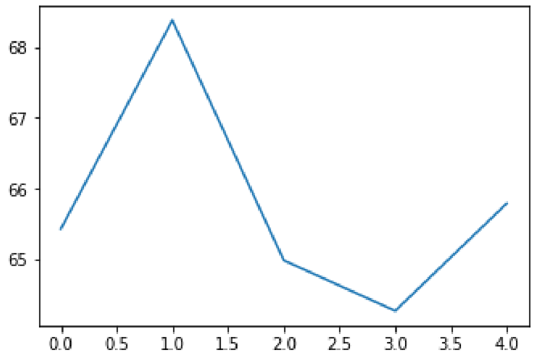
```
def check_accuracy_3d(model, loader, pretrained=None):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    dtype = torch.cuda.FloatTensor
    for t, sample in enumerate(loader):
        if t+1 % 1100==0:
            print('wait')
        x_var = Variable(sample['clip'].type(dtype))
        y_var = sample['Label'].type(dtype)
        model = model.type(dtype)
        y_var=y_var.cpu()
        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)
        _, preds = scores.data.cpu().max(1)
        num_correct += (preds.numpy() == y_var.numpy()).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
    return acc * 100
```

In [87]:

```
train_tf = T.Compose([
    T.ToTensor(),
    T.RandomOrder([
        T.RandomChoice([
            T.RandomHorizontalFlip(p=1),
            T.RandomVerticalFlip(p=1),
        ]),
        T.RandomRotation(90),
        T.RandomErasing(p = 1, value = 'random'),
        T.RandomPerspective()
    ])

])
clip_dataset_train=ActionClipDataset(root_dir='./data/trainClips/',labels=label_train,t
ransform=train_tf)

clip_dataloader_train = DataLoader(clip_dataset_train, batch_size=8,
                        shuffle=True, num_workers=0)
clip_dataset_val=ActionClipDataset(root_dir='./data/valClips/',labels=label_val,transfo
rm=T.ToTensor())

clip_dataloader_val = DataLoader(clip_dataset_val, batch_size=8,
                        shuffle=False, num_workers=0)

model = fixed_model_3d
optimizer = optim.SGD(model.parameters(), lr=1e-4)
loss_fn = nn.CrossEntropyLoss().type(dtype)
pretrained = None

num_epochs = 5
train_losses = []
val_losses = []
val_accs = []
best_acc = 50
PATH = 'saved_models/3d_{}.pt'
for epoch in range(num_epochs):
    losses = []
    dtype = torch.cuda.FloatTensor
    for t, sample in enumerate(clip_dataloader_train):
        model.train()

        x_var = sample['clip'].type(dtype)
        y_var = sample['Label'].type(dtype).long()


        model = model.type(dtype)

        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)

        loss = loss_fn(scores.cpu(), y_var.cpu())
        if (t + 1) % 200 == 0:
            print('t = %d, loss = %.4f' % (t + 1, loss.data.item()))
        losses.append(loss.data.item())
        optimizer.zero_grad()
        loss.backward()
#          torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
```

```
        optimizer.step()
    train_losses.append(np.mean(np.array(losses)))
    acc = check_accuracy_3d(model, clip_dataloader_val, pretrained=pretrained)

    val_losses.append(np.mean(np.array(losses)))

    val_accs.append(acc)
    if epoch > 0:
        clear_output(True)
        plt.figure(figsize=(12, 8))
        plt.subplot(221)
        plt.plot(train_losses)
        plt.subplot(222)
        plt.plot(val_accs)
        plt.show()
        print("Epoch", epoch+1, "=> ", "Val acc:", acc)
    if acc > best_acc:
        best_acc = acc
        torch.save(model, PATH.format(str(acc)[:5]))
```





```
Epoch 5 =>  Val acc: 65.7847533632287
```

In [89]:

```
fixed_model_3d = torch.load(PATH.format(str(best_acc)[:5]))
fixed_model_3d.eval()
check_accuracy_3d(fixed_model_3d, clip_dataloader_val)
```

```
Got 1525 / 2230 correct (68.39)
```

Out[89]:

```
68.38565022421524
```

The next experiment is done in different notebook file, here I have shared the code and his val accuracy using saved model. Please find the experiment in the file named '**video.ipynb**'.

In [ ]:

```
pretrained = torchvision.models.video.r3d_18(pretrained=True)
model = pretrained
model.fc = nn.Linear(512,10)
pretrained = None

loss_fn = nn.CrossEntropyLoss().type(dtype)
optimizer = optim.SGD(model.parameters(), lr=1e-4, momentum = 0.8)

train_tf = T.Compose([
    T.Resize(112),
    T.ToTensor(),
    T.RandomOrder([
        T.RandomVerticalFlip(p=0.8)
    ]),
    T.Normalize(mean = [0.43216, 0.394666, 0.37645], std = [0.22803, 0.22145, 0.216989
])
])
val_tf = T.Compose([
    T.Resize(112),
    T.ToTensor(),
    T.Normalize(mean = [0.43216, 0.394666, 0.37645], std = [0.22803, 0.22145, 0.216989
])
])

clip_dataset_train=ActionClipDataset(root_dir='./data/trainClips/',labels=label_train,t
ransform=train_tf)

clip_dataloader_train = DataLoader(clip_dataset_train, batch_size=10,
                        shuffle=True, num_workers=0)
clip_dataset_val=ActionClipDataset(root_dir='./data/valClips/',labels=label_val,transfo
rm=val_tf)

clip_dataloader_val = DataLoader(clip_dataset_val, batch_size=10,
                        shuffle=False, num_workers=0)

num_epochs = 15
train_losses = []
val_losses = []
val_accs = []
best_acc = 87.71
PATH = 'saved_models/Norm3d_{}.pt'
for epoch in range(num_epochs):
    losses = []
    dtype = torch.cuda.FloatTensor
    for t, sample in enumerate(clip_dataloader_train):
        model.train()

        x_var = sample['clip'].type(dtype)
        y_var = sample['Label'].type(dtype).long()


        model = model.type(dtype)

        if pretrained:
            pretrained = pretrained.type(dtype)
            x_var = pretrained(x_var)
        scores = model(x_var)

        loss = loss_fn(scores.cpu(), y_var.cpu())
        if (t + 1) % 200 == 0:
            print('t = %d, loss = %.4f' % (t + 1, loss.data.item()))
        losses.append(loss.data.item())
        optimizer.zero_grad()
        loss.backward()
#         torch.nn.utils.clip_grad_norm_(model.parameters(), 3)
        optimizer.step()
    train_losses.append(np.mean(np.array(losses)))
    acc = check_accuracy_3d(model, clip_dataloader_val, pretrained=pretrained)

    val_losses.append(np.mean(np.array(losses)))

    val_accs.append(acc)

    if acc > best_acc:
        best_acc = acc
        torch.save(model, PATH.format(str(acc)[:5]))

    if epoch > 0:
        clear_output(True)
        plt.figure(figsize=(12, 8))
        plt.subplot(221)
        plt.plot(train_losses)
        plt.subplot(222)
        plt.plot(val_accs)
        plt.show()
        print("Epoch", epoch+1, "=> ", "Val acc:", acc, "=> Best Till now", best_acc)
```

In [9]:

```
# model experiment is there in video.ipynb
dtype = torch.cuda.FloatTensor
fixed_model_3d = torch.load('saved_models/Norm3d_87.71.pt')
fixed_model_3d.eval()
check_accuracy_3d(fixed_model_3d, clip_dataloader_val)
```

```
Got 1956 / 2230 correct (87.71)
```

Test your 3d convolution model on the validation set. You don't need to submit the result of this part to kaggle.

Test your model on the test set, predict_on_test_3d() will generate a file named 'results_3d.csv'. Please submit the csv file to kaggle https://www.kaggle.com/c/cse512springhw3video (https://www.kaggle.com/c/cse512springhw3video) The highest 3 entries get extra 10 points.

In [91]:

```
def predict_on_test_3d(model, loader):
    '''
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    '''
    num_correct = 0
    num_samples = 0
    model.eval() # Put the model in test mode (the opposite of model.train(), essential
ly)
    results=open('results_3d.csv','w')
    count=0
    results.write('Id'+','+'Class'+'\n')
    for t, sample in enumerate(loader):
        x_var = Variable(sample['clip'].type(dtype))
        scores = model(x_var)
        _, preds = scores.data.max(1)
        for i in range(len(preds)):
            results.write(str(count)+','+str(preds[i].item())+'\n')
            count+=1
    results.close()
    return count

count=predict_on_test_3d(fixed_model_3d, clip_dataloader_test)
print(count)
```

```
3270
```

In [0]:

```

```

In [ ]:

```

```