VGG

Paper

Architecture

1. All filters of same size (3 x 3) : smallest size to capture left/right/up/down/center

2. Sometimes used 1x1 but the models without it seemed to be better : purpose of such filter is to add non-linearity within convolution but is this reaallly necessary?

3. Spatial resolution conserved during convolution (padding=1)

Paper

Training

Q

1. Optimize multinomial logistic regression (== softmax classifier?)
2. Learning rate decreased by factor of 10 when learning stagnates
3. 74 epochs == 370K iterations required to learn well : learns faster than previous models due to 1) implicit regularization imposed by greater depth 2) pre-initialization of certain layers
4. Random initialization == normal_
5. Pretraining with configuration A with random initialization
6. RandomCrop / RandomHorizontalFlip / RGB Color Shift

Paper

Others

1. Single/Multi-scale evaluation : larger the better, range of scale is even better

2. ConvNet Fusion : averaging softmax outputs

Paper

Others…

Multi-crop evaluation

1. Using a large set of crops

Q

2. feature map is padded with zero : from the input ??

Dense evaluation

Q

1. What is dense evaluation : using the whole image as opposed to multi-crop? Or something more?

Paper

Q Configuration D or E?

Table 7: **Comparison with the state of the art in ILSVRC classification.** Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Code

Summary : what I've done

1. Pretraining on configuration A for CIFAR10 – 20 epochs
2. Training on configuration for 74 epochs
3. Pretraining on configuration A for ImageWoof – 20 epochs
4. Training on configuration for 74 epochs

Summary : what I didn't do

1. RGB color shift for image / data augmentation

Q

1. How to do RGB color shift in torchvision.transforms

Code

Snaps : Beginning notes

```
# going to use imagewoof dataset for training VGG net D 16

# brief note on VGG 16 architecture :

# seems like there is no batch normalisation
# all hidden layers "equipped" with ReLU
# dropout on first two layers of fully connected layers (p=0.5)
# L2 regularizer
# mini-batch gradient descent with momentum (batchsize=256) (based on LeNet)
# initial learning rate 10^-2
# learning rate decreased fater of 10 when val acc stop improving (ReduceLRonPlateau?)
# 370K iterations = 74 epochs
# augmentations : randomcrop, horizontalflip, randomrgbcolorshift
# shallow train net A first and then do transfer learning?
# random initialization with zero mean 10-^2 variance
```

Code

Snaps #2 What I use

```
[ ] import torch
```

```
[ ] import torch.nn as nn
    import torch.nn.functional as F
    from torch.utils.data import DataLoader
```

Double-click (or enter) to edit

```
[ ] torch.__version__
```

```
'1.7.0+cu101'
```

```
[ ] import torchvision
    import torchvision.transforms as transforms
```

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
```

```
[ ] print("check device:", torch.cuda.get_device_name())
    print("how many?:", torch.cuda.device_count())
    print("so can i use it?", torch.cuda.is_available())
```

```
check device: Tesla P4
how many?: 1
so can i use it? True
```

Code

Snaps #3 Augmentation

```
# page 2
# "The only preprocessing we do is subtracting the mean RGB value,
# computed on the training set, from each pixel."

mean = 0.0
for image, _ in train_set:
    mean += image.mean([1,2])

mean = mean/len(train_set)
print(mean)

del train_set, train_transform
```

```
tensor([0.4914, 0.4822, 0.4465])
```

```
[ ]  # using torchvision transformation
     # augmentations : randomcrop, horizontalflip, randomrgbcolorshift
     # image for CIFAR10 is too small for randomcrop... so add padding
     # googled std :P

     train_transform = transforms.Compose([
                                 transforms.RandomCrop(32, padding=4),
                                 transforms.RandomHorizontalFlip(p=0.5),
                                 transforms.ToTensor(),
                                 transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010))
     ])

     test_transform = transforms.Compose([
                                transforms.ToTensor(),
                                transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010))
     ])
```

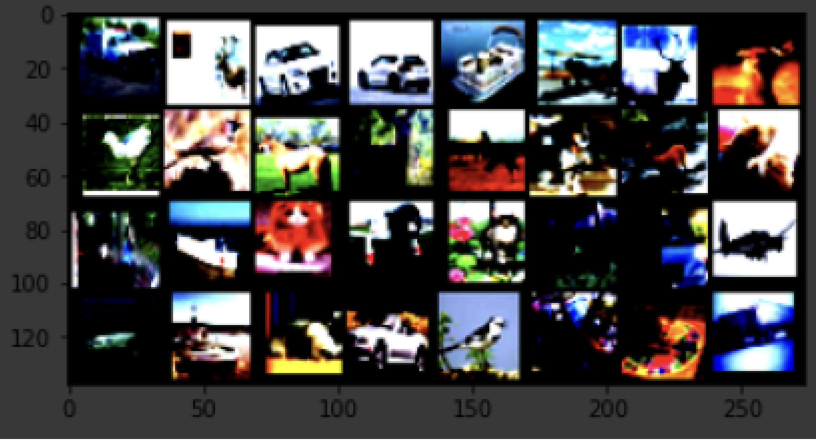
Code

Snaps #4 How the images turn out

```
dataiter = iter(train_loader)
images, labels = dataiter.next()

imshow(torchvision.utils.make_grid(images))
```

```
Clipping input data to the valid range for imshow with
```



Code

Snaps #5 Construction

```
class VGGa(nn.Module):
    def __init__(self):
        super(VGGa, self).__init__()

        self.conv1a = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU()
        )
        self.conv1d = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU()
        )
        self.conv2a = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU()
        )
        self.conv2d = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU()
        )
```

…

Code

Snaps #6 Construction

```python
def forward(self, x):
    x = self.conv1a(x)
    x = self.maxpool(x)
    x = self.conv2a(x)
    x = self.maxpool(x)
    x = self.conv3a(x)
    x = self.maxpool(x)
    x = self.conv4a(x)
    x = self.maxpool(x)
    x= self.conv5a(x)
    x = self.maxpool(x)
    #print('before view', x.shape)

    # 224 --> 112 --> 56 --> 28 --> 14 --> 7
    # 512 * 7 * 7 = 25088
    x = x.view(-1, 512)
    #print('after view', x.shape)
    x = self.dropout(x)
    x = self.fca(x)
    #x = self.softmax(x)
    return x
```

Code

Snaps #7 Weight_init

```python
# as in the paper,
# init weights with normal distribution
# init biases with zeros

# init all weights and biases

# next time try making this inside the module

def init_weights(m):

    for i, child in enumerate(m.children()):
        if type(child) is (nn.Linear or nn.Conv2d):
            torch.nn.init.normal_(child.weight, mean=0.0, std=0.01)
            torch.nn.init.zeros_(child.bias)
        elif type(child) == nn.Sequential:
            for j, grandchild in enumerate(child.children()):
                #print(j, grandchild)
                if type(child) is (nn.Linear or nn.Conv2d):
                    torch.nn.init.normal_(grandchild.weight, mean=0.0, std=0.01)
                    torch.nn.init.zeros_(grandchild.bias)
```

Code

Snaps #8 lossfn / optim / sched

```
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(vgga.parameters(), lr=learning_rate,
                            momentum=momentum, weight_decay=weight_decay)

# patience=5 chosen arbitrarily
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=0.1, patience=5, verbose=True
    )
```

Code

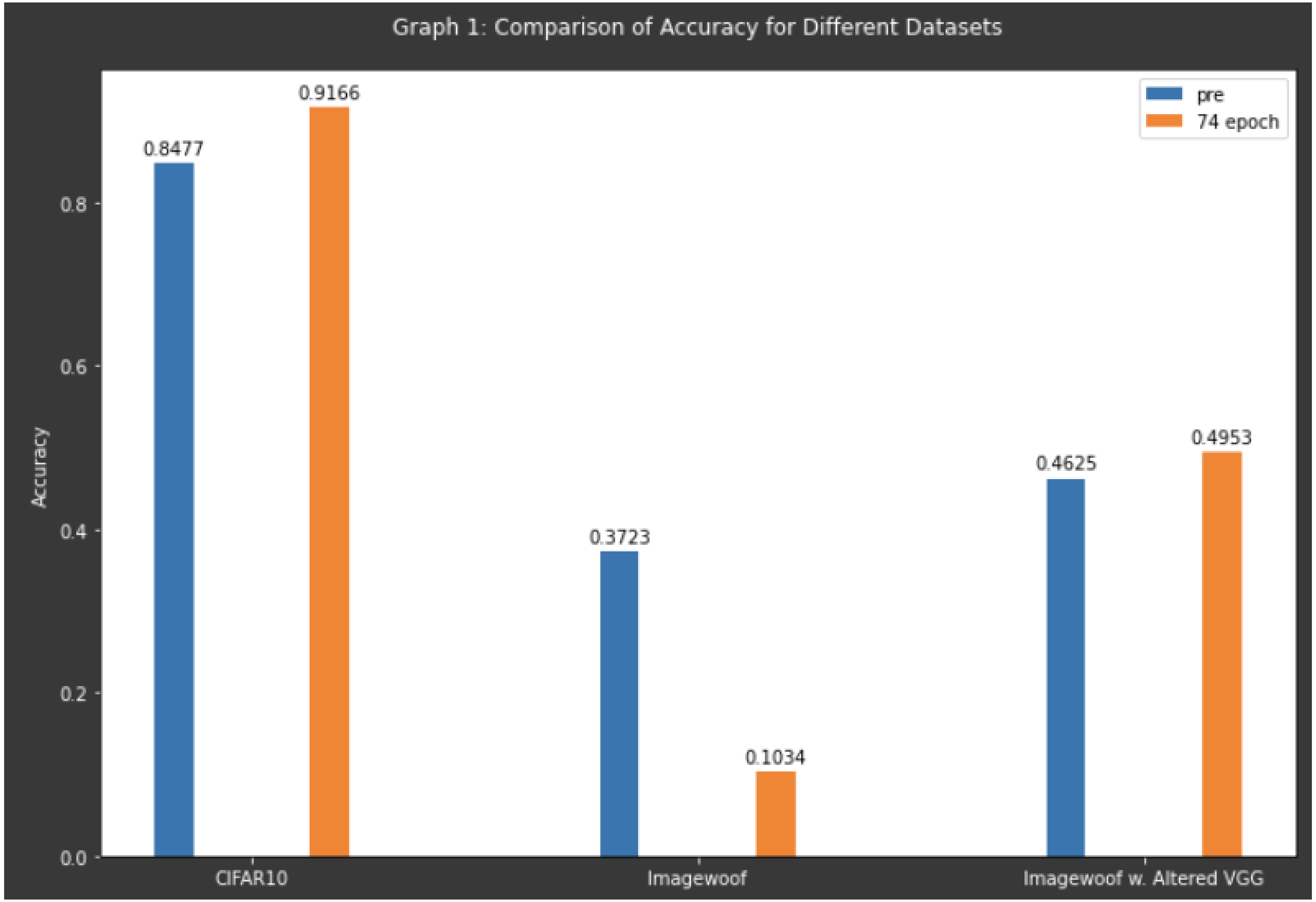
Snaps #9 Pretraining after 20 epoch

```
after looking at 32 images, running_acc is 0.9062
after looking at 1632 images, running_acc is 0.8536
after looking at 3232 images, running_acc is 0.8493
after looking at 4832 images, running_acc is 0.8506
after looking at 6432 images, running_acc is 0.8484
after looking at 8032 images, running_acc is 0.8461
after looking at 9632 images, running_acc is 0.8478
epoch 19 : acc 0.8477
### saving current model...###
```

Thoughts :
20 epoch might make overfit the model, since it is “pre-training” the training could have been lighter

Code

Snaps #10 Result Graph



Code

Notes on result

1. FC layer for CIFAR10 is just one nn.Linear(512,10)
2. FC layer for Imagewoof is three as given in the paper
3. The FC layer for Imagewoof seems to be too big – too complicated for the given problem / given dataset size – needs simplifying (or even simplifying conv layers – reduce padding to make spatial resolution shrink)