Q5: Analysis (20 points)

By now you should know how to train networks from scratch or using from pre-trained models. You should also understand the relative performance in either scenarios. Needless to say, the performance of these models is stronger than previous non-deep architectures used until 2012. However, final performance is not the only metric we care about. It is important to get some intuition of what these models are really learning. Lets try some standard techniques.

FEEL FREE TO WRITE UTIL CODE IN ANOTHER FILE AND IMPORT IN THIS NOTEBOOK FOR EASE OF READABILITY

5.1 Nearest Neighbors (7 pts)

Pick 3 images from PASCAL test set from different classes, and compute 4 nearest neighbors of those images over the test set. You should use and compare the following feature representations for the nearest neighbors:

1. fc7 features from the ResNet (finetuned from ImageNet)
2. pool5 features from the CaffeNet (trained from scratch)

CaffeNet 4 Nearest Neighbors

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import matplotlib.pyplot as plt
%matplotlib inline

import trainer
from utils import ARGS
from simple_cnn import SimpleCNN
from voc_dataset import VOCDataset

import os
import numpy as np

#load CaffeNet model
def get_fc(inp_dim, out_dim, non_linear='relu'):
    """
    Mid-level API. It is useful to customize your own for large code repo.
    :param inp_dim: int, intput dimension
    :param out_dim: int, output dimension
    :param non_linear: str, 'relu', 'softmax'
    :return: list of layers [FC(inp_dim, out_dim), (non linear layer)]
    """
    layers = []
    layers.append(nn.Linear(inp_dim, out_dim))
    if non_linear == 'relu':
        layers.append(nn.ReLU())
    elif non_linear == 'softmax':
        layers.append(nn.Softmax(dim=1))
    elif non_linear == 'none':
        pass
    else:
        raise NotImplementedError
    return layers

class CaffeNet(nn.Module):
    def __init__(self):
        super().__init__()
        c_dim = 3
        self.conv1 = nn.Conv2d(c_dim,96,11,4,padding=0) # valid padding
        self.pool1 = nn.MaxPool2d(3,2)
        self.conv2 = nn.Conv2d(96, 256, 5,padding=2) # same padding
        self.pool2 = nn.MaxPool2d(3,2)
        self.conv3 = nn.Conv2d(256,384,3,padding=1) # same padding
        self.conv4 = nn.Conv2d(384,384,3,padding=1) # same padding
        self.conv5 = nn.Conv2d(384,256,3,padding=1) # same padding
        self.pool3 = nn.MaxPool2d(3,2)
        self.flat_dim = 5*5*256 # replace with the actual value
        self.fc1 = nn.Sequential(*get_fc(self.flat_dim, 4096, 'relu'))
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Sequential(*get_fc(4096, 4096, 'relu'))
        self.dropout2 = nn.Dropout(p=0.5)
        self.fc3 = nn.Sequential(*get_fc(4096, 20, 'none'))

        self.nonlinear = lambda x: torch.clamp(x,0)
```

```
    def forward(self, x):
        N = x.size(0)
        x = self.conv1(x)
        x = self.nonlinear(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.nonlinear(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.nonlinear(x)
        x = self.conv4(x)
        x = self.nonlinear(x)
        x = self.conv5(x)
        x = self.nonlinear(x)
        x = self.pool3(x)
        x = x.view(N, self.flat_dim) # flatten the array

        out = self.fc1(x)
        out = self.nonlinear(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        out = self.nonlinear(out)
        out = self.dropout2(out)
        out = self.fc3(out)

        return x
```

In [ ]:

```
import utils
from sklearn.neighbors import KNeighborsClassifier

args = ARGS(batch_size = 32, use_cuda = True)

if __name__ == "__main__":
    modelCaffe = CaffeNet()
    torchLoadCaffe = torch.load('saved_models/CaffeNet-50.pth')
    modelCaffe.load_state_dict(torchLoadCaffe['model_state_dict'])
    modelCaffe = modelCaffe.to(args.device)
    modelCaffe.eval()
    testfindex, testOutput, testTarget = trainer.train_output_CaffeNet(modelCa
ffe, args)

    # build k- nearest neighbors
    clf = KNeighborsClassifier(n_neighbors = 4)
    clf.fit(testOutput, testfindex)
```

In [ ]:
```
vocTest = VOCDataset(split='test',size=64)
# how big are train and test sets?
print("Test Set is: ", testOutput.shape)

# select three random images from the test set
num = 3
indexArr = []
sampleTestOutputArr = np.ones((1,6400))
sampleTestfindexArr = np.ones((1,1))

for i in range(num):
    randNum = int(np.random.rand()*len(testOutput))
    if randNum not in indexArr:
        indexArr.append(randNum)
        sampleTestOutputArr = np.concatenate((sampleTestOutputArr,testOutput[r
andNum][np.newaxis,:]))
        sampleTestfindexArr = np.concatenate((sampleTestfindexArr,testfindex[r
andNum][np.newaxis,:]))

sampleTestOutputArr = sampleTestOutputArr[1:,:]
sampleTestfindexArr = sampleTestfindexArr[1:,:]

print("Size of Test Sample Output is: ", sampleTestOutputArr.shape)
print("Size of Test Sample Output is: ", sampleTestfindexArr.shape)
```

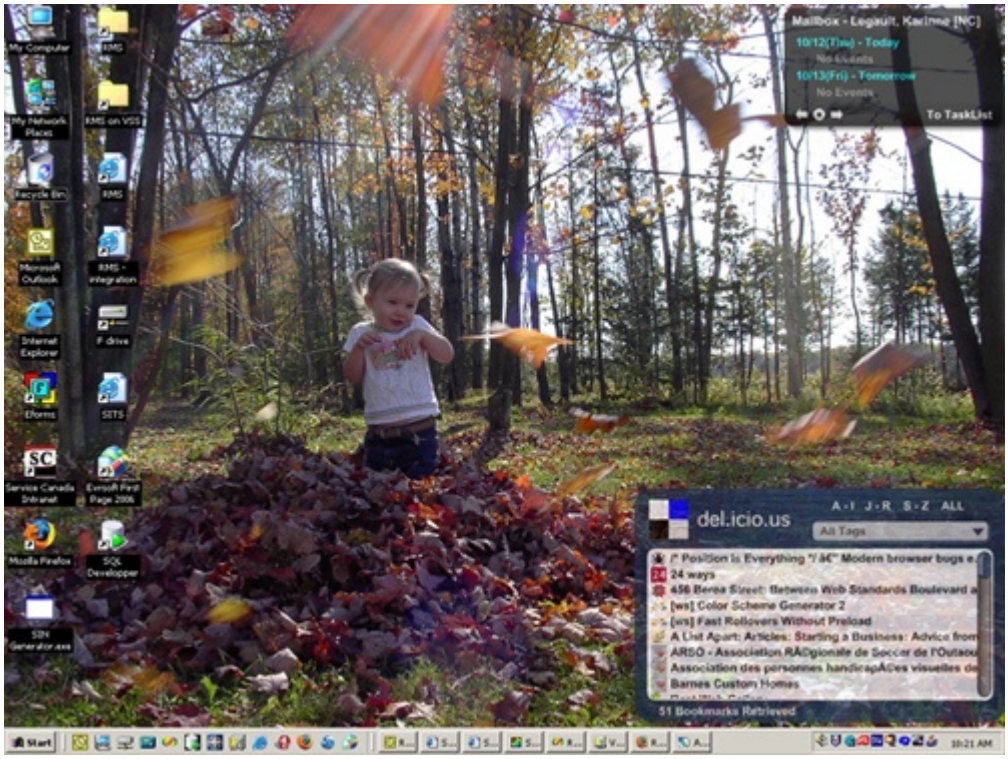
In [ ]:
```
# choose four values between 1 and 5011
testPred = clf.kneighbors(sampleTestOutputArr, return_distance=False)
print("The input is: ", np.transpose(sampleTestfindexArr))
# print("The prediction is: ",testPred)

numVal = 3

neighborImageIndex = np.zeros((numVal,4))

for i in range(numVal):
    for j in range(4):
        neighborImageIndex[i,j] = vocTest.index_list[testPred[i,j]]


print("The prediction is: ", neighborImageIndex)
```

Sample prediction output:

```
The input is:  [['008515' '008131' '006491']]
The prediction is:  [[8515. 2016.  562. 3659.]
 [8131. 8356. 3707. 3514.]
 [6491. 1089. 5673. 5048.]]
```

For the input image shown below ('008515.jpg'):

The follow 4 neighboring images were given:

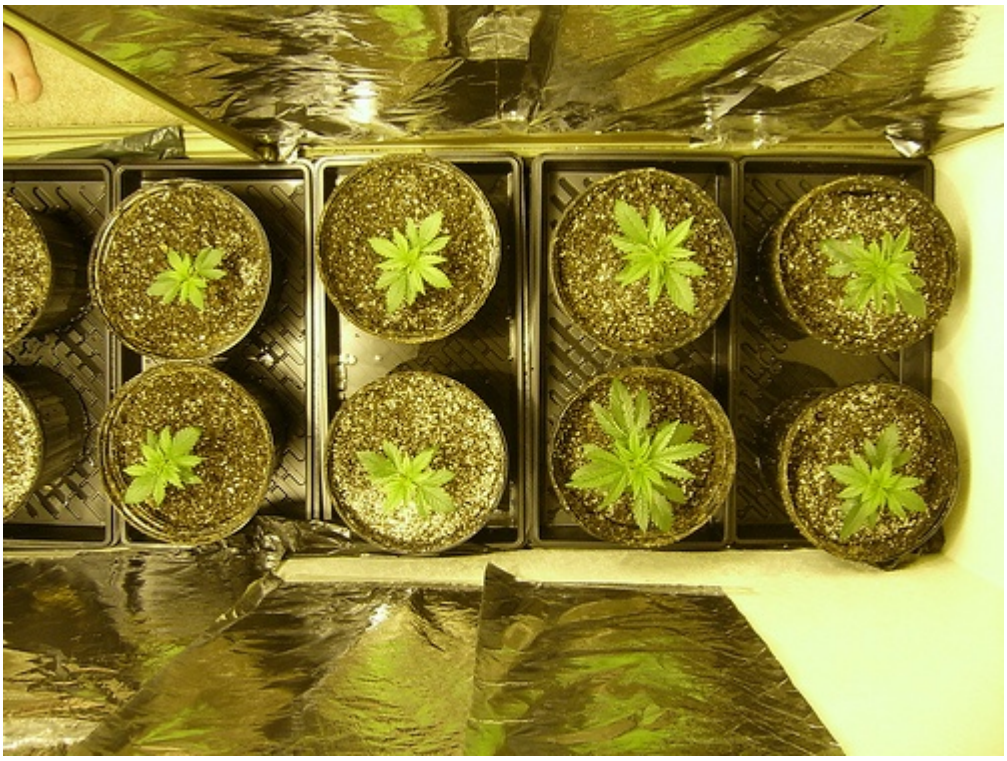


For the input image shown below ('008131.jpg'):

The follow 4 neighboring images were given:

For the input image shown below ('006491.jpg'):

The follow 4 neighboring images were given:



ResNet 4 Nearest Neighbors Section

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import matplotlib.pyplot as plt
%matplotlib inline

import trainer
from utils import ARGS
from simple_cnn import SimpleCNN
from voc_dataset import VOCDataset

#
import os
import numpy as np
```

In [ ]:
```
# Pre-trained weights up to second-to-last layer
# final layers should be initialized from scratcH!
class PretrainedResNet(nn.Module):
    def __init__(self):
        super().__init__()
        # load resnet model
        self.modelres = models.resnet18(pretrained = True)
        for params in self.modelres.parameters():
            params.requires_grad = False

        self.model= nn.Sequential(self.modelres,nn.Linear(1000,20,bias=True))

    def forward(self, x):
        return self.model(x)
```

In [ ]:
```
import utils
from sklearn.neighbors import KNeighborsClassifier

args = ARGS(batch_size = 16, use_cuda = True)

# Create ResNet model
if __name__ == '__main__':
    m = PretrainedResNet()
    model = m.model
    model = torch.load('saved_models/PreTrainedResNet-Model.pth')
    #model.load_state_dict(torch.load('q4_resnet_pretrained_statedict.pth'))
    model = model.to(args.device)
    testfindex, testOutput, testTarget = trainer.train_output_ResNet(model, ar
gs)

    clf = KNeighborsClassifier(n_neighbors = 4)
    clf.fit(testOutput, testfindex)
```

In [ ]:
```
vocTest = VOCDataset(split='test',size=64)

# how big are train and test sets?
print("Test Set is: ", testOutput.shape)

# select three random images from the test set
num = 3
indexArr = []
sampleTestOutputArr = np.ones((1,512))
sampleTestfindexArr = np.ones((1,1))

for i in range(num):
    randNum = int(np.random.rand()*len(testOutput))
    if randNum not in indexArr:
        indexArr.append(randNum)
        sampleTestOutputArr = np.concatenate((sampleTestOutputArr,testOutput[r
andNum][np.newaxis,:]))
        sampleTestfindexArr = np.concatenate((sampleTestfindexArr,testfindex[r
andNum][np.newaxis,:]))

sampleTestOutputArr = sampleTestOutputArr[1:,:]
sampleTestfindexArr = sampleTestfindexArr[1:,:]

print("Size of Test Sample Output is: ", sampleTestOutputArr.shape)
print("Size of Test Sample Output is: ", sampleTestfindexArr.shape)
```

In [ ]:

```
# choose four values between 1 and 5011
testPred = clf.kneighbors(sampleTestOutputArr, return_distance=False)
print("The input is: ", np.transpose(sampleTestfindexArr))
# print("The prediction is: ",testPred)

numVal = 3

neighborImageIndex = np.zeros((numVal,4))

for i in range(numVal):
    for j in range(4):
        neighborImageIndex[i,j] = vocTest.index_list[testPred[i,j]]


print("The prediction is: ", neighborImageIndex)
```

Sample prediction output:

```
The input is:  [['000517' '003326' '001374']]
The prediction is:  [[ 517. 6194.  205. 5196.]
 [3326. 3246. 3977. 1000.]
 [1374. 9521. 9632. 6422.]]
```

For the input image shown below ('000517.jpg'):

The follow 4 neighboring images were given:

For the input image shown below ('003326.jpg'):

The follow 4 neighboring images were given:


swell

For the input image shown below ('001374.jpg'):

The follow 4 neighboring images were given:

5.2 t-SNE visualization of intermediate features (7pts)

We can also visualize how the feature representations specialize for different classes. Take 1000 random images from the test set of PASCAL, and extract caffenet (scratch) fc7 features from those images. Compute a 2D t-SNE projection of the features, and plot them with each feature color coded by the GT class of the corresponding image. If multiple objects are active in that image, compute the color as the ”mean” color of the different classes active in that image. Legend the graph with the colors for each object class.

CaffeNet t-SNE

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import matplotlib.pyplot as plt
%matplotlib inline

import trainer
from utils import ARGS
from simple_cnn import SimpleCNN
from voc_dataset import VOCDataset

import os
import numpy as np

#load CaffeNet model
def get_fc(inp_dim, out_dim, non_linear='relu'):
    """
    Mid-level API. It is useful to customize your own for large code repo.
    :param inp_dim: int, intput dimension
    :param out_dim: int, output dimension
    :param non_linear: str, 'relu', 'softmax'
    :return: list of layers [FC(inp_dim, out_dim), (non linear layer)]
    """
    layers = []
    layers.append(nn.Linear(inp_dim, out_dim))
    if non_linear == 'relu':
        layers.append(nn.ReLU())
    elif non_linear == 'softmax':
        layers.append(nn.Softmax(dim=1))
    elif non_linear == 'none':
        pass
    else:
        raise NotImplementedError
    return layers

class CaffeNet(nn.Module):
    def __init__(self):
        super().__init__()
        c_dim = 3
        self.conv1 = nn.Conv2d(c_dim,96,11,4,padding=0) # valid padding
        self.pool1 = nn.MaxPool2d(3,2)
        self.conv2 = nn.Conv2d(96, 256, 5,padding=2) # same padding
        self.pool2 = nn.MaxPool2d(3,2)
        self.conv3 = nn.Conv2d(256,384,3,padding=1) # same padding
        self.conv4 = nn.Conv2d(384,384,3,padding=1) # same padding
        self.conv5 = nn.Conv2d(384,256,3,padding=1) # same padding
        self.pool3 = nn.MaxPool2d(3,2)
        self.flat_dim = 5*5*256 # replace with the actual value
        self.fc1 = nn.Sequential(*get_fc(self.flat_dim, 4096, 'relu'))
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Sequential(*get_fc(4096, 4096, 'relu'))
        self.dropout2 = nn.Dropout(p=0.5)
        self.fc3 = nn.Sequential(*get_fc(4096, 20, 'none'))

        self.nonlinear = lambda x: torch.clamp(x,0)
```

```
    def forward(self, x):
        N = x.size(0)
        x = self.conv1(x)
        x = self.nonlinear(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.nonlinear(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.nonlinear(x)
        x = self.conv4(x)
        x = self.nonlinear(x)
        x = self.conv5(x)
        x = self.nonlinear(x)
        x = self.pool3(x)
        x = x.view(N, self.flat_dim) # flatten the array

        out = self.fc1(x)
        out = self.nonlinear(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        out1 = out
        out = self.nonlinear(out)
        out = self.dropout2(out)
        out = self.fc3(out)

        return out1
```

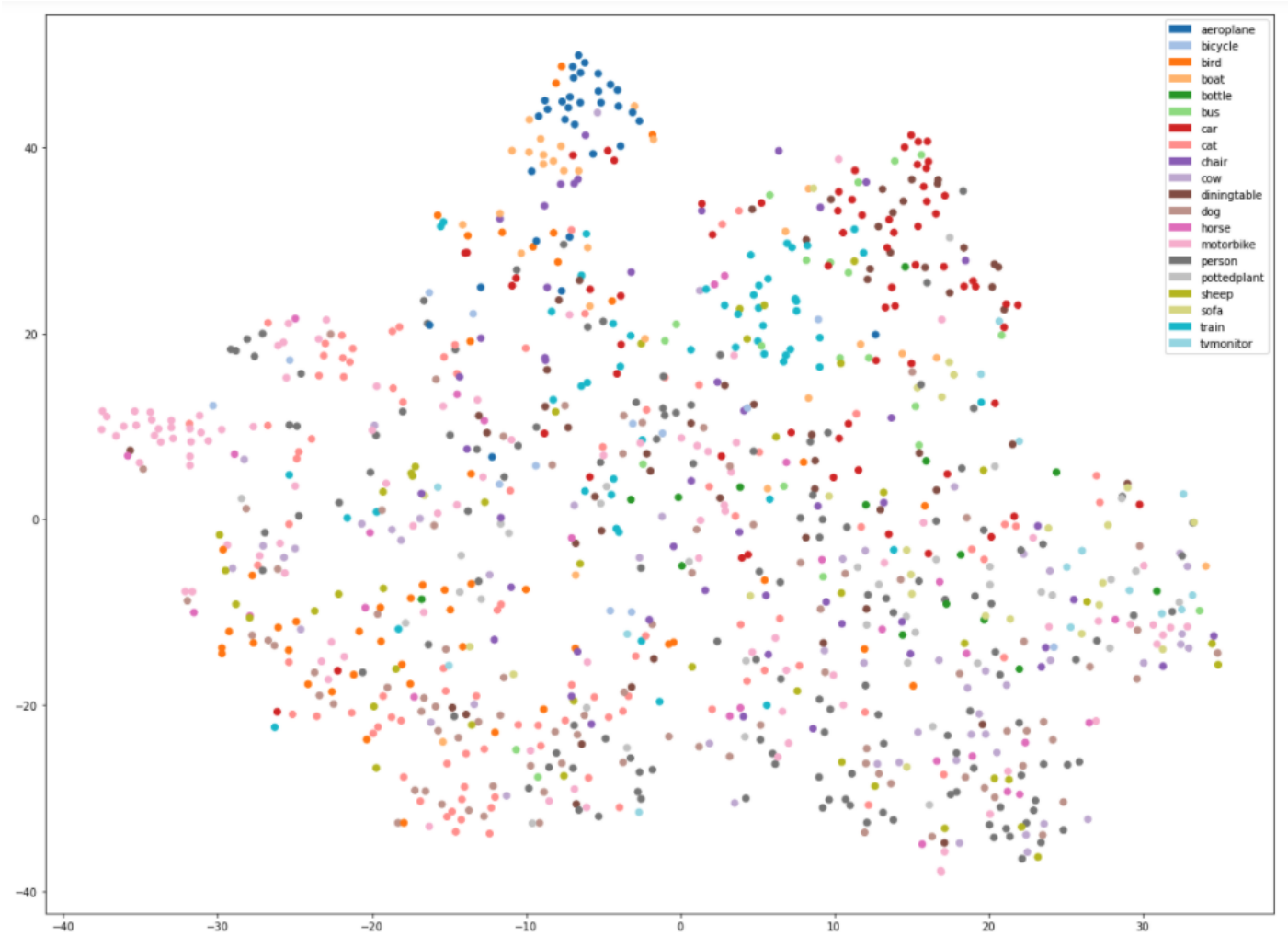
In [ ]:
```
if __name__ == '__main__':
    modelCaffe = CaffeNet()
    torchLoadCaffe = torch.load('saved_models/CaffeNet-50.pth')
    modelCaffe.load_state_dict(torchLoadCaffe['model_state_dict'])
    modelCaffe = modelCaffe.to(args.device)
    modelCaffe.eval()
    testfindex, testOutput, testTarget = trainer.train_output_CaffeNet(modelCa
ffe, args)
```

In [ ]:
```
# define "color" averaging function
def avg_color_label(labels):
    oneHotSum = np.sum(labels,axis=1)
    indexHotSum = np.sum(labels*np.arange(1,21)[np.newaxis,:],axis=1)

    returnVect = indexHotSum/oneHotSum


    return returnVect[:,np.newaxis]
```

In [ ]:
```
import numpy as np
from sklearn.manifold import TSNE

# select 1000 random images from the test set
num = 1000
indexArr = []
sneTestOutputArr = np.ones((1,6400))
sneTestTargetArr = np.ones((1,20))

while len(indexArr) < 1000:
    randNum = int(np.random.rand()*len(testOutput))
    if randNum not in indexArr:
        indexArr.append(randNum)
        sneTestOutputArr = np.concatenate((sneTestOutputArr,testOutput[randNum
][np.newaxis,:]))
        sneTestTargetArr = np.concatenate((sneTestTargetArr,testTarget[randNum
][np.newaxis,:]))
    i += 1

sneTestOutputArr = sneTestOutputArr[1:,:]
sneTestTargetArr = sneTestTargetArr[1:,:]

x_inbedded = TSNE(n_components=2).fit_transform(sneTestOutputArr)
```

In [ ]:
```
import matplotlib

meanLabels = avg_color_label(sneTestTargetArr)
#meanLabels = meanLabels/np.amax(meanLabels)
meanLabels = np.squeeze(meanLabels,axis=1)
plt.figure(figsize=(20,15))
cmap = plt.get_cmap('tab20')
recs = []
legend = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car',
                  'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'moto
rbike',
                  'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvmonit
or']
plt.scatter(x_inbedded[:,0],x_inbedded[:,1],c = meanLabels, cmap = plt.get_cma
p('tab20'))
for i in range(0,len(meanLabels)):
    recs.append(matplotlib.patches.Rectangle((0,0),1,1,fc=cmap(i)))
plt.legend(recs,legend,loc=1)
```

The output plot is shown as the following figure below:



5.3 Are some classes harder? (6pts)

Show the per-class performance of your caffenet (scratch) and ResNet (finetuned) models. Try to explain, by observing examples from the dataset, why some classes are harder or easier than the others (consider the easiest and hardest class). Do some classes see large gains due to pre-training? Can you explain why that might happen?

YOUR ANSWER HERE

CaffeNet and PreTrained ResNet

The class that had the highest AP performance was the "person" while the class that had the lowest performance was the "bottle".

There can be many reasons as to why the "person" performs the best. Two images of the class are shown below for reference. One reason could be that the features of a person are the most distinct from the other classes in the dataset such as "cow" or "aeroplane". In addition, a nondifficult image of a person in the dataset exposes a lot of features such as the upright position, arms, legs, facial features, colored attire, which can help to differentiate it from the other images in the dataset during training.

Two images of the class "bottle" are shown below for reference. Compared to the features of a human, there a significantly less features of a bottle for the model to detect. In some cases, the background may make the profile harder to detect, as shown in the image below where a close-up picture was taken of some wine bottles in a dark room. In addition, features of bottles may be more and more difficult to detect as the object is further away from the camera. For example, the label of a bottle may be easier to extract as a feature when the camera is close-up, but when the bottle is a couple of meters away from the camera, then the label would be difficult for the model to spot.


Farmer's
Daughter
2004

TreTrained ResNet vs From-Scratch ResNet

For reference, the per-class performance of the From-Scratch ResNet model is shown below:

aeroplane: 0.695663796029232
bicycle: 0.5622480632184709
bird: 0.41245039413080287
boat: 0.5710383505097775
bottle: 0.15982507088326053
bus: 0.5238514422134894
car: 0.7453615983966118
cat: 0.46971015685819406
chair: 0.4283246799656059
cow: 0.28834160996828456
diningtable: 0.40786228993246554
dog: 0.3652161054748284
horse: 0.7533615316892681
motorbike: 0.6526196860641374
person: 0.8447806695750432
pottedplant: 0.2523940858888379
sheep: 0.38764743196135676

sofa: 0.3606603351942273
train: 0.7182136028211066
tvmonitor: 0.4197671608743151

Though most classes saw some level of performance gain by using the transfer learning approach, the classes that saw the most significant gain were the cow, dog, and sheep classes. The gain percentage was abou 3-5% in the pretrained model. One reason for this specific gain is attributed to that the model was getting better at differentiating between the three classes. As seen through some iteration with the 4 nearest neighbors model, cows, dogs, and sheeps were frequently predicted between each other, probably due to the similar features and shape between classes. Using a transfer learning approached allow the model to run through another set of iterations to differentiate between the classes. Freezing the previous layers also allowed the model to focus on training on layer of the model, which proved to be successful in gaining performance compared to its From-Scratch counterpart.

CaffeNet Class Performance Analysis

In [ ]:

```
# redefine CaffeNet without the pool5 output
class CaffeNet(nn.Module):
    def __init__(self):
        super().__init__()
        c_dim = 3
        self.conv1 = nn.Conv2d(c_dim,96,11,4,padding=0) # valid padding
        self.pool1 = nn.MaxPool2d(3,2)
        self.conv2 = nn.Conv2d(96, 256, 5,padding=2) # same padding
        self.pool2 = nn.MaxPool2d(3,2)
        self.conv3 = nn.Conv2d(256,384,3,padding=1) # same padding
        self.conv4 = nn.Conv2d(384,384,3,padding=1) # same padding
        self.conv5 = nn.Conv2d(384,256,3,padding=1) # same padding
        self.pool3 = nn.MaxPool2d(3,2)
        self.flat_dim = 5*5*256 # replace with the actual value
        self.fc1 = nn.Sequential(*get_fc(self.flat_dim, 4096, 'relu'))
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Sequential(*get_fc(4096, 4096, 'relu'))
        self.dropout2 = nn.Dropout(p=0.5)
        self.fc3 = nn.Sequential(*get_fc(4096, 20, 'none'))

        self.nonlinear = lambda x: torch.clamp(x,0)

    def forward(self, x):
        N = x.size(0)
        x = self.conv1(x)
        x = self.nonlinear(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.nonlinear(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.nonlinear(x)
        x = self.conv4(x)
        x = self.nonlinear(x)
        x = self.conv5(x)
        x = self.nonlinear(x)
        x = self.pool3(x)
        x = x.view(N, self.flat_dim) # flatten the array

        out = self.fc1(x)
        out = self.nonlinear(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        out = self.nonlinear(out)
        out = self.dropout2(out)
        out = self.fc3(out)

        return out
```

In [ ]:
```
# output performance for each class

if __name__ == '__main__':
    modelCaffe = CaffeNet()
    torchLoadCaffe = torch.load('saved_models/CaffeNet-50.pth')
    modelCaffe.load_state_dict(torchLoadCaffe['model_state_dict'])
    modelCaffe = modelCaffe.to(args.device)
    modelCaffe.eval()

    test_loader = utils.get_data_loader('voc', train=False, batch_size=args.te
st_batch_size, split='test')
    ap, map = utils.eval_dataset_map(modelCaffe, args.device, test_loader)


    # output values from CaffeNet, compare with the target values
    classNames = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'ca
r',
                          'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse',
'motorbike',
                          'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvm
onitor']

    print("Accuracy Precision of CaffeNet Among Individual Classes: ")
    for i in range(len(classNames)):
        print("{}: {}".format(classNames[i],ap[i]))
```

Accuracy Precision of CaffeNet Among Individual Classes:
aeroplane: 0.6581140150639725
bicycle: 0.4561887044476487
bird: 0.35825532448861536
boat: 0.5089389243663608
bottle: 0.16395719924468152
bus: 0.3739061674684568
car: 0.6706960925944321
cat: 0.38321888683805233
chair: 0.39942472719290306
cow: 0.2317710070030363
diningtable: 0.29885085506558173
dog: 0.3274386268412313
horse: 0.6648294880873716
motorbike: 0.5684288971068936
person: 0.8086725636680325
pottedplant: 0.23318258506171002
sheep: 0.25812258618853845
sofa: 0.3496855596933779
train: 0.5634974187463156
tvmonitor: 0.36742932492025593

mAP:

0.4322304477043734

ResNet Class Performance Analysis

In [ ]:
```
if __name__ == '__main__':
    modelRes.eval()
    test_loader = utils.get_data_loader('voc', train=False, batch_size=args.te
st_batch_size, split='test')
    ap, map = utils.eval_dataset_map(modelRes, args.device, test_loader)


    # output values from CaffeNet, compare with the target values
    classNames = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'ca
r',
                          'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse',
'motorbike',
                          'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvm
onitor']

    print("Accuracy Precision of PreTrained ResNet Among Individual Classes: "
)
    for i in range(len(classNames)):
        print("{}: {}".format(classNames[i],ap[i]))
```

Accuracy Precision of PreTrained ResNet Among Individual Classes:
aeroplane: 0.7213222117720576
bicycle: 0.5694538600068458
bird: 0.4367706344884813
boat: 0.5773203950369964
bottle: 0.1818182675529317
bus: 0.5025989148818082
car: 0.7382160645945922
cat: 0.4581863265778226
chair: 0.43791839747272837
cow: 0.32130702694874635
diningtable: 0.4435263229389825
dog: 0.4043789661300173
horse: 0.7495443845590672
motorbike: 0.6876729041809619
person: 0.854338868879449
pottedplant: 0.2733504707389632
sheep: 0.4169879431619927
sofa: 0.3385033253895445
train: 0.6906721981250176
tvmonitor: 0.46132828158342576

mAP:

0.5132607882510216