In [1]:

```
import os, copy, re, time
import os.path as osp
import csv, math, ast, glob
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import DataLoader
from torch.utils.data import sampler

from sklearn.utils import shuffle
import torchvision.models as models
from torchvision import transforms, utils
from pyt_utils import *
%matplotlib inline
```

In [2]:

```
os.environ["CUDA_VISIBLE_DEVICES"] = "2, 3, 4, 5"
print(os.environ['CUDA_VISIBLE_DEVICES'])
```

```
2, 3, 4, 5
```

In [3]:

```
def get_data(data_dir, num_training=46000, num_validation=5000, num_test=4000, folde
    data_dir = data_dir + folder + '/'
    y_file = data_dir + 'coords.txt'
    num = num_training + num_validation + num_test
    Y = []
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    trn = transforms.Compose(
            [transforms.Resize([224, 224]),
            transforms.ToTensor(),
#             normalize
            ])
    with open(y_file, 'r') as f:
        lines = f.readlines()
        for i in range(num):
            line = lines[i]
            Y.append(ast.literal_eval(line.rstrip("\n")))
#         Y.pop()
    Y = np.asarray(Y)
    X = [trn(Image.open(data_dir + 'train%d.png' %i)) for i in range(num)]
    X, Y = shuffle(X, Y)
    X_train = X[0:num_training]
    Y_train = Y[0:num_training]
    X_val = X[num_training: num_training + num_validation]
    Y_val = Y[num_training: num_training + num_validation]
    X_test = X[-num_test : num]
    Y_test = Y[-num_test : num]

    return X_train, Y_train, X_val, Y_val, X_test, Y_test
```

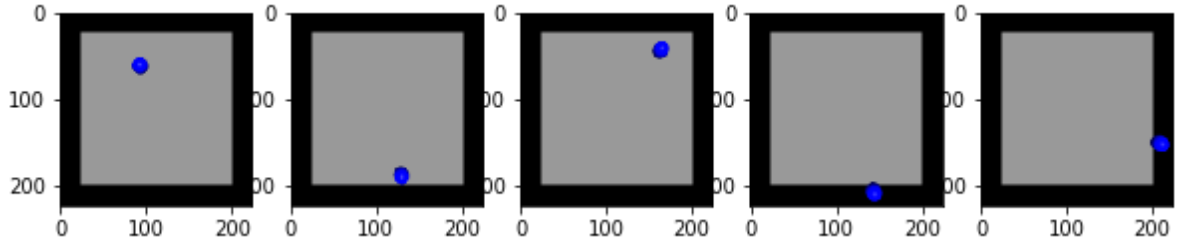
In [4]:

```
direc = '/home/shivanik/lab/code/Models/'
data_dir = osp.join(direc, 'Data256/')
result_dir = osp.join(direc, 'Results/')
folders = ['agent0_pic']
for folder in folders:
    X_train, Y_train, X_val, Y_val, X_test, Y_test = get_data(data_dir, folder=folde
folder = folders[0]
# img_size = X_train.shape[1:]
num_coords = Y_test.shape[-1]
# for d in [X_train, Y_train, X_val, Y_val, X_test, Y_test]:
#     print(d.shape)
```

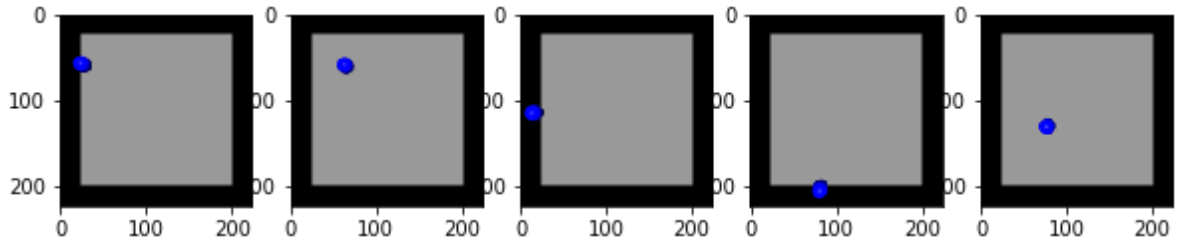
In [5]:

```
#Visualize some examples from the dataset
for X, Y in [(X_train, Y_train), (X_val, Y_val), (X_test, Y_test)]:
    f = plt.figure(figsize=(10,10))
    plt.tight_layout(pad=3, w_pad = 10)
    for y in range(1, 6):
        c = np.random.randint(len(X))
        plt.subplot(1, 5, y)
        trans = transforms.ToPILImage()
        d = trans(X[c])
        d = d.convert('RGB')
        plt.imshow(d)
        print(Y[c])
    plt.show()
```

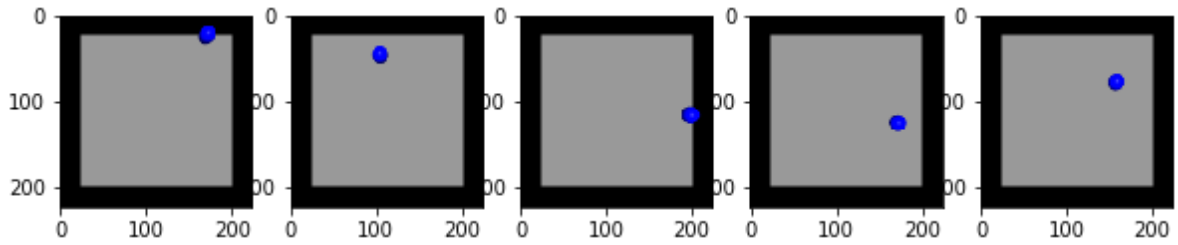
```
[-0.06027968  0.15751438]
[ 0.05098207 -0.24659687]
[0.16365945 0.21622837]
[ 0.10335389 -0.30756113]
[ 0.30704088 -0.12914991]
```



```
[-0.27468158  0.16821521]
[-0.15697922  0.16330204]
[-0.30752213 -0.01237432]
[-0.09636327 -0.29434625]
[-0.11168169 -0.06274364]
```



```
[0.18802663 0.28099924]
[-0.02766391  0.20712187]
[ 0.27281902 -0.01611649]
[ 0.19183976 -0.0460089 ]
[0.14382028 0.10582228]
```



In [6]:

```
train_data = []
for i in range(len(X_train)):
   train_data.append([X_train[i], Y_train[i]])

trainloader = torch.utils.data.DataLoader(train_data, \
                                          shuffle=True, batch_size=128)

val_data = []
for i in range(len(X_val)):
   val_data.append([X_val[i], Y_val[i]])
valloader = torch.utils.data.DataLoader(val_data, \
                                        shuffle=True, batch_size=128)

test_data = []
for i in range(len(X_test)):
   test_data.append([X_test[i], Y_test[i]])
testloader = torch.utils.data.DataLoader(test_data, \
                                        shuffle=True, batch_size=128)

data = {}
data['train'] = trainloader
data['eval'] = valloader
data['test'] = testloader
```

In [7]:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
torch.cuda.set_device(1)
print(device)
```

```
cuda:0
```

In [8]:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# with torch.cuda.device(0):
myresnet18 = Resnet18Coords().to(device)
alexnet = models.alexnet(pretrained=True).to(device)
vgg16 = models.vgg16(pretrained=True).to(device)
resnet18 = models.resnet18(pretrained=True).to(device)
inception = models.inception_v3(pretrained=True).to(device)
```

In [9]:

```
alexnet.classifier = nn.Sequential(*[alexnet.classifier[i] for i in range(4)] + [Fla
vgg16.classifier = nn.Sequential(*[vgg16.classifier[i] for i in range(3)] + [Flatter
```

In [10]:

```
model_random_xavier = nn.Sequential(
    nn.Conv2d(in_channels=3,out_channels=32,kernel_size=7,stride=4),
    nn.ReLU(inplace=True),
    nn.Conv2d(in_channels=32,out_channels=64,kernel_size=4,stride=2),
    nn.ReLU(inplace=True),
    nn.Conv2d(in_channels=64,out_channels=64,kernel_size=3,stride=1),
    nn.ReLU(inplace=True),
    Flatten(),
    )
model_random_xavier.to(device)
model_random_xavier.apply(weights_init_xavier)
model_random_xavier_t = copy.deepcopy(model_random_xavier)
```

In [11]:

```
# model_random_uniform = nn.Sequential(
#     nn.Conv2d(in_channels=3,out_channels=32,kernel_size=7,stride=4),
#     nn.ReLU(inplace=True),
#     nn.Conv2d(in_channels=32,out_channels=64,kernel_size=4,stride=2),
#     nn.ReLU(inplace=True),
#     nn.Conv2d(in_channels=64,out_channels=64,kernel_size=3,stride=1),
#     nn.ReLU(inplace=True),
#     Flatten(),
#     )
# model_random_uniform.cuda()
# model_random_uniform.apply(weights_init_xavier)
# model_random_uniform_t = copy.deepcopy(model_random_uniform)
```

In [12]:

```
MODELS = {
    "vgg16": vgg16,
    "alexnet": alexnet,
    "resnet18": myresnet18,
#     "random_uniform" : model_random_uniform,
    "random_xavier" : model_random_xavier,
#     "random_uniform_trainable" : model_random_uniform_t,
    "random_xavier_trainable" : model_random_xavier_t,
}

for name in MODELS.keys():
    if 'trainable' not in name:
        for param in MODELS[name].parameters():
            param.requires_grad = False
```

In [13]:

```
# from torchsummary import summary
# summary(MODELS['random_uniform'], (3, 224, 224))
out_shapes = {}
out_shapes['alexnet'] = 4096
out_shapes['vgg16'] = 4096
out_shapes['resnet18'] = 25088
for name in ['random_uniform', 'random_xavier', 'random_uniform_trainable', 'random_
    out_shapes[name] = 36864
```

In [14]:

```
class Mul(nn.Module):
    def __init__(self, const):
        super().__init__()
        self.const = const
    def forward(self, x):
        return x * self.const
```

In [15]:

```
for name, model in MODELS.items():
    model = nn.Sequential(
            model,
            nn.Linear(out_shapes[name],256),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Linear(256, 256),
            nn.LeakyReLU(0.1, inplace=True),
            nn.Linear(256, num_coords),
            nn.Tanh(),
            Mul(0.3)
            )
    model.cuda()
    MODELS[name] = model
```

In [16]:

```
gpu_dtype = torch.cuda.FloatTensor
thresh = 0.03
def train(model, loss_fn, optimizer, num_epochs = 5):
    since = time.time()
    print_every = 500
    train_loss_history = []
    val_acc_history = []

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch %d / %d' % (epoch + 1, num_epochs))

        for phase in ['train', 'eval', 'test']:
            running_loss = 0.0
            running_corrects = 0
            if phase == 'train':
                model.train()
            else:
                model.eval()

            for t, (x, y) in enumerate(data[phase]):
                x_var = Variable(x.type(gpu_dtype))
                y_var = Variable(y.type(gpu_dtype))

                y_pred = model(x_var)
                loss = loss_fn(y_pred, y_var)

                if 'train' in phase:
                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * x_var.size(0)
                running_corrects += torch.sum(torch.norm(y_pred - y_var, dim=1) < th

#                 if t == 10000:
#                     break

            epoch_loss = running_loss / len(data[phase].dataset)
            epoch_acc = running_corrects * 1.0 / len(data[phase].dataset)
            print('PHASE : %s  loss = %.4f, accuracy = %f' % (phase, epoch_loss, epo

            if phase == 'eval' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
            if phase == 'eval':
                val_acc_history.append(epoch_acc)
            else:
                train_loss_history.append(epoch_loss)


    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_ela
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
```

```
    return model, val_acc_history, train_loss_history
```

In [17]:

```
FINAL_MODELS = {}
lr = 1e-3
for name, model in MODELS.items():
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    num_epochs = 1 if 'trainable' in name else 3
    FINAL_MODELS[name] = train(model, loss_fn, optimizer, num_epochs=num_epochs)
```

```
Epoch 1 / 3
PHASE : train  loss = 0.0012, accuracy = 0.706652
PHASE : eval  loss = 0.0001, accuracy = 0.990200
PHASE : test  loss = 0.0001, accuracy = 0.994750
Epoch 2 / 3
PHASE : train  loss = 0.0001, accuracy = 0.943870
PHASE : eval  loss = 0.0001, accuracy = 0.987600
PHASE : test  loss = 0.0001, accuracy = 0.989000
Epoch 3 / 3
PHASE : train  loss = 0.0001, accuracy = 0.976630
PHASE : eval  loss = 0.0000, accuracy = 1.000000
PHASE : test  loss = 0.0000, accuracy = 1.000000
Training complete in 8m 9s
Best val Acc: 1.000000
Epoch 1 / 3
PHASE : train  loss = 0.0086, accuracy = 0.251826
PHASE : eval  loss = 0.0002, accuracy = 0.912800
PHASE : test  loss = 0.0002, accuracy = 0.917000
Epoch 2 / 3
PHASE : train  loss = 0.0009, accuracy = 0.457543
PHASE : eval  loss = 0.0002, accuracy = 0.930000
PHASE : test  loss = 0.0002, accuracy = 0.927750
Epoch 3 / 3
PHASE : train  loss = 0.0006, accuracy = 0.559348
PHASE : eval  loss = 0.0001, accuracy = 0.967800
PHASE : test  loss = 0.0001, accuracy = 0.969000
Training complete in 1m 18s
Best val Acc: 0.967800
Epoch 1 / 3
PHASE : train  loss = 0.0336, accuracy = 0.011413
PHASE : eval  loss = 0.0330, accuracy = 0.014200
PHASE : test  loss = 0.0325, accuracy = 0.011250
Epoch 2 / 3
PHASE : train  loss = 0.0325, accuracy = 0.011761
PHASE : eval  loss = 0.0319, accuracy = 0.014200
PHASE : test  loss = 0.0311, accuracy = 0.011500
Epoch 3 / 3
PHASE : train  loss = 0.0322, accuracy = 0.011783
PHASE : eval  loss = 0.0362, accuracy = 0.011800
PHASE : test  loss = 0.0352, accuracy = 0.010000
Training complete in 2m 12s
Best val Acc: 0.014200
Epoch 1 / 3
PHASE : train  loss = 0.0104, accuracy = 0.191152
PHASE : eval  loss = 0.0008, accuracy = 0.780200
PHASE : test  loss = 0.0008, accuracy = 0.776500
Epoch 2 / 3
PHASE : train  loss = 0.0012, accuracy = 0.458087
PHASE : eval  loss = 0.0008, accuracy = 0.446400
PHASE : test  loss = 0.0007, accuracy = 0.440500
```

```
Epoch 3 / 3
PHASE : train  loss = 0.0007, accuracy = 0.587065
PHASE : eval  loss = 0.0006, accuracy = 0.497200
PHASE : test  loss = 0.0006, accuracy = 0.505250
Training complete in 0m 57s
Best val Acc: 0.780200
Epoch 1 / 1
PHASE : train  loss = 0.0022, accuracy = 0.879848
PHASE : eval  loss = 0.0000, accuracy = 0.999800
PHASE : test  loss = 0.0000, accuracy = 0.999250
Training complete in 0m 22s
Best val Acc: 0.999800
```

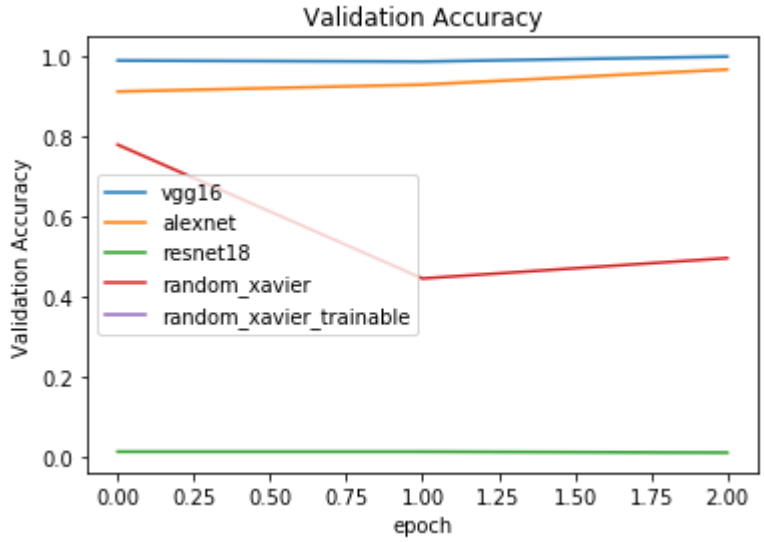
In [18]:

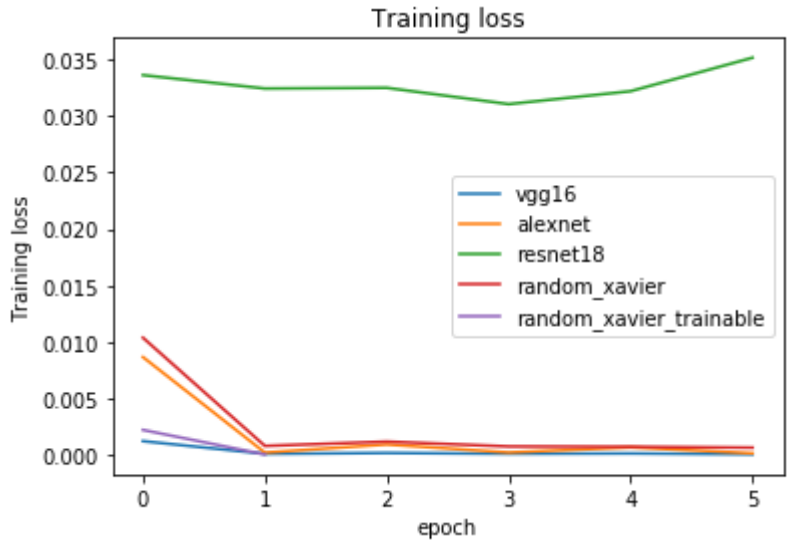
```
def make_dir(folder, cont):
    if not os.path.exists(folder):
        print('Making directory: {}'.format(folder))
        os.makedirs(folder)
        return folder
    elif cont != '' and not os.path.exists(cont):
        print('Making directory: {}'.format(cont))
        os.makedirs(cont)
        return cont
    else:
        print('Existing directory: {}'.format(folder))
        folder_name = folder.split('/')[-1]
        print(folder_name)
        count = sum([folder_name in name for name in os.listdir(result_dir)])
        cont = folder + "_" + str(count + 1)
        return make_dir(folder, cont)
```

In [19]:

```
rate = np.format_float_scientific(lr)
mkfolder = make_dir(result_dir + folder, '')
keys = ['Validation Accuracy', 'Training loss']
for i, key in enumerate(keys):
    for model in FINAL_MODELS.keys():
        history = FINAL_MODELS[model]
        plt.plot(history[1 + i], label = model)
        plt.title(key)
        plt.ylabel(key)
        plt.xlabel('epoch')
    plt.legend()
    plt.show()
    plt.savefig('%s/%s-%s.png' %(mkfolder, key, rate))
    plt.clf()
```

```
Making directory: /home/shivanik/lab/code/Models/Results/agent0_pic
```

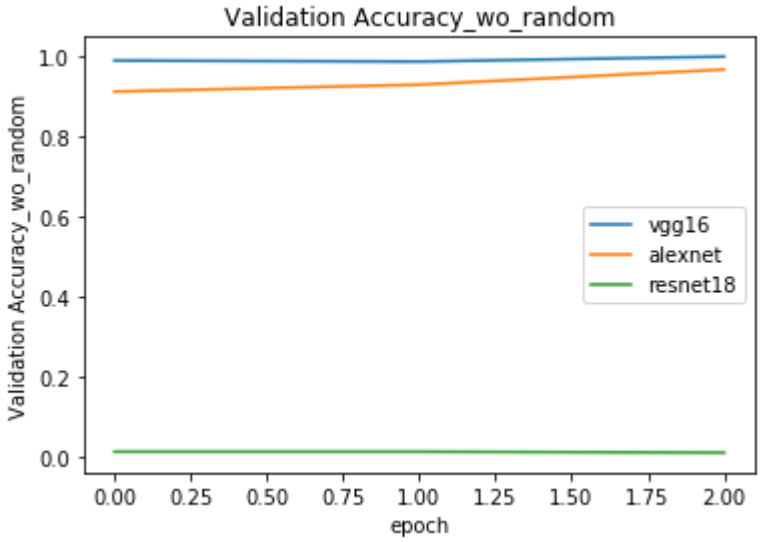


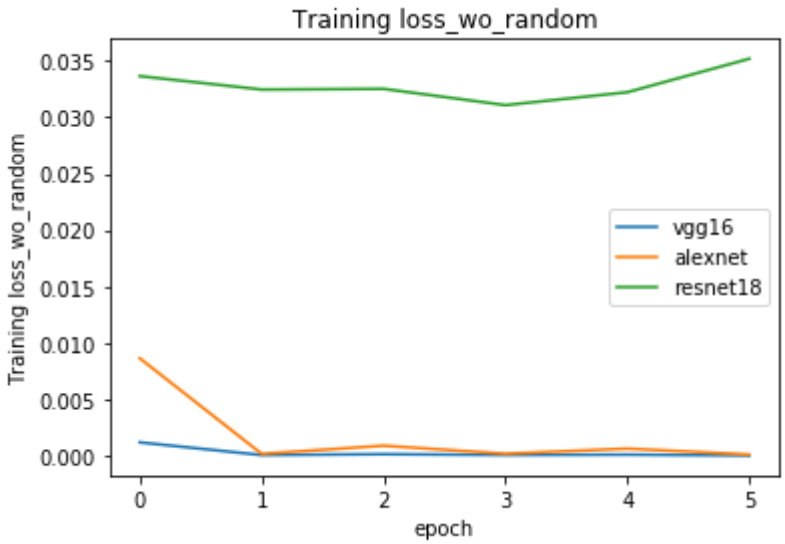



```
<Figure size 432x288 with 0 Axes>
```

In [20]:

```
rate = np.format_float_scientific(lr)
keys = ['Validation Accuracy_wo_random', 'Training loss_wo_random']
for i, key in enumerate(keys):
    for model in FINAL_MODELS.keys():
        if 'random_xavier' not in model:
            history = FINAL_MODELS[model]
            plt.plot(history[1 + i], label = model)
            plt.title(key)
            plt.ylabel(key)
            plt.xlabel('epoch')
    plt.legend()
    plt.show()
    plt.savefig('%s/%s-%s.png' %(mkfolder, key, rate))
    plt.clf()
```





```
<Figure size 432x288 with 0 Axes>
```

In [21]:

```
results = {}
thresh = 0.03
t = 0
for name, model in FINAL_MODELS.items():
    running_corrects = 0.0
    model = model[0]
    for x, y in data['test']:
        t += len(x)
        x_var = Variable(x.type(gpu_dtype))
        y_var = Variable(y.type(gpu_dtype))
        model.eval()
        y_pred = model(x_var)
        running_corrects += torch.sum(torch.norm(y_pred - y_var, dim=1) < thresh)
    acc = running_corrects * 1.0 / len(data['test'].dataset)
    results[name] = (acc.cpu().numpy(), y[0:5], y_pred[0:5])
```

In [22]:

```
results
```

Out[22]:

```
{'vgg16': (array(1., dtype=float32), tensor([[ 0.1886,  0.0748],
          [-0.2245,  0.2582],
          [-0.1814, -0.1798],
          [ 0.1770, -0.1620],
          [ 0.2567, -0.0289]], dtype=torch.float64), tensor([[ 0.1892,
0.0768],
          [-0.2286,  0.2655],
          [-0.1806, -0.1755],
          [ 0.1778, -0.1591],
          [ 0.2557, -0.0324]], device='cuda:1', grad_fn=<SliceBackward
>)),
 'alexnet': (array(0.96900004, dtype=float32), tensor([[ 0.0994, -0.20
86],
          [ 0.0601, -0.1571],
          [-0.1474,  0.2191],
          [-0.3071,  0.2433],
          [-0.0931,  0.1944]], dtype=torch.float64), tensor([[ 0.0882,
-0.2192],
          [ 0.0512, -0.1656],
          [-0.1452,  0.2149],
          [-0.2794,  0.2602],
          [-0.0766,  0.1972]], device='cuda:1', grad_fn=<SliceBackward
>)),
 'resnet18': (array(0.01125, dtype=float32), tensor([[-0.2142,  0.070
4],
          [-0.1549,  0.0627],
          [ 0.1835, -0.3101],
          [ 0.0314, -0.0373],
          [-0.1583,  0.0737]], dtype=torch.float64), tensor([[-0.3000,
0.3000],
          [-0.3000,  0.3000],
          [ 0.3000, -0.3000],
          [ 0.3000, -0.3000],
          [-0.3000,  0.3000]], device='cuda:1', grad_fn=<SliceBackward
>)),
 'random_xavier': (array(0.77650005, dtype=float32),
  tensor([[-0.0771,  0.0725],
          [-0.1242, -0.1256],
          [ 0.2084, -0.1855],
          [ 0.3073, -0.0386],
          [ 0.1564,  0.1459]], dtype=torch.float64),
  tensor([[-0.0815,  0.0946],
          [-0.1212, -0.1069],
          [ 0.2412, -0.2097],
          [ 0.2211, -0.0234],
          [ 0.1707,  0.1747]], device='cuda:1', grad_fn=<SliceBackward
>)),
 'random_xavier_trainable': (array(0.99925005, dtype=float32),
  tensor([[-0.1420, -0.1432],
          [ 0.2097,  0.0689],
          [ 0.0304,  0.0012],
          [ 0.2132,  0.2241],
          [-0.2148,  0.1572]], dtype=torch.float64),
  tensor([[-0.1385, -0.1393],
          [ 0.2133,  0.0657],
```

```
          [ 0.0356,  0.0004],
          [ 0.2148,  0.2225],
          [-0.2098,  0.1552]], device='cuda:1', grad_fn=<SliceBackward
>))}
```

In [23]:

```
t
```

Out[23]:

```
20000
```

In [24]:

```
torch.norm(y_pred - y_var, dim=1) < thresh
```

Out[24]:

```
tensor([True, True, True, True, True, True, True, True, True, True, Tr
ue, True,
        True, True, True, True, True, True, True, True, True, True, Tr
ue, True,
        True, True, True, True, True, True, True, True], device='cuda:
1')
```

In [25]:

```
file = csv.writer(open("%s/result_on_test.csv" %(mkfolder), "w"))
for key, val in results.items():
    file.writerow([key, val[0], val[1], val[2]])
```

In [26]:

```
# results_val = {}
# thresh = 0.03
# for name, model in FINAL_MODELS.items():
#     running_corrects = 0.0
#     model = model[0]
#     for x, y in data['eval']:
#         x_var = Variable(x.type(gpu_dtype))
#         y_var = Variable(y.type(gpu_dtype))
#         model.eval()
#         y_pred = model(x_var)
#         running_corrects += torch.sum(torch.norm(y_pred - y_var, dim=1) < thresh)
#     acc = running_corrects * 1.0 / len(data['eval'].dataset)
#     results_val[name] = (acc.cpu().numpy(), y[0:5], y_pred[0:5])
```

In [27]:

```
# results_val
```

In [28]:

```
# file = csv.writer(open("%s/result_on_test.csv" %(mkfolder), "w"))
# for key, val in results.items():
#     file.writerow([key, val[0], val[1], val[2]])
```