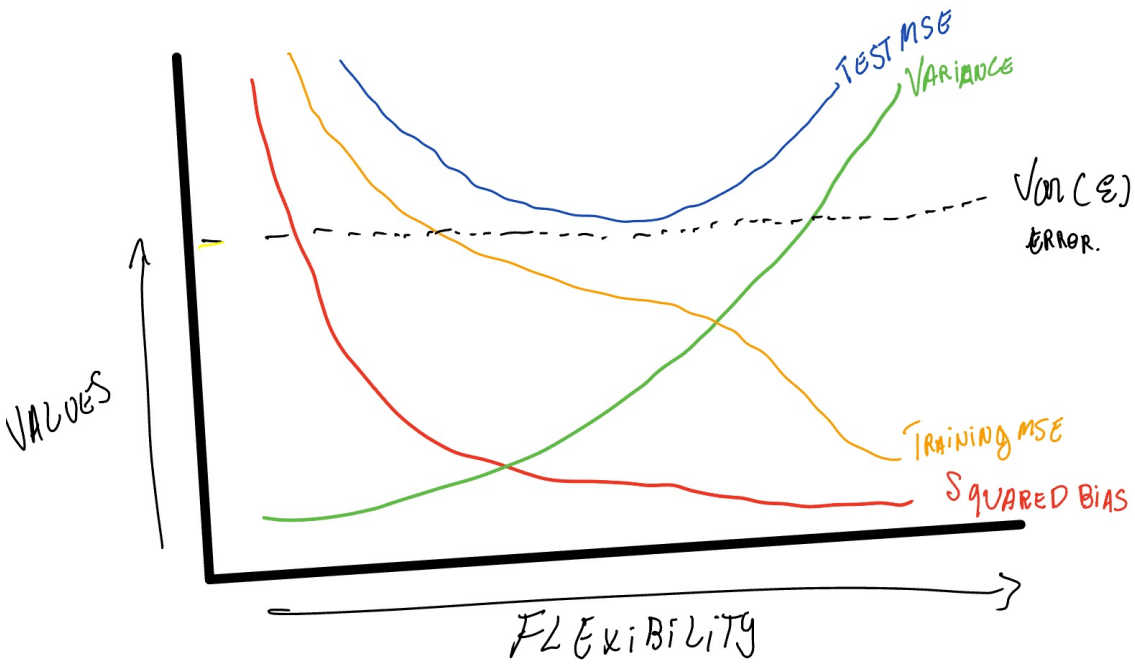
Problem 1 - 5 points

1. (2.5) Provide a sketch of typical (squared) bias, variance, training error, test error, andirreducible error curves, on a single plot, as model complexity increases. The x-axis shouldrepresent the model complexity, and the y-axis should represent the values for each curve.There should be five curves. Make sure to label each one.

Drawing attached



1. (2.5) Explain why each of the five curves has the shape displayed in part 1.

Flexibity. in the picture can also be swapped by the complexity. ***Answer:*** REview

The training MSE declines at a constant rate without oscilations the as flexibility increases, this is because as flexibility increases the f curve fits the observed data more closely.

The test MSE has a intial decline as the flexibility increases but at some point increases as the flexibility or complexity fo the model increase, this happes because we start to overfit the model and the model stops to predict correctly to the test set.

The squared bias decreases at a constant rate without oscilations and the variance increases at a constant rate without oscilations, in general as we use more complex methods, the variance will increase and the bias will decrease.

The irreducible error is a constant parallel to the test MSE curve because the expected test MSE will always be greater the Var(ε).

Problem 2 - 5 points

Table 1 in Figure-1 provides a training data set containing six observations, three features, and onetarget label. Suppose we wish to use this data set to make a prediction forYwhenX1=X2=X3= 0 usingK-nearest neighbors.

1. (2) Compute the Euclidean distance between each observation and the test point,X1 =X2 =X3 = 0.

Answer:

Distances:

obs1 = (0+3+0)/3 = 3

obs2 = (2+0+0)/3 = 2

obs3 = (0+1+3)/3 = 3.16

obs4 = (0+1+2)/3 = 2.23

obs5 = (-1 + 0 + 1) / 3 = 1.41

obs6 = (1+1+1)/3 = 1.73

1. (1.5) What is our prediction withK= 1 for this test point? Why?

Answer:

Since the testset X1=X2=X3=0, this is close to Green which is at a distance of sqrt(2)=1.414. Thus, the prediction will be Green.

1. (1.5) What is our prediction withK= 3? Why?

Answer:

Since the testset X1=X2=X3=0, closest ones are Red (Obs 2), Green(Obs 5) and Red(Obs 6). Thus, the Prediction will be Red

Problem 3 - 5 points

Suppose we collect data for a group of students in a statistics class with variablesX1= hours studied,X2=undergrad GPA, andY= receive an A. We fit a logistic regression and produce estimated coeffi-cient,β0=−6,β1= 0.05,β2= 1.

1. (1) Write the expression for probability of getting an A using the logistic function.

Answer:

$Log(\frac{P(x)}{1-P(x)})$ = $\beta_0 + \beta_1 + X_1 + \beta_2 + X_2$

1. (2) Estimate the probability that a student who studies for 40 h and has an undergrad GPAof 3.5 gets an A in the class.

Answer:

$Log(\frac{P(x)}{1-P(x)}) = -5 + 0.075 * 40 + 1.1 * 3.5 = 1.85$

And then we that the log on both sides to get

$\frac{P(x)}{1-P(x)} = 6.35982$

$\therefore$

P(X)=0.864

1. (2) How many hours would the student in part 2 need to study to have a 50% chance ofgetting an A in the class?

Answer:

0.5 = $\frac{e^{(-5+0.075*X_1+1.1*X_2)}}{1+e^{(-5+0.075*X_1+1.1*X_2)}}$

if $e^{(-5+0.075*X_1+1.1*X_2)}$ = 1

Then $-5 + 0.075 * X_1 + 1.1 * X_2$ = 0

X_2 = 3.5

X_1 = 15.3

Problem 4 - 5 points

1. (2) While learning rate is decayed using a learning rate schedule as training progresses, it needsto be done wisely. If the decay is too aggressive, it may prevent any learning. To understandthis an experiment was done in which four different decay schedules we used. Look at thecode in Figure 2 that was used to decay the learning rate in this experiment.

Running the code in Figure 2 creates a line plot showing learning rates over updates fordifferent decay values as shown in Figure 3.Figure 4 shows the training and test accuracy for the 4 training jobs, each with one of thefour learning rate schedules. Here each schedule is identified by the decay value. Each figureis labeled with an alphabet for the decay value.The four decay values are:0.1, 0.01, 0.001, 0.0001You need to match the decay value with the right alphabet. •A•B•C•D

Answer:

A - 0.0001 B- 0.1 C -0.001 D - 0.01

1. (3) An early stopping criterion is created using the generalization loss as: ″Stop as soon asthe generalization loss (GL) exceeds a certain threshold″, where GL (expressed in percentage)is defined as:GL(t) = 100 Eva(t)Eopt(t)−1 whereEva(t) is the validation error

at epocht, andEopt(t) is the minimum validation errortill epocht, which is defined as:Eopt(t) = mint′≤tEva(t′

From the validation error plot (Figure 5) identify the following:(a) Early stopping epoch for 28% threshold(b) Early stopping epoch for 150% threshold(c) Epoch at which overfitting is correctly detected(d) Epoch at which the model has the smallest generalization errorYour answers for the above should be in terms of epochs A, B, C, D, E, and F only. Theseepochs are marked in the validation plot.

Problem 5 - 5 points

Let there be four source datasets labeled A,B,C,D, each with an average feature representationgiven as a 4 dimensional vector:

A: [1,0,0,0] B: [0,1,0,0] C: [1,0,1,0] D: [0,1,0,1]

Consider the following approach for creating pseudolabels for an image:

•First take cosine similarity between the image's feature representation and the feature repre-sentation of each of the four datasets. Recall that the cosine similarity between two vectorsxandyis given byx•y∥x∥∥y∥, where•is for dot product between the two vectors.

•Concatenate the labels of nearestndatasets to create a pseudolabel for the image. Given twodatasets, the one with which the cosine similarity of image is higher, is nearer than the other.For example if for an image, the cosine similarity with these 4 datasets is [A: -0.2, B: 0.5, C:0.7, D: -0.3] then its pseudolabels under different Nearest-nschemes is as follows:

Nearest-1 :C Nearest-2 :CB Nearest-3 :CBA Nearest-4 :CBAD

Using the approach given above, find the pseudolabels for the following two images under the Nearest-1, Nearest-2, Nearest-3 and Nearest-4 schemes. The image feature vectors are given as:

1. Image X : [−1,1,−1,1]
2. Image Y : [1,−1,1,−1]

In [ ]:
```
import numpy as np
vector_a = [1,0,0,0]
vector_b = [0,1,0,0]
vector_c = [1,0,1,0]
vector_d = [0,1,0,1]
img_x = [-1,1,-1,1]
img_y = [1,-1,1,-1]

def cosine_similarity(x,y):
    return np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))

# similarities image x
print("For Image X")
print("A:",cosine_similarity(img_x,vector_a),
      "B:",cosine_similarity(img_x,vector_b),
      "C",cosine_similarity(img_x,vector_c),
      "D:",cosine_similarity(img_x,vector_d))

print("For Image Y")
print("A:",cosine_similarity(img_y,vector_a),
```

```
        "B:",cosine_similarity(img_y,vector_b),
        "C",cosine_similarity(img_y,vector_c),
        "D:",cosine_similarity(img_y,vector_d))
```

```
For Image X
A: -0.5 B: 0.5 C -0.7071067811865475 D: 0.7071067811865475
For Image Y
A: 0.5 B: -0.5 C 0.7071067811865475 D: -0.7071067811865475
```

Answer 5:

For X: Nearest-1:D Nearest-2:DB Nearest-3:DBA Nearest-4 :DBAC

For Y: Nearest-1:C Nearest-2:CA Nearest-3:CAB Nearest-4:CABD

*Problem 6 - 5 points

This question is based on LeNet architecture as presented in the following paper:LeCun et al. Gradient Based Learning Applied to Document Understanding.

1. (1) How many connections are there in the first convolutional layer?

Answer:

In the first layer:

Our Filter size are 5 x 5

The number of filters = 6

These filters have strides = S = 1

With Padding = 0

The output featuremap size is = 28 x 28

The number of neurons is equal = 28*28*6 = 4,704

number of learning parameters = (Weights + Bias)per filter *number of filters = (5* 5 + 1) * 6 = 156

Therefore number of connections = 156 *28* 28 = 1,22,304

1. (1) Why third convolutional layer is technically a fully connected layer?

Answer:

The third convolutional layer has 120 neuron units, and each of these unis is connected to (5 x 5) neighborhood on all 16 of the previous average pooling layer feature maps. Every unit of the third Conv layer is connected to all the feature maps of the previous average pooling layer and, thus the current conv layer is known as Fully Connected Convolution Layer.

1. (1) What type of layers are contributing the maximum number of trainable parameters?

Answer:

The first Fully connected layer with 120 neuron units is the layer with largest contributions to the total of trainable parameters.

1. (2) How did we get 1516 trainable parameters in the second convolution layer?

Answer:

As here we can see that input coming from the first average pooling layer has 6 layers and the output of the second conv layer has 16 layers. Therefore we can not have a 1-to-1 mapping between these layers. Therefore, each unit in each feature map on the second conv layer is connected to several (5 x 5) areas at identical locations in different subset of the previous avg. pooling layer feature maps.

There are different combinations of feature maps taken from this avg.pooling layer:

Type 1 - The first 6 convolution layers of the second conv layer are made with inputs from every contiguous subset of 3 feature maps from the previous avg.pooling.

Type 2 - The next 6 convolution layers of the second conv layer are made with inputs from every contiguous subset of 4 feature maps from the previous avg.pooling.

Type 3 - The next 3 layers of the second conv layer were made with inputs from the discontinuous subset of 4 feature maps from the previous avg.pooling layer.

Type 4 - The last layer of the second conv layer is made with all the feature maps.

Number of learning parameters = (Parameters in type-1) + (Parameters in n type-2) + (Parameters in type-3) + (Parameters in type-4) = [6 *(553 + 1)] + [6* (554 + 1)] + [3 *(554 + 1)] + [1* (556 + 1)]

= 456 + 606 + 303 + 151 = 1516

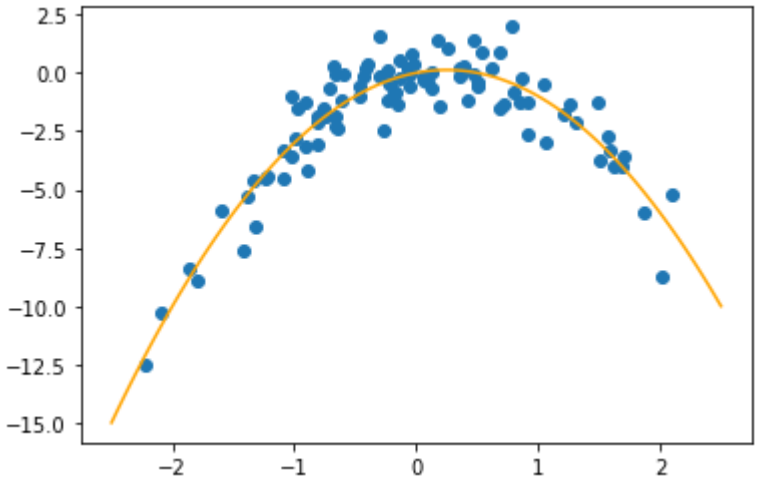
*Problem 7 - 10 points

Consider the case whenf(x) =x−2x2andy(x) =f(x) +N(0,1), whereN(0,1) is normaldistribution with mean 0 and standard deviation 1. Create a dataset of size 100 points by randomlysamplingxfromN(0,1) and then using the expression fory(x) to get the correspondingy.

1. (2) Display the dataset andf(x). Use scatter plot foryand smooth line plot forf(x).

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt


np.random.seed(1999)

def f_x(x):
    return x - 2*(x**2)

mu, sigma = 0, 1 # mean and standard deviation
s = np.random.normal(mu, sigma, 1)

data_x = []
```

```
data_y = []

for i in range(100):
    x = np.random.normal(mu, sigma, 1)
    data_x.append(x)
    data_y.append(f_x(x)+np.random.normal(mu, sigma, 1))

plt.scatter(x = data_x,y = data_y )

x_new = np.linspace(-2.5, 2.5, 100)
y_new = []

for i in x_new:
    y_new.append(f_x(i))

plt.plot(x_new,y_new,color = "orange")
plt.show()
```



1. (4) Set a random seed, and then compute the LOOCV (Leave One Out Cross Validation)errors that result from fitting the following four models using least squares:

(a)$Y=\beta0+\beta1X+\varepsilon$

(b)$Y=\beta0+\beta1X+\beta2X2+\varepsilon$

(c)$Y=\beta0+\beta1X+\beta2X2+\beta3X3+\varepsilon$

(d)$Y=\beta0+\beta1X+\beta2X2+\beta3X3+\beta4X4+\varepsilon$

In [ ]:

```
import pandas as pd
import numpy as np
import sklearn.linear_model as skl_lm
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import PolynomialFeatures
np.random.seed(1998)
loocv = LeaveOneOut()
lm = skl_lm.LinearRegression()

for i in range(1,5):
    poly = PolynomialFeatures(degree=i)
    X_current = poly.fit_transform(data_x)
    model = lm.fit(X_current, data_y)
    scores = cross_val_score(model, X_current, data_y, scoring="neg_mean_squa
 n_jobs=1)
```

```
    print("Degree-"+str(i)+" polynomial MSE: " + str(np.mean(np.abs(scores))))
```

```
Degree-1 polynomial MSE: 6.806724503963531, STD: 13.808798207245323
Degree-2 polynomial MSE: 1.0100562390793604, STD: 1.493097599848512
Degree-3 polynomial MSE: 1.0523654834251268, STD: 1.6273979196798125
Degree-4 polynomial MSE: 1.1119174013512805, STD: 1.8392423505177806
```

1. (2) Repeat 2 using another random seed, and report your results. Are your results the sameas what you got in 2 ? Why?

Answer:

Exact same, because LOOCV will be the same since it evaluates n folds of a single observation.

In [ ]:
```
np.random.seed(1997)
loocv = LeaveOneOut()
lm = skl_lm.LinearRegression()

for i in range(1,5):
    poly = PolynomialFeatures(degree=i)
    X_current = poly.fit_transform(data_x)
    model = lm.fit(X_current, data_y)
    scores = cross_val_score(model, X_current, data_y, scoring="neg_mean_squa
 n_jobs=1)

    print("Degree-"+str(i)+" polynomial MSE: " + str(np.mean(np.abs(scores))))
```

```
Degree-1 polynomial MSE: 6.806724503963531, STD: 13.808798207245323
Degree-2 polynomial MSE: 1.0100562390793604, STD: 1.493097599848512
Degree-3 polynomial MSE: 1.0523654834251268, STD: 1.6273979196798125
Degree-4 polynomial MSE: 1.1119174013512805, STD: 1.8392423505177806
```

1. (2) Which of the models in part 2 had the smallest LOOCV error? Is this what you expected?Explain your answer.

Answer:

The quatratic model and yes I expected that because the true data is of quatratic form.

*Problem 8 - 10 points

You will create a Neural Network in PyTorch to classify a dataset of images. The dataset we aregoing to use is CIFAR10, which contains 50K 32x32 color images. The model we are going tobuild is ResNet-18, as described in https://arxiv.org/abs/1512.03385 . The reference code is athttps://github.com/kuangliu/pytorch-cifar.

1. (4) Create a PyTorch program with a DataLoader that loads the images and the related labelsfrom the torchvision CIFAR10 dataset. Import CIFAR10 dataset for the torchvision package,with the following sequence of transformations:

(a) Random cropping, with size 32x32 and padding 4

(b) Random horizontal flipping with a probability 0.5

(c) Normalize each image's RGB channel with mean(0.4914, 0.4822, 0.4465) and variance(0.2023, 0.1994, 0.2010)

In [ ]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

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

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=100, shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

The DataLoader for the training set uses a minibatch size of 128 and numworkers=2. TheDataLoader for the testing set uses minibatch size of 100 and numworkers =2. You can referto: https://pytorch.org/tutorials/beginner/blitz/cifar10tutorial.html

1. (4) Create a main function that creates the DataLoaders for the training set and the neuralnetwork, then run 5 epochs with a complete training phase on all the minibatches of thetraining set.

In [ ]:
```
net = torchvision.models.resnet18(pretrained=True)
net = net.cuda() if device else net
net
```

Out[ ]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bi
as=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mo
de=False)
  (layer1): Sequential(
    (0): BasicBlock(
```

```
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      )
    )
    (1): BasicBlock(
```

```
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [ ]:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)

def accuracy(out, labels):
    _,pred = torch.max(out, dim=1)
    return torch.sum(pred==labels).item()

num_ftrs = net.fc.in_features
net.fc = nn.Linear(num_ftrs, 128)
```

In [ ]:

```
import argparse
parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Training')
parser.add_argument('--lr', default=0.1, type=float, help='learning rate')
parser.add_argument('--resume', '-r', action='store_true',
                    help='resume from checkpoint')
args = parser.parse_args()

device = 'cuda' if torch.cuda.is_available() else 'cpu'
best_acc = 0  # best test accuracy
start_epoch = 0

net = torchvision.models.resnet18(pretrained=True)
net = net.to(device)
```

```
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True

if args.resume:
    # Load checkpoint.
    print('==> Resuming from checkpoint..')
    assert os.path.isdir('checkpoint'), 'Error: no checkpoint directory found
    checkpoint = torch.load('./checkpoint/ckpt.pth')
    net.load_state_dict(checkpoint['net'])
    best_acc = checkpoint['acc']
    start_epoch = checkpoint['epoch']

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=args.lr,
                      momentum=0.9, weight_decay=5e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)


# Training
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

        progress_bar(batch_idx, len(trainloader), 'Loss: %.3f | Acc: %.3f%% (
                     % (train_loss/(batch_idx+1), 100.*correct/total, correct


def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

            progress_bar(batch_idx, len(testloader), 'Loss: %.3f | Acc: %.3f%
                         % (test_loss/(batch_idx+1), 100.*correct/total, corr

    # Save checkpoint.
```

```
    acc = 100.*correct/total
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, './checkpoint/ckpt.pth')
        best_acc = acc


for epoch in range(start_epoch, start_epoch+200):
    train(epoch)
    test(epoch)
    scheduler.step()
```

```
usage: ipykernel_launcher.py [-h] [--lr LR] [--resume]
ipykernel_launcher.py: error: unrecognized arguments: -f /root/.local/share/ju
pyter/runtime/kernel-b718b451-6090-4f37-bd43-3e39812c80d9.json
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:3334:
UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

1. (2) For each minibatch calculate the training loss value, the training accuracy of the predic-tions, measured on training data.

*Problem 9 - 10 points

Using the VGG19 as pretrained model (pretrained on Imagenet1K) architecture build a customclassifier for MNIST handwritten digits classification by following two approaches:

https://www.kaggle.com/code/muerbingsha/mnist-vgg19/notebook

and lab 7

1. (4) Use VGG19 as a feature extractor and send them to dense layers which are trained according to our data set.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
IMG_SIZE = 32
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
import cv2

def resize(img_array):
    tmp = np.empty((img_array.shape[0], IMG_SIZE, IMG_SIZE))

    for i in range(len(img_array)):
        img = img_array[i].reshape(28, 28).astype('uint8')
```

```
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        img = img.astype('float32')/255
        tmp[i] = img

    return tmp

train_x_resize = resize(x_train)
test_x_resize = resize(x_test)

train_x_final = np.stack((train_x_resize,)*3, axis=-1)
test_x_final = np.stack((test_x_resize,)*3, axis=-1)
print(train_x_final.shape)
print(test_x_final.shape)

from keras.utils import to_categorical
train_y_final = to_categorical(y_train, num_classes=10)
print(train_y_final.shape)
```

```
(60000, 32, 32, 3)
(10000, 32, 32, 3)
(60000, 10)
```

In [ ]:

```
from keras.models import Sequential
from keras.applications import VGG19
from keras.layers import Dense, Flatten

vgg19 = VGG19(weights = 'imagenet',
              include_top = False,
              input_shape=(IMG_SIZE, IMG_SIZE, 3)
              )

model = Sequential()
model.add(vgg19)
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
base_learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten_2 (Flatten)         (None, 512)               0

 dense_6 (Dense)             (None, 10)                5130

=================================================================
Total params: 20,029,514
Trainable params: 20,029,514
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train_x_final, train_y_fi
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(48000, 32, 32, 3)
```

```
(12000, 32, 32, 3)
(48000, 10)
(12000, 10)
```

In [ ]:
```
from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_acc', verbose=1, patience=5)
mc = ModelCheckpoint(filepath='mnist-vgg19.h5', verbose=1, monitor='val_acc')
cb = [es, mc]
```

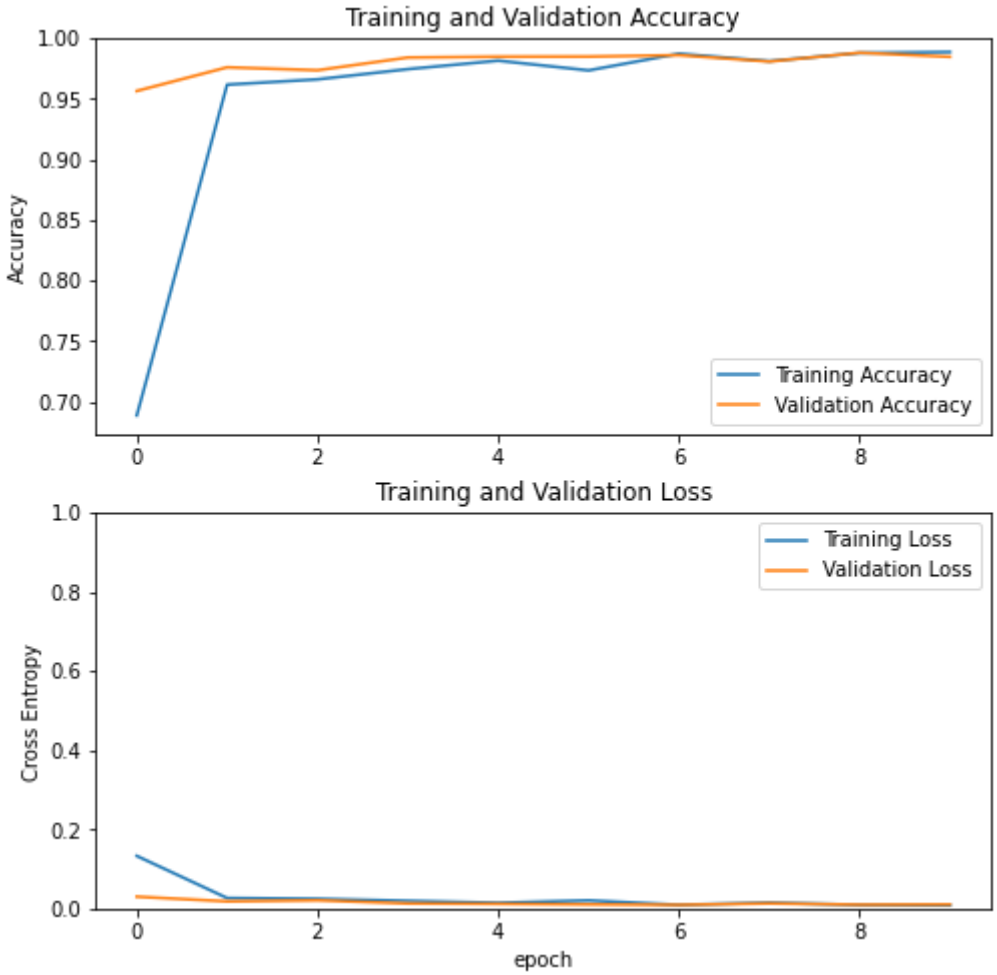
In [ ]:
```
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_test, y_test),
                    callbacks=cb)
```

```
Epoch 1/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.1330 - accurac
y: 0.6887
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 1: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 61s 39ms/step - loss: 0.1329 - ac
curacy: 0.6889 - val_loss: 0.0299 - val_accuracy: 0.9563
Epoch 2/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0264 - accurac
y: 0.9615
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 2: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 59s 40ms/step - loss: 0.0264 - ac
curacy: 0.9615 - val_loss: 0.0181 - val_accuracy: 0.9758
Epoch 3/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0244 - accurac
y: 0.9658
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 3: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 60s 40ms/step - loss: 0.0244 - ac
curacy: 0.9658 - val_loss: 0.0207 - val_accuracy: 0.9733
Epoch 4/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0193 - accurac
y: 0.9742
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 4: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 59s 39ms/step - loss: 0.0193 - ac
curacy: 0.9743 - val_loss: 0.0128 - val_accuracy: 0.9838
Epoch 5/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0143 - accurac
y: 0.9812
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 5: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 60s 40ms/step - loss: 0.0143 - ac
curacy: 0.9812 - val_loss: 0.0118 - val_accuracy: 0.9845
Epoch 6/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0204 - accurac
y: 0.9732
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 6: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 62s 41ms/step - loss: 0.0204 - ac
curacy: 0.9732 - val_loss: 0.0104 - val_accuracy: 0.9846
Epoch 7/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0093 - accurac
y: 0.9870
```

```
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 7: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 61s 40ms/step - loss: 0.0093 - ac
curacy: 0.9870 - val_loss: 0.0091 - val_accuracy: 0.9858
Epoch 8/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0146 - accurac
y: 0.9808
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 8: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 60s 40ms/step - loss: 0.0146 - ac
curacy: 0.9808 - val_loss: 0.0129 - val_accuracy: 0.9805
Epoch 9/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0092 - accurac
y: 0.9876
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 9: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 59s 39ms/step - loss: 0.0092 - ac
curacy: 0.9876 - val_loss: 0.0096 - val_accuracy: 0.9877
Epoch 10/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0081 - accurac
y: 0.9884
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 10: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 58s 39ms/step - loss: 0.0081 - ac
curacy: 0.9884 - val_loss: 0.0102 - val_accuracy: 0.9844
```

In [ ]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



1. (4) Freeze the weights of first 10 layers of the VGG19 network, while retrain the subsequentlayers.

In [ ]:
```
vgg19 = VGG19(weights = 'imagenet',
              include_top = False,
              input_shape=(IMG_SIZE, IMG_SIZE, 3)
              )

vgg19.trainable = False
model = Sequential()
model.add(vgg19)
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

base_learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten_1 (Flatten)         (None, 512)               0

 dense_5 (Dense)             (None, 10)                5130

=================================================================
```

```
Total params: 20,029,514
Trainable params: 5,130
Non-trainable params: 20,024,384
_________________________________________________________________
```

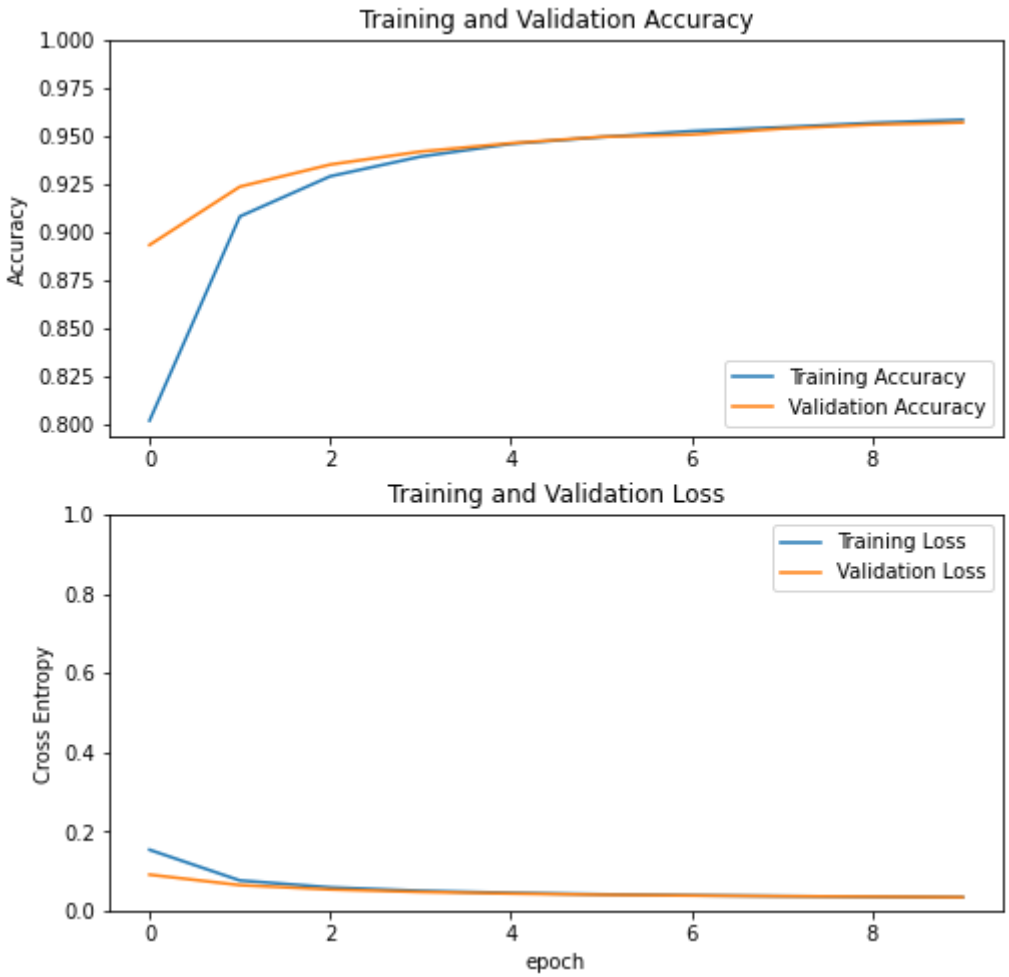
In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train_x_final, train_y_fi
from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_acc', verbose=1, patience=5)
mc = ModelCheckpoint(filepath='mnist-vgg19.h5', verbose=1, monitor='val_acc')
cb = [es, mc]
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_test, y_test),
                    callbacks=cb)
```

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:108
2: UserWarning: "`binary_crossentropy` received `from_logits=True`, but the `o
utput` argument was produced by a sigmoid or softmax activation and thus does
not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
1496/1500 [============================>.] - ETA: 0s - loss: 0.1536 - accurac
y: 0.8017
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 1: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 25s 15ms/step - loss: 0.1534 - ac
curacy: 0.8020 - val_loss: 0.0906 - val_accuracy: 0.8933
Epoch 2/10
1496/1500 [============================>.] - ETA: 0s - loss: 0.0759 - accurac
y: 0.9082
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 2: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 22s 15ms/step - loss: 0.0759 - ac
curacy: 0.9082 - val_loss: 0.0642 - val_accuracy: 0.9237
Epoch 3/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0584 - accurac
y: 0.9291
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 3: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 21s 14ms/step - loss: 0.0584 - ac
curacy: 0.9291 - val_loss: 0.0532 - val_accuracy: 0.9352
Epoch 4/10
1499/1500 [============================>.] - ETA: 0s - loss: 0.0498 - accurac
y: 0.9393
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 4: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 22s 15ms/step - loss: 0.0498 - ac
curacy: 0.9393 - val_loss: 0.0469 - val_accuracy: 0.9419
Epoch 5/10
1496/1500 [============================>.] - ETA: 0s - loss: 0.0445 - accurac
y: 0.9459
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 5: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 21s 14ms/step - loss: 0.0445 - ac
curacy: 0.9459 - val_loss: 0.0428 - val_accuracy: 0.9463
Epoch 6/10
1496/1500 [============================>.] - ETA: 0s - loss: 0.0409 - accurac
y: 0.9495
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
```

```
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 6: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 22s 15ms/step - loss: 0.0409 - ac
curacy: 0.9494 - val_loss: 0.0398 - val_accuracy: 0.9496
Epoch 7/10
1498/1500 [============================>.] - ETA: 0s - loss: 0.0383 - accurac
y: 0.9526
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 7: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 21s 14ms/step - loss: 0.0383 - ac
curacy: 0.9525 - val_loss: 0.0379 - val_accuracy: 0.9507
Epoch 8/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0363 - accurac
y: 0.9546
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 8: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 22s 15ms/step - loss: 0.0363 - ac
curacy: 0.9546 - val_loss: 0.0361 - val_accuracy: 0.9538
Epoch 9/10
1500/1500 [==============================] - ETA: 0s - loss: 0.0347 - accurac
y: 0.9568
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 9: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 21s 14ms/step - loss: 0.0347 - ac
curacy: 0.9568 - val_loss: 0.0350 - val_accuracy: 0.9559
Epoch 10/10
1498/1500 [============================>.] - ETA: 0s - loss: 0.0333 - accurac
y: 0.9584
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
Epoch 10: saving model to mnist-vgg19.h5
1500/1500 [==============================] - 21s 14ms/step - loss: 0.0333 - ac
curacy: 0.9584 - val_loss: 0.0337 - val_accuracy: 0.9570
```

In [ ]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



1. (2) Which approach gives a better transferability and why? ***Answer:***

The first approach using the feature extractor. Because it allows us to continue training the weights and adapting it better to the current data set we are predicting.

By freezing the weights, our model does not learn the deep features of our dataset.

Train for 10 epochs and use a learning rate of 0.001 with Adam optimizer with a batch size of 32.You can create models either in Tensorflow or Pytorch

Problem 10 - 10 points

1. (5) Refer to VGG16 architecture on page 3 in Table 1 of the paper by Simonyan et al availableat https://arxiv.org/pdf/1409.1556.pdf. You need to complete Table 1 below for calculatingactivation units and parameters at each layer in VGG16 (without counting biases).

Layers	Number of activiations	Parameters
Input	224 *224* 3=150K	0
CONV3-64	224 *224*64=3.2M	(3 *3* 3)*64 =1792
CONV3-64	224 *224* 64=3.2M	(3 *3* 64)*64=36928
POOL2	112 *112* 64=800K	0
CONV3-128	112 *112* 128=1.6M	(3 *3* 64)*128 =73856

Layers	Number of activiations	Parameters
CONV3-128	112 *112* 128=1.6M	(3 *3* 128)*128 =147584
POOL2	56 *56* 128=400K	0
CONV3-256	56 *56* 256=800K	(3 *3* 128)*256 295168
CONV3-256	56 *56* 256=800K	(3 *3* 256)*256 =590080
CONV3-256	56 *56* 256=800K	(3 *3* 256)*256 =590080
POOL2	28 *28* 256=200K	0
CONV3-512	28 *28* 512=400K	(3 *3* 256)*512=1180160
CONV3-512	28 *28* 512=400K	(3 *3* 512)*512=2359808
CONV3-512	28 *28* 512=400K	(3 *3* 512)*512 =2359808
POOL2	14 *14* 512=100K	0
CONV3-512	14 *14* 512=100K	(3 *3* 512)*512 =2359808
CONV3-512	14 *14* 512=100K	(3 *3* 512)*512 =2359808
CONV3-512	14 *14* 512=100K	(3 *3* 512)*512 =2359808
POOL2	7 *7* 512=25K	0
FC	4096	7 *7* 512* 4096 = 102764544
FC	4096	4096 *4096 = 16,777,216
FC	1000	4096 *1000 = 4097000
Total		138,357,544

1. The original Googlenet paper by Szegedy et al. (available at https://arxiv.org/pdf/1409.4842.pdf)proposes two architectures for Inception module, shown in Figure 2 on page 5 of the paper,referred to as naive and dimensionality reduction respectively.

(a) (2.5) Assuming the input to inception module (referred to as ”previous layer” in Figure2 of the paper) has size 32x32x256, calculate the output size after filter concatenationfor the naive and dimensionality reduction inception architectures with number of filtersgiven in Figure 6.

Answer:

Naive inception:

1x1 convolutions, 128 filters

3x3 convolutions, 192 filters

5x5 convolutions, 96 filters

3x3 max pooling,256

= 32x32x(128 + 192 + 96 +256) = 672 output

Dimentionality reduction inception:

1x1 convolutions , 128 filters

1x1 convolutions 64 filters + 3x3 convolutions ,192 filters

1x1 convolutions , 64 filters + 5x5 convolutions , 96 filters

3x3 max pooling, 256 filters + 1x1 convolutions ,64 filters

= 32x32x480 output

(b) (2.5) Next calculate the total number of convolutional operations for each of the twoinception architecture again assuming the input to the module has dimensions 32x32x256and number of filters given in Figure 6.

Answer:

Naive inception:

1x1 convolutions, 128 filters

computation = (32 *32)* 128 *(1* 1) *256 = 33,554,432

3x3 convolutions, 192 filters computation = (32 *32)* 192 *(3* 3) *256 = 452,884,832

5x5 convolutions, 96 filters computation = (32 *32)* 96 *(5* 5) *256 = 629,145,600

3x3 max pooling,256 filters

computation = 0

Total = 1.11 Billion computations

Dimentionality reduction inception:

1x1 convolutions , 128 filters Computations= (32 *32)* 128 *(1* 1) *256 = 33,554,432

1x1 convolutions 64 filters + 3x3 convolutions ,192 filters Computations= (32 *32)* 64 *(1* 1)*256

+ (32 *32)* 192 *(3* 3) * 64 = 130,023,424

1x1 convolutions , 64 filters + 5x5 convolutions , 96 filters

Computations= (32 *32)* 64 *(1* 1)*256

+ (32 *32)* 96 *(5* 5) * 64 = 174,063,616

3x3 max pooling , 256 filters+ 1x1 convolutions ,64 filters

Computations= (32 *32)* 64 *(1* 1) *64 = 4,194,304 total computation = 341,835,776

*Problem 11 - 10 points

This question focuses on batch normalization.

1. (2) Which of the following statements are true about batch normalization? (Select all thatapply.)

(a) Batch normalization makes processing a single batch faster, reducing the training timewhile keeping the number of updates fixed. This allows the network to spend the sameamount of time performing more updates to reach the minima.

Answer:

False, instead adding a batch normalization layer as extra processing time to the training iteration, making it slower. Plus the extra hyperparameters of the batch norm layer will require training duing the back propagation.

(b) Batch normalization weakens the coupling between earlier/later layers, which allows for independent learning.

Answer:

True, it will be normalizing the input to each next layer with batch normalization, this helps with loosening the coupling in between layers and allows the learning process more independent.

(c) Batch normalization normalizes the output distribution to be more uniform across dimensions.

Answer:

True. The purpose of batch normalization is to normalize each layer's inputs which are going to be the outputs of the layer before it given the mean and variance of the values in the current mini-batch.

(d) Batch normalization mitigates the effects of poor weight initialization and allows the network to initialize our weights to smaller values close to zero.

Answer:

False. The Batch normalization does mitigate the effects of poor weight initialization, so this part is true. But the second part of the statement is not true, because we are limiting the consequences of bad initialization, the batch norm does not activily determine weights to have a certain range.

1. (4) In the code provided in Figure 7, complete the implementation of batch normalization'sforward propagation in numpy code. The following formulas may be helpful:$z(i)norm=z(i)-\mu\sqrt{\sigma 2+\varepsilon}\tilde{z}(i)=\gamma z(i)norm+\beta$

In [ ]:
```
def forward_batchnorm(Z, gamma, beta, eps, cache_dict, beta_avg, mode):
  out = None
  if mode == 'train':
    mu = np.mean(X, axis=0)
    var = np.var(X, axis=0)
    cache_dict['mu'] = beta_avg * cache_dict['mu'] + \(1 - beta_avg) * mu
    cache_dict['var'] = beta_avg * cache_dict['var'] + \(1 - beta_avg) * var
  elif mode == 'test':
    mu = cache_dict['mu']
```

```
    var = cache_dict['var']
Z_norm = (X - mu) / np.sqrt(var + eps)
out = gamma * Z_norm + beta
return out
```

1. (1) What is the role of the ε hyperparameter in batch normalization?

Answer: The ϵ is a hyperparameter that helps prevent the division by 0 for features that have varience equal 0.

1. (1) What problem does using batch normalization have with a batch size of 1?

Answer:

If we have a minibatch size of 1, the output from the Bacth Norm layer will always be 0 because we only have one sample subtracted by the mean of that sample which is itself, and we multipy that by gamma, which will be 0 which subsequent we add to beta which is bias, but we initalize bias to 0, so there wont be any meaninful learning.

Applying Batch normal over a small amount of sample doenst makes sense. We will have a noisy mean and varience during training because we just have a few samples of the population. And the same will happen during test.

1. (2) You are applying batch normalization to a fully connected (dense) layer with an input size of 20 and output size of 30. How many training parameters does this layer have, including batch normalization parameters?

Answer:

There are 20 × 30 = 600 weights in the dense layer the bias des not enter this calculations.

Gamma and beta are scalars for each input therefore we have 30 gammas and 30 betas. We will have a total of 660 parameters.

Problem 12 - 20 points

Assume you have an input of size H×W×C. You need to create a 2-D convolution layer with a receptive field of 7×7. The output size should be alsoH×W×C.

1. (10) Provide five different architectures that can achieve this.Hint: Remember receptive fieldequivalence of stacked convolution layers and power of small filters.

Answer:

architecture 1:

ConvNet1 = 2x2 kernel size

ConvNet2 = 6x6 kernel size

architecture 2:

ConvNet1 = 2x2 kernel size

ConvNet2 = 5x5 kernel size

ConvNet3 = 2x2 kernel size

architecture 3:

ConvNet1 = 2x2 kernel size

ConvNet2 = 6x6 kernel size

ConvNet3 = 1x1 kernel size

architecture 4:

ConvNet1 = 1x1 kernel size

ConvNet2 = 2x2 kernel size

ConvNet3 = 3x3 kernel size

ConvNet4 = 4x4 kernel size

architecture 5:

ConvNet1 = 2x2 kernel size

ConvNet2 = 3x3 kernel size

ConvNet3 = 3x3 kernel size

ConvNet4 = 2x2 kernel size

ConvNet5 = 1x1 kernel size

1. (5) Calculate the total number of trainable parameters for each of the five architectures.Which has the most number of trainable parameters and which has the least?

Assuming that the input are of depth 1 and the conv layers have 1 filter.

architecture 1:

ConvNet1 = 2* 2 + 1 = 5

ConvNet2 = 6 * 6 +1 = 37

Total = 42

architecture 2:

ConvNet1 = 2* 2 + 1 = 5

ConvNet2 = 5 * 5 +1 = 26

ConvNet1 = 2 * 2 +1 = 5

Total = 36

architecture 3:

ConvNet1 = 2* 2 + 1 = 5

ConvNet2 = 6 * 6 +1 = 37

ConvNet1 = 1 * 1 +1 = 2

Total = 44

architecture 4:

ConvNet1 = 1 * 1 +1 = 2

ConvNet2 = 2 * 2 +1 = 5

ConvNet3 = 3 * 3 +1 = 10

ConvNet4 = 4 * 4 +1 = 17

Total = 34

architecture 5:

ConvNet1 = 2 *2 +1 = 5

ConvNet2 = 3 * 3 +1 = 10

ConvNet3 = 3 * 3 +1 = 10

ConvNet4 = 2 *2 +1 = 5

ConvNet5 =1 * 1 +1 = 2

Total = 32

1. (5) Compute the total computations required for each of the five architectures. Which has the largest computational requirement and which has the smallest?

largest comutational requirement is in my architecture number 3

smallest computational cost is on my architecture number 5

With an input of size 15 x 15 x 1

still with 1 filter per conv layer.

architecture 1:

ConvNet1 = (2 *2)* (15 *15)* 1* 1 = 900

ConvNet2 =(6 *6)* (15 *15)* 1* 1 = 8100

total = 9000

architecture 2:

ConvNet1 = 2 *2 (15* 15) *1* 1 = 900

ConvNet2 = 5 *5* (15 *15)* 1* 1 = 5625

ConvNet1 = 2 *2* (15 *15)* 1* 1 = 900

total = 7425

architecture 3:

ConvNet1 = 2 *2* (15 *15)* 1* 1 = 900

ConvNet2 = 6 *6* (15 *15)* 1* 1 = 8100

ConvNet1 = 1 *1* (15 *15)* 1* 1 = 225

Total = 9225

architecture 4:

ConvNet1 = 1 *1* (15 *15)* 1* 1 = 225

ConvNet2 = 2 *2* (15 *15)* 1* 1 = 900

ConvNet3 = 3 *3* (15 *15)* 1* 1 = 2025

ConvNet4 = 4 *4* (15 *15)* 1* 1 = 3600

Total = 6750

architecture 5:

ConvNet1 = 2 *2* (15 *15)* 1* 1= 900

ConvNet2 = 3 *3* (15 *15)* 1* 1= 2025

ConvNet3 = 3 *3* (15 *15)* 1* 1 = 2025

ConvNet4 = 2 *2* (15 *15)* 1* 1= 900

ConvNet5 =1 *1* (15 *15)* 1* 1 = 225

total = 6075