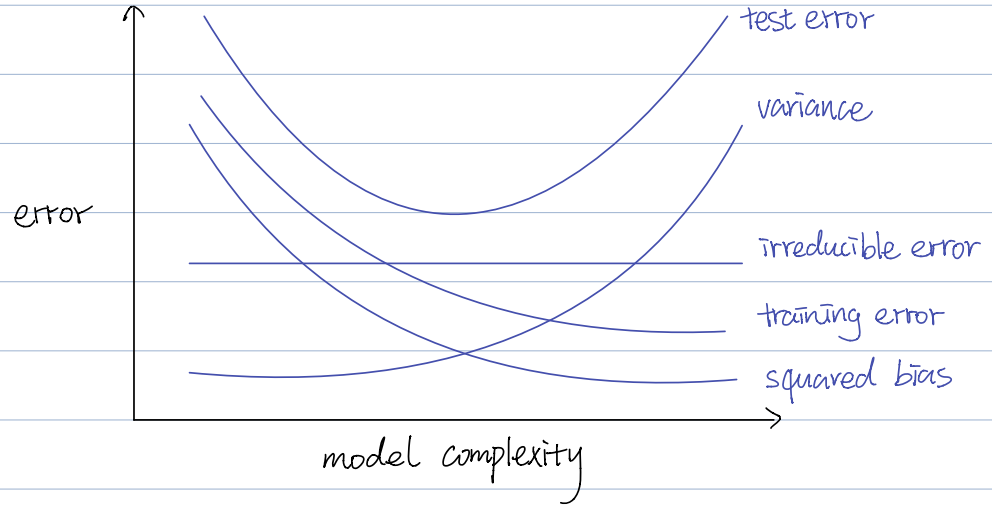
Problem 1

Q1.



Q2.

① squared bias: It shows whether the model's predictions predict the real model well. Model with low complexity has high bias, but model with high complexity has low bias, because a more complex model represents better the real-problem, so the squared bias has a decreasing curve.

② variance: It is low for not complex model, and it starts to increase as the complexity increasing, because the noise in training set will be captured by the model.

③ training error: It is high for not complex model, and it starts to decrease as the complexity increasing, because the difference between predictions of the model and the observations decreases.

④ irreducible error: It is constant because it is from dataset and not influenced by model.

⑤ test error: It is high for not complex model, and it starts to decrease as the complexity increasing. Until the point when variance dominates, causing the overfitting of the model, the test error starts to increase.

Problem 2

Q1.

① $\sqrt{3^2} = 3$

② $\sqrt{2^2} = 2$

③ $\sqrt{1^2 + 3^2} = \sqrt{10} \approx 3.16$

④ $\sqrt{1^2 + 2^2} = \sqrt{5} \approx 2.24$

⑤ $\sqrt{1^2 + 1^2} = \sqrt{2} \approx 1.41$

⑥ $\sqrt{1^2 + 1^2 + 1^2} = \sqrt{3} \approx 1.73$

Q2.

Obs.5 is the nearest neighbour to test point with euclidean distance 1.41, and obs.5 is green, so our prediction with K=1 will also be green.

Q3.

Obs.5, obs.6, obs.2 are the three nearest neighbour to test point with euclidean distance 1.41, 1.73, 2 respectively, and obs.5 is green, obs.6 is red, obs.2 is green, so our prediction with K=3 will be red, which is more than the number of green.

Problem 3

Q1.

Y = getting an A

$$P(Y) = \frac{e^{-6 + 0.05X_1 + X_2}}{1 + e^{-6 + 0.05X_1 + X_2}}$$

Q2.

Given that $X_1 = 40$ and $X_2 = 3.5$,

$$P(Y) = \frac{e^{-6 + 0.05 \times 40 + 3.5}}{1 + e^{-6 + 0.05 \times 40 + 3.5}} \approx 0.378$$

Q3.

Given that $\frac{e^{-6 + 0.05h + 3.5}}{1 + e^{-6 + 0.05h + 3.5}} = \frac{1}{2}$,

$$e^{-6 + 0.05h + 3.5} = 1$$

$\because e^0 = 1$

$\therefore -6 + 0.05h + 3.5 = 0$

$0.05h = 2.5$

$h = 50$

Thus, the student in part 2 needs to study 50 hours to have a 50% chance of getting an A in the class.

Problem 4

Q1.

A. 0.1

B. 0.001

C. 0.01

D. 0.0001

Q2.

(a) A

(b) C

(c) F

(d) E

Problem 5

Q1.

$X = [-1, 1, -1, 1]$

① $A = [1, 0, 0, 0]$

$$\text{cosine similarity} = \frac{-1}{\sqrt{4} \times 1} = -\frac{1}{2} = -0.5$$

② $B = [0, 1, 0, 0]$

$$\text{cosine similarity} = \frac{1}{\sqrt{4} \times 1} = \frac{1}{2} = 0.5$$

③ $C = [1, 0, 1, 0]$

$$\text{cosine similarity} = \frac{-2}{\sqrt{4} \times \sqrt{2}} = -\frac{1}{\sqrt{2}} = -0.707$$

④ $D = [0, 1, 0, 1]$

$$\text{cosine similarity} = \frac{2}{\sqrt{4} \times \sqrt{2}} = \frac{1}{\sqrt{2}} = 0.707$$

∴ Nearest-1 : D, Nearest-2 : DB, Nearest-3 : DBA, Nearest-4 : DBAC

Q2.

$Y = [1, -1, 1, -1]$

① $A = [1, 0, 0, 0]$

$$\text{cosine similarity} = \frac{1}{\sqrt{4} \times 1} = \frac{1}{2} = 0.5$$

② $B = [0, 1, 0, 0]$

$$\text{cosine similarity} = \frac{-1}{\sqrt{4} \times 1} = -\frac{1}{2} = -0.5$$

③ $C = [1, 0, 1, 0]$

$$\text{cosine similarity} = \frac{2}{\sqrt{4} \times \sqrt{2}} = \frac{1}{\sqrt{2}} = 0.707$$

④ $D = [0, 1, 0, 1]$

$$\text{cosine similarity} = \frac{-2}{\sqrt{4} \times \sqrt{2}} = -\frac{1}{\sqrt{2}} = -0.707$$

∴ Nearest-1 : C, Nearest-2 : CA, Nearest-3 : CAB, Nearest-4 : CABD

Problem 6

Q1.

$(5 \times 5 + 1) \times 6 \times 28 \times 28 = 122304$

Q2.

The third convolutional layer C5 takes all the output of the previous layer S4, "flattens" them and turns them into a singe vector that can all be the input for the next layer F6.

Q3.

convolutional layer

Q4.

① Connect the first 6 feature maps of layer C3 to the adjacent 3 feature maps in the previous layer S2 as the input for C3.

② Connect the next 6 feature maps of layer C3 to the neighbour 4 feature maps in the previous layer S2 as the input for C3.

③ Connect the next 3 feature maps of layer C3 to non-adjacent 4 feature maps in the previous layer S2 as the input for C3.

④ Connect the last feature map of layer C3 to all feature maps in the previous layer S2 as the input for C3.

Thus, the trainable parameters of layer C3 with kernel size 5x5 are $6 \times (3 \times 5 \times 5 + 1) + 6 \times (4 \times 5 \times 5 + 1) + 3 \times (4 \times 5 \times 5 + 1) + 1 \times (6 \times 5 \times 5 + 1) = 1516$.

Problem 7

Q1.

```
# importing module
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# numpy.random.normal() method
np.random.seed(0)
x = np.random.normal(0, 1, 100)
err = np.random.normal(0, 1, 100)

# f(x) = x - 2x^2 , y(x) = f(x) + N(0, 1)
f = []
y = []
for i in range(len(x)):
  num1 = x[i] - 2 * (x[i] ** 2)
  num2 = x[i] - 2 * (x[i] ** 2) + err[i]
  f.append(num1)
  y.append(num2)


# Display the dataset
y_dataset = pd.DataFrame(data=[x, y]).T
y_dataset.columns = ['x','y']
display(y_dataset)
```

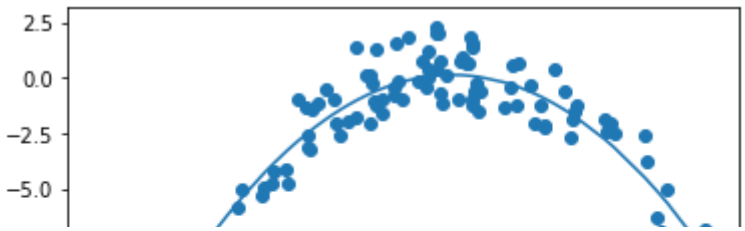
	x	y
0	1.764052	−2.576558
1	0.400157	−1.267853

```
# Display f(x)
x_sort = np.sort(x)
f_sort = []
for i in range(len(x_sort)):
  num1 = x_sort[i] - 2 * (x_sort[i] ** 2)
  f_sort.append(num1)
f_dataset = pd.DataFrame(data=[x_sort, f_sort]).T
f_dataset.columns=['x','f(x)']
display(f_dataset)
```

	x	f(x)
0	−2.552990	−15.588504
1	−1.980796	−9.827906
2	−1.726283	−7.686386
3	−1.706270	−7.528986
4	−1.630198	−6.945292
...	...	...
95	1.867558	−5.107988
96	1.895889	−5.292902
97	1.950775	−5.660274
98	2.240893	−7.802311
99	2.269755	−8.033817

100 rows × 2 columns

```
# Use scatter plot for y and smooth line plot for f(x).
plt.scatter(y_dataset['x'],y_dataset['y'])
plt.plot(f_dataset['x'],f_dataset['f(x)'])
plt.show()
```



Q2.

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd


model = LinearRegression()
cv = LeaveOneOut()
X = np.vstack([x, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 1:",RMSE)
X = np.vstack([x,x**2, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 2:",RMSE)
X = np.vstack([x,x**2,x**3, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 3:",RMSE)
X = np.vstack([x,x**2,x**3,x**4, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 4:",RMSE)
```

```
Model 1: 2.981309415774586
Model 2: 1.056481513788703
Model 3: 1.0780658052226322
Model 4: 1.0710499255446229
```

Q3.

```
np.random.seed(1)
cv = LeaveOneOut()
```

```
model = LinearRegression()


# x = np.random.normal(0, 1, 100)
# err = np.random.normal(0, 1, 100)
# # f(x) = x − 2x^2 , y(x) = f(x) + N(0, 1)
# y = []
# for i in range(len(x)):
#   num2 = x[i] - 2 * (x[i] ** 2) + err[i]
#   y.append(num2)

X = np.vstack([x, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 1:",RMSE)
X = np.vstack([x,x**2, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 2:",RMSE)
X = np.vstack([x,x**2,x**3, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 3:",RMSE)
X = np.vstack([x,x**2,x**3,x**4, np.ones(len(x))]).T
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',cv=cv, n_jobs=
RMSE = sqrt(mean(absolute(scores)))
print("Model 4:",RMSE)
```

```
Model 1: 2.981309415774586
Model 2: 1.056481513788703
Model 3: 1.0780658052226322
Model 4: 1.0710499255446229
```

My results are the same as what I got in 2, because LOOCV evaluates every iteration of splitting the data into groups containing n-1 observations and the remaining observation regardless different random seeds.

Q4.

The second model with quadratic polynomial had the smallest LOOCV error. It matches my expectation because it shows the true form of y, which also has a quadratic polynomial.

Problem 8

Q1.

```
!git clone https://github.com/kuangliu/pytorch-cifar.git
%cd pytorch-cifar/models
!python resnet.py install
```

```
Cloning into 'pytorch-cifar'...
remote: Enumerating objects: 382, done.
remote: Total 382 (delta 0), reused 0 (delta 0), pack-reused 382
Receiving objects: 100% (382/382), 85.69 KiB | 12.24 MiB/s, done.
Resolving deltas: 100% (195/195), done.
/content/pytorch-cifar/models/pytorch-cifar/models
```

```
'''Train CIFAR10 with PyTorch.'''
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn

import torchvision
import torchvision.transforms as transforms

import os
import argparse
import resnet
import tqdm as tqdm


# Data
print('==> Preparing data..')
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    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)

best_acc = 0
start_epoch = 0
# Model
print('==> Building model..')
net = resnet.ResNet18()
```

```
==> Preparing data..
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/ci
100%                                                 170498071/170498071 [00:01<00:00,
                                                     87535412.63it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```
import os
import sys
import time
import math

import torch.nn as nn
import torch.nn.init as init


# utils.py
term_width = 100
TOTAL_BAR_LENGTH = 65.
last_time = time.time()
begin_time = last_time
def progress_bar(current, total, msg=None):
    global last_time, begin_time
    if current == 0:
        begin_time = time.time()  # Reset for new bar.

    cur_len = int(TOTAL_BAR_LENGTH*current/total)
    rest_len = int(TOTAL_BAR_LENGTH - cur_len) - 1

    sys.stdout.write(' [')
    for i in range(cur_len):
        sys.stdout.write('=')
    sys.stdout.write('>')
    for i in range(rest_len):
        sys.stdout.write('.')
    sys.stdout.write(']')

    cur_time = time.time()
    step_time = cur_time - last_time
    last_time = cur_time
```

```
        tot_time = cur_time - begin_time

        L = []
        L.append('  Step: %s' % format_time(step_time))
        L.append(' | Tot: %s' % format_time(tot_time))
        if msg:
            L.append(' | ' + msg)

        msg = ''.join(L)
        sys.stdout.write(msg)
        for i in range(term_width-int(TOTAL_BAR_LENGTH)-len(msg)-3):
            sys.stdout.write(' ')

        # Go back to the center of the bar.
        for i in range(term_width-int(TOTAL_BAR_LENGTH/2)+2):
            sys.stdout.write('\b')
        sys.stdout.write(' %d/%d ' % (current+1, total))

        if current < total-1:
            sys.stdout.write('\r')
        else:
            sys.stdout.write('\n')
        sys.stdout.flush()

    def format_time(seconds):
        days = int(seconds / 3600/24)
        seconds = seconds - days*3600*24
        hours = int(seconds / 3600)
        seconds = seconds - hours*3600
        minutes = int(seconds / 60)
        seconds = seconds - minutes*60
        secondsf = int(seconds)
        seconds = seconds - secondsf
        millis = int(seconds*1000)

        f = ''
        i = 1
        if days > 0:
            f += str(days) + 'D'
            i += 1
        if hours > 0 and i <= 2:
            f += str(hours) + 'h'
            i += 1
        if minutes > 0 and i <= 2:
            f += str(minutes) + 'm'
            i += 1
        if secondsf > 0 and i <= 2:
            f += str(secondsf) + 's'
            i += 1
        if millis > 0 and i <= 2:
            f += str(millis) + 'ms'
```

```
            i += 1
        if f == '':
            f = '0ms'
        return f

    # main.py
    # Training
    def train(epoch,optimizer,criterion,trainloader,device,net):
        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%% (%d/%d)'
                         % (train_loss/(batch_idx+1), 100.*correct/total, correct, total)


    def test(epoch,optimizer,criterion,trainloader,device,net):
        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%% (%d/%d
                             % (test_loss/(batch_idx+1), 100.*correct/total, correct, tot

        # 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
```

Q2.

```
# main.py
def main(start_epoch, trainloader, testloader,net, best_acc):
  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 checkpoin
  args,_ = parser.parse_known_args()

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

  device= ""

  if torch.cuda.is_available():
    device = 'cuda'
  else:
    device = 'cpu'

  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)

for epoch in range(start_epoch, start_epoch+5):
    train(epoch,optimizer,criterion,trainloader,device,net)
    test(epoch,optimizer,criterion,trainloader,device,net)
    scheduler.step()
```

Q3.

```
main(start_epoch,trainloader,testloader,net,best_acc)
```

```
Epoch: 0
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-153-82eb9ba9fa6f> in <module>
----> 1 main(start_epoch,trainloader,testloader,net,best_acc)

                                 10 frames

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in
_conv_forward(self, input, weight, bias)
    452                             _pair(0), self.dilation, self.groups)
    453         return F.conv2d(input, weight, bias, self.stride,
--> 454                         self.padding, self.dilation, self.groups)
    455
    456     def forward(self, input: Tensor) -> Tensor:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

Problem 9

Q1.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from keras.datasets import mnist
import cv2
import keras.utils as kr
```

```
def reshape(img, size):
  result = np.ones((img.shape[0],size,size))
  for i in range(img[i].shape[0]):
    data = cv2.resize(img[i].reshape(img[i].shape[0],img[i].shape[0]).astype("uint8")
    result[i] = data
  result = np.stack((result,result,result),axis = -1)
  return result
```

```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
x_train = reshape(X_train,32)
x_test = reshape(X_test, 32)
y_train = kr.to_categorical(Y_train,num_classes=10)
y_test = kr.to_categorical(Y_test,num_classes=10)
```

```
from torch.nn.modules import activation
model = tf.keras.applications.VGG19(include_top=False,weights="imagenet",input_tensor
model.trainable = False
lr = 0.001
globalAVG = tf.keras.layers.GlobalAveragePooling2D()
predict = tf.keras.layers.Dense(10,activation="softmax")
inputSize = tf.keras.Input(shape=(32,32,3))
output = predict(globalAVG(model(inputSize,training = False)))
Model = tf.keras.Model(inputSize,output)
Model.compile(loss= tf.keras.losses.BinaryCrossentropy(from_logits=True),optimizer=tf
info = Model.fit(x_train,y_train,epochs=10,validation_data=(x_test,y_test),batch_size
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
80134624/80134624 [==============================] - 1s 0us/step
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082:
  return dispatch_target(*args, **kwargs)
1875/1875 [==============================] - 900s 480ms/step - loss: 0.1398 - ac
Epoch 2/10
1875/1875 [==============================] - 874s 466ms/step - loss: 0.0678 - ac
Epoch 3/10
1875/1875 [==============================] - 877s 468ms/step - loss: 0.0524 - ac
Epoch 4/10
1875/1875 [==============================] - 890s 475ms/step - loss: 0.0451 - ac
Epoch 5/10
1875/1875 [==============================] - 876s 467ms/step - loss: 0.0407 - ac
Epoch 6/10
1875/1875 [==============================] - ETA: 0s - loss: 0.0376 - accuracy:
```

Q2.

```
model = tf.keras.applications.VGG19(include_top=False,weights="imagenet",input_tensor
model.trainable = False
```

```
for i,layer in enumerate(model.layers):
  if i >=10 :
    layer.trainable = True
lr = 0.001
globalAVG = tf.keras.layers.GlobalAveragePooling2D()
predict = tf.keras.layers.Dense(10,activation="softmax")
inputSize = tf.keras.Input(shape=(32,32,3))
output = predict(globalAVG(model(inputSize,training = False)))
Model = tf.keras.Model(inputSize,output)
Model.compile(loss= tf.keras.losses.BinaryCrossentropy(from_logits=True),optimizer=tf
info = Model.fit(x_train,y_train,epochs=10,validation_data=(x_test,y_test),batch_size
```

Q3.

The second one should give a better transferability, becuase it trains the data using the higher layers.

Problem 10

Q1.

Layer	Number of Activations (Memory)	Parameters (Compute)
Input	224*224*3=150K	0
CONV3-64	224×224×64 = 3211264	3x3x3x64 = 1728
CONV3-64	224×224×64 = 3211264	3x3x64x64 = 36864
POOL2	112×112×64 = 802816	0
CONV3-128	112x112x128 = 1605632	3x3x64x128 = 73728
CONV3-128	112x112x128 = 1605632	3x3x128x128 = 147456
POOL2	56x56x128 = 401408	0
CONV3-256	56x56x256 = 802816	3x3x128x256 = 294912
CONV3-256	56x56x256 = 802816	3x3x256x256 = 589824
CONV3-256	56x56x256 = 802816	3x3x256x256 = 589824
POOL2	28x28x256 = 200704	0
CONV3-512	28x28x512 = 401408	3x3x256x512 = 1179648
CONV3-512	28x28x512 = 401408	3x3x512x512 = 2359296
CONV3-512	28x28x512 = 401408	3x3x512x512 = 2359296
POOL2	14x14x512 = 100352	0
CONV3-512	14x14x512 = 100352	3x3x512x512 = 2359296
CONV3-512	14x14x512 = 100352	3x3x512x512 = 2359296
CONV3-512	14x14x512 = 100352	3x3x512x512 = 2359296
POOL2	7x7x512 = 25088	0
FC	4096	7x7x512x4096 = 102760448
FC	4096	4096*4096 = 16,777,216
FC	1000	4096x1000 = 4096000
TOTAL	15237608	138344128

Table 1: VGG16 memory and weights

Q2(a).

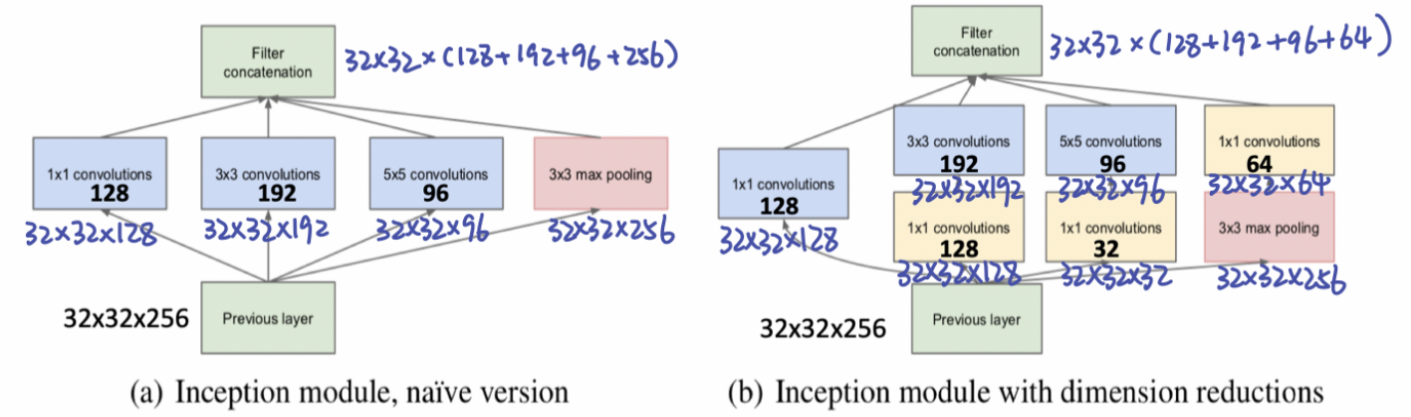


Figure 6: Inception blocks

Q2(b).

- Inception module, naive version

  total number of convolutional operations = (32 x 32 x 128) x (1 x 1 x 256) + (32 x 32 x 192) x (3 x 3 x 256) + (32 x 32 x 96) x (5 x 5 x 256) = 1115684864

- Inception module with dimension reductions

  total number of convolutional operations = (32 x 32 x 128) x (1 x 1 x 256) + (32 x 32 x 128) x (1 x 1 x 256) + (32 x 32 x 192) x (3 x 3 x 128) + (32 x 32 x 32) x (1 x 1 x 256) + (32 x 32 x 96) x (5 x 5 x 32) + (32 x 32 x 64) x (1 x 1 x 256) = 397410304

Problem 11

Q1.

- (b)
- (c)

Q2.

```
def forward_batchnorm(Z, gamma, beta, eps, cache_dict, beta_avg, mode):
  if mode == 'train':
    mu = np.mean(Z, axis = 0)
    var = np.var(Z, 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 = (Z - mu) / np.sqrt(var + eps)
  out = gamma * Z_norm + beta
  return out
```

Q3.

The role of ϵ hyperparameter in batch normalization is to avoid division by zero when variance is 0.

Q4.

If the batch size is 1, then mean will be the same as the output, which causes the output to be 0. In this case, we cannot learn anything meaningful.

Q5.

20 x 30 + 30 + 30 = 660

Problem 12

Q1.

1. CONV 7x7, C filters
2. CONV 1x7, C filters → CONV 7X1, C filters
3. CONV 4X4, C filters → CONV 4X4, C filters
4. CONV 3X3, C filters → CONV 3X3, C filters → CONV 3X3, C filters

5. CONV 1X1, C/2 filters → CONV 3X3, C/2 filters → CONV 1X1, C filters

Q2.

1. (7 x 7 x C) x C = $49C^2$
2. (1 x 7 x C + 7 x 1 x C) x C = $14C^2$
3. 2 x (4 x 4 x C) x C = $32C^2$
4. 3 x (3 x 3 x C) x C = $27C^2$
5. 3 x (1 x 1 x C x $\frac{C}{2}$ + 3 x 3 x $\frac{C}{2}$ x $\frac{C}{2}$ + 1 x 1 x $\frac{C}{2}$ x C) = $\frac{39}{4}C^2$ = $9.75C^2$

Q3.

1. (H x W x C) x (7 x 7 x C) = $49HWC^2$
2. (H x W x C) x (1 x 7 x C + 7 x 1 x C) = $14HWC^2$
3. 2 x (H x W x C) x (4 x 4 x C) = $32HWC^2$
4. 3 x (H x W x C) x (3 x 3 x C) = $27HWC^2$
5. 3 x (H x W x $\frac{C}{2}$ x 1 x 1 x C + H x W x $\frac{C}{2}$ x 3 x 3 x $\frac{C}{2}$ + H x W x C x 1 x 1 x $\frac{C}{2}$) = $\frac{39}{4}HWC^2$ = $9.75HWC^2$

Architecture 1 has the largest computational requirement and architecture 5 has the smallest.