15/11/2021 ML- HW5 Sai Satya Charan.Malladi

Q1) (a)

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i(w^T x_i + b)) + \lambda \|w\|^2$$

$$\phi(t) = \log(1 + e^{-t})$$

$$J = \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(w^T x_i + b)}\right) + \lambda \|w\|^2$$

$$\frac{\partial J}{\partial w} \Rightarrow \underbrace{\frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 + e^{-(y_i(w^T x_i + b))}} \left[-y_i x_i e^{-y_i(w^T x_i + b)}\right]}_{} + 2\lambda w$$

dividing the numerator and denominator by $e^{-y_i(w^T x_i + b)}$

We get,

$$\boxed{\frac{\partial J}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i}{1 + e^{y_i(w^T x_i + b)}} + 2\lambda w}$$

Similarly,

$$\frac{\partial J}{\partial b} = \underbrace{\frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 + e^{-y_i(w^T x_i + b)}} \left[-y_i e^{-y_i(w^T x_i + b)}\right]}_{} + 0$$

dividing numerator and denominator by $e^{-y_i(w^T x_i + b)}$

We get,

$$\boxed{\frac{\partial J}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \frac{-y_i}{1 + e^{y_i(w^T x_i + b)}}}$$

Gradient descent update rule

$$w^{k+1} = w^k - \eta \frac{\partial J}{\partial w^k}$$

$$b^{k+1} = b^k - \eta \frac{\partial J}{\partial b^k}$$

where w^k, b^k indicate hyperparameters at kth iteration.

$$\boxed{\begin{aligned} w^{k+1} &= w^k - \eta \left[\frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i}{1+e^{y_i(w^{k^T} x_i + b^k)}} + 2\lambda w^k \right] \\ b^{k+1} &= b^k - \eta \left[\frac{1}{n} \sum_{i=1}^{n} \frac{-y_i}{1+e^{y_i(w^{k^T} x_i + b^k)}} \right] \end{aligned}}$$

→ This is $(w^k)^T$

→ Update rules

Q1 (b) In Q1 (a) we have got

$$\frac{\partial J}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i}{1+e^{y_i(w^T x_i + b)}} + 2\lambda w$$

As the objective J is given to be convex, a unique global minimizer exists., (w^*, b^*). And @ global minimizer

$$\underbrace{\frac{\partial J}{\partial w^*} = 0}_{\downarrow} \qquad \frac{\partial J}{\partial b^*} = 0$$

We will just take $\frac{\partial J}{\partial w^*} = 0$ and examine it.

$$\frac{\partial J}{\partial w^*} = 0 \Rightarrow \frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i}{1+e^{y_i((w^*)^T x_i + b^*)}} + 2\lambda w^* = 0$$

$$\Rightarrow \quad 2\lambda w^* = +\frac{1}{n}\sum_{i=1}^{n} \frac{y_i x_i}{1+e^{y_i(w^{*T}x_i+b^*)}}$$

$$\Rightarrow \quad 2\lambda w^* = \frac{1}{n} X^T \begin{bmatrix} \frac{y_1}{1+e^{y_1(w^{*T}x_1+b^*)}} \\ \vdots \\ \frac{y_n}{1+e^{y_n(w^{*T}x_n+b)}} \end{bmatrix}$$

where $X^T = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix}$

$$\Rightarrow \quad w^* = X^T \underbrace{\left[\frac{1}{2\lambda n} \begin{bmatrix} \frac{y_1}{1+e^{y_1(w^{*T}x_1+b^*)}} \\ \vdots \\ \frac{y_n}{1+e^{y_n(w^{*T}x_n+b^*)}} \end{bmatrix} \right]}_{\alpha^*}$$

$\therefore \boxed{w^* = X^T \alpha^*}$ for some α^*

Q1

(c) (a) $w^{old} = X^T \alpha^{old}$

We know from Q1 (a) update formula

$$w^{new} = w^{old} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} \frac{-y_i x_i}{1+e^{y_i(\underbrace{w^{oldT}x_i}_{\text{This is } (w^{old})^T}+b^{old})}} + 2\lambda w^{old} \right] \quad -①$$

Let us substitute $w^{old} = X^T\alpha^{old}$ in eqn ①

$$\cancel{X^T\alpha^{old}} \quad w^{new} = X^T\alpha^{old} - \eta\left[\underbrace{\frac{1}{n}\sum\frac{-y_i x_i}{1+e^{y_i((X^T\alpha^{old})^T x_i + b^{old})}}} + 2\lambda X^T\alpha^{old}\right] \quad -②$$

Let us try to analayse this term

$$\Rightarrow \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i}{1+e^{y_i(\alpha^{old\,T}Xx_i + b^{old})}} \Rightarrow \frac{1}{n}X^T\begin{bmatrix}\dfrac{-y_1}{1+e^{y_1(\alpha^{old\,T}Xx_1 + b^{old})}}\\ \vdots \\ \dfrac{-y_n}{1+e^{y_n(\alpha^{old\,T}Xx_n + b^{old})}}\end{bmatrix} \quad -③$$

Substituting eqn ③ back in ②

$$w^{new} = X^T\alpha^{old} - \eta\left[\frac{1}{n}X^T\begin{bmatrix}\dfrac{-y_1}{1+e^{y_1(\alpha^{old\,T}Xx_1 + b^{old})}}\\ \vdots \\ \dfrac{-y_n}{1+e^{y_n(\alpha^{old}Xx_n + b^{old})}}\end{bmatrix} + 2\lambda X^T\alpha^{old}\right]$$

$$\Rightarrow w^{new} = X^T\underbrace{\left[\alpha^{old} - \eta\left[\frac{1}{n}\begin{bmatrix}\dfrac{-y_1}{1+e^{y_1(\alpha^{old\,T}Xx_1 + b^{old})}}\\ \vdots \\ \dfrac{-y_n}{1+e^{y_n(\alpha^{old\,T}Xx_n + b^{old})}}\end{bmatrix} + 2\lambda\alpha^{old}\right]\right]}_{\alpha^{new}}$$

$$\boxed{\Rightarrow w^{new} = X^T\alpha^{new}}$$

where

$$\alpha^{new} = \left[\alpha^{old} - \eta\left[\frac{1}{n}\begin{bmatrix}\dfrac{-y_1}{1+e^{y_1(\alpha^{old\,T}Xx_1 + b^{old})}}\\ \vdots \\ \dfrac{-y_n}{1+e^{y_n(\alpha^{old\,T}Xx_n + b^{old})}}\end{bmatrix} + 2\lambda\alpha^{old}\right]\right] \rightarrow \text{update}$$

Q1 (d) From Q1(c) we know that

$$\alpha^{new} = \alpha^{old} - \eta \left[\frac{1}{n} \underbrace{\begin{bmatrix} \dfrac{-y_1}{1+e^{y_1(\alpha^{old\,T} X x_1 + b^{old})}} \\ \vdots \\ \dfrac{-y_n}{1+e^{y_n(\alpha^{old\,T} X x_n + b^{old})}} \end{bmatrix}} + 2\lambda\alpha^{old} \right] \quad -(1)$$

Let us just examine a general term in this vector

$$\Rightarrow \frac{-y_i}{1+e^{y_i(\alpha^{old\,T} \underbrace{X x_i} + b^{old})}}$$

we see that Xx_i is an inner product

(ie) $$X x_i = \begin{bmatrix} x_1^T x_i \\ x_2^T x_i \\ \vdots \\ x_n^T x_i \end{bmatrix} \quad -(2)$$

This inner product can be replaced by using a kernel.

$$X x_i = \begin{bmatrix} k(x_1, x_i) \\ k(x_2, x_i) \\ \vdots \\ k(x_n, x_i) \end{bmatrix} = K(X, x_i) \quad -(3)$$

I am denoting this column vector as $K(X, x_i)$

Now substituting the same, back in eqn ① in eq ③

We get

$$\alpha^{new} = \alpha^{old} - \eta \left[\frac{1}{n} \left[\frac{y_i}{1 + e^{y_i(\alpha^{old^T} K(X, x_i) + b^{old})}} \right]_{i=1 \ldots n} + 2\lambda \alpha^{old} \right]$$

also

This is b^{old}

$$b^{new} = b - \eta \sum_{i=1}^{n} \frac{-y_i}{1 + e^{y_i((\alpha^{old})^T K(X, x_i) + b^{old})}}$$

Thus the update rule is kernelized.

We can set α^{old} to zero vector to start with and keep updating a^{new}, b^{new} till we end up with (α^*, b^*), the we know $w^* = X^T \alpha^*$

Final classifier,

$$\eta(y=1|x) = \frac{1}{1 + e^{-(\alpha^{*T} K(X, x_i) + b^*)}}$$

This is the probability that Class = 1,

and

$$\begin{cases} \eta(y=1|x) \geq \frac{1}{2} & \text{predict class} = 1 \\ \eta(y=1|x) < \frac{1}{2} & \text{predict class} = -1 \end{cases}$$

This is -1

Kernel Logistic Regression

Q 1(e) Accuracy = 0.796

Error ⇒ 1 - 0.796 ⇒ 0.204

Q 2(a) Statistics X_mean = [0.50161345 , 0.45612671, 0.38244407]

X std = [0.24617303 , 0.23615181 , 0.23905821]

(1)

- Resizing:

✓ Benefit – resizing is useful to have images with the same pixels in the data set
The raw dataset might contain different images with different dimensions for example 1000 x 800 , 200 x 400 , etc , but resizing helps us to work with uniform dimensions through out data set.

✓ Draw back: By doing resizing, some important features are interpolated, and also, when ever we resize a smaller image, we would create a padding around, it which migh mis lead our classifier, if there are several images with padding.

- Normalizing:

✓ Benifit: Normalizing the pixel intensities, will help the convergence of the optimizer, there wontbe much of ~~zig~~ zig-zagging during optimization. The ~~Drawback~~ centered data will keep the gradients in control when we use back propagation

Normalizing data

* **Drawback** - Normalizing data can sometimes lead to loss of information.

For example, Normalizing just preserves relative variations in data.

If we want to classify day and night images; normalizing our data will affect the classification performance as it only captures relative information.

Q2(a)

(2) per-channel mean and standard deviation are use to scale our training data. Validation set is not a part of our training data, when we scale our training data using training per channel mean and per-channel standard deviation, it helps us center the data for training, but during validation and testing phase we should use this same scaling as the neural network that is trained is learnt for the information scaled this particular way. Hence, it doesnot make sense to use a different scaling metric for validation images.

Q 2(b)

(1) Layer - 1

Filter = 5x5

Input channel = 3

Output channel = 16

This is for bias

No. of parameters = $(5\times5\times3+1)\times16$

$\boxed{= 1216}$ - Convolution layer-1

Layer-2

Filter = 5x5

Input channel = 16

Output channels = 32

Bias

No. of parameters = $(5\times5\times16+1)\times32$

$\boxed{= 12832}$ - Convolution layer-2

Layer-3

Filter = 5x5

Input channel = 32

Output channel = 64

No. of parameters = $(5 \times 5 \times 32 + \overset{\text{Bias}}{1}) \times 64$

$= \boxed{51264}$ - convolution Layer-3

Layer-4

Filter = 5x5

Input channel = 64

Output Channels = 128

No. of parameters = $(5 \times 5 \times 64 + \overset{\text{Bias}}{1}) \times 128$

$= \boxed{204{,}928}$ - convolution Layer-4

Layer-5 (Fully connected

Inputs = 128x2x2

Output = 64

No. of parameters = $(128 \times 2 \times 2 + \overset{\text{Bias}}{1}) \times 64$

$= \boxed{32{,}832}$ - fully connected Layer-1

Layer-6 (Fully connected)

Inputs = 64

Outputs = 5

No. of parameters = $(64 + \overset{\text{Bias}}{1}) \times 5$

$= \boxed{325}$ - fully connected Layer-2

Total parameters =

$$\begin{array}{r} 1216\ + \\ 12832\ + \\ 51264\ + \\ 204928\ + \\ 32832\ + \\ 325\ \\ \hline 303397 \end{array}$$

Q2(b) Q. One reason as to why we don't want to initialize our neural network to zero is because, when initialized with 0's then all the layers perform the same calculation, there won't be any symmetry breaking, the gradients computed take the learning no where, This is the reason why we should initialize the network weights randomly.

Q2 (d) Final accuracies and Losses

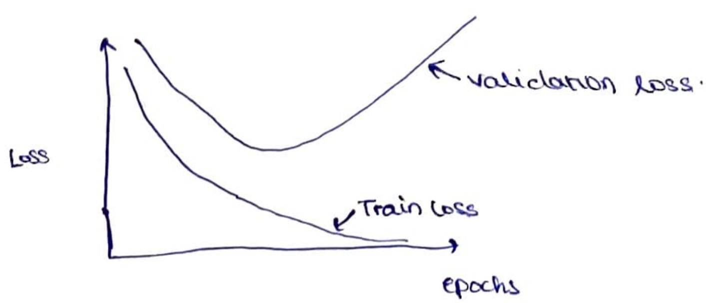
Validation Loss :- 2.3131

Validation accuracy :- 0.4872

Train Loss :- 0.007707

Train accuracy :- 1

(e) 1) The training Loss keeps decreasing, and the validation Loss drops at first and then starts increasing after a point.



As we keep on training the model our training loss would be going to zero, while our validation loss will shoot up to higher values.

This means we have done an overfit to our data.

Q2 (e) a. We should always try to minimize Validation loss so the training loss.

Based on the plot, we should stop training our model after 2 epochs as after 2 epochs the validation loss starts increasing.

We should not try to maximize training accuracy because, it leads to overfitting for the data, and the model learnt, won't be a good generalization of the true nature of data.

~~Q2 (g)~~ Q 2(f) Final accuracies and Losses:

Validation Loss :- 0.18

Validation accuracy = 0.935897

train Loss : 0.11598

train accuracy: 0.956

Q 2 (g) We can't use accuracy during an im balance dataset case because, it is not fully reflective of the correct classification for each class. Per-class accuracy metric makes more sense in this scenario to asses how good our model is.

Q 2 (h)

(Non-Weighted) Loss
Imbalance dataset (5 epochs)

Validation loss :- 0.1246

Validation accuracy :- 0.9545

train Loss: 0.03430

train accuracy: 0.99046

Per class accuracy :-

Cat	dog (+ve class)
1	0.5

(dog +ve class)

precision = 1

recall = 0.5

F1-Score = 0.66666

Weighted Cross entropy Loss
Imbalance dataset (5 epochs)

Validation Loss :- 0.12327

Validation accuracy :- 0.963636

train loss :- 0.047591

train accuracy :- 0.9857142

Per class accuracy :-

Cat	dog
0.98	0.8

Precision = 0.8

Recall = 0.8

F1 Score = 0.8

(dog is +ve class)

The un-weighted model has more training accuracy and train loss, but as the data set is imbalanced we can see it performs poorly on other metrics like F-1 score, recall, per-class accuracy. What was happening here is that the model was being overfit for the features of Cat as the examples are more in number.

In the weighted case, it can be observed that weighting dog class by a factor improved the performance of the model, this can be seen from metrics like F1 score, precision, recall. Here due to weight, the ~~feat~~ model was able to distinguish features of dog from that of cat, though the examples are pretty less in number.

Problem 2 (d)

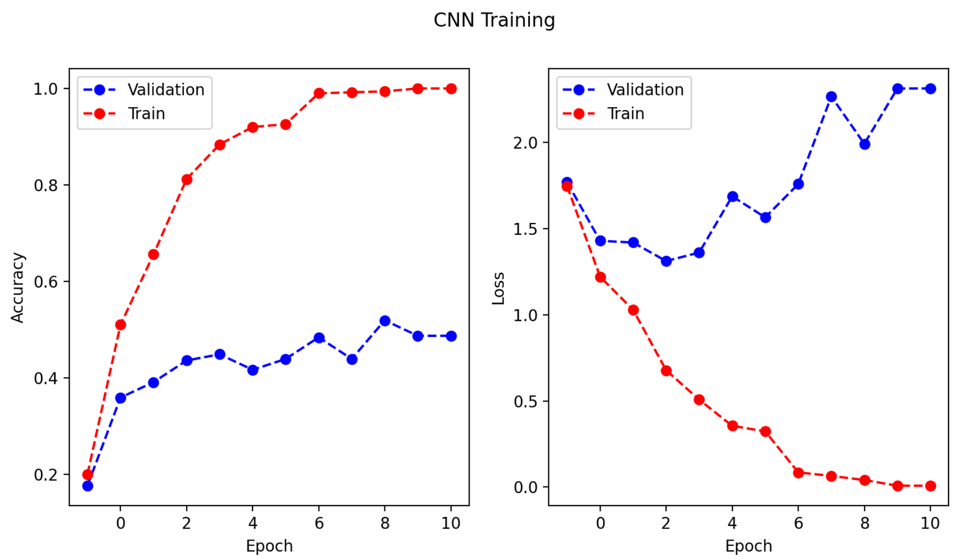


Figure 1: Problem 2 (d) CNN training plot

Problem 2 (f)

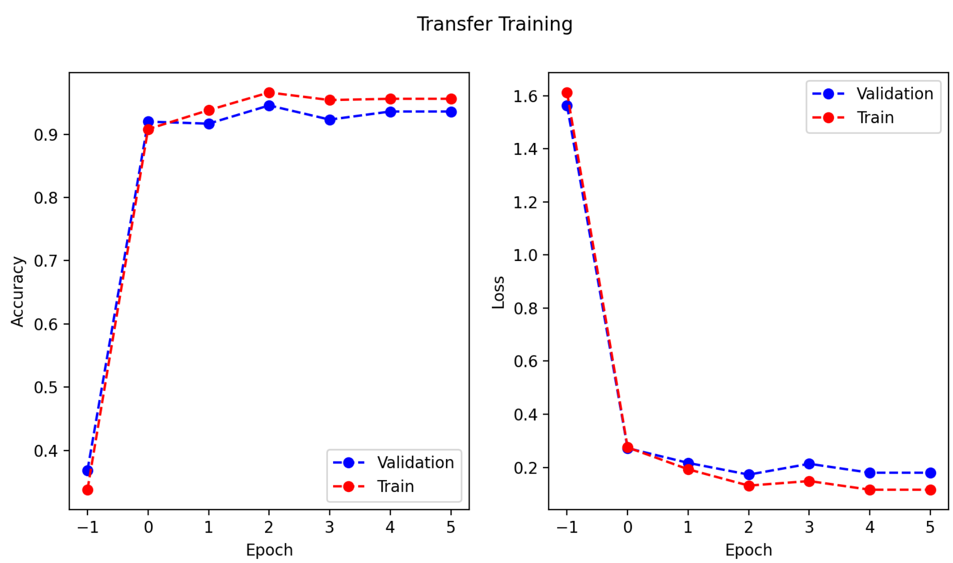


Figure 2: Problem 2 (f) Transfer training plot

Problem 2 (h)

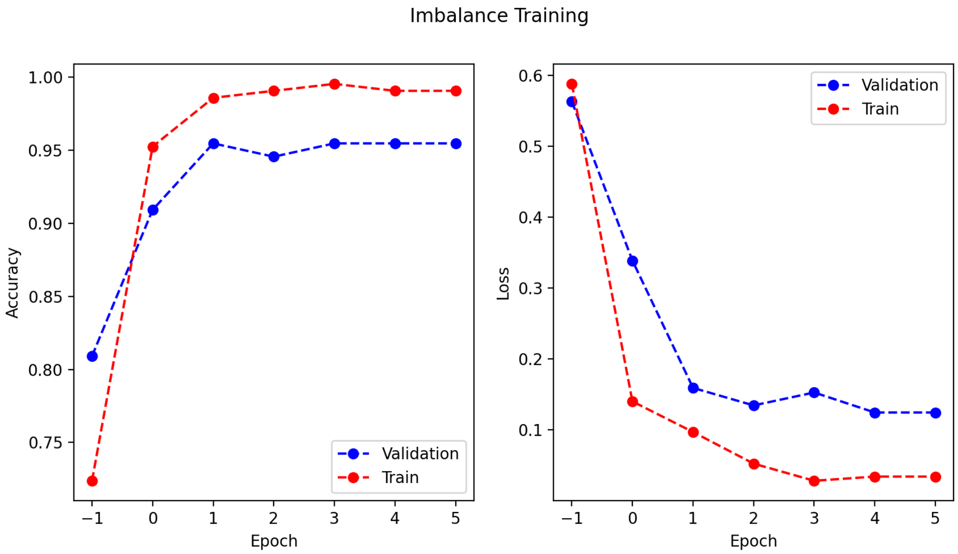


Figure 3: Problem 2 (h) Imbalance training plot

Problem 2 (h)

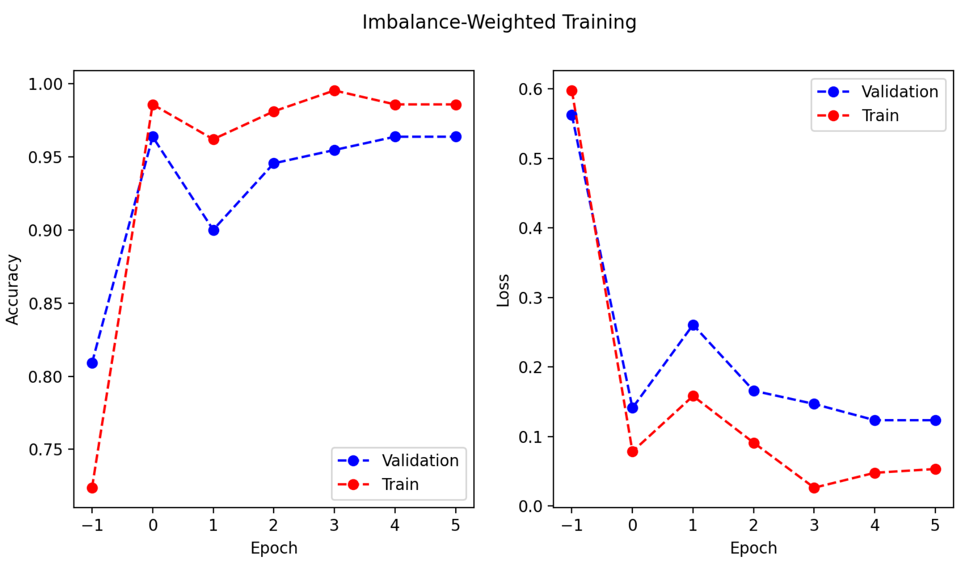


Figure 4: Problem 2 (h) Imbalance Weighted training plot

Problem 1

In [67]:

```python
# EECS 545 FA21 HW5 - Kernel Logistic Regression
import numpy as np
from sklearn.metrics.pairwise import rbf_kernel, linear_kernel
```

In [68]:

```python
# linear logistic regression
def linear_logistic_regression(x_train, y_train, x_test, y_test, step_size, reg_strength, num_iters):
    from sklearn.linear_model import LogisticRegression
    # only use sklearn's LogisticRegression
    clf = LogisticRegression(C=1/reg_strength)
    clf.fit(x_train, y_train)
    test_acc = clf.score(x_test, y_test)
    return test_acc
```

In [69]:

```python
# kernal logistic regression
def kernel_logistic_regression(x_train, y_train, x_test, y_test, step_size, reg_strength, num_iters, kernel_parameter):
    """
    x_train - (n_train, d)
    y_train - (n_train,)
    x_test - (n_test, d)
    y_test - (n_test,)
    step_size: gamma in problem description
    reg_strength: lambda in problem description
    num_iters: how many iterations of gradient descent to perform

    Implement KLR with the Gaussian Kernel.
    The only allowed sklearn usage is the rbf_kernel, which has already been imported.
    """
    # TODO
    ntrain = x_train.shape[0]
    nfeatures = x_train.shape[1]
    ntest = x_test.shape[0]

    # create kernel matrices
    ker_train = rbf_kernel(x_train,x_train,gamma = kernel_parameter)
    ker_test = rbf_kernel(x_train,x_test,gamma = kernel_parameter)

#     sanity check puroposes
#     ker_train = linear_kernel(x_train,x_train)
#     ker_test = linear_kernel(x_train,x_test)

    ### do gradient descent
    # set initial parameter
    alp = np.zeros(ntrain)
    b = 1e-7
    for i in range(num_iters):
        update_mat = np.array([-y_train[j]/(1 + np.exp(y_train[j]*(np.dot(alp,ker_train[:,j]) + b))) for j in range(ntrain)])
        b -= step_size*(1/ntrain*np.sum(update_mat))
        alp -= step_size*(1/ntrain*update_mat + 2*reg_strength*alp)

    y_pred = np.ones(ntest)
    # apply classifier on test set
    eta = np.array([1/(1 + np.exp(-(np.dot(alp,ker_test[:,j]) + b))) for j in range(ntest)])
    y_pred[eta < 1/2] = -1
    test_acc = np.sum(y_pred == y_test)/ntest
    return test_acc
```

In [71]:

```python
x_train = np.load("x_train.npy")    # shape (n_train, d)
x_test = np.load("x_test.npy")      # shape (n_test, d)

y_train = np.load("y_train.npy")    # shape (n_train,)
y_test = np.load("y_test.npy")        # shape (n_test,)

linear_acc = linear_logistic_regression(x_train, y_train, x_test, y_test, 1.0, 0.001, 200)
print("Linear LR accuracy:", linear_acc)

klr_acc = kernel_logistic_regression(x_train, y_train, x_test, y_test, 5.0, 0.001,200, 0.1)

# sanity check
# klr_acc = kernel_logistic_regression(x_train, y_train, x_test, y_test, 1.0, 0.001,200, 0.1)

print("Kernel LR accuracy:", klr_acc)
```

```
Linear LR accuracy: 0.769
Kernel LR accuracy: 0.796
```

Problem 2

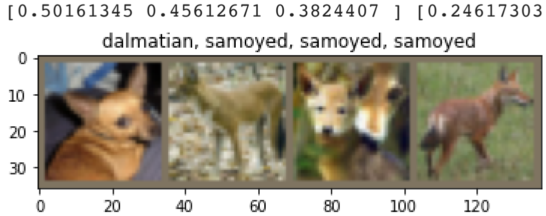
dataset.py

In [1]:

```python
# EECS 545 Fall 2021
import os
import matplotlib.pyplot as plt
import numpy as np

import torch
import torchvision
import torchvision.transforms as transforms


class DogDataset:
    """
    Dog Dataset.
    """
    def __init__(self, batch_size=4, dataset_path='data/images/dogs', if_resize=True):
        self.batch_size = batch_size
        self.dataset_path = dataset_path
        self.if_resize = if_resize
        self.train_dataset = self.get_train_numpy()
        self.x_mean, self.x_std = self.compute_train_statistics()
        self.transform = self.get_transforms()
        self.train_loader, self.val_loader = self.get_dataloaders()

    def get_train_numpy(self):
        train_dataset = torchvision.datasets.ImageFolder(os.path.join(self.dataset_path, 'train'))
        train_x = np.zeros((len(train_dataset), 224, 224, 3))
        # train_x = np.zeros((len(train_dataset), 64, 64, 3))
        for i, (img, _) in enumerate(train_dataset):
            train_x[i] = img
        return train_x / 255.0

    def compute_train_statistics(self):
        # TODO (part a): compute per-channel mean and std with respect to self.train_dataset
        x_mean = np.mean(self.train_dataset,axis=(0,1,2)) # per-channel mean
        x_std = np.std(self.train_dataset,axis=(0,1,2)) # per-channel std
        return x_mean, x_std

    def get_transforms(self):
        if self.if_resize:
            # TODO (part a): fill in the data transforms
            transform_list = [
                # resize the image to 32x32x3
                transforms.Resize((32,32)),
                # convert image to PyTorch tensor
                transforms.ToTensor(),
                # normalize the image (use self.x_mean and self.x_std)
                transforms.Normalize(self.x_mean,self.x_std)
            ]
        else:
            # TODO (part f): fill in the data transforms
            # Note: Only change from part a) is there is no need to resize the image
                transform_list = [
                # convert image to PyTorch tensor
                transforms.ToTensor(),
                # normalize the image (use self.x_mean and self.x_std)
                transforms.Normalize(self.x_mean,self.x_std)
            ]
        transform = transforms.Compose(transform_list)
        return transform

    def get_dataloaders(self):
        # train set
        train_set = torchvision.datasets.ImageFolder(os.path.join(self.dataset_path, 'train'), transform=self.transform)
        train_loader = torch.utils.data.DataLoader(train_set, batch_size=self.batch_size, shuffle=True)

        # validation set
        val_set = torchvision.datasets.ImageFolder(os.path.join(self.dataset_path, 'val'), transform=self.transform)
        val_loader = torch.utils.data.DataLoader(val_set, batch_size=self.batch_size, shuffle=False)

        return train_loader, val_loader

    def plot_image(self, image, label):
        image = np.transpose(image.numpy(), (1, 2, 0))
        image = image * self.x_std.reshape(1, 1, 3) + self.x_mean.reshape(1, 1, 3)  # un-normalize
        plt.title(label)
        plt.imshow((image*255).astype('uint8'))
        plt.show()

    def get_semantic_label(self, label):
```

```
        mapping = {'chihuahua': 0, 'dalmatian': 1, 'golden_retriever': 2, 'samoyed': 3, 'siberian_husky': 4}
        reverse_mapping = {v: k for k, v in mapping.items()}
        return reverse_mapping[label]


class DogCatDataset:
    """
    Cat vs. Dog Dataset.
    """
    def __init__(self, batch_size=4, dataset_path='data/images/dogs_vs_cats'):
        self.batch_size = batch_size
        self.dataset_path = dataset_path
        self.transform = self.get_transforms()
        self.train_loader, self.val_loader = self.get_dataloaders()

    def get_transforms(self):
        # TODO (part g): fill in the data transforms
        transform_list = [
            # resize the image to 256x256x3
            transforms.Resize((256,256)),
            # crop the image at the center of size 224x224x3
            transforms.CenterCrop((224,224)),
            # convert image to PyTorch tensor
            transforms.ToTensor(),
            # normalize the image
            transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
        ]
        transform = transforms.Compose(transform_list)
        return transform

    def get_dataloaders(self):
        # train set
        train_set = torchvision.datasets.ImageFolder(os.path.join(self.dataset_path, 'train'), transform=self.transform)
        train_loader = torch.utils.data.DataLoader(train_set, batch_size=self.batch_size, shuffle=True)

        # validation set
        val_set = torchvision.datasets.ImageFolder(os.path.join(self.dataset_path, 'val'), transform=self.transform)
        val_loader = torch.utils.data.DataLoader(val_set, batch_size=self.batch_size, shuffle=False)

        return train_loader, val_loader


if __name__ == '__main__':
    dataset = DogDataset()
    print(dataset.x_mean, dataset.x_std)
    images, labels = iter(dataset.train_loader).next()
    dataset.plot_image(
        torchvision.utils.make_grid(images),
        ', '.join([dataset.get_semantic_label(label.item()) for label in labels])
    )
```

```
[0.50161345 0.45612671 0.3824407 ] [0.24617303 0.23615181 0.23905821]
```



model.py

In [3]:

```
# EECS 545 Fall 2021
import math
# from typing_extensions import TypeVarTuple
import torch.nn as nn
import torch.nn.functional as F


class CNN(nn.Module):
    """
    Convolutional Neural Network.
    """
    def __init__(self):
        super().__init__()

        # TODO (part c): define layers
        self.conv1 = nn.Conv2d(3, 16, 5, stride=2, padding=2)  # convolutional layer 1
        self.conv2 = nn.Conv2d(16, 32, 5, stride=2, padding=2) # convolutional layer 2
        self.conv3 = nn.Conv2d(32, 64, 5, stride=2, padding=2) # convolutional layer 3
        self.conv4 = nn.Conv2d(64, 128, 5, stride=2, padding=2) # convolutional layer 4
        self.fc1 = nn.Linear(128*2*2, 64, bias=True) # fully connected layer 1
        self.fc2 = nn.Linear(64, 5, bias=True) # fully connected layer 2 (output layer)
```

```
        self.init_weights()

    def init_weights(self):
        for conv in [self.conv1, self.conv2, self.conv3, self.conv4]:
            C_in = conv.weight.size(1)
            nn.init.normal_(conv.weight, 0.0, 1/math.sqrt(5 * 2.5 * C_in))
            nn.init.constant_(conv.bias, 0.0)

        # TODO (part c): initialize parameters for fully connected layers
        nn.init.normal_(self.fc1.weight, 0.0, 1/math.sqrt(256))
        nn.init.constant_(self.fc1.bias, 0.0)
        nn.init.normal_(self.fc2.weight, 0.0, 1/math.sqrt(32))
        nn.init.constant_(self.fc2.bias, 0.0)

    def forward(self, x):
        N, C, H, W = x.shape

        # TODO (part c): forward pass of image through the network
        z = F.relu(self.conv1(x))
        z = F.relu(self.conv2(z))
        z = F.relu(self.conv3(z))
        z = F.relu(self.conv4(z))
        z = z.view(z.size(0),-1)
        z = F.relu(self.fc1(z))
        z = self.fc2(z)
        return z


def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)


if __name__ == '__main__':
    from dataset import DogDataset
    net = CNN()
    print(net)
    print('Number of CNN parameters: {}'.format(count_parameters(net)))
    dataset = DogDataset()
    images, labels = iter(dataset.train_loader).next()
    print('Size of model output:', net(images).size())
```

```
CNN(
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  (conv3): Conv2d(32, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  (conv4): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  (fc1): Linear(in_features=512, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=5, bias=True)
)
Number of CNN parameters: 303397
Size of model output: torch.Size([4, 5])
```

train.py

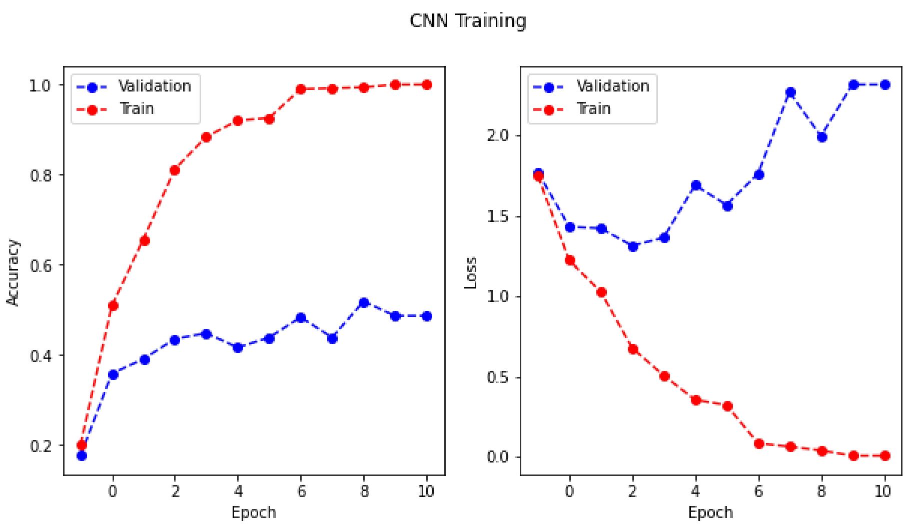
In [4]:

```
# EECS 545 Fall 2021
import torch
import numpy as np
import random

# from torch._C import FloatTensor
import checkpoint
from dataset import DogDataset, DogCatDataset
from model import CNN
from plot import Plotter

torch.manual_seed(0)
np.random.seed(0)
random.seed(0)

def predictions(logits):
    """
    Compute the predictions from the model.
    Inputs:
        - logits: output of our model based on some input, tensor with shape=(batch_size, num_classes)
    Returns:
        - pred: predictions of our model, tensor with shape=(batch_size)
    """
    # TODO (part d): compute the predictions
    pred = torch.argmax(logits, dim=1)
    return pred


def accuracy(y_true, y_pred):
    """
    Compute the accuracy given true and predicted labels.
    Inputs:
```

```python
        - y_true: true labels, tensor with shape=(num_examples)
        - y_pred: predicted labels, tensor with shape=(num_examples)
    Returns:
        - acc: accuracy, float
    """
    # TODO (part d): compute the accuracy
    num_examples = y_true.shape[0]
    acc = np.sum(y_true.numpy()==y_pred.numpy())/num_examples
    return acc


def _train_epoch(train_loader, model, criterion, optimizer):
    """
    Train the model for one iteration through the train set.
    """
    for i, (X, y) in enumerate(train_loader):
        # clear parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        output = model(X)
        loss = criterion(output, y)
        loss.backward()
        optimizer.step()


def _evaluate_epoch(plotter, train_loader, val_loader, model, criterion, epoch):
    """
    Evaluates the model on the train and validation set.
    """
    stat = []
    for data_loader in [val_loader, train_loader]:
        y_true, y_pred, running_loss = evaluate_loop(data_loader, model, criterion)
        total_loss = np.sum(running_loss) / y_true.size(0)
        total_acc = accuracy(y_true, y_pred)
        stat += [total_acc, total_loss]
    plotter.stats.append(stat)
    plotter.log_cnn_training(epoch)
    plotter.update_cnn_training_plot(epoch)


def evaluate_loop(data_loader, model, criterion=None):
    model.eval()
    y_true, y_pred, running_loss = [], [], []
    for X, y in data_loader:
        with torch.no_grad():
            output = model(X)
            predicted = predictions(output.data)
            y_true.append(y)
            y_pred.append(predicted)
            if criterion is not None:
                running_loss.append(criterion(output, y).item() * X.size(0))
    model.train()
    y_true, y_pred = torch.cat(y_true), torch.cat(y_pred)
    return y_true, y_pred, running_loss


def train(config, dataset, model):
    # Data loaders
    train_loader, val_loader = dataset.train_loader, dataset.val_loader

    if 'use_weighted' not in config:
        # TODO (part d): define loss function
        criterion = torch.nn.CrossEntropyLoss()
    else:
        # TODO (part h): define weighted loss function
        criterion = torch.nn.CrossEntropyLoss(weight=torch.FloatTensor([1,20]))

    # TODO (part d): define optimizer
    learning_rate = config['learning_rate']
    momentum = config['momentum']
    optimizer = torch.optim.SGD(model.parameters(),lr=learning_rate,momentum=momentum)

    # Attempts to restore the latest checkpoint if exists
    print('Loading model...')
    force = config['ckpt_force'] if 'ckpt_force' in config else False
    model, start_epoch, stats = checkpoint.restore_checkpoint(model, config['ckpt_path'], force=force)

    # Create plotter
    plot_name = config['plot_name'] if 'plot_name' in config else 'CNN'
    plotter = Plotter(stats, plot_name)

    # Evaluate the model
    _evaluate_epoch(plotter, train_loader, val_loader, model, criterion, start_epoch)

    # Loop over the entire dataset multiple times
    for epoch in range(start_epoch, config['num_epoch']):
        # Train model on training set
```

```
        _train_epoch(train_loader, model, criterion, optimizer)

        # Evaluate model on training and validation set
        _evaluate_epoch(plotter, train_loader, val_loader, model, criterion, epoch + 1)

        # Save model parameters
        checkpoint.save_checkpoint(model, epoch + 1, config['ckpt_path'], plotter.stats)

    print('Finished Training')

    # Save figure and keep plot open
    plotter.save_cnn_training_plot()
    plotter.hold_training_plot()


if __name__ == '__main__':
    # define config parameters for training
    config = {
        'dataset_path': 'data/images/dogs',
        'batch_size': 4,
        'if_resize': True,              # If resize of the image is needed
        'ckpt_path': 'checkpoints/cnn',  # directory to save our model checkpoints
        'num_epoch': 10,                 # number of epochs for training
        'learning_rate': 1e-3,           # learning rate
        'momentum': 0.9,                  # momentum
    }
    # create dataset
    dataset = DogDataset(config['batch_size'], config['dataset_path'],config['if_resize'])
    # create model
    model = CNN()
    # train our model on dataset
    train(config, dataset, model)
```

```
Loading model...
Which epoch to load from? Choose from epochs below:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter 0 to train from scratch.
>> 10
Loading from checkpoint checkpoints/cnn/epoch=10.checkpoint.pth.tar
=> Successfully restored checkpoint (trained for 10 epochs)
Setting up interactive graph...
Epoch 10
        Validation Loss: 2.3131151782372634
        Validation Accuracy: 0.48717948717948717
        Train Loss: 0.007707070170436054
        Train Accuracy: 1.0
```




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

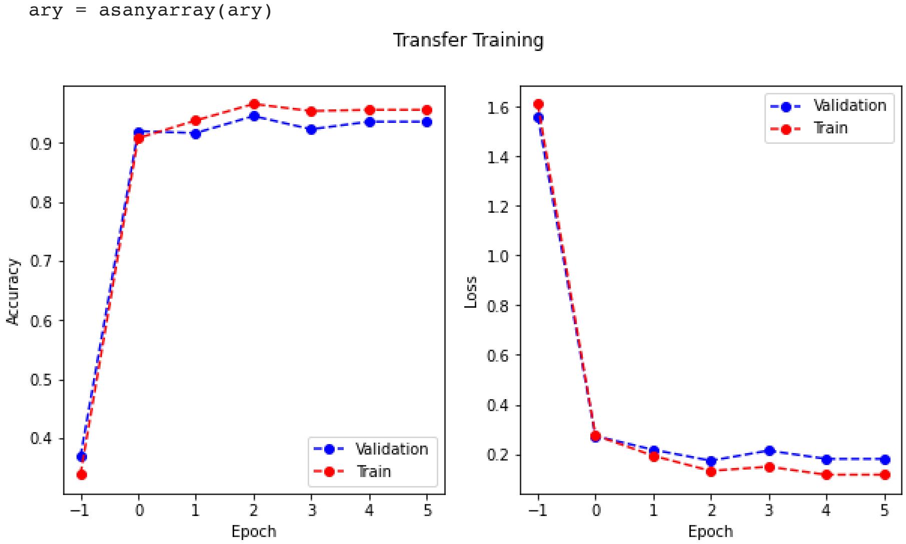
transfer.py

In [5]:

```
# EECS 545 Fall 2021
import torch
import torchvision.models as models
from dataset import DogDataset
from train import train


def load_pretrained(num_classes=5):
    """
    Load a ResNet-18 model from `torchvision.models` with pre-trained weights. Freeze all the parameters besides the
    final layer by setting the flag `requires_grad` for each parameter to False. Replace the final fully connected layer
    with another fully connected layer with `num_classes` many output units.
```

```
    Inputs:
        - num_classes: int
    Returns:
        - model: PyTorch model
    """
    # TODO (part f): load a pre-trained ResNet-18 model
    resnet18 = models.resnet18(pretrained=True)
    for param in resnet18.parameters():
        param.requires_grad = False

    # add a final fully connected layer
    num_ftrs = resnet18.fc.in_features
    resnet18.fc = torch.nn.Linear(num_ftrs, num_classes)
    return resnet18


if __name__ == '__main__':
    config = {
        'dataset_path': 'data/images/dogs',
        'batch_size': 4,
        'if_resize': False,
        'ckpt_path': 'checkpoints/transfer',
        'plot_name': 'Transfer',
        'num_epoch': 5,
        'learning_rate': 1e-3,
        'momentum': 0.9,
    }
    dataset = DogDataset(config['batch_size'], config['dataset_path'],config['if_resize'])
    model = load_pretrained()
    train(config, dataset, model)
```

```
Loading model...
Which epoch to load from? Choose from epochs below:
[0, 1, 2, 3, 4, 5]
Enter 0 to train from scratch.
>> 5
Loading from checkpoint checkpoints/transfer/epoch=5.checkpoint.pth.tar
=> Successfully restored checkpoint (trained for 5 epochs)
Setting up interactive graph...
Epoch 5
        Validation Loss: 0.18001348892740238
        Validation Accuracy: 0.9358974358974359
        Train Loss: 0.11598092106450349
        Train Accuracy: 0.956
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/numpy/core/shape_base.py:65: VisibleDeprecation
Warning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with diffe
rent lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```




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

imbalance.py

In [6]:

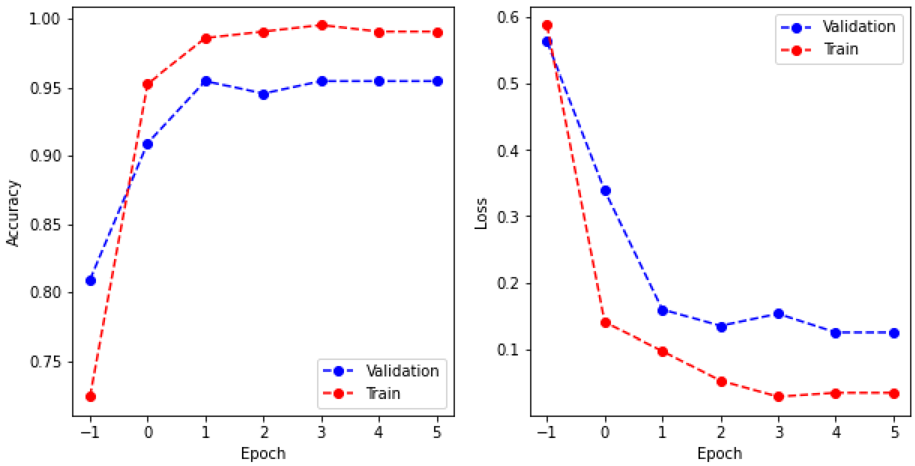
```
# EECS 545 Fall 2021
from dataset import DogCatDataset
from train import evaluate_loop, train
from transfer import load_pretrained

def per_class_accuracy(y_true, y_pred, num_classes=2):
    """
    Compute the per-class accuracy given true and predicted labels.
    Inputs:
        - y_true: true labels, tensor with shape=(num_examples)
```

```
        - y_pred: predicted labels, tensor with shape=(num_examples)
    Returns:
        - per_class_acc: per-class accuracy, list of floats
    """
    #
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    nlabel0 = 0
    nlabel1 = 0

    # # TODO (part h): compute the per-class accuracy
    for ii in range(y_true.shape[0]):
        if (y_true[ii] == 1):
            nlabel1 += 1
            if (y_pred[ii] == 1):
                TP += 1
            else:
                FN +=1
        else:
            nlabel0 += 1
            if (y_pred[ii] == 1):
                FP += 1
            else:
                TN +=1

    return [TN/nlabel0, TP/nlabel1]


def precision(y_true, y_pred):
    """
    Compute the precision given true and predicted labels. Treat the dog class (label=1) as the positive class.
    Precision = TP / (TP + FP)
    Inputs:
        - y_true: true labels, tensor with shape=(num_examples)
        - y_pred: predicted labels, tensor with shape=(num_examples)
    Returns:
        - prec: precision, float
    """
    #
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    nlabel0 = 0
    nlabel1 = 0

    # # TODO (part h): compute the per-class accuracy
    for ii in range(y_true.shape[0]):
        if (y_true[ii] == 1):
            nlabel1 += 1
            if (y_pred[ii] == 1):
                TP += 1
            else:
                FN +=1
        else:
            nlabel0 += 1
            if (y_pred[ii] == 1):
                FP += 1
            else:
                TN +=1
    return TP/(TP+FP)


def recall(y_true, y_pred):
    """
    Compute the recall given true and predicted labels. Treat the dog class (label=1) as the positive class.
    Recall = TP / (TP + FN)
    Inputs:
        - y_true: true labels, tensor with shape=(num_examples)
        - y_pred: predicted labels, tensor with shape=(num_examples)
    Returns:
        - rec: recall, float
    """
    # TODO (part h): compute the recall
    #
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    nlabel0 = 0
    nlabel1 = 0

    # # TODO (part h): compute the per-class accuracy
```

```python
    for ii in range(y_true.shape[0]):
        if (y_true[ii] == 1):
            nlabel1 += 1
            if (y_pred[ii] == 1):
                TP += 1
            else:
                FN +=1
        else:
            nlabel0 += 1
            if (y_pred[ii] == 1):
                FP += 1
            else:
                TN +=1
    return TP/(TP+FN)


def f1_score(y_true, y_pred):
    """
    Compute the f1-score given true and predicted labels. Treat the dog class (label=1) as the positive class.
    F1-score = 2 * (Precision * Recall) / (Precision + Recall)
    Inputs:
        - y_true: true labels, tensor with shape=(num_examples)
        - y_pred: predicted labels, tensor with shape=(num_examples)
    Returns:
        - f1: f1-score, float
    """
    # TODO (part h): compute the f1-score
    P = precision(y_true,y_pred)
    R = recall(y_true, y_pred)
    return 2*(P*R)/(P+R)


def compute_metrics(dataset, model):
    y_true, y_pred, _ = evaluate_loop(dataset.val_loader, model)
    print('Per-class accuracy: ', per_class_accuracy(y_true, y_pred))
    print('Precision: ', precision(y_true, y_pred))
    print('Recall: ', recall(y_true, y_pred))
    print('F1-score: ', f1_score(y_true, y_pred))


if __name__ == '__main__':
    # model with normal cross-entropy loss
    config = {
        'dataset_path': 'data/images/dogs_vs_cats_imbalance',
        'batch_size': 4,
        'ckpt_force': True,
        'ckpt_path': 'checkpoints/imbalance',
        'plot_name': 'Imbalance',
        'num_epoch': 5,
        'learning_rate': 1e-3,
        'momentum': 0.9,
    }
    dataset = DogCatDataset(config['batch_size'], config['dataset_path'])
    model = load_pretrained(num_classes=2)
    train(config, dataset, model)
    compute_metrics(dataset, model)

    # model with weighted cross-entropy loss
    config = {
        'ckpt_path': 'checkpoints/imbalance_weighted',
        'plot_name': 'Imbalance-Weighted',
        'num_epoch': 5,
        'learning_rate': 1e-3,
        'momentum': 0.9,
        'use_weighted': True,
    }
    model_weighted = load_pretrained(num_classes=2)
    train(config, dataset, model_weighted)
    compute_metrics(dataset, model_weighted)
```

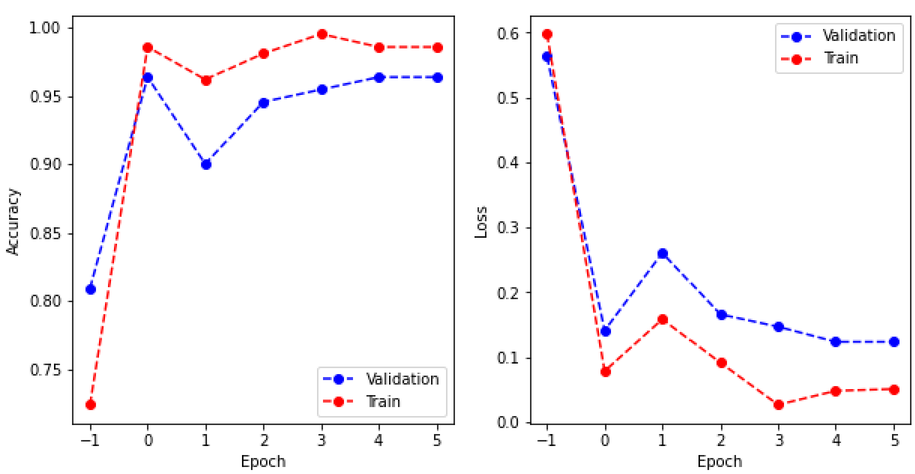
```
Loading model...
Which epoch to load from? Choose from epochs below:
[1, 2, 3, 4, 5]
>> 5
Loading from checkpoint checkpoints/imbalance/epoch=5.checkpoint.pth.tar
=> Successfully restored checkpoint (trained for 5 epochs)
Setting up interactive graph...
Epoch 5
        Validation Loss: 0.12463922413404692
        Validation Accuracy: 0.9545454545454546
        Train Loss: 0.034303230836632705
        Train Accuracy: 0.9904761904761905
```

Imbalance Training




```
Finished Training
<Figure size 432x288 with 0 Axes>
Per-class accuracy:  [1.0, 0.5]
Precision:  1.0
Recall:  0.5
F1-score:  0.6666666666666666
Loading model...
Which epoch to load from? Choose from epochs below:
[0, 1, 2, 3, 4, 5]
Enter 0 to train from scratch.
>> 5
Loading from checkpoint checkpoints/imbalance_weighted/epoch=5.checkpoint.pth.tar
=> Successfully restored checkpoint (trained for 5 epochs)
Setting up interactive graph...
Epoch 5
        Validation Loss: 0.12327551427720622
        Validation Accuracy: 0.9636363636363636
        Train Loss: 0.05039005388971418
        Train Accuracy: 0.9857142857142858
```

Imbalance-Weighted Training




```
Finished Training
<Figure size 432x288 with 0 Axes>
Per-class accuracy:  [0.98, 0.8]
Precision:  0.8
Recall:  0.8
F1-score:  0.8000000000000002
```