DEEP LEARNING ASSIGNMENT-1 REPORT

Part-1 SVM and Softmax

Observations

- For CIFAR-Dataset because of a lot of data variations the accuracy on CIFAR dataset is less compare to Mushroom dataset
- As CIFAR dataset is a large dataset, the model took more time to train.
- Softmax tries to minimize the error as much as possible but where as in svm we use hinge loss which stops training when the error is zero. So in case of cifar dataset softmax overfits so the accuracy is even worse.

Results

Classifier	CIFAR-Test	Mushroom-Test
SVM	22.70	79.38
Softmax	10.00	83.50

Part-2 Multilayer Classification

Model used :

Multi-layer fully connected neural network

Dimensions:

Input - `N X D` where N is the number of images

`Hi` : The number of neurons in the ith hidden layer

`Wi : H[i-1] X H[i]` : ith layer weights

`bi : H[i] X 1` : ith layer biases

`Zi : H[i] X N` : Linear output of the ith layer

`Ai : H[i] X N` : Output after activation in the ith layer

Classification performed over C classes

Structure of the model:

L fully connected layers
ReLu Activation function for the first L-1 layers (to introduce non-linearity into the network)
Softmax activation for the last layer, which outputs the scores for each class

Input -> [Linear -> ReLu](L-1 times) -> LINEAR -> SOFTMAX -> Output

Loss function used:

Cross entropy loss with L2 regularization

Implementation:

Initializing the parameters

- Initializing the parameters (weights and biases) Wi and bi for the L layers

Training the model

- Forward propagation module
    - Linear output Z followed by ReLu activation function for the first L-1 layers
    - Linear output Z followed by Softmax activation function for the last layer, which gives the scores for each class for all data samples as a matrix scores where scores[i, c] is the score for class c on the input X[i]
- Computing the loss function at the final layer
- Backward propagation module :
    - Compute the gradients for the Wi parameters at each layer
    - Update the parameters with the gradients obtained using SGD

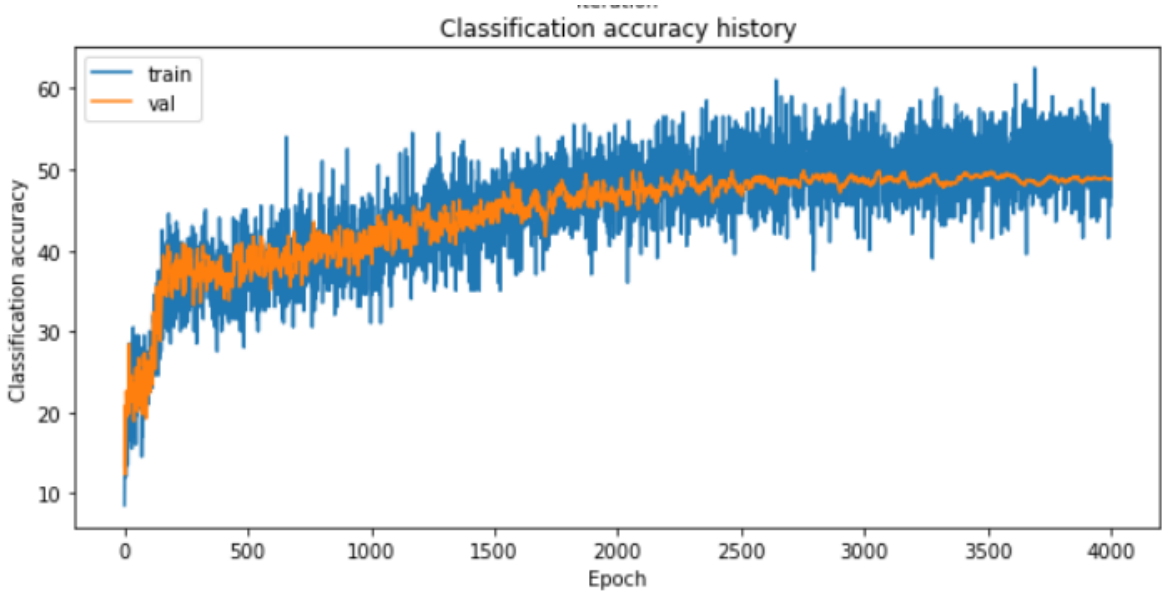
Testing

- With the trained parameters obtained, make a forward pass for the test set to obtain the output scores for each class and make appropriate predictions.

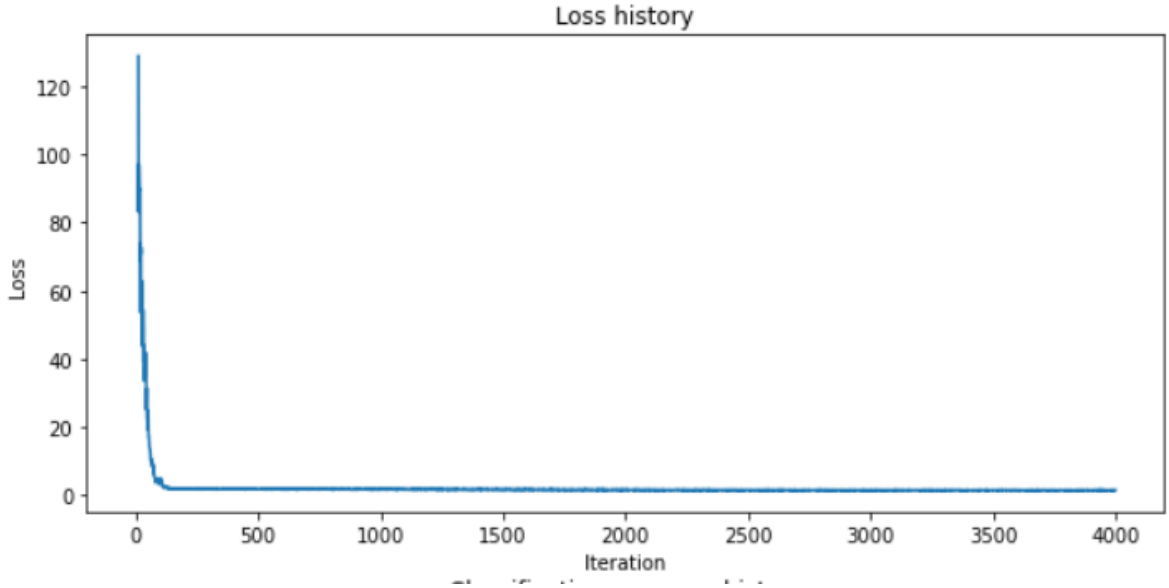
Observations and Results:

Accuracy on the test set obtained for SGD optimizer : 49.19%

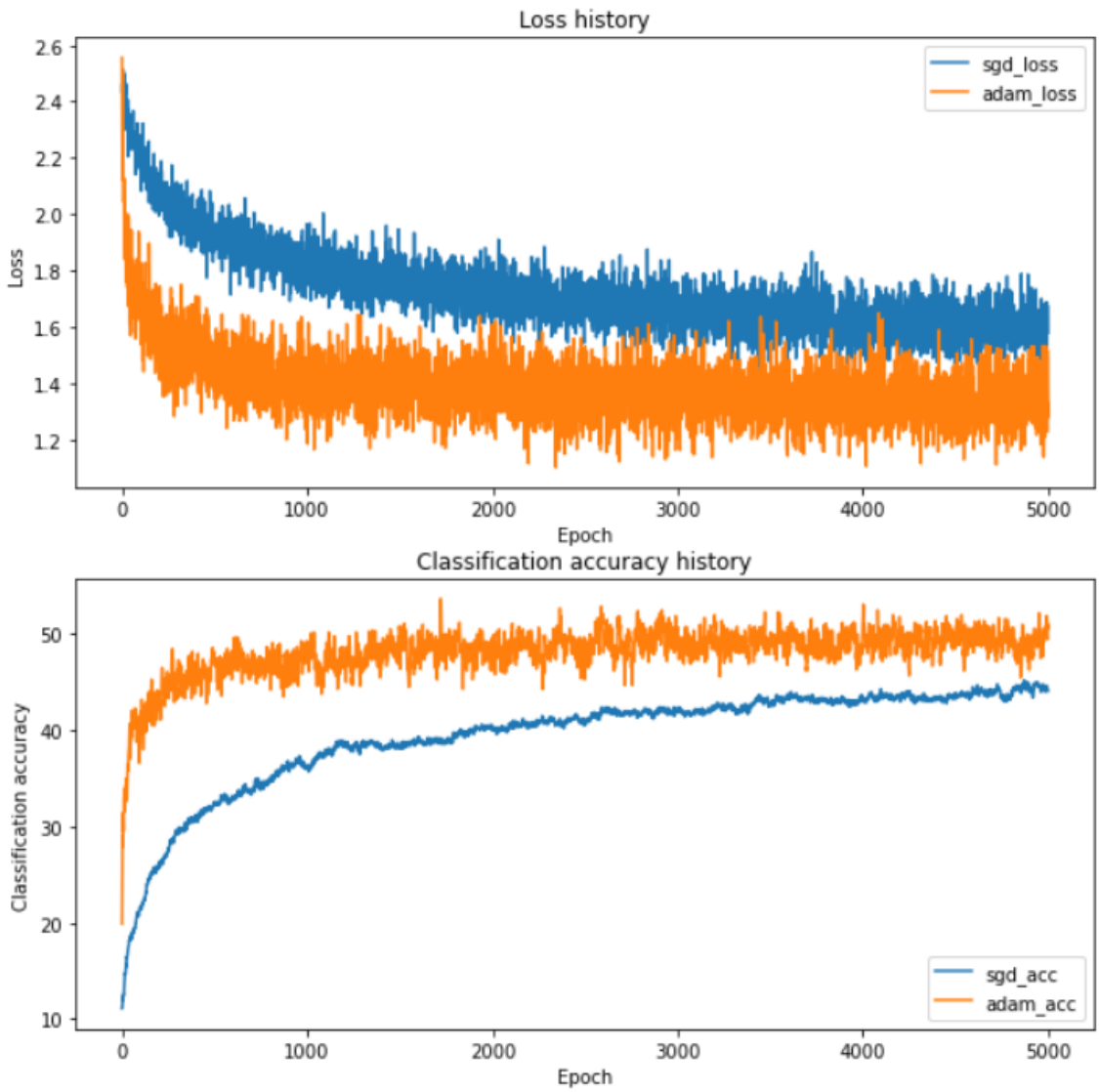
Variation of Classification accuracy with epochs for SGD



Variation of loss with iterations (SGD)



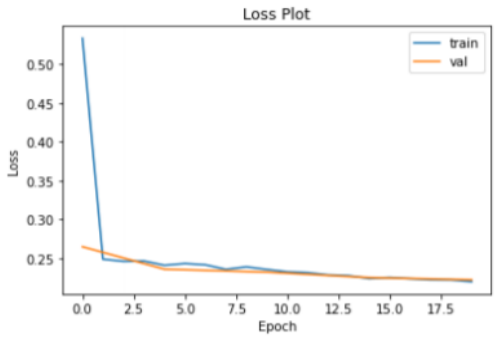
Comparing SGD optimizer with Adam Optimizer



Part-3 Multi-label Image Classification

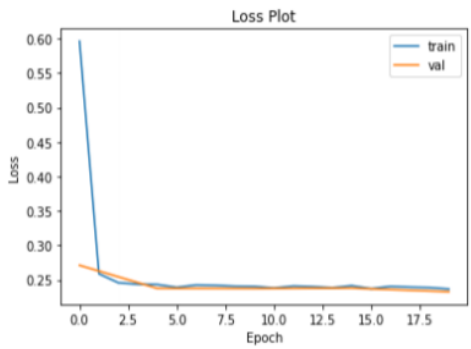
<u>Part 3A - Predefined Models</u>
Observations

Simple classifier:



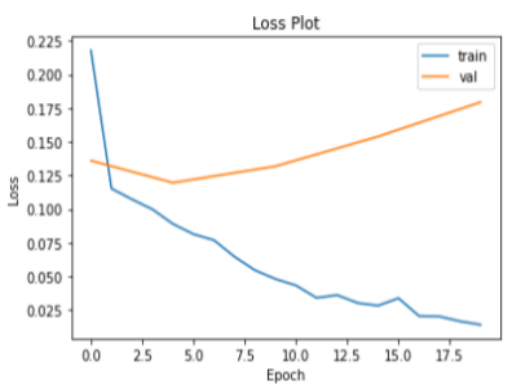
Here there is no gap between the train and validation error. This means we need to increase the capacity of the algorithm.

AlexNet :



Here there is no gap between the train and validation error. This means we need to increase the capacity of the algorithm.

Pretrained AlexNet :



Here the intersection of graphs says that the validation set is too small and statistics are not meaningful.

<u>Part 3B - Self models</u>

After studying VGG,AlexNet we tried some Conv layers and Fully connected layers we tried with different max pooling layers and COnv layers and Fully connected layers

```
class Classifier(nn.Module):
    class Classifier(nn.Module):
     # TODO: implement me
     def init(self):
         super(Classifier, self).init()
         self.features = nn.Sequential(nn.Conv2d(3, 64, 3),
                                       nn.Conv2d(64, 64, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(64, 128, 3),
                                       nn.Conv2d(128, 128, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(128, 256, 3),
                                       nn.Conv2d(256, 256, 3),
                                       nn.Conv2d(256, 256, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(256, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       )
         self.para = nn.Sequential(
             nn.Dropout(),
             nn.ReLU(inplace=True),
             nn.Dropout(),
             nn.Linear(4096, 4096),
             nn.ReLU(inplace=True),
             nn.Linear(4096, NUM_CLASSES),
         )

     def forward(self, x):
         x = self.features(x)
         x = torch.flatten(x)
         x = self.para(x)
         return x
```

Individual Contributions

Korupolu Saideepthi - (S20180010087)
Part1 -Softmax
Part3A - Predefined Models

Varakala Sowmya - (S201800100187)
Part1- SVM
Part2 -Multilayer Classification -ForwardPass

ManjuShree - (S20180010055)
Part3A -Predefined Models
Part3B - Developing Your Own Classifier

Swathi k - (S20180010172)
Part2 - Multilayer Classification - BackwardPass
Part3B - Developing Your Own Classifier

Sai deepthi(S20180010087)

Sowmya(S20180010187)

Manju shree(S20180010055)

Swathi(S20180010172)

In [1]:
```
import random
import numpy as np
from data_process import get_CIFAR10_data, get_MUSHROOM_data
from scipy.spatial import distance
#from models import Perceptron, SVM, Softmax
from models import SVM, Softmax
%matplotlib inline
```

Loading CIFAR-10

In the following cells we determine the number of images for each split and load the images.
TRAIN_IMAGES + VAL_IMAGES = (0, 50000], TEST_IMAGES = 10000

In [2]:
```
# You can change these numbers for experimentation
# For submission we will use the default values
TRAIN_IMAGES = 40000
VAL_IMAGES = 10000
```

In [3]:
```
data = get_CIFAR10_data(TRAIN_IMAGES, VAL_IMAGES)
X_train_CIFAR, y_train_CIFAR = data['X_train'], data['y_train']
X_val_CIFAR, y_val_CIFAR = data['X_val'], data['y_val']
X_test_CIFAR, y_test_CIFAR = data['X_test'], data['y_test']
n_class_CIFAR = len(np.unique(y_test_CIFAR))
```

Convert the sets of images from dimensions of **(N, 3, 32)** -> **(N, 3072)** where N is the number of images so that each **3x32x32** image is represented by a single vector.

In [4]:
```
X_train_CIFAR = np.reshape(X_train_CIFAR, (X_train_CIFAR.shape[0], -1))
X_val_CIFAR = np.reshape(X_val_CIFAR, (X_val_CIFAR.shape[0], -1))
X_test_CIFAR = np.reshape(X_test_CIFAR, (X_test_CIFAR.shape[0], -1))
```

Loading Mushroom

In the following cells we determine the splitting of the mushroom dataset.
TRAINING + VALIDATION = 0.8, TESTING = 0.2

In [5]:
```
# TRAINING = 0.6 indicates 60% of the data is used as the training dataset.
VALIDATION = 0.2
```

In [6]:
```
data = get_MUSHROOM_data(VALIDATION)
X_train_MR, y_train_MR = data['X_train'], data['y_train']
X_val_MR, y_val_MR = data['X_val'], data['y_val']
X_test_MR, y_test_MR = data['X_test'], data['y_test']
n_class_MR = len(np.unique(y_test_MR))

print("Number of train samples: ", X_train_MR.shape[0])
print("Number of val samples: ", X_val_MR.shape[0])
print("Number of test samples: ", X_test_MR.shape[0])
```

```
Number of train samples:  4874
Number of val samples:  1625
Number of test samples:  1625
```

Get Accuracy

This function computes how well your model performs using accuracy as a metric.

In [7]:
```
def get_acc(pred, y_test):
    return np.sum(y_test==pred)/len(y_test)*100
```

Support Vector Machines (with SGD)

First, you will implement a "soft margin" SVM. In this formulation you will maximize the margin between positive and negative training examples and penalize margin violations using a hinge loss.

We will optimize the SVM loss using SGD. This means you must compute the loss function with respect to model weights. You will use this gradient to update the model weights.

SVM optimized with SGD has 3 hyperparameters that you can experiment with :

- **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update.
- **Epochs** - similar to as defined above in Perceptron.
- **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case it is a coefficient on the term which maximizes the margin. You could try different values. The default value is set to 0.05.

You will implement the SVM using SGD in the **models/SVM.py**

The following code:

- Creates an instance of the SVM classifier class
- The train function of the SVM class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

Train SVM on CIFAR

In [37]:
```
lr = 0.01
n_epochs = 500
reg_const = 0.05

svm_CIFAR = SVM(n_class_CIFAR, lr, n_epochs, reg_const)
svm_CIFAR.train(X_train_CIFAR, y_train_CIFAR)
```

In [38]:
```
pred_svm = svm_CIFAR.predict(X_train_CIFAR)
print("The training accuracy is given by: %f" % (get_acc(pred_svm, y_train_CIFAR)))
```

```
The training accuracy is given by: 21.625000
```

Validate SVM on CIFAR

In [39]:
```
pred_svm = svm_CIFAR.predict(X_val_CIFAR)
print("The validation accuracy is given by: %f" % (get_acc(pred_svm, y_val_CIFAR)))
```

```
The validation accuracy is given by: 21.370000
```

Test SVM on CIFAR

In [40]:
```
pred_svm = svm_CIFAR.predict(X_test_CIFAR)
print("The testing accuracy is given by: %f" % (get_acc(pred_svm, y_test_CIFAR)))
```

```
The testing accuracy is given by: 21.700000
```

Train SVM on Mushroom

In [12]:
```
lr = 0.01
n_epochs = 100
reg_const = 0.05

svm_MR = SVM(n_class_MR, lr, n_epochs, reg_const)
svm_MR.train(X_train_MR, y_train_MR)
```

In [13]:
```
pred_svm = svm_MR.predict(X_train_MR)
print("The training accuracy is given by: %f" % (get_acc(pred_svm, y_train_MR)))
```

```
The training accuracy is given by: 79.154698
```

Validate SVM on Mushroom

In [14]:
```
pred_svm = svm_MR.predict(X_val_MR)
print("The validation accuracy is given by: %f" % (get_acc(pred_svm, y_val_MR)))
```

```
The validation accuracy is given by: 77.946154
```

Test SVM on Mushroom

In [15]:
```
pred_svm = svm_MR.predict(X_test_MR)
print("The testing accuracy is given by: %f" % (get_acc(pred_svm, y_test_MR)))
```

```
The testing accuracy is given by: 79.384615
```

Softmax Classifier (with SGD)

Next, you will train a Softmax classifier. This classifier consists of a linear function of the input data followed by a softmax function which outputs a vector of dimension C (number of classes) for each data point. Each entry of the softmax output vector corresponds to a confidence in one of the C classes, and like a probability distribution, the entries of the output vector sum to 1. We use a cross-entropy loss on this softmax output to train the model.

Check the following link as an additional resource on softmax classification: http://cs231n.github.io/linear-classify#softmax

Once again we will train the classifier with SGD. This means you need to compute the gradients of the softmax cross-entropy loss function according to the weights and update the weights using this gradient. Check the following link to help with implementing the gradient updates: https://deepnotes.io/softmax-crossentropy

The softmax classifier has 3 hyperparameters that you can experiment with :

- **Learning rate** - As above, this controls how much the model weights are updated with respect to their gradient.
- **Number of Epochs** - As described for perceptron.
- **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case, we minimize the L2 norm of the model weights as regularization, so the regularization constant is a coefficient on the L2 norm in the combined cross-entropy and regularization objective.

You will implement a softmax classifier using SGD in the **models/Softmax.py**

The following code:

- Creates an instance of the Softmax classifier class
- The train function of the Softmax class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

Train Softmax on CIFAR

In [57]:
```
lr = 0.001
n_epochs = 500
reg_const = 0.05

softmax_CIFAR = Softmax(n_class_CIFAR, lr, n_epochs, reg_const)
softmax_CIFAR.train(X_train_CIFAR, y_train_CIFAR)
```

In [58]:
```
pred_softmax = softmax_CIFAR.predict(X_train_CIFAR)
print("The training accuracy is given by: %f" % (get_acc(pred_softmax, y_train_CIFAR)))
```

```
The training accuracy is given by: 9.965000
```

Validate Softmax on CIFAR

In [61]:
```
pred_softmax = softmax_CIFAR.predict(X_val_CIFAR)
print("The validation accuracy is given by: %f" % (get_acc(pred_softmax, y_val_CIFAR)))
```

```
The validation accuracy is given by: 10.140000
```

Testing Softmax on CIFAR

In [62]:
```
pred_softmax = softmax_CIFAR.predict(X_test_CIFAR)
print("The testing accuracy is given by: %f" % (get_acc(pred_softmax, y_test_CIFAR)))
```

```
The testing accuracy is given by: 10.000000
```

Train Softmax on Mushroom

In [53]:
```
lr = 0.001
n_epochs = 100
reg_const = 0.05

softmax_MR = Softmax(n_class_MR, lr, n_epochs, reg_const)
softmax_MR.train(X_train_MR, y_train_MR)
```

Out[53]:
```
[0.68306643181027243,
 1.2552566030485784,
 1.0455000000000000,
 1.1841290133814442,
 1.6050499984967556,
 0.60305877992039447,
 0.77143720912774135,
 1.7859795087645737,
 1.2086218749649511,
 0.38419288642825236,
 0.30327054994411603,
 0.90000000000000000,
 0.95430093430314,
 1.2331360057289961,
 0.29515707972222781,
 0.58804397248078864,
 0.52977765380017,
 0.15842831840616398,
 0.33044325000000000,
 1.1439642856640125,
 0.43344495689132227,
 2.3343807731884463,
 0.81582281857612286,
 0.79517227051133395,
 1.1237829074856395,
 0.27174862542528075,
 0.57248248849960661,
 0.29000000000000000,
 1.2365481338854977,
 3.6384477534136336,
 0.55139380448387774,
 0.14940931289777751,
 0.96388188173343909,
 1.3536329392800657,
 2.5514373424164495,
 1.7221338117863454,
 0.35000000000000000,
 0.99544112630895163,
 1.9157389704538719,
 0.14971887523967567,
 0.83473863655539177,
 1.9297967185645155,
 0.68953424000845115,
 0.12618029862507136,
 0.53270365188282613,
 0.45000000000000000,
 1.0979636157765,
 1.7861898643069745,
 1.2812347918534985,
 0.64388537296916864,
 0.51020124921207,
 0.39601921088929033,
 0.45000000000000000,
 0.26700629186660986,
 0.41470780139606847,
 1.3905438114426602,
 0.86330865671687456,
 1.3495510027819306,
 1.4530775117750268,
 1.6332926527514487,
 0.75430952368202699,
 0.45000000000000000,
 0.32734059394645937,
 0.14592419036323574,
 0.61606421326365502,
 0.47338387070171112,
 0.45623754509282672,
 0.42244709662245572,
 0.13864215417675215,
 1.3904232734997713,
 0.55511258809977537,
 3.6270026466609739,
 1.5579489537511055,
 2.4203477483464137,
 0.23754428659745566,
 1.2754739746438041,
 0.32413207994493483,
 0.45000000000000000,
 0.46386264606474853,
 0.30720980134077813,
 1.8173504685785647,
 0.66521998692203661,
 1.2419645361187991,
 0.09629037326923119,
 0.13964658849368,
 0.12630664755343954,
 0.45000000000000000,
 0.90999951628988853,
 0.18941378238559549,
 0.77253904337235655,
 0.54803847131803878,
 1.9822738706754113,
 0.36126025725154746,
 0.45000000000000000,
 0.54254112587672111,
 0.21189629122734907,
 0.51464554729384464,
 0.70754313366308353,
 0.46659245738435111,
 0.18484160098976787]
```

In [54]:
```
pred_softmax = softmax_MR.predict(X_train_MR)
print("The training accuracy is given by: %f" % (get_acc(pred_softmax, y_train_MR)))
```

```
The training accuracy is given by: 83.709479
```

Validate Softmax on Mushroom

In [55]:
```
pred_softmax = softmax_MR.predict(X_val_MR)
print("The validation accuracy is given by: %f" % (get_acc(pred_softmax, y_val_MR)))
```

```
The validation accuracy is given by: 82.030769
```

Testing Softmax on Mushroom

In [56]:
```
pred_softmax = softmax_MR.predict(X_test_MR)
print("The testing accuracy is given by: %f" % (get_acc(pred_softmax, y_test_MR)))
```

```
The testing accuracy is given by: 83.507692
```

In [ ]:

In [ ]:

In [ ]:

Assignment 1 - part 2

Sai deepthi(S20180010087)

Sowmya(S20180010187)

Manjju shree(S20180010055)

Swathi(S20180010172)

In [15]:
```
import matplotlib.pyplot as plt
import numpy as np

from models.neural_net import NeuralNetwork
from utils.data_process import get_CIFAR10_data

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 10.0)  # set default size of pl
ots

# For auto-reloading external modules
# See http://stackoverflow.com/questions/1907993/autoreload-of-modules-
in-ipython
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Loading CIFAR-10

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Now that you have implemented a neural network that passes gradient checks and works on toy data, you will test your network on the CIFAR-10 dataset.

In [16]:
```
# You can change these numbers for experimentation
# For submission be sure they are set to the default values
TRAIN_IMAGES = 49000
VAL_IMAGES = 1000
TEST_IMAGES = 10000

data = get_CIFAR10_data(TRAIN_IMAGES, VAL_IMAGES, TEST_IMAGES)
X_train, y_train = data['X_train'], data['y_train']
X_val, y_val = data['X_val'], data['y_val']
X_test, y_test = data['X_test'], data['y_test']
```

In [17]:
```
def get_acc(pred, y_test):
    return np.sum(y_test==pred)/len(y_test)*100
```

Train using SGD

To train our network we will use SGD. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

You can try different numbers of layers and other hyperparameters on the CIFAR-10 dataset below.

In [18]:
```
# Hyperparameters

input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 110
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 4000
batch_size = 200
learning_rate = 7e-2
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
learning_rate_decay = 0.9
regularization =  1e-2

def train(hidden_sizes, num_layers, learning_rate, learning_rate_decay,
regularization, optimizer = "SGD"):
    # Initialize a new neural network model
    net = NeuralNetwork(input_size, hidden_sizes, num_classes, num_laye
rs)

    # Variables to store performance for each epoch
    train_loss = np.zeros(epochs)
    train_accuracy = np.zeros(epochs)
    val_accuracy = np.zeros(epochs)


    # For each epoch...
    for epoch in tqdm(range(epochs)):

        batch_idx = np.random.choice(TRAIN_IMAGES, batch_size, replace=
True)

        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]

        y_pred = np.argmax(net.forward(X_batch), axis = 1)
        train_accuracy[epoch] = get_acc(y_pred, y_batch)
        train_loss[epoch] = net.backward(X_batch, y_batch, learning_rat
e, regularization, mode = optimizer)

        # Validation
        # No need to run the backward pass here, just run the forward p
ass to compute accuracy
        y_val_pred = np.argmax(net.forward(X_val), axis = 1)
        val_accuracy[epoch] = get_acc(y_val_pred, y_val)

        if epoch % 50 == 0:
             learning_rate *= learning_rate_decay

    return net, train_loss, train_accuracy, val_accuracy
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
net, train_loss, train_accuracy, val_accuracy = train(hidden_sizes, num
_layers, learning_rate, learning_rate_decay, regularization, "Adam")
```

```
  0%|          | 1/4000 [00:00<45:41,  1.46it/s]C:\Users\MYPC\Documents
\Bachelors\sem6\DL\assignment1-part2\models\neural_net.py:158: RuntimeW
arning: divide by zero encountered in log
  Total loss for this batch of training samples
100%|██████████| 4000/4000 [03:44<00:00, 17.80it/s]
```

In [19]:

```
y_test_pred = np.argmax(net.forward(X_test), axis = 1)
test_acc = get_acc(y_test_pred, y_test)
print(test_acc)
```

```
49.19
```

Graph loss and train/val accuracies

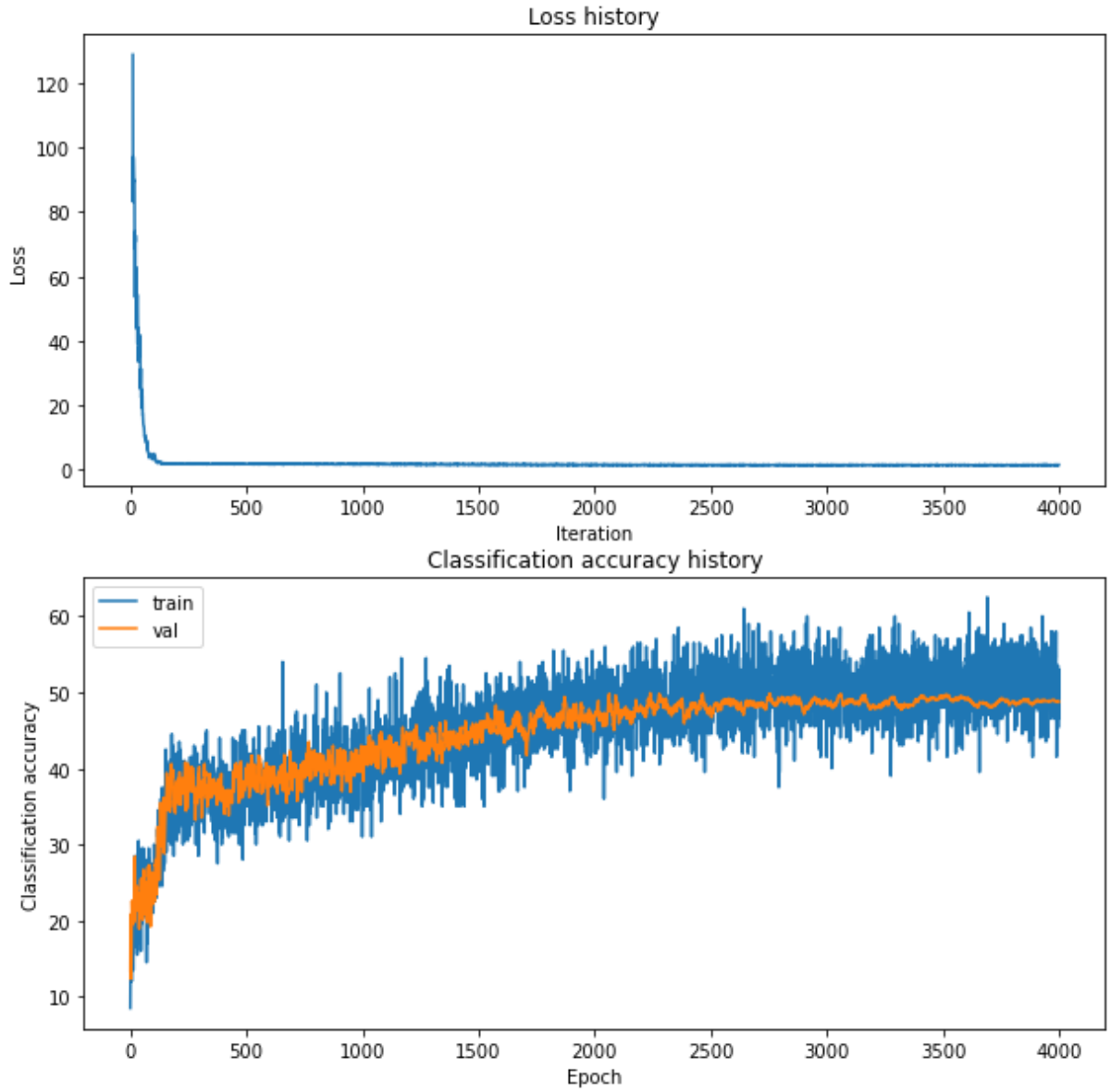
Examining the loss graph along with the train and val accuracy graphs should help you gain some intuition for the hyperparameters you should try in the hyperparameter tuning below. It should also help with debugging any issues you might have with your network.

In [20]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy, label='train')
plt.plot(val_accuracy, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



Train using Adam

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Next we will train the same model using the Adam optimizer. You should take the above code for SGD and modify it to use Adam instead. For implementation details, see the lecture slides. The original paper that introduced Adam is also a good reference, and contains suggestions for default values: https://arxiv.org/pdf/1412.6980.pdf

In [21]:
```
# TODO: implement me
```

Hyperparameter tuning

Once you have successfully trained a network you can tune your hyparameters to increase your accuracy.

Based on the graphs of the loss function above you should be able to develop some intuition about what hyperparameter adjustments may be necessary. A very noisy loss implies that the learning rate might be too high, while a linearly decreasing loss would suggest that the learning rate may be too low. A large gap between training and validation accuracy would suggest overfitting due to large model without much regularization. No gap between training and validation accuracy would indicate low model capacity.

You will compare networks of two and three layers using the different optimization methods you implemented.

The different hyperparameters you can experiment with are:

- **Batch size**: We recommend you leave this at 200 initially which is the batch size we used.
- **Number of iterations**: You can gain an intuition for how many iterations to run by checking when the validation accuracy plateaus in your train/val accuracy graph.
- **Initialization** Weight initialization is very important for neural networks. We used the initialization `W = np.random.randn(n) / sqrt(n)` where `n` is the input dimension for layer corresponding to `W`. We recommend you stick with the given initializations, but you may explore modifying these. Typical initialization practices: http://cs231n.github.io/neural-networks-2/#init
- **Learning rate**: Generally from around 1e-4 to 1e-1 is a good range to explore according to our implementation.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

- **Learning rate decay**: We recommend a 0.95 decay to start.
- **Hidden layer size**: You should explore up to around 120 units per layer. For three-layer network, we fixed the two hidden layers to be the same size when obtaining the target numbers. However, you may experiment with having different size hidden layers.
- **Regularization coefficient**: We recommend trying values in the range 0 to 0.1.

Hints:

- After getting a sense of the parameters by trying a few values yourself, you will likely want to write a few for-loops to traverse over a set of hyperparameters.
- If you find that your train loss is decreasing, but your train and val accuracy start to decrease rather than increase, your model likely started minimizing the regularization term. To prevent this you will need to decrease the regularization coefficient.

In [22]:
```
input_size = 32 * 32 * 3
num_layers = 2
num_classes = 10
epochs = 2000
batch_size = 200

# lrs= [1e-3]
lrs = [7e-2]
# hidden_size = [100, 110]
hidden_size = [200]
lrds = [0.9]
# lrds = [0.95, 0.9, 0.94]
regs = [1e-2]
# regs = [3e-2, 1e-1, 5e-2]

best_acc = 0
best_net = None

results = {}
for _ in range(3):
    for hs in hidden_size:
        for lr in lrs:
            for lrd in lrds:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                for reg in regs:
                    hidden_sizes = [hs] * (num_layers - 1)
                    net, _, _, _ = train(hidden_sizes, num_layers, lr,
lrd, reg, optimizer="Adam")
                    y_val_pred = np.argmax(net.forward(X_val), axis = 1
)
                    val_acc= get_acc(y_val_pred, y_val)
                    if val_acc > best_acc:
                        best_net = net
                        best_acc = val_acc
                    results[(hs, lr, lrd, reg)] = val_acc
                    print('hs %d, lr %e, lrd %f, reg %e, val accuracy:
%f' % (hs, lr, lrd, reg,  val_acc))
#     for r in results:
#         (hs, lr, lrd, reg) = r
#         val_acc = results[r]
#         print('hs %d, lr %e, lrd %f, reg %e, val accuracy: %f' % (hs,
lr, lrd, reg,  val_acc))
print("Best accuracy:", best_acc)
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]C:\Users\MYPC\Documents\Bachelo
rs\sem6\DL\assignment1-part2\models\neural_net.py:158: RuntimeWarning:
divide by zero encountered in log
  Total loss for this batch of training samples
100%|██████████| 2000/2000 [03:21<00:00,  9.93it/s]
  0%|          | 2/2000 [00:00<02:20, 14.22it/s]

hs 200, lr 7.000000e-02, lrd 0.900000, reg 1.000000e-02, val accuracy:
47.000000

100%|██████████| 2000/2000 [03:24<00:00,  9.77it/s]
  0%|          | 1/2000 [00:00<04:03,  8.22it/s]

hs 200, lr 7.000000e-02, lrd 0.900000, reg 1.000000e-02, val accuracy:
47.700000

100%|██████████| 2000/2000 [03:12<00:00, 10.37it/s]

hs 200, lr 7.000000e-02, lrd 0.900000, reg 1.000000e-02, val accuracy:
47.700000
Best accuracy: 47.699999999999996
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [23]:
```
y_test_pred = np.argmax(best_net.forward(X_test), axis = 1)
test_acc= get_acc(y_test_pred, y_test)
print(test_acc)
```

```
47.64
```

Run on the test set

When you are done experimenting, you should evaluate your final trained networks on the test set.

In [24]:
```
best_2layer_sgd_prediction = None
best_3layer_sgd_prediction = None
best_2layer_adam_prediction = None
best_3layer_adam_prediction = None
```

Compare SGD and Adam

Create graphs to compare training loss and validation accuracy between SGD and Adam. The code is similar to the above code, but instead of comparing train and validation, we are comparing SGD and Adam.

In [26]:
```
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 110
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 5000
batch_size = 200
learning_rate = 1e-3
learning_rate_decay = 1
regularization =  1e-2
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sgd_loss = np.zeros(epochs)
adam_loss = np.zeros(epochs)
sgd_accuracy = np.zeros(epochs)
adam_accuracy = np.zeros(epochs)

_, sgd_loss, _, sgd_accuracy = train(hidden_sizes, num_layers, learning
_rate, learning_rate_decay, regularization, "SGD")
_, adam_loss, _, adam_accuracy = train(hidden_sizes, num_layers, learni
ng_rate, learning_rate_decay, regularization, "Adam")
```

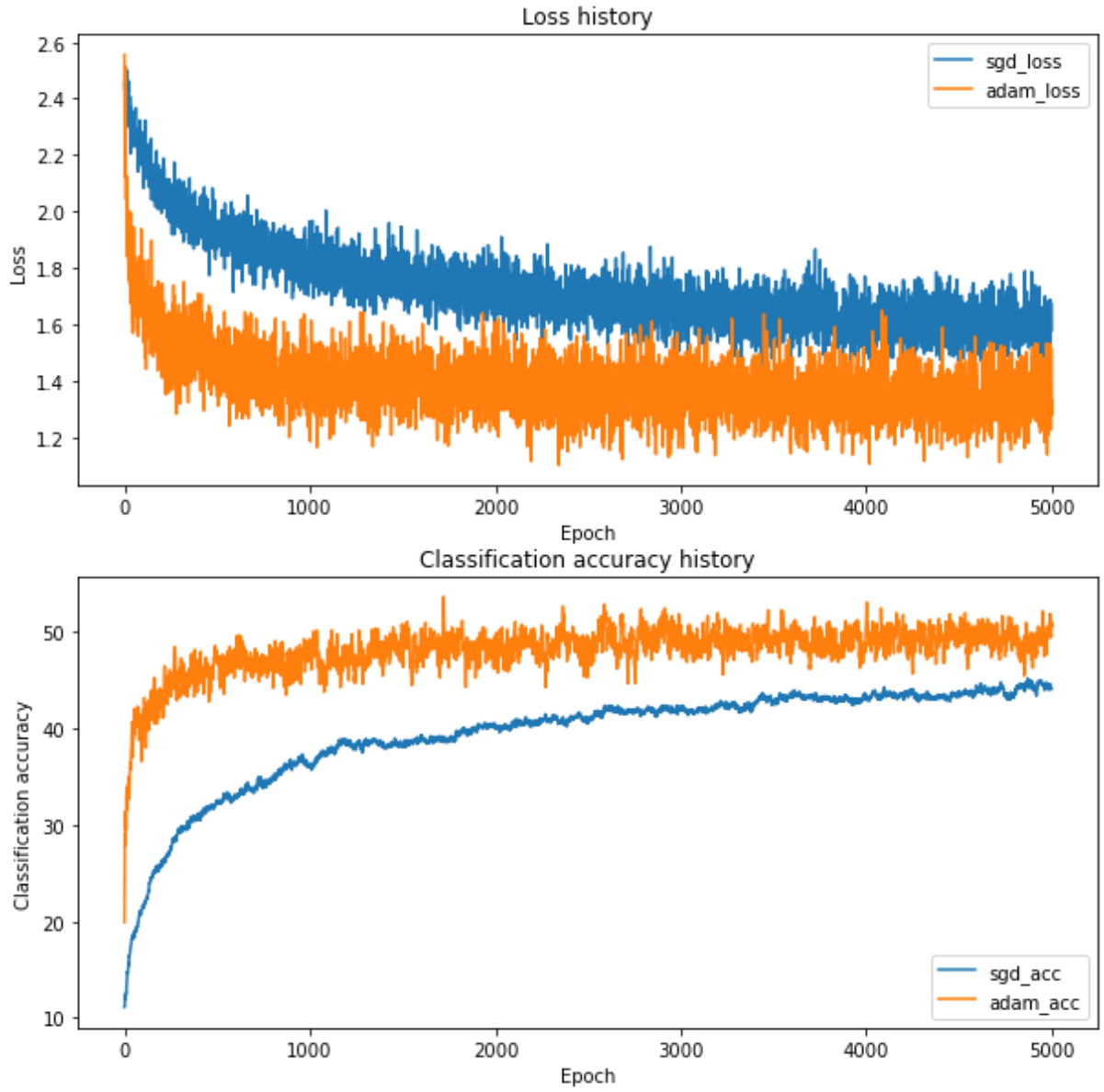
```
100%|██████████| 5000/5000 [03:14<00:00, 25.66it/s]
100%|██████████| 5000/5000 [04:45<00:00, 17.53it/s]
```

In [27]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(sgd_loss, label='sgd_loss')
plt.plot(adam_loss, label='adam_loss')
plt.title('Loss history')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(sgd_accuracy, label='sgd_acc')
plt.plot(adam_accuracy, label='adam_acc')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
In [ ]:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Assignment 1 Part 1b Introduction: Multi-label Image Classification

Jai Vaishnav

Kolkata

Jadavpur University

Batch 2

CSE


```
[1] !pip install -qqq wget
    !pip install -qqq torch torchvision torchmetric
    import os, sys, shutil
    os.environ['LC_ALL'] = 'C.UTF-8'
    os.environ['LANG'] = 'C.UTF-8'
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/classes.txt
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/bbox_labels_600_hierarchy.txt
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/class-descriptions-boxable.txt
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/train/annotations-human-bbox.csv
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/validation/annotations-human-bbox.csv
    !wget -qqq https://raw.githubusercontent.com/jcxldn/CVPR-2020-Open-Images-Challenge-Baseline/master/dataset/test/annotations-human-bbox.csv
    !pip -qqq install -U scikit-learn scipy matplotlib
    !echo "oiscript1.train.1000 -c /tmp/train-images-boxable.txt -d dataset/train/data" > class.txt
    !echo "oiscript1.validation -c /tmp/validation-images-boxable.txt -d dataset/validation/data" >> class.txt
    !echo "oiscript1.test -c /tmp/test-images-boxable.txt -d dataset/test/data" >> class.txt
    !cat class.txt
    !sed -i "s/^/ln -s /" class.txt
    !sed -i "s/$/ /" class.txt
    !mkdir -p dataset/{train,validation,test}/data
    !chmod +x class.txt
    !./class.txt
    !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Multi-label Classification

In this assignment, we will be working with the multi-label classification on Open Images. We will begin by looking at the dataset and its various annotations, and then we will be training a deep neural network to predict the labels for a given image. Finally, we will evaluate our trained model on the test set and submit our predictions to the Kaggle competition.

Part 1b

Part 1b of the assignment is split into parts.

Part 1b

In part 1b, we will be continue from yesterday. We will be using Open Images annotations for the task.

Specifically, we will be performing the following tasks:

- Loading the annotations and class descriptions
- Mapping the class descriptions to the class labels
- Visualizing the bounding boxes
- Training a simple neural network on images

What is added

We will be adding the ability to visualize segmentation masks. You may also be looking through the dataset and adding other ideas.

Feel free to add any other ideas you have related to image classification or segmentation to the notebook!

Reading Pascal Data

Loading Training Data

We now have the class descriptions, so we can now load in the training annotations as well.

```
[2] !head -n 1000000 dataset/train/data/annotations-human-bbox.csv > train.csv
    !head -n 1 dataset/train/data/annotations-human-bbox.csv > header.csv
    !cat header.csv train.csv > train.csv
    !rm -rf header.csv
    !head -n 1 dataset/validation/data/annotations-human-bbox.csv > header.csv
    !cat header.csv dataset/validation/data/annotations-human-bbox.csv > validation.csv
    !rm -rf header.csv
    !head -n 1 dataset/test/data/annotations-human-bbox.csv > header.csv
    !cat header.csv dataset/test/data/annotations-human-bbox.csv > test.csv
    !rm -rf header.csv
    !head -n 1 train.csv
    !head -n 1 validation.csv
    !head -n 1 test.csv
```

Loading Validation Data

We now have the train.csv, validation.csv and test.csv. We can now load the training annotations into pandas.

```
[3] !head -n 1000000 dataset/train/data/annotations-human-bbox.csv > train.csv
    !head -n 1 dataset/validation/data/annotations-human-bbox.csv > header.csv
    !cat header.csv dataset/validation/data/annotations-human-bbox.csv > validation.csv
    !rm -rf header.csv
    !head -n 1 dataset/test/data/annotations-human-bbox.csv > header.csv
    !cat header.csv dataset/test/data/annotations-human-bbox.csv > test.csv
    !rm -rf header.csv
    !head -n 1 train.csv
    !head -n 1 validation.csv
    !head -n 1 test.csv
```

Visualizing the Data

Now that we have the annotations in a pleasant to use format, we can begin to visualize the data. We will be using the `matplotlib` library to visualize the data.

```
[4] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Classification

```
[5] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Training the network

We now have the data in a pleasant to use format. We can now begin to train the network. We will be using a simple convolutional neural network to predict the labels for a given image. We will be using the `keras` library to train the network.

```
[6] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Evaluate on test set

```
[7] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Another Baseline (Frame Scoped)

Similarly, we will be using the frame scoped baseline to predict the labels for a given image. In this case, we will be using the `keras` library to train the network. We will be using the `keras` library to train the network. We will be using the `keras` library to train the network. We will be using the `keras` library to train the network.

Running Another

We now have the frame scoped baseline. We can now train the network and evaluate on the validation set.

```
[8] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Evaluate on test set

```
[9] !ln -s /usr/bin/sed
    !ln -s /usr/bin/cut
    !ln -s /usr/bin/tr
    !ln -s /usr/bin/sort
    !ln -s /usr/bin/uniq
    !ln -s /usr/bin/head
    !ln -s /usr/bin/tail
    !ln -s /usr/bin/cat
    !ln -s /usr/bin/tee
    !ln -s /usr/bin/mkdir
    !ln -s /usr/bin/ln
    !ln -s /usr/bin/rm
    !ln -s /usr/bin/wc
    !ln -s /usr/bin/sleep
    !ln -s /usr/bin/seq
    !ln -s /usr/bin/xargs
    !ln -s /usr/bin/find
    !ln -s /usr/bin/dirname
    !ln -s /usr/bin/basename
```

Potential Alterations

Some of the alterations that can be made to improve the model performance are as follows:

- Adding more layers to the model
- Adding more data augmentation
- Using a pre-trained model
- Using a different loss function
- Using a different optimizer
- Using a different learning rate
- Using a different batch size
- Using a different number of epochs
- Using a different number of hidden units
- Using a different activation function
- Using a different weight initialization
- Using a different regularization
- Using a different dropout rate
- Using a different learning rate decay
- Using a different learning rate schedule
- Using a different learning rate warm-up
- Using a different learning rate cosine annealing
- Using a different learning rate cosine annealing with warm restarts
- Using a different learning rate cosine annealing with warm restarts and decay
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up
- Using a different learning rate cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing and cosine annealing with warm restarts and decay and warm-up and cosine annealing

Assignment 1 Part 3B: Developing Your Own Classifier

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [3]:
```
!pip3 install torch==1.5 torchvision==0.6
```

```
Collecting torch==1.5
  Downloading https://files.pythonhosted.org/packages/76/58/668ffb25215
b3f8231a550a227be7f905f514859c70a65ca59d28f9b7f60/torch-1.5.0-cp37-cp37
m-manylinux1_x86_64.whl (752.0MB)
     |[redacted]| 752.0MB 24kB/s
Collecting torchvision==0.6
  Downloading https://files.pythonhosted.org/packages/7b/ed/a894f274a77
33d6492e438a5831a95b507c5ec777edf6d8c3b97574e08c4/torchvision-0.6.0-cp3
7-cp37m-manylinux1_x86_64.whl (6.6MB)
     |[redacted]| 6.6MB 45.0MB/s
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-
packages (from torch==1.5) (0.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-p
ackages (from torch==1.5) (1.19.5)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.
7/dist-packages (from torchvision==0.6) (7.0.0)
Installing collected packages: torch, torchvision
  Found existing installation: torch 1.7.1+cu101
    Uninstalling torch-1.7.1+cu101:
      Successfully uninstalled torch-1.7.1+cu101
  Found existing installation: torchvision 0.8.2+cu101
    Uninstalling torchvision-0.8.2+cu101:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
      Successfully uninstalled torchvision-0.8.2+cu101
Successfully installed torch-1.5.0 torchvision-0.6.0
```

In [2]:
```
%cd '/content/drive/MyDrive/DL/assignment1-part3/assignment1-part3'
```

```
/content/drive/MyDrive/DL/assignment1-part3/assignment1-part3
```

In [3]:
```
!ls
```

```
A1_P3A_Introduction.ipynb        download_data.sh   VOCdevkit
A1_P3B_Develop_Classifier.ipynb  __pycache__        voc_simple_classifi
er.pth
classifier.py                    voc_dataloader.py  VOCtrainval_06-Nov-
2007.tar
```

In [4]:
```
import os
import numpy as np
import torch
import torch.nn as nn
import torchvision

from torchvision import transforms
from sklearn.metrics import average_precision_score
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
from classifier import SimpleClassifier, Classifier#, AlexNet
from voc_dataloader import VocDataset, VOC_CLASSES

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

Part 3B: Design your own network

In this notebook, your task is to create and train your own model for multi-label classification on VOC Pascal.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

What to do

1. You will make change on network architecture in `classifier.py`.
2. You may also want to change other hyperparameters to assist your training to get a better performances. Hints will be given in the below instructions.

What to submit

Check the submission template for details what to submit.

In [5]:
```
def train_classifier(train_loader, classifier, criterion, optimizer):
    classifier.train()
    loss_ = 0.0
    losses = []
    for i, (images, labels, _) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        logits = classifier(images)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        losses.append(loss)
    return torch.stack(losses).mean().item()
```

In [6]:
```
def test_classifier(test_loader, classifier, criterion, print_ind_class
es=True, print_total=True):
    classifier.eval()
    losses = []
    with torch.no_grad():
        y_true = np.zeros((0,21))
        y_score = np.zeros((0,21))
        for i, (images, labels, _) in enumerate(test_loader):
            images, labels = images.to(device), labels.to(device)
            logits = classifier(images)
            y_true = np.concatenate((y_true, labels.cpu().numpy()), axi
s=0)
            y_score = np.concatenate((y_score, logits.cpu().numpy()), a
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
xis=0)
            loss = criterion(logits, labels)
            losses.append(loss.item())
        aps = []
        # ignore first class which is background
        for i in range(1, y_true.shape[1]):
            ap = average_precision_score(y_true[:, i], y_score[:, i])
            if print_ind_classes:
                print('-------  Class: {:<12}     AP: {:>8.4f}  -------
'.format(VOC_CLASSES[i], ap))
            aps.append(ap)

        mAP = np.mean(aps)
        test_loss = np.mean(losses)
        if print_total:
            print('mAP: {0:.4f}'.format(mAP))
            print('Avg loss: {}'.format(test_loss))

    return mAP, test_loss, aps
```

In [7]:

```
def plot_losses(train, val, test_frequency, num_epochs):
    plt.plot(train, label="train")
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency ==
0 or i ==0)]
    plt.plot(indices, val, label="val")
    plt.title("Loss Plot")
    plt.ylabel("Loss")
    plt.xlabel("Epoch")
    plt.legend()
    plt.show()

def plot_mAP(train, val, test_frequency, num_epochs):
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency ==
0 or i ==0)]
    plt.plot(indices, train, label="train")
    plt.plot(indices, val, label="val")
    plt.title("mAP Plot")
    plt.ylabel("mAP")
    plt.xlabel("Epoch")
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    plt.legend()
    plt.show()
```

In [8]:
```
def train(classifier, num_epochs, train_loader, val_loader, criterion,
optimizer, test_frequency=5):
    train_losses = []
    train_mAPs = []
    val_losses = []
    val_mAPs = []

    for epoch in range(1,num_epochs+1):
        print("Starting epoch number " + str(epoch))
        train_loss = train_classifier(train_loader, classifier, criteri
on, optimizer)
        train_losses.append(train_loss)
        print("Loss for Training on Epoch " +str(epoch) + " is "+ str(t
rain_loss))
        if(epoch%test_frequency==0 or epoch==1):
            mAP_train, _, _ = test_classifier(train_loader, classifier,
criterion, False, False)
            train_mAPs.append(mAP_train)
            mAP_val, val_loss, _ = test_classifier(val_loader, classifi
er, criterion)
            print('Evaluating classifier')
            print("Mean Precision Score for Testing on Epoch " +str(epo
ch) + " is "+ str(mAP_val))
            val_losses.append(val_loss)
            val_mAPs.append(mAP_val)

    return classifier, train_losses, val_losses, train_mAPs, val_mAPs
```

Developing Your Own Model

Goal

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

To meet the benchmark for this assignment you will need to improve the network. Note you should have noticed pretrained Alenxt performs really well, but training Alexnet from scratch performs much worse. We hope you can design a better architecture over both the simple classifier and AlexNet to train from scratch.

How to start

You may take inspiration from other published architectures and architectures discussed in lecture. However, you are NOT allowed to use predefined models (e.g. models from torchvision) or use pretrained weights. Training must be done from scratch with your own custom model.

Some hints

There are a variety of different approaches you should try to improve performance from the simple classifier:

- Network architecture changes
  - Number of layers: try adding layers to make your network deeper
  - Batch normalization: adding batch norm between layers will likely give you a significant performance increase
  - Residual connections: as you increase the depth of your network, you will find that having residual connections like those in ResNet architectures will be helpful
- Optimizer: Instead of plain SGD, you may want to add a learning rate schedule, add momentum, or use one of the other optimizers you have learned about like Adam. Check the `torch.optim` package for other optimizers
- Data augmentation: You should use the `torchvision.transforms` module to try adding random resized crops and horizontal flips of the input data. Check `transforms.RandomResizedCrop` and `transforms.RandomHorizontalFlip` for this. Feel free to apply more transforms for data augmentation which can lead to better performance.
- Epochs: Once you have found a generally good hyperparameter setting try training for more epochs
- Loss function: You might want to add weighting to the `MultiLabelSoftMarginLoss` for classes that are less well represented or experiment with a different loss function

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Note

We will soon be providing some initial expectations of mAP values as a function of epoch so you can get an early idea whether your implementation works without waiting a long time for training to converge.

What to submit

Submit your best model and save all plots for the writeup.

In [10]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std= [0.229, 0.224, 0.225])

train_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize
        ])

test_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize,
        ])

ds_train = VocDataset('VOCdevkit/VOC2007/','train',train_transform)
ds_val = VocDataset('VOCdevkit/VOC2007/','val',test_transform)
ds_test = VocDataset('VOCdevkit/VOC2007test/','test', test_transform)
```

```
/content/drive/My Drive/DL/assignment1-part3/assignment1-part3/voc_data
loader.py:109: VisibleDeprecationWarning: Creating an ndarray from ragg
ed nested sequences (which is a list-or-tuple of lists-or-tuples-or nda
rrays with different lengths or shapes) is deprecated. If you meant to
do this, you must specify 'dtype=object' when creating the ndarray
  return np.array(names), np.array(labels).astype(np.float32), np.array
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
(box_indices), label_order
```

In [11]:
```
num_epochs = 100
test_frequency = 5
batch_size = 64

train_loader = torch.utils.data.DataLoader(dataset=ds_train,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=1)

val_loader = torch.utils.data.DataLoader(dataset=ds_val,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=1)

test_loader = torch.utils.data.DataLoader(dataset=ds_test,
                                           batch_size=batch_size,
                                           shuffle=False,
                                           num_workers=1)
```

In [55]:
```
c = SimpleClassifier()
c.parameters()
```

```
[autoreload of classifier failed: Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/IPython/extensions/autor
eload.py", line 247, in check
    superreload(m, reload, self.old_objects)
ValueError: __init__() requires a code object with 1 free vars, not 0
]
```

Out[55]:
```
<generator object Module.parameters at 0x7f95e673e950>
```

In [56]:
```
cl = Classifier()
cl.parameters()
```

Out[56]:
```
<generator object Module.parameters at 0x7f95e673e4d0>
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [60]:
```
# TODO: Run your own classifier here
c = Classifier()

criterion = nn.MultiLabelSoftMarginLoss()

# optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentu
m=0.9)
optimizer = torch.optim.Adam(c.parameters(), lr=1e-4)

classifier, train_losses, val_losses, train_mAPs, val_mAPs = train(c, n
um_epochs, train_loader, val_loader, criterion, optimizer, test_frequen
cy)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-60-7b8d18f8e118> in <module>()
      5
      6 # optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.9)
----> 7 optimizer = torch.optim.Adam(c.parameters(), lr=1e-4)
      8
      9 classifier, train_losses, val_losses, train_mAPs, val_mAPs = train(c, num_epochs, train_loader, val_loader, criterion, optimizer, test_frequency)

/usr/local/lib/python3.7/dist-packages/torch/optim/adam.py in __init__(self, params, lr, betas, eps, weight_decay, amsgrad)
     42         defaults = dict(lr=lr, betas=betas, eps=eps,
     43                         weight_decay=weight_decay, amsgrad=amsgrad)
---> 44         super(Adam, self).__init__(params, defaults)
     45
     46     def __setstate__(self, state):

/usr/local/lib/python3.7/dist-packages/torch/optim/optimizer.py in __init__(self, params, defaults)
     44         param_groups = list(params)
     45         if len(param_groups) == 0:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
---> 46             raise ValueError("optimizer got an empty parameter
 list")

     47         if not isinstance(param_groups[0], dict):
     48             param_groups = [{'params': param_groups}]

ValueError: optimizer got an empty parameter list
```

In [ ]:
```
plot_losses(train_losses, val_losses, test_frequency, num_epochs)
plot_mAP(train_mAPs, val_mAPs, test_frequency, num_epochs)
```

In [ ]:
```
mAP_test, test_loss, test_aps = test_classifier(test_loader, classifier
, criterion)
print(mAP_test)
```

In [ ]:
```
torch.save(classifier.state_dict(), './voc_my_best_classifier.pth')
output_submission_csv('my_solution.csv', test_aps)
```

In [ ]:
```

```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD