classification_nn

February 17, 2024

1 ECE 176 Assignment 4: Classification using Neural Network

Now that you have developed and tested your model on the toy dataset set. It's time to get down and get dirty with a standard dataset such as cifar10. At this point, you will be using the provided training data to tune the hyper-parameters of your network such that it works with cifar10 for the task of multi-class classification.

Important: Recall that now we have non-linear decision boundaries, thus we do not need to do one vs all classification. We learn a single non-linear decision boundary instead. Our non-linear boundaries (thanks to relu non-linearity) will take care of differentiating between all the classes

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

[1]:
```
!python get_datasets.py
```

```
Downloading cifar-10
Done
```

[2]:
```
# Prepare Packages
import numpy as np
import matplotlib.pyplot as plt

from utils.data_processing import get_cifar10_data
from utils.evaluation import get_classification_accuracy


%matplotlib inline
plt.rcParams["figure.figsize"] = (10.0, 8.0)  # set default size of plots

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

# Use a subset of CIFAR10 for the assignment
dataset = get_cifar10_data(
    subset_train=5000,
    subset_val=250,
```

```
    subset_test=500,
)

print(dataset.keys())
print("Training Set Data  Shape: ", dataset["x_train"].shape)
print("Training Set Label Shape: ", dataset["y_train"].shape)
print("Validation Set Data  Shape: ", dataset["x_val"].shape)
print("Validation Set Label Shape: ", dataset["y_val"].shape)
print("Test Set Data  Shape: ", dataset["x_test"].shape)
print("Test Set Label Shape: ", dataset["y_test"].shape)
```

```
dict_keys(['x_train', 'y_train', 'x_val', 'y_val', 'x_test', 'y_test'])
Training Set Data  Shape:  (5000, 3072)
Training Set Label Shape:  (5000,)
Validation Set Data  Shape:  (250, 3072)
Validation Set Label Shape:  (250,)
Test Set Data  Shape:  (500, 3072)
Test Set Label Shape:  (500,)
```

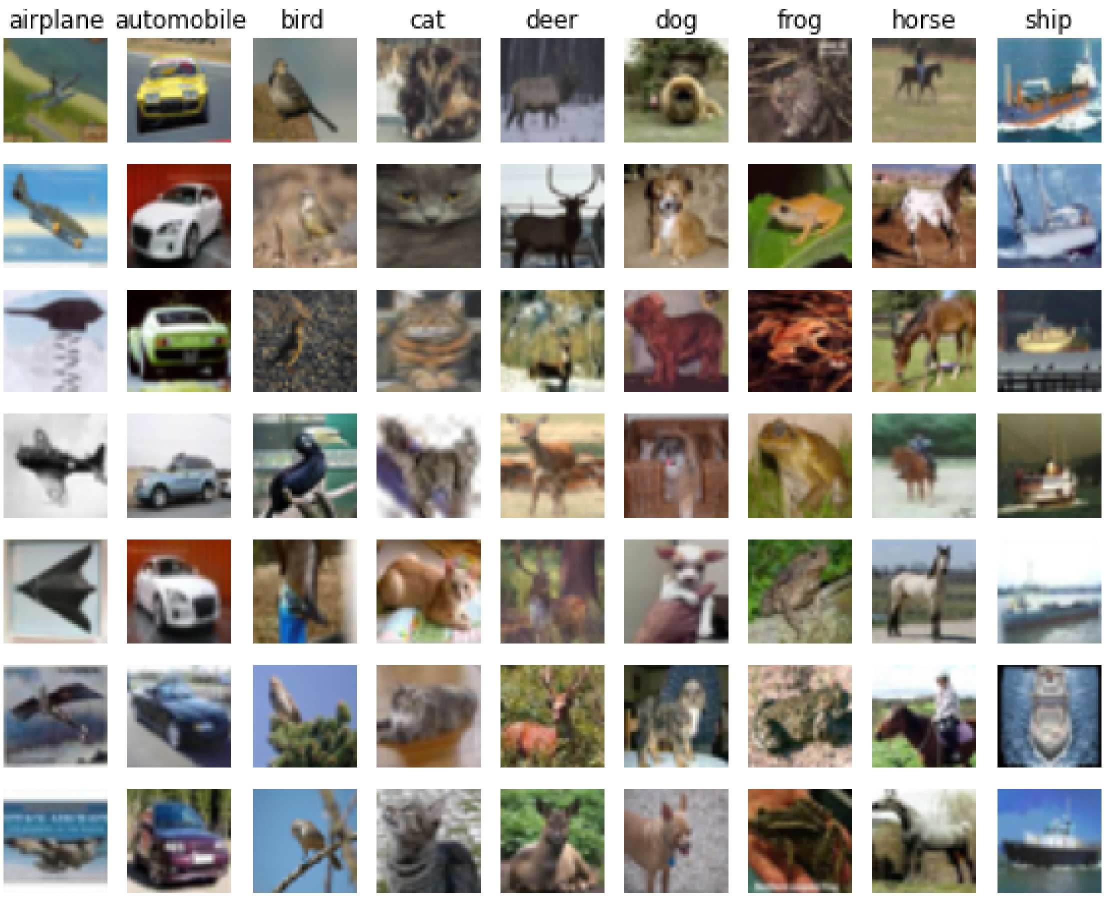
[3]:
```
x_train = dataset["x_train"]
y_train = dataset["y_train"]
x_val = dataset["x_val"]
y_val = dataset["y_val"]
x_test = dataset["x_test"]
y_test = dataset["y_test"]
```

[4]:
```
# Import more utilies and the layers you have implemented
from layers.sequential import Sequential
from layers.linear import Linear
from layers.relu import ReLU
from layers.softmax import Softmax
from layers.loss_func import CrossEntropyLoss
from utils.optimizer import SGD
from utils.dataset import DataLoader
from utils.trainer import Trainer
```

1.1 Visualize some examples from the dataset.

[5]:
```
# We show a few examples of training images from each class.
classes = [
    "airplane",
    "automobile",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
```

```
    "horse",
    "ship",
]
samples_per_class = 7


def visualize_data(dataset, classes, samples_per_class):
    num_classes = len(classes)
    for y, cls in enumerate(classes):
        idxs = np.flatnonzero(y_train == y)
        idxs = np.random.choice(idxs, samples_per_class, replace=False)
        for i, idx in enumerate(idxs):
            plt_idx = i * num_classes + y + 1
            plt.subplot(samples_per_class, num_classes, plt_idx)
            plt.imshow(dataset[idx])
            plt.axis("off")
            if i == 0:
                plt.title(cls)
    plt.show()


# Visualize the first 10 classes
visualize_data(
    x_train.reshape(5000, 3, 32, 32).transpose(0, 2, 3, 1),
    classes,
    samples_per_class,
)
```



1.2 Initialize the model

[6]:
```
input_size = 3072
hidden_size = 100  # Hidden layer size (Hyper-parameter)
num_classes = 10  # Output

# For a default setting we use the same model we used for the toy dataset.
# This tells you the power of a 2 layered Neural Network. Recall the Universal␣
↪Approximation Theorem.
# A 2 layer neural network with non-linearities can approximate any function,␣
↪given large enough hidden layer
def init_model():
    # np.random.seed(0) # No need to fix the seed here
    l1 = Linear(input_size, hidden_size)
    l2 = Linear(hidden_size, num_classes)

    r1 = ReLU()
    softmax = Softmax()
```

```
    return Sequential([l1, r1, l2, softmax])
```

[7]:
```
# Initialize the dataset with the dataloader class
dataset = DataLoader(x_train, y_train, x_val, y_val, x_test, y_test)
net = init_model()
optim = SGD(net, lr=0.01, weight_decay=0.01)
loss_func = CrossEntropyLoss()
epoch = 200  # (Hyper-parameter)
batch_size = 200  # (Reduce the batch size if your computer is unable to handle␣
 ↪it)
```

[8]:
```
# Initialize the trainer class by passing the above modules
trainer = Trainer(
    dataset, optim, net, loss_func, epoch, batch_size, validate_interval=3
)
```

[9]:
```
# Call the trainer function we have already implemented for you. This trains␣
 ↪the model for the given
# hyper-parameters. It follows the same procedure as in the last ipython␣
 ↪notebook you used for the toy-dataset
train_error, validation_accuracy = trainer.train()
```

```
Epoch Average Loss: 2.302538
Validate Acc: 0.084
Epoch Average Loss: 2.302363
Epoch Average Loss: 2.302160
Epoch Average Loss: 2.301863
Validate Acc: 0.096
Epoch Average Loss: 2.301457
Epoch Average Loss: 2.300843
Epoch Average Loss: 2.299990
Validate Acc: 0.096
Epoch Average Loss: 2.298836
Epoch Average Loss: 2.297372
Epoch Average Loss: 2.295567
Validate Acc: 0.088
Epoch Average Loss: 2.293456
Epoch Average Loss: 2.290957
Epoch Average Loss: 2.287896
Validate Acc: 0.084
Epoch Average Loss: 2.284023
Epoch Average Loss: 2.278987
Epoch Average Loss: 2.272815
Validate Acc: 0.096
Epoch Average Loss: 2.265853
Epoch Average Loss: 2.258431
Epoch Average Loss: 2.250831
```

```
Validate Acc: 0.100
Epoch Average Loss: 2.243134
Epoch Average Loss: 2.235766
Epoch Average Loss: 2.228620
Validate Acc: 0.112
Epoch Average Loss: 2.222014
Epoch Average Loss: 2.215715
Epoch Average Loss: 2.210025
Validate Acc: 0.124
Epoch Average Loss: 2.204761
Epoch Average Loss: 2.199846
Epoch Average Loss: 2.195497
Validate Acc: 0.132
Epoch Average Loss: 2.191222
Epoch Average Loss: 2.187065
Epoch Average Loss: 2.183200
Validate Acc: 0.136
Epoch Average Loss: 2.180274
Epoch Average Loss: 2.176472
Epoch Average Loss: 2.173230
Validate Acc: 0.144
Epoch Average Loss: 2.170278
Epoch Average Loss: 2.167194
Epoch Average Loss: 2.164604
Validate Acc: 0.140
Epoch Average Loss: 2.161871
Epoch Average Loss: 2.159627
Epoch Average Loss: 2.156986
Validate Acc: 0.144
Epoch Average Loss: 2.154553
Epoch Average Loss: 2.152324
Epoch Average Loss: 2.150128
Validate Acc: 0.144
Epoch Average Loss: 2.148125
Epoch Average Loss: 2.146041
Epoch Average Loss: 2.144083
Validate Acc: 0.148
Epoch Average Loss: 2.142252
Epoch Average Loss: 2.140154
Epoch Average Loss: 2.138903
Validate Acc: 0.156
Epoch Average Loss: 2.136762
Epoch Average Loss: 2.135166
Epoch Average Loss: 2.133179
Validate Acc: 0.144
Epoch Average Loss: 2.132032
Epoch Average Loss: 2.130354
Epoch Average Loss: 2.128594
```

```
Validate Acc: 0.156
Epoch Average Loss: 2.127247
Epoch Average Loss: 2.125219
Epoch Average Loss: 2.123963
Validate Acc: 0.148
Epoch Average Loss: 2.122806
Epoch Average Loss: 2.120940
Epoch Average Loss: 2.119623
Validate Acc: 0.168
Epoch Average Loss: 2.117560
Epoch Average Loss: 2.116460
Epoch Average Loss: 2.114619
Validate Acc: 0.168
Epoch Average Loss: 2.112802
Epoch Average Loss: 2.111394
Epoch Average Loss: 2.109295
Validate Acc: 0.172
Epoch Average Loss: 2.107921
Epoch Average Loss: 2.105392
Epoch Average Loss: 2.103318
Validate Acc: 0.168
Epoch Average Loss: 2.101340
Epoch Average Loss: 2.098694
Epoch Average Loss: 2.096403
Validate Acc: 0.176
Epoch Average Loss: 2.094321
Epoch Average Loss: 2.091355
Epoch Average Loss: 2.088423
Validate Acc: 0.200
Epoch Average Loss: 2.085658
Epoch Average Loss: 2.081976
Epoch Average Loss: 2.079208
Validate Acc: 0.228
Epoch Average Loss: 2.077023
Epoch Average Loss: 2.073326
Epoch Average Loss: 2.069776
Validate Acc: 0.236
Epoch Average Loss: 2.066918
Epoch Average Loss: 2.063695
Epoch Average Loss: 2.060293
Validate Acc: 0.236
Epoch Average Loss: 2.057211
Epoch Average Loss: 2.054165
Epoch Average Loss: 2.050852
Validate Acc: 0.232
Epoch Average Loss: 2.047772
Epoch Average Loss: 2.045366
Epoch Average Loss: 2.042617
```

```
Validate Acc: 0.244
Epoch Average Loss: 2.039720
Epoch Average Loss: 2.036696
Epoch Average Loss: 2.034236
Validate Acc: 0.248
Epoch Average Loss: 2.031654
Epoch Average Loss: 2.029122
Epoch Average Loss: 2.026496
Validate Acc: 0.264
Epoch Average Loss: 2.024426
Epoch Average Loss: 2.022311
Epoch Average Loss: 2.019357
Validate Acc: 0.260
Epoch Average Loss: 2.018062
Epoch Average Loss: 2.015623
Epoch Average Loss: 2.013454
Validate Acc: 0.264
Epoch Average Loss: 2.011797
Epoch Average Loss: 2.009439
Epoch Average Loss: 2.008201
Validate Acc: 0.264
Epoch Average Loss: 2.005785
Epoch Average Loss: 2.003532
Epoch Average Loss: 2.002000
Validate Acc: 0.268
Epoch Average Loss: 1.999968
Epoch Average Loss: 1.997993
Epoch Average Loss: 1.996816
Validate Acc: 0.268
Epoch Average Loss: 1.994891
Epoch Average Loss: 1.992646
Epoch Average Loss: 1.991058
Validate Acc: 0.264
Epoch Average Loss: 1.989383
Epoch Average Loss: 1.987373
Epoch Average Loss: 1.987078
Validate Acc: 0.268
Epoch Average Loss: 1.984275
Epoch Average Loss: 1.982816
Epoch Average Loss: 1.980759
Validate Acc: 0.284
Epoch Average Loss: 1.979304
Epoch Average Loss: 1.977111
Epoch Average Loss: 1.975656
Validate Acc: 0.280
Epoch Average Loss: 1.973401
Epoch Average Loss: 1.971649
Epoch Average Loss: 1.969992
```

```
Validate Acc: 0.296
Epoch Average Loss: 1.968767
Epoch Average Loss: 1.966490
Epoch Average Loss: 1.964231
Validate Acc: 0.292
Epoch Average Loss: 1.960804
Epoch Average Loss: 1.958659
Epoch Average Loss: 1.957416
Validate Acc: 0.308
Epoch Average Loss: 1.955159
Epoch Average Loss: 1.952510
Epoch Average Loss: 1.947802
Validate Acc: 0.308
Epoch Average Loss: 1.946458
Epoch Average Loss: 1.942763
Epoch Average Loss: 1.939922
Validate Acc: 0.296
Epoch Average Loss: 1.936995
Epoch Average Loss: 1.932804
Epoch Average Loss: 1.930225
Validate Acc: 0.300
Epoch Average Loss: 1.928316
Epoch Average Loss: 1.923058
Epoch Average Loss: 1.921420
Validate Acc: 0.296
Epoch Average Loss: 1.919260
Epoch Average Loss: 1.916774
Epoch Average Loss: 1.913924
Validate Acc: 0.276
Epoch Average Loss: 1.910829
Epoch Average Loss: 1.909111
Epoch Average Loss: 1.905870
Validate Acc: 0.288
Epoch Average Loss: 1.904934
Epoch Average Loss: 1.901818
Epoch Average Loss: 1.899995
Validate Acc: 0.312
Epoch Average Loss: 1.897434
Epoch Average Loss: 1.895294
Epoch Average Loss: 1.893233
Validate Acc: 0.300
Epoch Average Loss: 1.891155
Epoch Average Loss: 1.889347
Epoch Average Loss: 1.886142
Validate Acc: 0.296
Epoch Average Loss: 1.886025
Epoch Average Loss: 1.883206
Epoch Average Loss: 1.880699
```

```
Validate Acc: 0.300
Epoch Average Loss: 1.879615
Epoch Average Loss: 1.878772
Epoch Average Loss: 1.876192
Validate Acc: 0.300
Epoch Average Loss: 1.873852
Epoch Average Loss: 1.871608
Epoch Average Loss: 1.869782
Validate Acc: 0.300
Epoch Average Loss: 1.869138
Epoch Average Loss: 1.866602
Epoch Average Loss: 1.865560
Validate Acc: 0.304
Epoch Average Loss: 1.862459
Epoch Average Loss: 1.861574
Epoch Average Loss: 1.861144
Validate Acc: 0.312
Epoch Average Loss: 1.858137
Epoch Average Loss: 1.855692
Epoch Average Loss: 1.853509
Validate Acc: 0.288
Epoch Average Loss: 1.852843
Epoch Average Loss: 1.850432
Epoch Average Loss: 1.849172
Validate Acc: 0.324
Epoch Average Loss: 1.846087
Epoch Average Loss: 1.845643
Epoch Average Loss: 1.841321
Validate Acc: 0.300
Epoch Average Loss: 1.839812
Epoch Average Loss: 1.838377
Epoch Average Loss: 1.835869
Validate Acc: 0.320
Epoch Average Loss: 1.834917
Epoch Average Loss: 1.832104
Epoch Average Loss: 1.830938
Validate Acc: 0.324
Epoch Average Loss: 1.829928
Epoch Average Loss: 1.826212
Epoch Average Loss: 1.823800
Validate Acc: 0.308
Epoch Average Loss: 1.823045
Epoch Average Loss: 1.823667
Epoch Average Loss: 1.817594
Validate Acc: 0.316
Epoch Average Loss: 1.817124
Epoch Average Loss: 1.816326
Epoch Average Loss: 1.815354
```

```
Validate Acc: 0.332
Epoch Average Loss: 1.810587
```

1.2.1 Print the training and validation accuracies for the default hyper-parameters provided

[10]:
```python
from utils.evaluation import get_classification_accuracy

out_train = net.predict(x_train)
acc = get_classification_accuracy(out_train, y_train)
print("Training acc: ", acc)
out_val = net.predict(x_val)
acc = get_classification_accuracy(out_val, y_val)
print("Validation acc: ", acc)
```

```
Training acc:  0.3464
Validation acc:  0.332
```

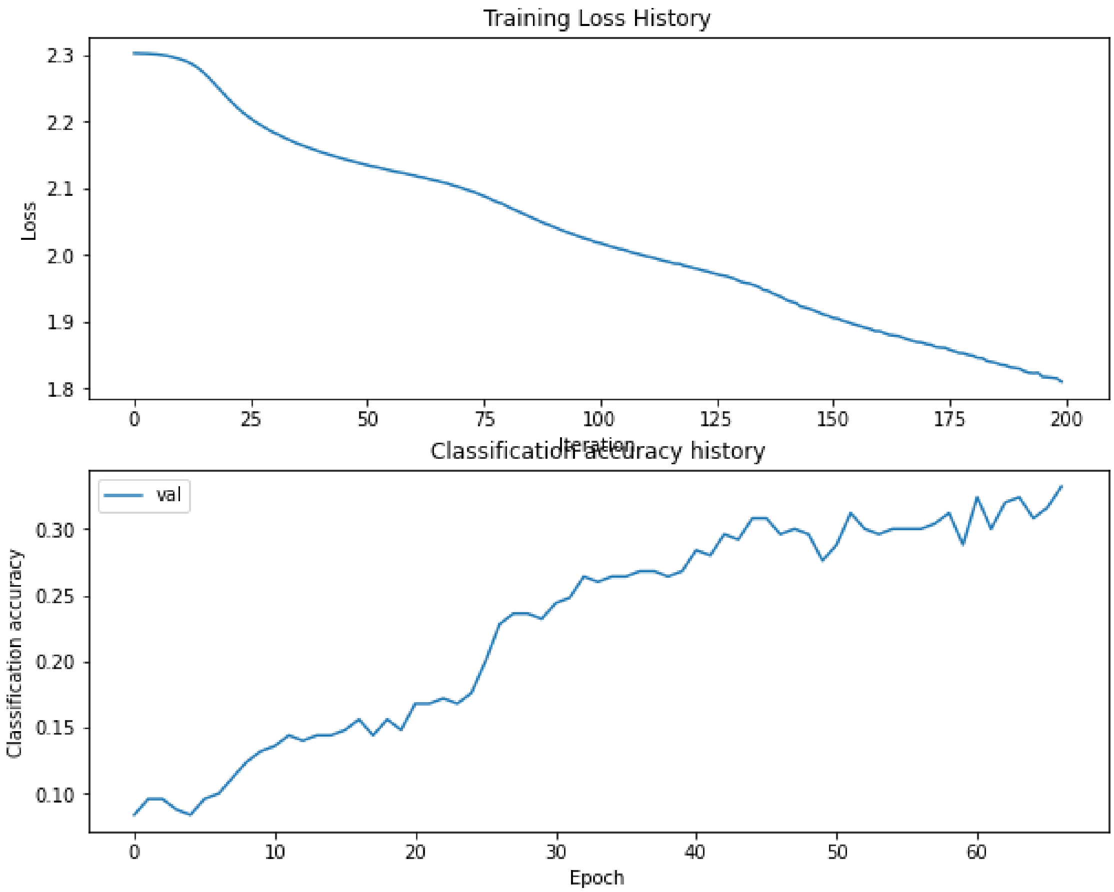
1.2.2 Debug the training

With the default parameters we provided above, you should get a validation accuracy of around 0.2~0.3 on the validation set. This isn't very good.

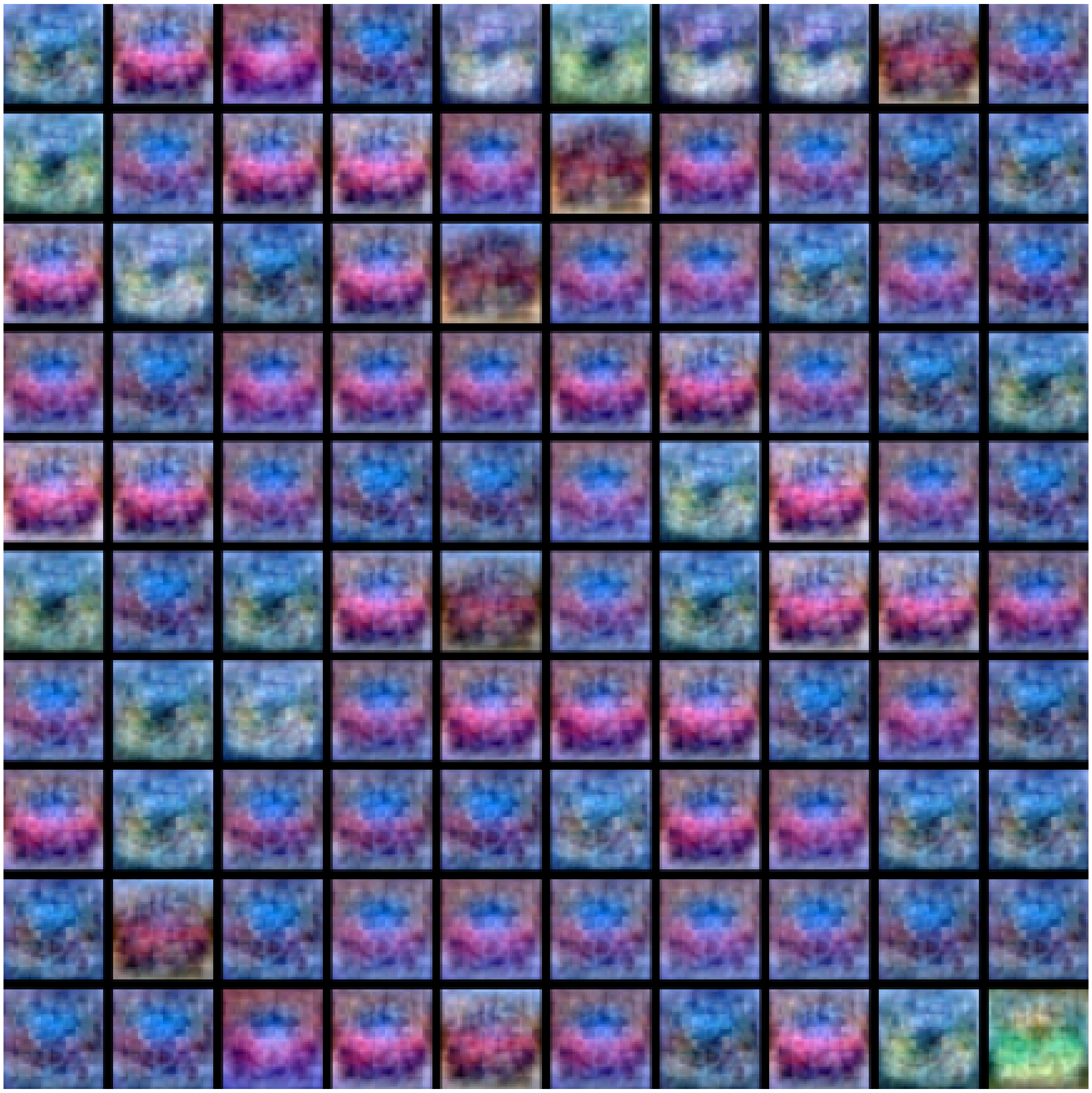
One strategy for getting insight into what's wrong is to plot the training loss function and the validation accuracies during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

[11]:
```python
# Plot the training loss function and validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_error)
plt.title("Training Loss History")
plt.xlabel("Iteration")
plt.ylabel("Loss")

plt.subplot(2, 1, 2)
# plt.plot(stats['train_acc_history'], label='train')
plt.plot(validation_accuracy, label="val")
plt.title("Classification accuracy history")
plt.xlabel("Epoch")
plt.ylabel("Classification accuracy")
plt.legend()
plt.show()
```



[12]:
```
from utils.vis_utils import visualize_grid

# Credits: http://cs231n.stanford.edu/

# Visualize the weights of the network

def show_net_weights(net):
    W1 = net._modules[0].parameters[0]
    W1 = W1.reshape(3, 32, 32, -1).transpose(3, 1, 2, 0)
    plt.imshow(visualize_grid(W1, padding=3).astype("uint8"))
    plt.gca().axis("off")
    plt.show()


show_net_weights(net)
```



2 Tune your hyperparameters (50%)

What's wrong?. Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer size, learning rate, numer of training epochs, and regularization strength.

Approximate results. You should be aim to achieve a classification accuracy of greater than 40% on the validation set. Our best network gets over 40% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on cifar10 as you can (40% could serve as a reference), with a fully-connected Neural Network.

Explain your hyperparameter tuning process below.

Your Answer:

[13]:
```
################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained ␣
↪#
# model hyperparams in best_net.                                               ␣
↪#
#                                                                              ␣
↪#
# To help debug your network, it may help to use visualizations similar to the ␣
↪#
# ones we used above; these visualizations will have significant qualitative   ␣
↪#
# differences from the ones we saw above for the poorly tuned network.        ␣
↪#
#                                                                              ␣
↪#
# You are now free to test different combinations of hyperparameters to build ␣
↪#
# various models and test them according to the above plots and visualization  ␣
↪#

# TODO: Show the above plots and visualizations for the default params (already␣
↪#
# done) and the best hyper-params you obtain. You only need to show this for 2 ␣
↪#
# sets of hyper-params.                                                        ␣
↪#
# You just need to store values for the hyperparameters in best_net_hyperparams␣
↪#
# as a list in the order
# best_net_hyperparams = [lr, weight_decay, epoch, hidden_size]
################################################################################

best_net_hyperparams = [0.1, 0.001, 200, 300]  # store the best model into this
# USED FOR RUNNING TESTS AND TUNING
# best_accuracy = 0

input_size = 3072
hidden_size = 300
```

```
num_classes = 10  # Output

# USED FOR RUNNING TESTS AND TUNING
# learning_rates = [0.1, 0.01, 0.001]
# weight_decays = [0.1, 0.01, 0.001]
# epochs = [100, 150, 200]
# hidden_sizes = [100, 200, 300]

# For a default setting we use the same model we used for the toy dataset.
# This tells you the power of a 2 layered Neural Network. Recall the Universal␣
↪Approximation Theorem.
# A 2 layer neural network with non-linearities can approximate any function,␣
↪given large enough hidden layer
def init_model():
    # np.random.seed(0) # No need to fix the seed here
    l1 = Linear(input_size, hidden_size)
    l2 = Linear(hidden_size, num_classes)

    r1 = ReLU()
    softmax = Softmax()
    return Sequential([l1, r1, l2, softmax])

# USED FOR RUNNING TESTS AND TUNING
# for lr in learning_rates:
#     for weight_decay in weight_decays:
#         for epoch in epochs:
#             for hidden_size in hidden_sizes:
#                 print("lr, weight_decay, epoch, hidden_size: ", lr,␣
↪weight_decay, epoch, hidden_size)

# Initialize the dataset with the dataloader class
dataset = DataLoader(x_train, y_train, x_val, y_val, x_test, y_test)
best_net = init_model()
optim = SGD(best_net, lr=0.1, weight_decay=0.001)
loss_func = CrossEntropyLoss()
epoch = 200
batch_size = 200

# Initialize the trainer class by passing the above modules
trainer = Trainer(
    dataset, optim, best_net, loss_func, epoch, batch_size, validate_interval=3
)

# Call the trainer function we have already implemented for you. This trains␣
↪the model for the given
# hyper-parameters. It follows the same procedure as in the last ipython␣
↪notebook you used for the toy-dataset
```

```
train_error, validation_accuracy = trainer.train()

# out_train = net.predict(x_train)
# acc = get_classification_accuracy(out_train, y_train)
# print("Training acc: ", acc)
# out_val = net.predict(x_val)
# acc = get_classification_accuracy(out_val, y_val)
# print("Validation acc: ", acc)

                # USED FOR FINDING IDEAL HYPERPARAMETERS
#                 if acc > best_accuracy:
#                     best_accuracy = acc
#                     best_net_hyperparams = [lr, weight_decay, epoch,
 ↪hidden_size]
#                     print("NEW best validation accuracy:", best_accuracy)
#                     print("NEW best hyperparameters:", best_net_hyperparams)

# USED FOR HYPERPARAMETER TUNING
# print("Best validation accuracy:", best_accuracy)
# print("Best hyperparameters:", best_net_hyperparams)
```

```
Epoch Average Loss: 2.299955
Validate Acc: 0.084
Epoch Average Loss: 2.270726
Epoch Average Loss: 2.213987
Epoch Average Loss: 2.187136
Validate Acc: 0.132
Epoch Average Loss: 2.162906
Epoch Average Loss: 2.152636
Epoch Average Loss: 2.124264
Validate Acc: 0.156
Epoch Average Loss: 2.111862
Epoch Average Loss: 2.083653
Epoch Average Loss: 2.078761
Validate Acc: 0.188
Epoch Average Loss: 2.051981
Epoch Average Loss: 2.042481
Epoch Average Loss: 2.016782
Validate Acc: 0.176
Epoch Average Loss: 1.996013
Epoch Average Loss: 2.001382
Epoch Average Loss: 1.978823
Validate Acc: 0.276
Epoch Average Loss: 1.980588
Epoch Average Loss: 1.939177
Epoch Average Loss: 1.923781
Validate Acc: 0.296
```

```
Epoch Average Loss: 1.906502
Epoch Average Loss: 1.897976
Epoch Average Loss: 1.886186
Validate Acc: 0.292
Epoch Average Loss: 1.867194
Epoch Average Loss: 1.862610
Epoch Average Loss: 1.844268
Validate Acc: 0.308
Epoch Average Loss: 1.824108
Epoch Average Loss: 1.810816
Epoch Average Loss: 1.814872
Validate Acc: 0.348
Epoch Average Loss: 1.820020
Epoch Average Loss: 1.795814
Epoch Average Loss: 1.773385
Validate Acc: 0.364
Epoch Average Loss: 1.781959
Epoch Average Loss: 1.774161
Epoch Average Loss: 1.747094
Validate Acc: 0.328
Epoch Average Loss: 1.744176
Epoch Average Loss: 1.742060
Epoch Average Loss: 1.704766
Validate Acc: 0.348
Epoch Average Loss: 1.726548
Epoch Average Loss: 1.717709
Epoch Average Loss: 1.714920
Validate Acc: 0.352
Epoch Average Loss: 1.697586
Epoch Average Loss: 1.689310
Epoch Average Loss: 1.697571
Validate Acc: 0.360
Epoch Average Loss: 1.675556
Epoch Average Loss: 1.694102
Epoch Average Loss: 1.678752
Validate Acc: 0.356
Epoch Average Loss: 1.657484
Epoch Average Loss: 1.678972
Epoch Average Loss: 1.639546
Validate Acc: 0.408
Epoch Average Loss: 1.621193
Epoch Average Loss: 1.606400
Epoch Average Loss: 1.661723
Validate Acc: 0.404
Epoch Average Loss: 1.636762
Epoch Average Loss: 1.644639
Epoch Average Loss: 1.669538
Validate Acc: 0.380
```

```
Epoch Average Loss: 1.614463
Epoch Average Loss: 1.614155
Epoch Average Loss: 1.587876
Validate Acc: 0.416
Epoch Average Loss: 1.573375
Epoch Average Loss: 1.595524
Epoch Average Loss: 1.548443
Validate Acc: 0.372
Epoch Average Loss: 1.557256
Epoch Average Loss: 1.554913
Epoch Average Loss: 1.573235
Validate Acc: 0.392
Epoch Average Loss: 1.531509
Epoch Average Loss: 1.588887
Epoch Average Loss: 1.540950
Validate Acc: 0.400
Epoch Average Loss: 1.526240
Epoch Average Loss: 1.554460
Epoch Average Loss: 1.523146
Validate Acc: 0.412
Epoch Average Loss: 1.516681
Epoch Average Loss: 1.515318
Epoch Average Loss: 1.483480
Validate Acc: 0.384
Epoch Average Loss: 1.468042
Epoch Average Loss: 1.482263
Epoch Average Loss: 1.497431
Validate Acc: 0.384
Epoch Average Loss: 1.466014
Epoch Average Loss: 1.485431
Epoch Average Loss: 1.480871
Validate Acc: 0.348
Epoch Average Loss: 1.457225
Epoch Average Loss: 1.457309
Epoch Average Loss: 1.493792
Validate Acc: 0.404
Epoch Average Loss: 1.429325
Epoch Average Loss: 1.430952
Epoch Average Loss: 1.445326
Validate Acc: 0.420
Epoch Average Loss: 1.408376
Epoch Average Loss: 1.394314
Epoch Average Loss: 1.440444
Validate Acc: 0.376
Epoch Average Loss: 1.457244
Epoch Average Loss: 1.429524
Epoch Average Loss: 1.433498
Validate Acc: 0.384
```

```
Epoch Average Loss: 1.411078
Epoch Average Loss: 1.416967
Epoch Average Loss: 1.395350
Validate Acc: 0.416
Epoch Average Loss: 1.361577
Epoch Average Loss: 1.374661
Epoch Average Loss: 1.360049
Validate Acc: 0.412
Epoch Average Loss: 1.377408
Epoch Average Loss: 1.409981
Epoch Average Loss: 1.391746
Validate Acc: 0.412
Epoch Average Loss: 1.348527
Epoch Average Loss: 1.313542
Epoch Average Loss: 1.308795
Validate Acc: 0.360
Epoch Average Loss: 1.332325
Epoch Average Loss: 1.377506
Epoch Average Loss: 1.327121
Validate Acc: 0.416
Epoch Average Loss: 1.302008
Epoch Average Loss: 1.282744
Epoch Average Loss: 1.345911
Validate Acc: 0.436
Epoch Average Loss: 1.316196
Epoch Average Loss: 1.295503
Epoch Average Loss: 1.308162
Validate Acc: 0.452
Epoch Average Loss: 1.260518
Epoch Average Loss: 1.327422
Epoch Average Loss: 1.292926
Validate Acc: 0.396
Epoch Average Loss: 1.264853
Epoch Average Loss: 1.287758
Epoch Average Loss: 1.259866
Validate Acc: 0.420
Epoch Average Loss: 1.212267
Epoch Average Loss: 1.239255
Epoch Average Loss: 1.294158
Validate Acc: 0.424
Epoch Average Loss: 1.320279
Epoch Average Loss: 1.245029
Epoch Average Loss: 1.282160
Validate Acc: 0.392
Epoch Average Loss: 1.255220
Epoch Average Loss: 1.232639
Epoch Average Loss: 1.182401
Validate Acc: 0.440
```

```
Epoch Average Loss: 1.187378
Epoch Average Loss: 1.209614
Epoch Average Loss: 1.181744
Validate Acc: 0.380
Epoch Average Loss: 1.211768
Epoch Average Loss: 1.233755
Epoch Average Loss: 1.219021
Validate Acc: 0.448
Epoch Average Loss: 1.209262
Epoch Average Loss: 1.211301
Epoch Average Loss: 1.180716
Validate Acc: 0.384
Epoch Average Loss: 1.231443
Epoch Average Loss: 1.142070
Epoch Average Loss: 1.181439
Validate Acc: 0.404
Epoch Average Loss: 1.146595
Epoch Average Loss: 1.249510
Epoch Average Loss: 1.284436
Validate Acc: 0.360
Epoch Average Loss: 1.257404
Epoch Average Loss: 1.178403
Epoch Average Loss: 1.149683
Validate Acc: 0.420
Epoch Average Loss: 1.140289
Epoch Average Loss: 1.114127
Epoch Average Loss: 1.101906
Validate Acc: 0.412
Epoch Average Loss: 1.202242
Epoch Average Loss: 1.144806
Epoch Average Loss: 1.098978
Validate Acc: 0.448
Epoch Average Loss: 1.129251
Epoch Average Loss: 1.106074
Epoch Average Loss: 1.097307
Validate Acc: 0.440
Epoch Average Loss: 1.030552
Epoch Average Loss: 1.100278
Epoch Average Loss: 1.036471
Validate Acc: 0.424
Epoch Average Loss: 1.053966
Epoch Average Loss: 1.067288
Epoch Average Loss: 1.065489
Validate Acc: 0.424
Epoch Average Loss: 1.024165
Epoch Average Loss: 1.108489
Epoch Average Loss: 1.113359
Validate Acc: 0.448
```

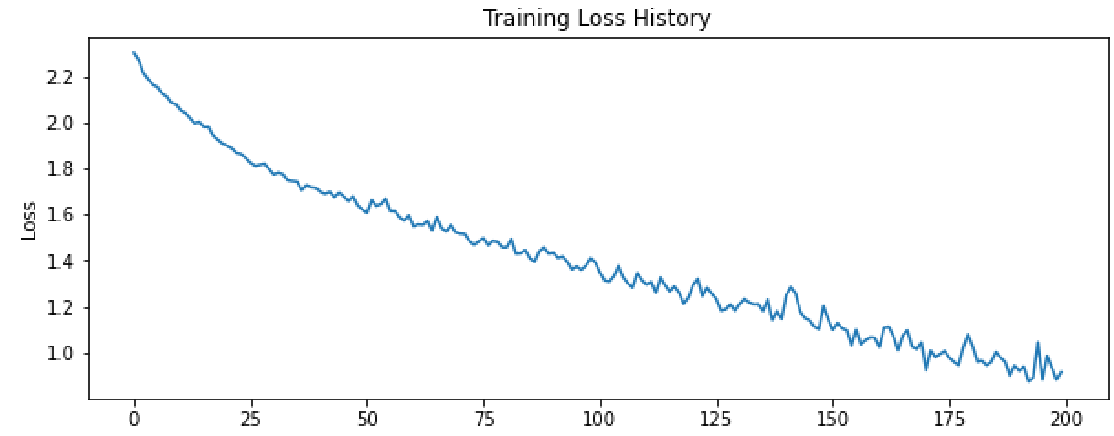
```
Epoch Average Loss: 1.072252
Epoch Average Loss: 1.010491
Epoch Average Loss: 1.074792
Validate Acc: 0.360
Epoch Average Loss: 1.097769
Epoch Average Loss: 1.025978
Epoch Average Loss: 1.014913
Validate Acc: 0.392
Epoch Average Loss: 1.044297
Epoch Average Loss: 0.923521
Epoch Average Loss: 1.008597
Validate Acc: 0.444
Epoch Average Loss: 0.979989
Epoch Average Loss: 0.993732
Epoch Average Loss: 1.006875
Validate Acc: 0.420
Epoch Average Loss: 0.978501
Epoch Average Loss: 0.959854
Epoch Average Loss: 0.945397
Validate Acc: 0.436
Epoch Average Loss: 1.023493
Epoch Average Loss: 1.080449
Epoch Average Loss: 1.028283
Validate Acc: 0.420
Epoch Average Loss: 0.961112
Epoch Average Loss: 0.965554
Epoch Average Loss: 0.945951
Validate Acc: 0.408
Epoch Average Loss: 0.959090
Epoch Average Loss: 1.002890
Epoch Average Loss: 0.979633
Validate Acc: 0.432
Epoch Average Loss: 0.960181
Epoch Average Loss: 0.900069
Epoch Average Loss: 0.944420
Validate Acc: 0.452
Epoch Average Loss: 0.920850
Epoch Average Loss: 0.940054
Epoch Average Loss: 0.875253
Validate Acc: 0.416
Epoch Average Loss: 0.893088
Epoch Average Loss: 1.044991
Epoch Average Loss: 0.882823
Validate Acc: 0.400
Epoch Average Loss: 0.986138
Epoch Average Loss: 0.936065
Epoch Average Loss: 0.883015
Validate Acc: 0.416
```

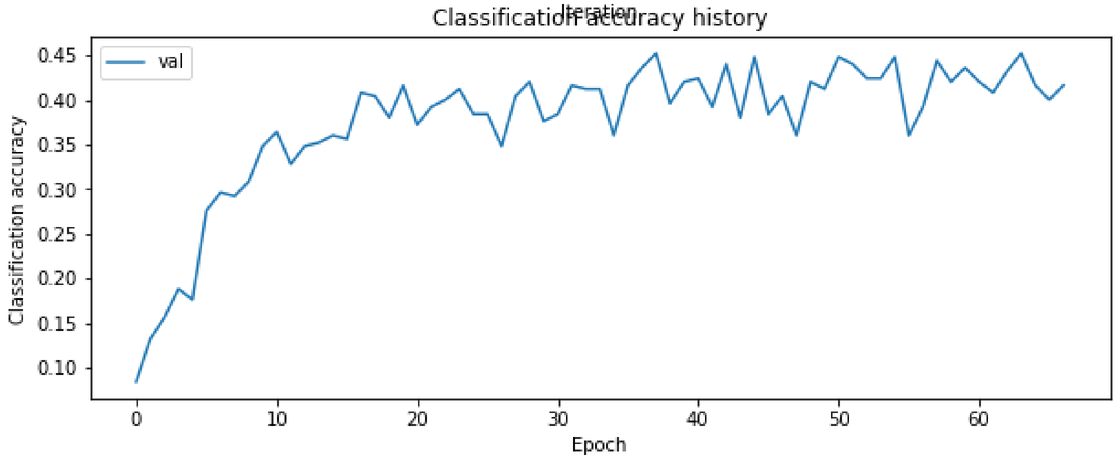
```
Epoch Average Loss: 0.915794
```

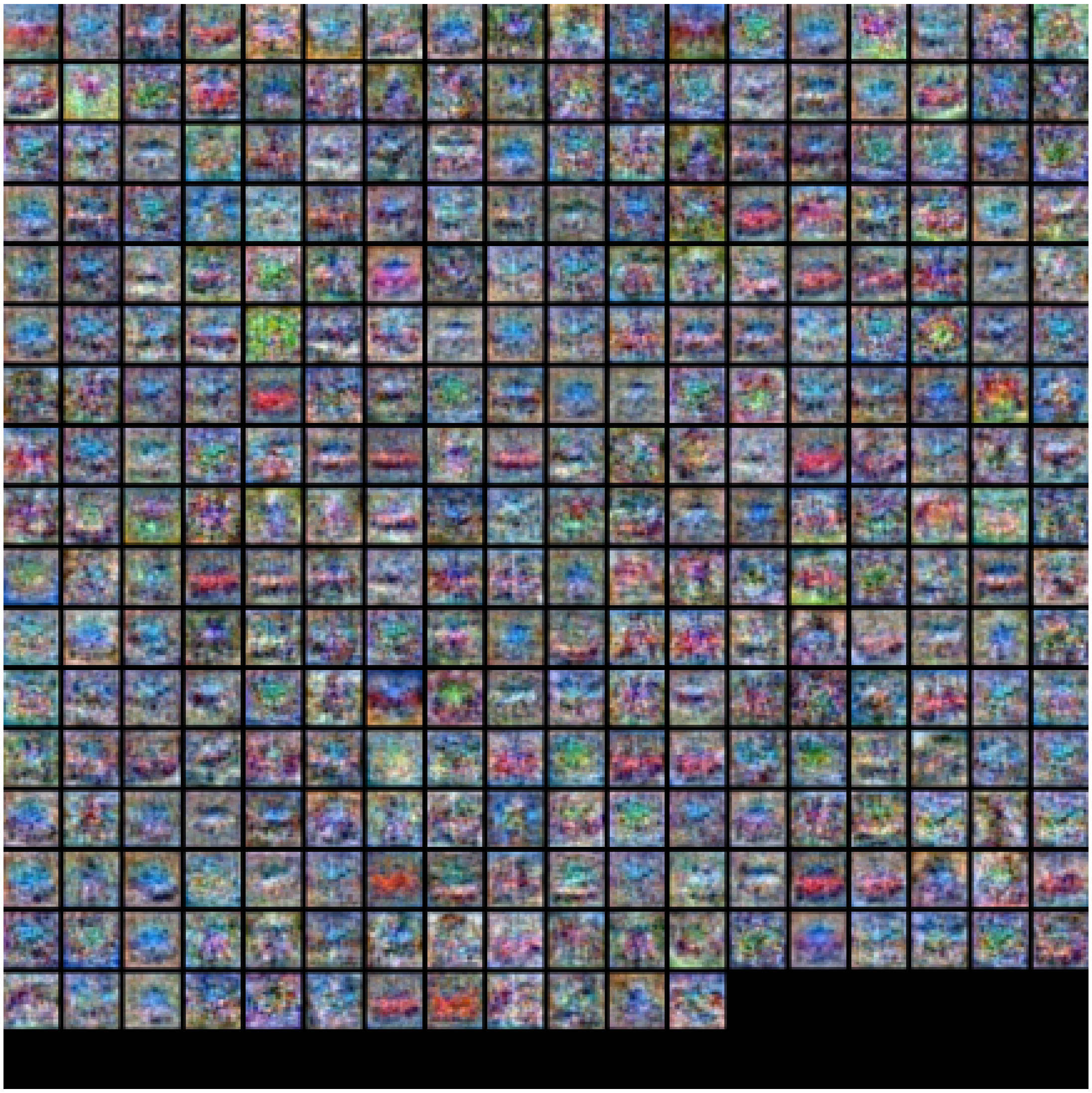
[14]:

```python
out_train = best_net.predict(x_train)
acc = get_classification_accuracy(out_train, y_train)
print("Training acc: ", acc)
out_val = best_net.predict(x_val)
acc = get_classification_accuracy(out_val, y_val)
print("Validation acc: ", acc)

best_accuracy = acc
best_net_hyperparams = [0.1, 0.001, 200, 300]

print("Best validation accuracy:", best_accuracy)
print("Best hyperparameters:", best_net_hyperparams)

# TODO: Plot the training_error and validation_accuracy of the best network (5%)
plt.subplot(2, 1, 1)
plt.plot(train_error)
plt.title("Training Loss History")
plt.xlabel("Iteration")
plt.ylabel("Loss")

plt.subplot(2, 1, 2)
# plt.plot(stats['train_acc_history'], label='train')
plt.plot(validation_accuracy, label="val")
plt.title("Classification accuracy history")
plt.xlabel("Epoch")
plt.ylabel("Classification accuracy")
plt.legend()
plt.show()

# TODO: visualize the weights of the best network (5%)
def show_net_weights(net):
    W1 = best_net._modules[0].parameters[0]
    W1 = W1.reshape(3, 32, 32, -1).transpose(3, 1, 2, 0)
    plt.imshow(visualize_grid(W1, padding=3).astype("uint8"))
    plt.gca().axis("off")
    plt.show()

show_net_weights(best_net)
```

```
Training acc:  0.698
Validation acc:  0.436
Best validation accuracy: 0.436
Best hyperparameters: [0.1, 0.001, 200, 300]
```


Training Loss History
Loss
Iteration
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0
25
50
75
100
125
150
175
200


Classification accuracy history
val
Classification accuracy
Epoch
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
10
20
30
40
50
60



3 Run on the test set (30%)

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 35%.

[15]:
```
test_acc = (best_net.predict(x_test) == y_test).mean()
print("Test accuracy: ", test_acc)
```

```
Test accuracy:  0.39
```

Inline Question (9%) Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

Your Answer: I would say 1. train on a larger dataset, and 3. increase the regularization strength.

Your Explanation: Increasing the amount of training data can help the model generalize better to unseen examples, which helps reduce overfitting. Increasing the regularization strength would also help, as it can help prevent overfitting by reducing complexity and improving generalization.

3.1 Survey (1%)

3.1.1 Question:

How many hours did you spend on this assignment?

3.1.2 Your Answer:

12 hours (the bulk of it being waiting for the training)