

```

1 # Week7- Homework
2 # Taken from https://www.kaggle.com/datduyn/2-layer-net-on-weeds-discriminant/
3 # From the set of images captured by the UAV, all those with occurrence of weeds were selected resulting
4 # a total of 400 images. Through the Pynovisão software, using the SLIC algorithm, these images were segmented
5 # and segments annotated manually with their respective class. These segments were used in construction of image dataset.
6 # This image dataset has 15336 segments, being 3249 of soil, 7376 of soybean, 3520 grass and 1191 of broadleaf weeds.
7 # Import Dependencies
8 import os #for OS dependent functionalities
9 from os import listdir, makedirs, getcwd, remove
10 import numpy as np
11 import cv2
12 import glob
13 import matplotlib.pyplot as plt
14 %matplotlib inline
15 from google.colab import drive # to save model etc on google colab drive
16
17 # Configuration
18 # set default size of plots
19 plt.rcParams['figure.figsize'] = (19.0, 17.0)
20 plt.rcParams['image.interpolation'] = 'nearest'
21 plt.rcParams['image.cmap'] = 'gray'
22
23 # To load an extension while IPython is running, use the %load_ext magic
24 # Autoreload to reload external python modules in Notebook
25 # see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
26 %load_ext autoreload
27 %autoreload 2
28
29 ##### Download Soybean Image Data From Kaggle #####
30
31 print("\n" + '\033[1m \033[4mDownload Soybean Data From Kaggle\033[0m' + "\n")
32 # Setup the Kaggle environemt for image download
33 os.environ['KAGGLE_USERNAME'] = 'XXXX'
34 os.environ['KAGGLE_KEY'] = 'XXXXXXXX'
35 # Download the Kaggle Soybean dataset
36 !kaggle datasets download -d fpeccia/weed-detection-in-soybean-crops
37 # unzip the Soybean images
38 !unzip weed-detection-in-soybean-crops.zip
39
40 # Data preprocessing
41 data_dir = r'./dataset/dataset/'
42 classes = ['broadleaf', 'grass', 'soil', 'soybean']
43
44 num_file = 1100
45 all_files = []
46 num_data = num_file*len(classes)
47 Y = np.zeros(num_data)
48
49 # get all files names into all_files array
50 for i, cls in enumerate(classes):
51     all_files += [f for f in glob.glob(data_dir+cls+'/*.tif')][:num_file]
52     Y[i*num_file:(i+1)*num_file] = i # label all classes with int [0.. len(classes)]
53
54 # Image dimension
55 im_width = 200
56 im_height = 200
57 im_channel = 3
58 dim = im_width * im_height * im_channel
59
60 X = np.ndarray(shape=(num_data, im_width, im_height, im_channel), dtype=np.float64)
61
62
63
64 # OpenCV-Python is a library of Python bindings designed to solve computer vision problems.
65 # All the OpenCV array structures are converted to and from Numpy arrays.
66 # This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib

```

```
00 # This makes it easier to integrate with other libraries that use numpy such as scipy and matplotlib.
67 # Resizing an image means changing the dimensions of it, be it width alone, height alone or both.
68 # Also, the aspect ratio of the original image could be preserved in the resized image.
69 # enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.
70 # This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.
71 for idx, file in enumerate(all_files):
72     X[idx] = cv2.resize(cv2.imread(file), (im_width, im_height))
73
74
75 # Prepare data for training, validation and testing the model
76
77 X_train = np.empty(shape=(4000,im_width, im_height, im_channel), dtype=np.float64)
78 X_val = np.empty(shape=(200,im_width, im_height, im_channel), dtype=np.float64)
79 X_test = np.empty(shape=(200,im_width, im_height, im_channel), dtype=np.float64)
80
81 y_train = np.empty(4000)
82 y_val = np.empty(200)
83 y_test = np.empty(200)
84
85 for i, cls in enumerate(classes):
86     X_test[50*i:50*(i+1)] = X[np.where(Y == i)[0][:50]]
87     X_val[50*i:50*(i+1)] = X[np.where(Y == i)[0][50:100]]
88     X_train[1000*i:1000*(i+1)] = X[np.where(Y == i)[0][100:]]
89
90     y_test[50*i:50*(i+1)] = i
91     y_val[50*i:50*(i+1)] = i
92     y_train[1000*i:1000*(i+1)] = i
93
94 del Y
95 del X
96
97 # Extract features
98 # Shuffle training index
99 train_idxs = np.random.permutation(X_train.shape[0])
100 # DataFrame.astype() method is used to cast a pandas object to a specified dtype.
101 # astype() function also provides the capability to convert any suitable existing column to categorical type.
102 # DataFrame.astype() function comes very handy when we want to case a particular column data type to another data type.
103 y_train = y_train[train_idxs].astype(int)
104 X_train = X_train[train_idxs]
105
106 X_train = np.reshape(X_train, (X_train.shape[0], -1)).astype('float64')
107 X_test = np.reshape(X_test, (X_test.shape[0], -1)).astype('float64')
108 X_val = np.reshape(X_val, (X_val.shape[0], -1)).astype('float64')
109
110 X_tiny = X_train[100:110].astype('float64')
111 y_tiny = y_train[100:110].astype(int)
112 num_dev = 500
113
114 X_dev = X_train[0:num_dev].astype('float64')
115 y_dev = y_train[0:num_dev].astype(int)
116 print("X_train shape", X_train.shape, "| y_train shape:", y_train.shape)
117 print("X_test shape", X_test.shape, "| y_test shape:", y_test.shape)
118 print("X_val shape", X_val.shape, "| y_val shape:", y_val.shape)
119 print("X_dev shape", X_dev.shape, "| y_dev shape:", y_dev.shape)
120 print("X_tiny shape", X_tiny.shape, "| y_tiny shape:", y_tiny.shape)
121
122 #Subtract out the mean image
123 #first: compute the mean image
124 # mean_image = np.mean(X_train, axis=0) #axis=0. stack horizontally
125 mean_image = 128
126 #Second subtract the mean image from train and test data
127 X_train -= mean_image
128 X_val -= mean_image
129 X_test -= mean_image
130 X_dev -= mean_image
131 X_tiny -= mean_image
132
133 #Third append the bias dimension using linear algebra trick
```

```

134 # Not for net
135 # X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
136 # X_val = np.hstack([X_val, np.ones((X_val.shape[0], 1))])
137 # X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
138 # X_dev = np.hstack([X_dev, np.ones((X_dev.shape[0], 1))])
139 # X_tiny = np.hstack([X_tiny, np.ones((X_tiny.shape[0], 1))])
140
141 print('=====STACK BIAS term=====')
142 print("X_train shape", X_train.shape)
143 print("X_test shape", X_test.shape)
144 print("X_val shape", X_val.shape)
145 print("X_dev shape", X_dev.shape)
146 print("X_tiny shape", X_tiny.shape)
147
148 # Visualize some images
149 # Make sure that everything when OK
150 classes = ['broadleaf', 'grass', 'soil', 'soybean']
151 n_class = len(classes)
152 samples_per_class = 4
153
154 for y, cls in enumerate(classes):
155     idxes = np.flatnonzero(y == y_train)
156     idxes = np.random.choice(idxes, samples_per_class, replace = False)
157     for i, idx in enumerate(idxes):
158         plt_idx = i * n_class + y + 1
159         plt.subplot(samples_per_class,n_class, plt_idx)
160         plt.imshow(X_train[idx].reshape(im_width, im_height, im_channel).astype('uint8'))
161         if(i==0): plt.title(cls)
162
163 plt.show()
164
165 # 2 Layer Neural net
166 # Stacked RELU activation function.
167 class TwoLayerNet():
168     def __init__(self, input_size, hidden_size, output_size, std= 1e-4):
169         """
170             std: weight initialization term
171             W1: first layer weight, shape(D x H)
172             W2: second layer weight shape(H x C)
173             C: num_classes(output_size) , H: hidden_size, D: data_dim(input_size)
174         """
175         self.params = {}
176         self.params['W1'] = std * np.random.randn(input_size, hidden_size)
177         self.params['b1'] = np.zeros(hidden_size)
178         self.params['W2'] = std * np.random.randn(hidden_size, output_size)
179         self.params['b2'] = np.zeros(output_size)
180
181     def loss(self, X, y = None, reg=0.0):
182         """
183             reg: regularization strength
184             X: ndarray shape(N x C). N: num of data
185             y: vector of training label
186         """
187         # Define relu activation function
188         relu = lambda x:np.maximum(0,x)
189
190         # unpack
191         W1, b1 = self.params['W1'], self.params['b1']
192         W2, b2 = self.params['W2'], self.params['b2']
193         N, D = X.shape
194
195         # Forward prop
196         layer1 = relu(X.dot(W1) + b1)  # (N,D) x (D,H) = (N,H)
197         scores = layer1.dot(W2) + b2
198
199         # if target is not given then jump out
200

```

```

200     if(y is None):
201         return scores
202
203     # compute the loss
204     ## Normalization trick to prevent overflow when compute exp
205     scores -= scores.max()#stack vertically
206
207     scores = np.exp(scores)
208     scores_sumexp = np.sum(scores, axis=1)#stack vertically
209
210     ## Normalize all score
211     softmax = scores / scores_sumexp.reshape(N,1)  #Shape: (N, C)
212     #total loss of all training. -log of all correct score
213     loss = (-1.0) * np.sum(np.log(softmax[range(N),y]))
214
215     ##Normalize the loss and add regularization strength
216     loss /= N
217     loss += reg * np.sum(W1 * W1)
218     loss += reg * np.sum(W2 * W2)
219
220     #Backward pass on the net
221     grads = {}
222
223     correct_class_scores = scores[range(N), y]
224     softmax[range(N), y] = (-1.0) * (scores_sumexp - correct_class_scores)/scores_sumexp
225     softmax /= N
226
227
228     #Want to find dW2(dL/dW2)
229     # Derivation: dL/dW2 = dL/dscore * dscore/dW2(chain rule)
230     #dL/dscore = softmax since L(score) = softmax(variable)
231     #dscore/dW2 = relu_(hidden layer output)
232     grads['W2'] = layer1.T.dot(softmax)
233     grads['b2'] = np.sum(softmax, axis=0)#stack horizontally
234     grads['W2'] += reg * 2 * W2
235
236     #dL/dW1 = dL/dscore * dscore/drelu(layler1) * drelu(layer1)/dW1
237     #dL/dW1 = dW1 = softmax * W2 * X
238     hidden = softmax.dot(W2.T)
239
240     #derivative of a max gate
241     #Intuition: in forward pass if neuron didn't fire that mean. the derivative of that neuron
242     # is 0. This might be bad since this will kill gradient.
243     hidden[layer1 == 0] = 0
244
245     grads['W1'] = X.T.dot(hidden)
246     grads['b1'] = np.sum(hidden, axis=0) #stack horizontally
247     grads['W1'] += reg * 2 * W1
248
249     return loss, grads
250
251 def train(self, X, y, X_val, y_val,
252           learning_rate =1e-3, learning_rate_decay=0.95,
253           reg=5e-6, num_iters=100,
254           batch_size=200, it_verbose = 1, verbose=False):
255 ...
256
257     Train using SGD
258     Input:
259         X: nd array shape(N x D)
260         y: vector of train label
261         X_val: nd array shape( n_VAL , D) Use as validation set after each epoch
262         y_val: vector of validation label
263 ...
264         N, D = X.shape
265         N_val = X_val.shape[0]
266         iteration_per_epoch = max(N/batch_size, 1)
267

```

```

267     loss_hist = []
268     train_acc_hist = []
269     val_acc_hist = []
270
271     for it in range(num_iters):
272         sampling = np.random.choice(np.arange(N), batch_size, replace=False)
273         X_batch = X[sampling]
274         y_batch = y[sampling]
275
276         #compute loss and gradients
277         loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
278         loss_hist.append(loss)
279
280         #Update rule
281         self.params['W1'] += (-1.0) * learning_rate * grads['W1']
282         self.params['b1'] += (-1.0) * learning_rate * grads['b1']
283         self.params['W2'] += (-1.0) * learning_rate * grads['W2']
284         self.params['b2'] += (-1.0) * learning_rate * grads['b2']
285
286         if(verbose and it%it_verbose==0):
287             print('iteration: %d / %d | Loss: %f' % (it, num_iters, loss))
288         # Every epoch, check train and val accuracy and decay learning rate.
289         if (it % iteration_per_epoch == 0):
290             # Check accuracy
291             train_acc = (self.predict(X_batch) == y_batch).mean()
292             val_acc = (self.predict(X_val) == y_val).mean()
293             train_acc_hist.append(train_acc)
294             val_acc_hist.append(val_acc)
295
296             # Decay learning rate
297             learning_rate *= learning_rate_decay
298
299     return {
300         'loss_hist':loss_hist,
301         'train_acc_hist':train_acc_hist,
302         'val_acc_hist':val_acc_hist
303     }
304
305     def predict(self, X):
306         """
307             Use the trained weights of this two-layer network to predict labels for
308             data points. For each data point we predict scores for each of the C
309             classes, and assign each data point to the class with the highest score.
310
311             Inputs:
312             - X: A numpy array of shape (N, D) giving N D-dimensional data points to
313                 classify.
314
315             Returns:
316             - y_pred: A numpy array of shape (N,) giving predicted labels for each of
317                 the elements of X. For all i, y_pred[i] = c means that X[i] is predicted
318                 to have class c, where 0 <= c < C.
319
320             y_pred = None
321             relu = lambda x:np.maximum(0,x)
322             # Unpack variables from the params dictionary
323             W1, b1 = self.params['W1'], self.params['b1']
324             W2, b2 = self.params['W2'], self.params['b2']
325
326             #Forward propagation though the network
327             layer1 = relu(X.dot(W1) + b1)
328             scores = layer1.dot(W2) + b2 #shape: (N x C)
329             y_pred = np.argmax(scores, axis=1)
330
331             return y_pred
332
333             input_size = im_width * im_height * im_channel
334             hidden_size = 200

```

```

334 output_size = n_class
335 std = 1e-3 # size initialization parameter
336
337 net = TwoLayerNet(input_size, hidden_size, output_size, std )
338 stats = net.train(X_dev, y_dev, X_val, y_val,
339             learning_rate = 1e-5, learning_rate_decay=0.95,
340             reg=0.0, num_iters=70,
341             batch_size=100, it_verbose = 10, verbose=True)
342
343 # plot loss history and train/ validation accuracies history
344 plt.rcParams['figure.figsize'] = (15.0, 15.0) # set default size of plots
345 plt.subplot(2,1,1)
346 plt.plot(stats['loss_hist'])
347 plt.title('Loss History')
348 plt.xlabel('Iteration')
349 plt.ylabel('Loss')
350
351 plt.subplot(2,1,2)
352 plt.plot(stats['train_acc_hist'], label='train')
353 plt.plot(stats['val_acc_hist'], label='val')
354 plt.xlabel('Epoch')
355 plt.ylabel('Classification Accuracies')
356 plt.legend()
357 plt.show()
358
359 print((net.predict(X_test) == y_test).mean())
360
361 # Hyperparameter search
362 best_net = None # store the best model into this
363 best_val= -1 #highest validation accuracy
364
365
366 hidden_unit = [200]
367 learn_rates = [7.6e-5]
368 regularizations = [0.0]
369 iterations = [600]
370
371 result = {}
372 best_stats = None
373
374 input_size = im_height * im_width * im_channel #
375 output_size = 4 #4 class
376 for hidden in hidden_unit:
377     for learn in learn_rates:
378         for r in regularizations:
379             for iter in iterations:
380                 tune_net = TwoLayerNet(input_size,
381                             hidden_size=hidden,
382                             output_size=output_size, std=1e-3)
383                 stats = tune_net.train(X_train, y_train, X_val, y_val,
384                             num_iters=iter, batch_size=200,
385                             learning_rate=learn, learning_rate_decay=0.94,
386                             reg=r, it_verbose = 100, verbose=True)
387                 train_acc = stats['train_acc_hist'][-1]#get last value
388                 val_acc = stats['val_acc_hist'][-1]
389                 result[(hidden, learn)] = (train_acc, val_acc)
390                 #print log
391                 print('hs:',hidden,'learn:',learn,'reg',r,'iter',iter,'train-acc:',train_acc,'val_acc',val_acc)
392                 if(val_acc > best_val):
393                     best_val = val_acc
394                     #create best net
395                     best_stats = stats
396                     best_net = tune_net
397                     del tune_net
398                     del stats
399
400 print("Accuracy on Test set", (best_net.predict(X_test) == y_test).mean())

```

```
401
402 # plot loss history and train/ validation accuracies history
403 plt.rcParams['figure.figsize'] = (15.0, 15.0) # set default size of plots
404 plt.subplot(2,1,1)
405 plt.plot(best_stats['loss_hist'])
406 plt.title('Loss History')
407 plt.xlabel('Iteration')
408 plt.ylabel('Loss')
409
410 plt.subplot(2,1,2)
411 plt.plot(best_stats['train_acc_hist'], label='train')
412 plt.plot(best_stats['val_acc_hist'], label='val')
413 plt.xlabel('Epoch')
414 plt.ylabel('Classification Accuracies')
415 plt.legend()
416 plt.show()
417
418 plt.hist(best_net.predict(X_test))
419
```

⇨

Download Soybean Data From Kaggle

```
Downloading weed-detection-in-soybean-crops.zip to /content
100% 2.36G/2.37G [00:36<00:00, 22.0MB/s]
100% 2.37G/2.37G [00:36<00:00, 70.3MB/s]
Archive: weed-detection-in-soybean-crops.zip
  inflating: dataset/broadleaf/1.tif
  inflating: dataset/broadleaf/10.tif
  inflating: dataset/broadleaf/100.tif
  inflating: dataset/broadleaf/1000.tif
  inflating: dataset/broadleaf/1001.tif
  inflating: dataset/broadleaf/1002.tif
  inflating: dataset/broadleaf/1003.tif
  inflating: dataset/broadleaf/1004.tif
  inflating: dataset/broadleaf/1005.tif
  inflating: dataset/broadleaf/1006.tif
  inflating: dataset/broadleaf/1007.tif
  inflating: dataset/broadleaf/1008.tif
  inflating: dataset/broadleaf/1009.tif
  inflating: dataset/broadleaf/101.tif
  inflating: dataset/broadleaf/1010.tif
  inflating: dataset/broadleaf/1011.tif
  inflating: dataset/broadleaf/1012.tif
  inflating: dataset/broadleaf/1013.tif
  inflating: dataset/broadleaf/1014.tif
  inflating: dataset/broadleaf/1015.tif
  inflating: dataset/broadleaf/1016.tif
  inflating: dataset/broadleaf/1017.tif
  inflating: dataset/broadleaf/1018.tif
  inflating: dataset/broadleaf/1019.tif
  inflating: dataset/broadleaf/102.tif
  inflating: dataset/broadleaf/1020.tif
  inflating: dataset/broadleaf/1021.tif
  inflating: dataset/broadleaf/1022.tif
  inflating: dataset/broadleaf/1023.tif
  inflating: dataset/broadleaf/1024.tif
  inflating: dataset/broadleaf/1025.tif
  inflating: dataset/broadleaf/1026.tif
  inflating: dataset/broadleaf/1027.tif
  inflating: dataset/broadleaf/1028.tif
  inflating: dataset/broadleaf/1029.tif
  inflating: dataset/broadleaf/103.tif
  inflating: dataset/broadleaf/1030.tif
  inflating: dataset/broadleaf/1031.tif
  inflating: dataset/broadleaf/1032.tif
  inflating: dataset/broadleaf/1033.tif
  inflating: dataset/broadleaf/1034.tif
  inflating: dataset/broadleaf/1035.tif
  inflating: dataset/broadleaf/1036.tif
  inflating: dataset/broadleaf/1037.tif
  inflating: dataset/broadleaf/1038.tif
  inflating: dataset/broadleaf/1039.tif
  inflating: dataset/broadleaf/104.tif
  inflating: dataset/broadleaf/1040.tif
  inflating: dataset/broadleaf/1041.tif
  inflating: dataset/broadleaf/1042.tif
  inflating: dataset/broadleaf/1043.tif
  inflating: dataset/broadleaf/1044.tif
  inflating: dataset/broadleaf/1045.tif
  inflating: dataset/broadleaf/1046.tif
  inflating: dataset/broadleaf/1047.tif
  inflating: dataset/broadleaf/1048.tif
  inflating: dataset/broadleaf/1049.tif
  inflating: dataset/broadleaf/105.tif
  inflating: dataset/broadleaf/1050.tif
  inflating: dataset/broadleaf/1051.tif
  inflating: dataset/broadleaf/1052.tif
  inflating: dataset/broadleaf/1053.tif
  inflating: dataset/broadleaf/1054.tif
  inflating: dataset/broadleaf/1055.tif
  inflating: dataset/broadleaf/1056.tif
  inflating: dataset/broadleaf/1057.tif
  inflating: dataset/broadleaf/1058.tif
```



```
-----  
inflating: dataset/soil/2749.tif  
inflating: dataset/soil/2751.tif  
inflating: dataset/soil/2750.tif  
inflating: dataset/soil/2752.tif  
inflating: dataset/soil/2753.tif  
inflating: dataset/soil/2754.tif  
inflating: dataset/soil/2755.tif  
inflating: dataset/soil/2756.tif  
inflating: dataset/soil/2757.tif  
inflating: dataset/soil/2758.tif  
inflating: dataset/soil/2759.tif  
inflating: dataset/soil/2760.tif  
inflating: dataset/soil/2761.tif  
inflating: dataset/soil/2762.tif  
inflating: dataset/soil/2763.tif  
inflating: dataset/soil/2764.tif  
inflating: dataset/soil/2765.tif  
inflating: dataset/soil/2766.tif  
inflating: dataset/soil/2767.tif  
inflating: dataset/soil/2768.tif  
inflating: dataset/soil/2769.tif  
inflating: dataset/soil/2770.tif  
inflating: dataset/soil/2771.tif  
inflating: dataset/soil/2772.tif  
inflating: dataset/soil/2773.tif  
inflating: dataset/soil/2774.tif  
inflating: dataset/soil/2775.tif  
inflating: dataset/soil/2776.tif  
inflating: dataset/soil/2777.tif  
inflating: dataset/soil/2778.tif  
inflating: dataset/soil/2779.tif  
inflating: dataset/soil/2780.tif  
inflating: dataset/soil/2781.tif  
inflating: dataset/soil/2782.tif  
inflating: dataset/soil/2783.tif  
inflating: dataset/soil/2784.tif  
inflating: dataset/soil/2785.tif  
inflating: dataset/soil/2786.tif  
inflating: dataset/soil/2787.tif  
inflating: dataset/soil/2788.tif  
inflating: dataset/soil/2789.tif  
inflating: dataset/soil/2790.tif  
inflating: dataset/soil/2791.tif  
inflating: dataset/soil/2792.tif  
inflating: dataset/soil/2793.tif  
inflating: dataset/soil/2794.tif  
inflating: dataset/soil/2795.tif  
inflating: dataset/soil/2796.tif  
inflating: dataset/soil/2797.tif  
inflating: dataset/soil/2798.tif  
inflating: dataset/soil/2799.tif  
inflating: dataset/soil/2800.tif  
inflating: dataset/soil/2801.tif  
inflating: dataset/soil/2802.tif  
inflating: dataset/soil/2803.tif  
inflating: dataset/soil/2804.tif  
inflating: dataset/soil/2805.tif  
inflating: dataset/soil/2806.tif  
inflating: dataset/soil/2807.tif  
inflating: dataset/soil/2808.tif  
inflating: dataset/soil/2809.tif  
inflating: dataset/soil/2810.tif  
inflating: dataset/soil/2811.tif  
inflating: dataset/soil/2812.tif  
inflating: dataset/soil/2813.tif  
inflating: dataset/soil/2814.tif  
inflating: dataset/soil/2815.tif
```


inflating: dataset/soybean/132.tif
inflating: dataset/soybean/133.tif
inflating: dataset/soybean/134.tif
inflating: dataset/soybean/135.tif
inflating: dataset/soybean/136.tif
inflating: dataset/soybean/137.tif
inflating: dataset/soybean/138.tif
inflating: dataset/soybean/139.tif
inflating: dataset/soybean/1398.tif
inflating: dataset/soybean/1399.tif
inflating: dataset/soybean/14.tif
inflating: dataset/soybean/140.tif
inflating: dataset/soybean/1400.tif
inflating: dataset/soybean/1401.tif
inflating: dataset/soybean/1402.tif
inflating: dataset/soybean/1403.tif
inflating: dataset/soybean/1404.tif
inflating: dataset/soybean/1405.tif
inflating: dataset/soybean/1406.tif
inflating: dataset/soybean/1407.tif
inflating: dataset/soybean/1408.tif
inflating: dataset/soybean/1409.tif
inflating: dataset/soybean/141.tif
inflating: dataset/soybean/1410.tif
inflating: dataset/soybean/1411.tif
inflating: dataset/soybean/1412.tif
inflating: dataset/soybean/1413.tif
inflating: dataset/soybean/1414.tif
inflating: dataset/soybean/1415.tif
inflating: dataset/soybean/1416.tif
inflating: dataset/soybean/1417.tif
inflating: dataset/soybean/1418.tif
inflating: dataset/soybean/1419.tif
inflating: dataset/soybean/142.tif
inflating: dataset/soybean/1420.tif
inflating: dataset/soybean/1421.tif
inflating: dataset/soybean/1422.tif
inflating: dataset/soybean/1423.tif
inflating: dataset/soybean/1424.tif
inflating: dataset/soybean/1425.tif
inflating: dataset/soybean/1426.tif
inflating: dataset/soybean/1427.tif
inflating: dataset/soybean/1428.tif
inflating: dataset/soybean/1429.tif
inflating: dataset/soybean/143.tif
inflating: dataset/soybean/1430.tif
inflating: dataset/soybean/1431.tif
inflating: dataset/soybean/1432.tif
inflating: dataset/soybean/1433.tif
inflating: dataset/soybean/1434.tif
inflating: dataset/soybean/1435.tif
inflating: dataset/soybean/1436.tif
inflating: dataset/soybean/1437.tif
inflating: dataset/soybean/1438.tif
inflating: dataset/soybean/1439.tif

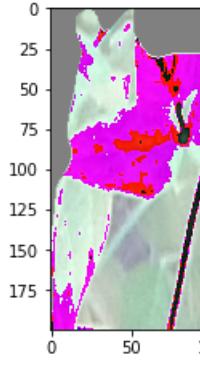
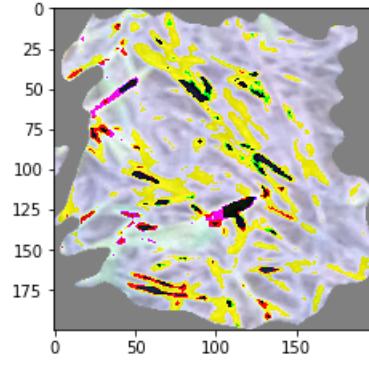
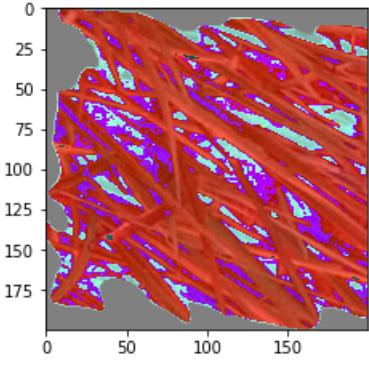
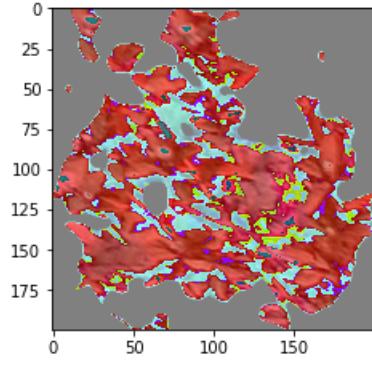
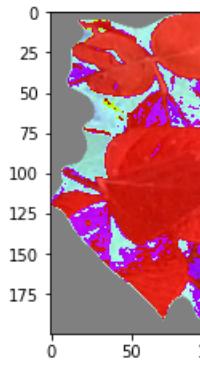
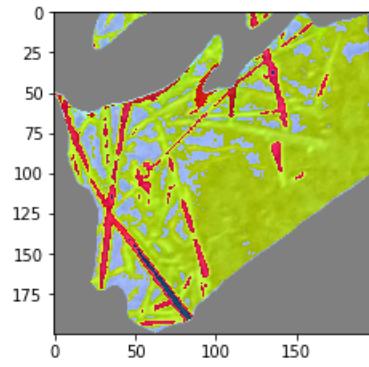
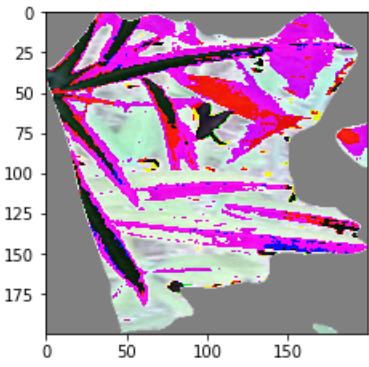
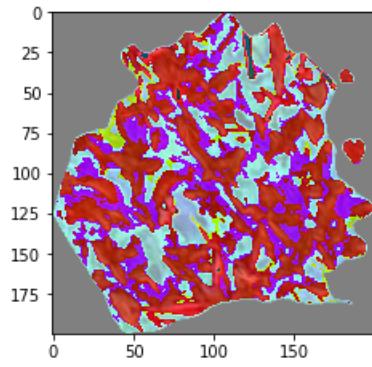
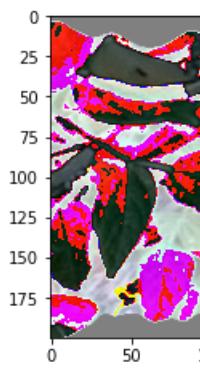
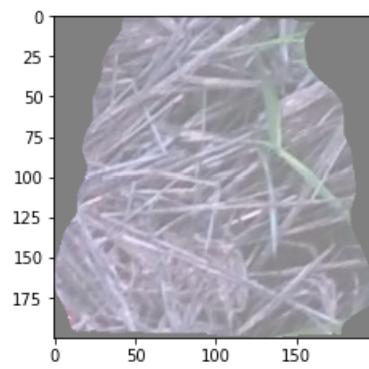
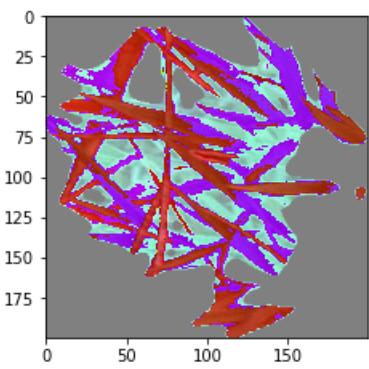
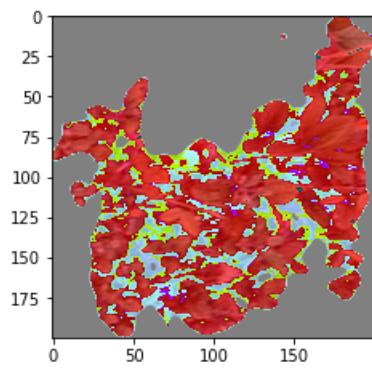
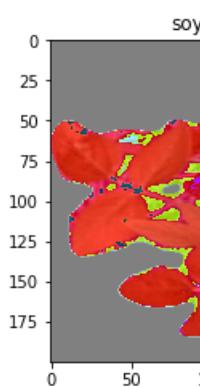
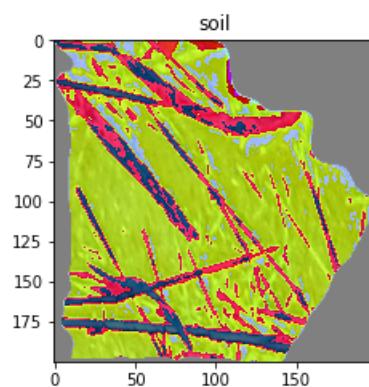
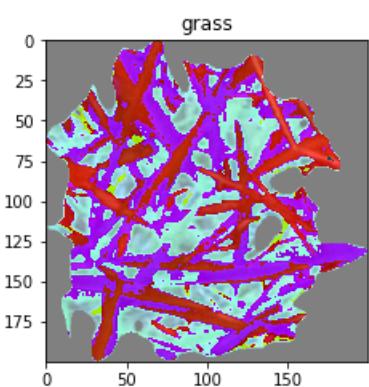
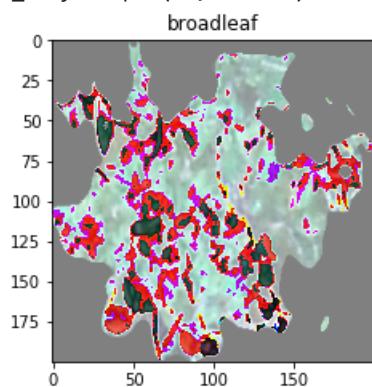

```
inflating: dataset/soybean/40.tif
inflating: dataset/soybean/401.tif
inflating: dataset/soybean/402.tif
inflating: dataset/soybean/403.tif
inflating: dataset/soybean/404.tif
inflating: dataset/soybean/405.tif
inflating: dataset/soybean/406.tif
inflating: dataset/soybean/407.tif
inflating: dataset/soybean/408.tif
inflating: dataset/soybean/409.tif
inflating: dataset/soybean/410.tif
inflating: dataset/soybean/411.tif
inflating: dataset/soybean/412.tif
inflating: dataset/soybean/413.tif
inflating: dataset/soybean/414.tif
inflating: dataset/soybean/415.tif
inflating: dataset/soybean/416.tif
inflating: dataset/soybean/417.tif
inflating: dataset/soybean/418.tif
inflating: dataset/soybean/419.tif
inflating: dataset/soybean/420.tif
inflating: dataset/soybean/421.tif
inflating: dataset/soybean/422.tif
inflating: dataset/soybean/423.tif
inflating: dataset/soybean/424.tif
inflating: dataset/soybean/425.tif
inflating: dataset/soybean/426.tif
inflating: dataset/soybean/427.tif
inflating: dataset/soybean/428.tif
inflating: dataset/soybean/429.tif
inflating: dataset/soybean/430.tif
inflating: dataset/soybean/431.tif
inflating: dataset/soybean/432.tif
inflating: dataset/soybean/433.tif
inflating: dataset/soybean/434.tif
inflating: dataset/soybean/435.tif
inflating: dataset/soybean/436.tif
inflating: dataset/soybean/437.tif
inflating: dataset/soybean/438.tif
inflating: dataset/soybean/439.tif
inflating: dataset/soybean/440.tif
inflating: dataset/soybean/441.tif
```



```

inflating: dataset/soybean/998.tif
inflating: dataset/soybean/999.tif
X_train shape (4000, 120000) | y_train shape: (4000,)
X_test shape (200, 120000) | y_test shape: (200,)
X_val shape (200, 120000) | y_val shape: (200,)
X_dev shape (500, 120000) | y_dev shape: (500,)
X_tiny shape (10, 120000) | y_tiny shape: (10,)
=====STACK BIAS term=====
X_train shape (4000, 120000)
X_test shape (200, 120000)
X_val shape (200, 120000)
X_dev shape (500, 120000)
X_tiny shape (10, 120000)

```

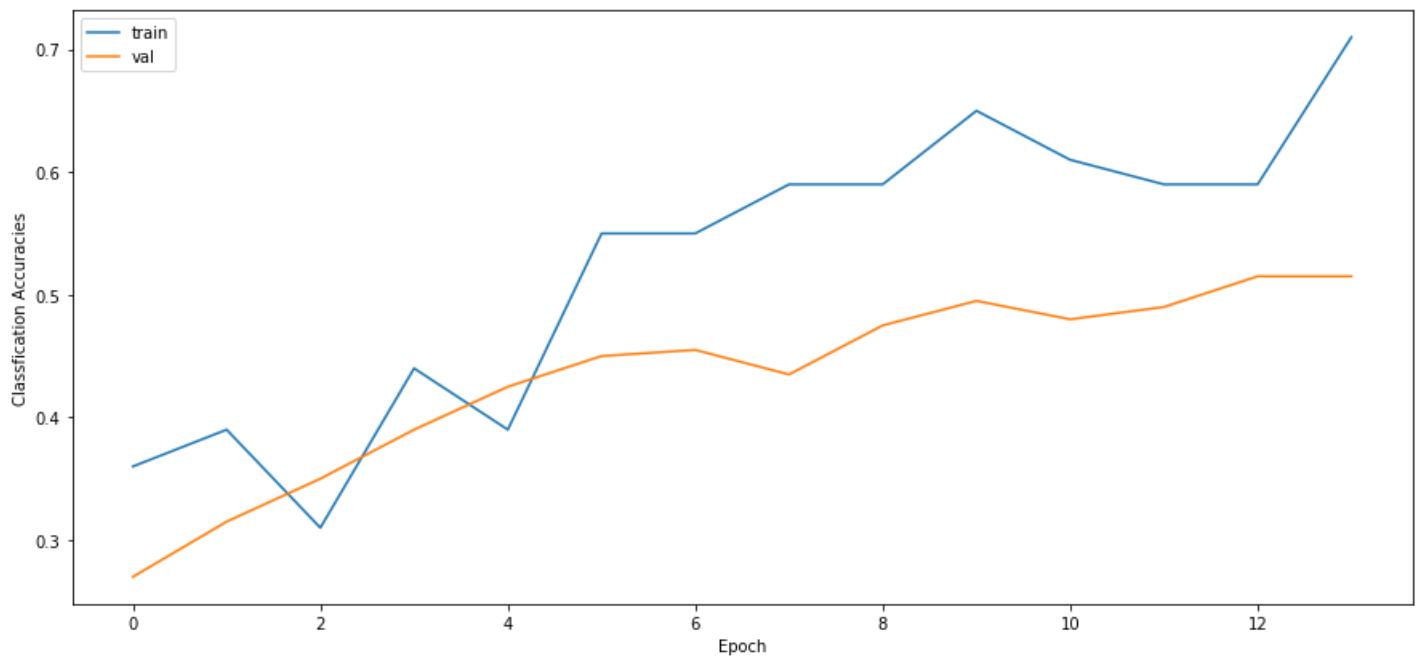
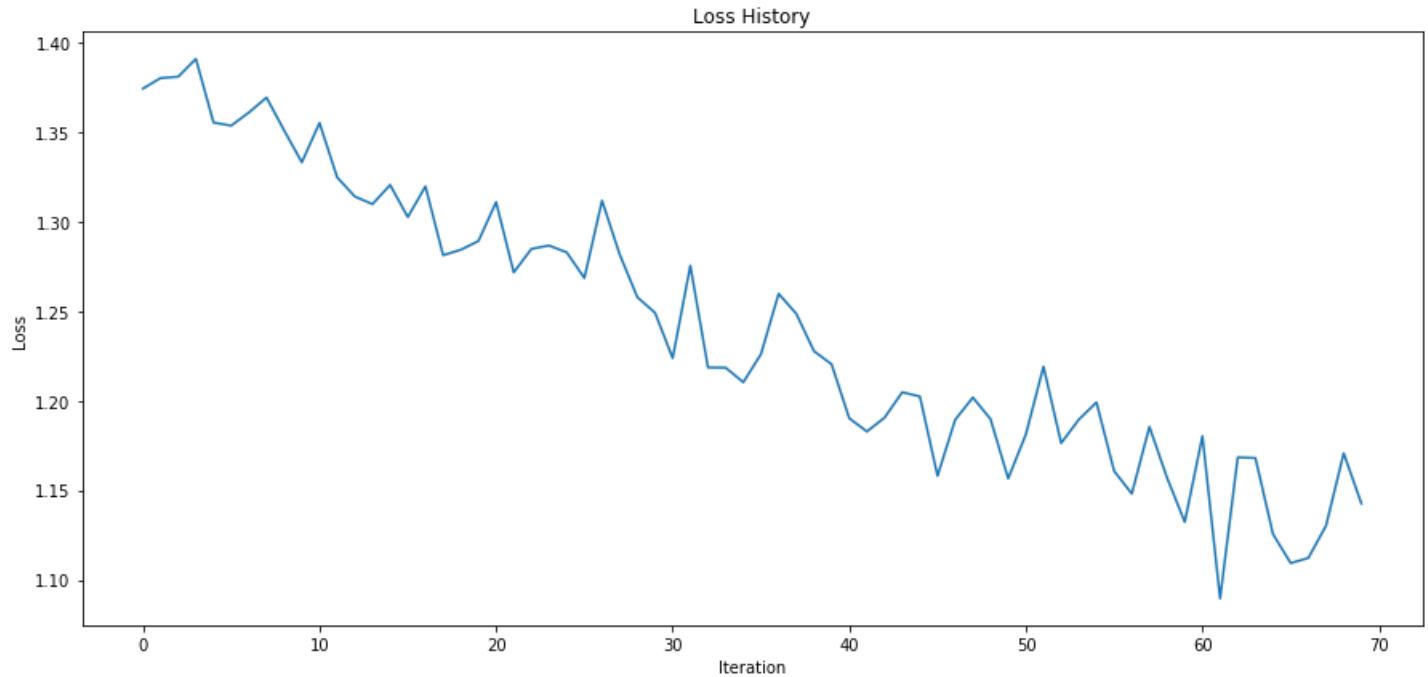


```

iteration: 0 / 70 | Loss: 1.374283
iteration: 10 / 70 | Loss: 1.355155
iteration: 20 / 70 | Loss: 1.310931
iteration: 30 / 70 | Loss: 1.223833
iteration: 40 / 70 | Loss: 1.190251

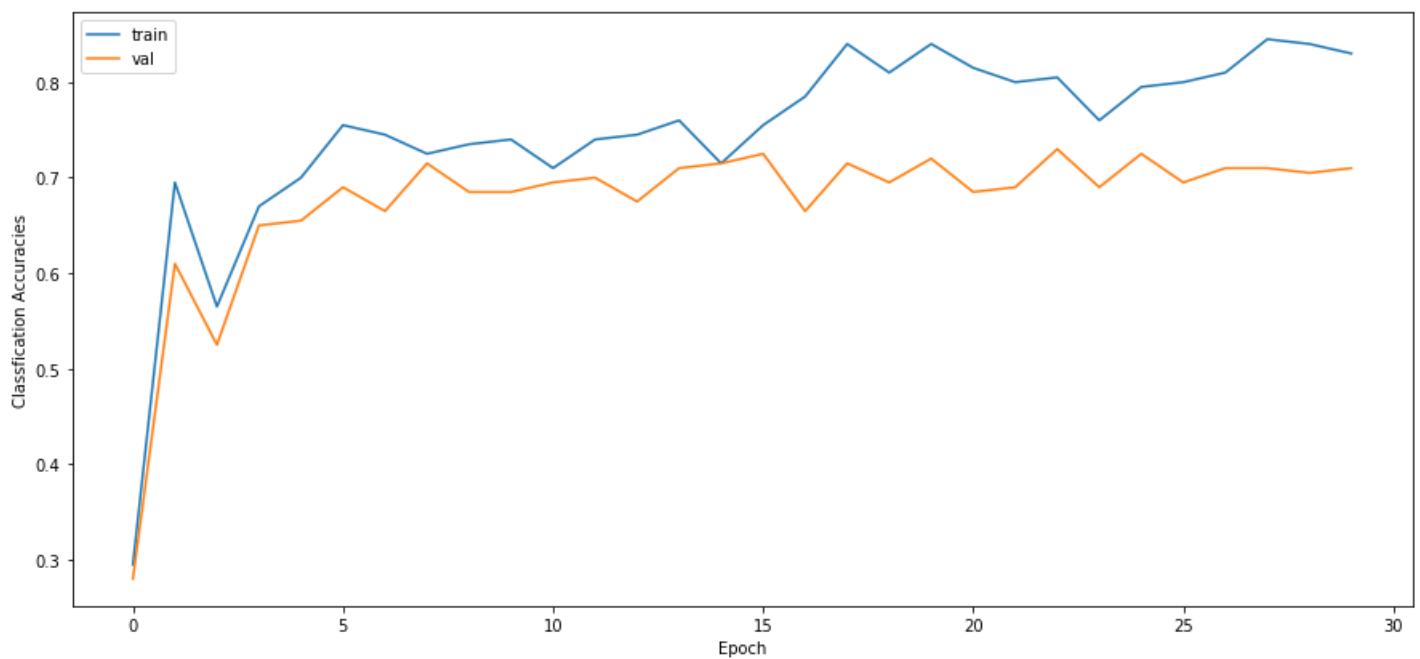
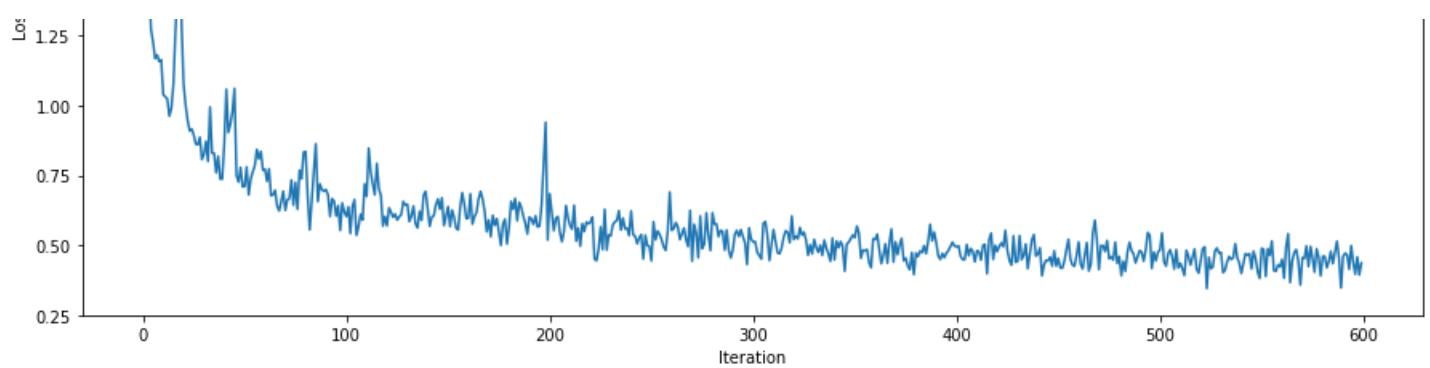
```

iteration: 50 / 70 | Loss: 1.181205
iteration: 60 / 70 | Loss: 1.180321



0.6
iteration: 0 / 600 | Loss: 1.436724
iteration: 100 / 600 | Loss: 0.603214
iteration: 200 / 600 | Loss: 0.684007
iteration: 300 / 600 | Loss: 0.512452
iteration: 400 / 600 | Loss: 0.494978
iteration: 500 / 600 | Loss: 0.483569
hs: 200 learn: 7.6e-05 reg 0.0 iter 600 train-acc: 0.83 val_acc 0.71
Accuracy on Test set 0.745





```
(array([56.,  0.,  0., 40.,  0.,  0., 51.,  0.,  0., 53.]),  
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),  
<a list of 10 Patch objects>)
```

