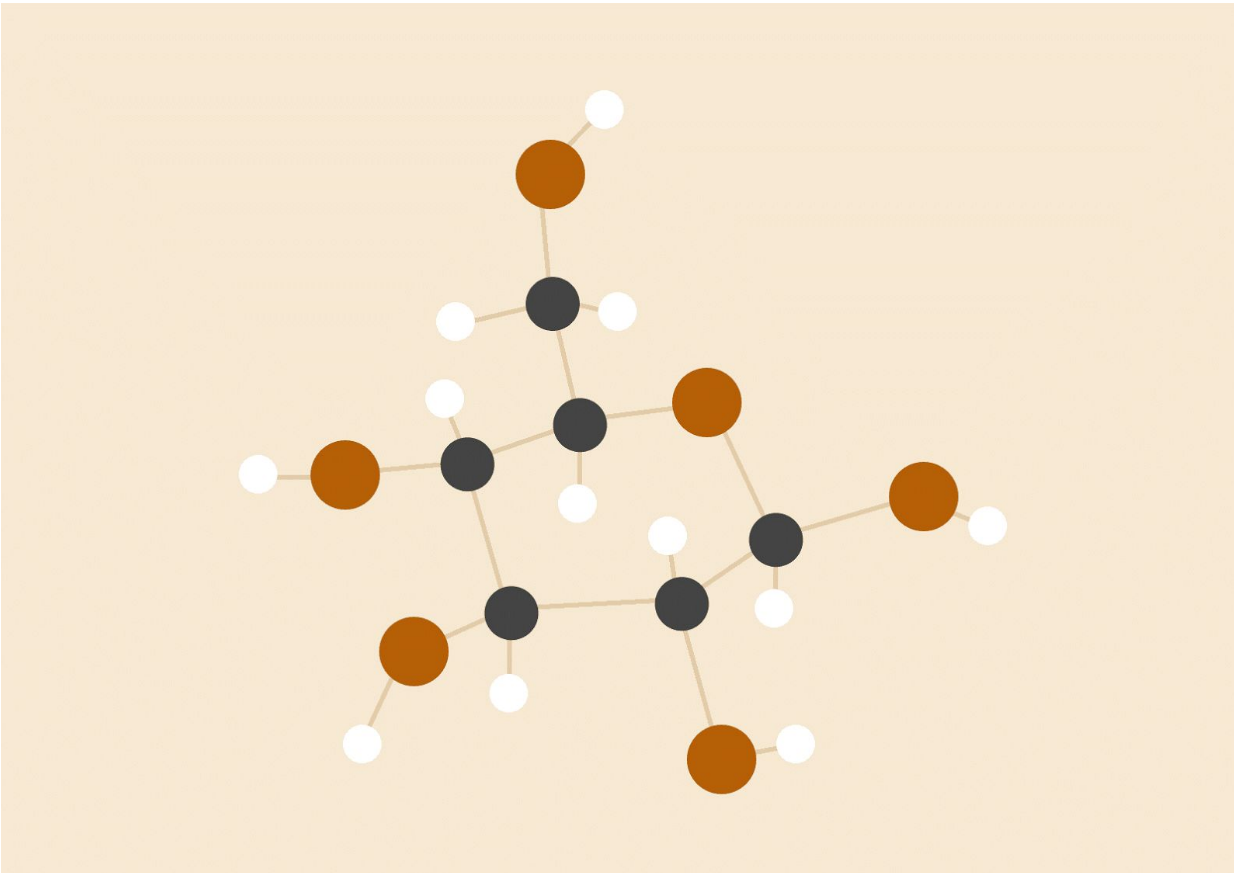
Big data analysis

Skin detection with ELM



Group3: Skin detection

31.08.2016

ARCADA

INTRODUCTION

This report explains the process and results of our big data analysis-project. The idea is to use ELM neural network model to identify skin pixels in JPEG images. ELM model can classify pixels after it is trained with a training set. Report explains how tests were done as well as the results and how we reached over 80% accuracy in pixel classification.

INPUT DATA AND PROCESSING

1. Programming and runtime environment (Arcada Big Data Lab)

- OS: Linux/Ubuntu
- GPU: Nvidia GeForce GTX
- Python 2.7
- HDF5
- Python based HP-ELM toolbox

2. Reading images into HDF5 files and normalizing data

Training set pictures and masks were read into a HDF5 database and normalized (formula). There were a total of 147 features (7x7x3=147 i.e. 7x7=49 pixels and three colors: RGB) in each pixel in the training set pictures. Each pixel is a one line in a HDF5 file. For masks we had only 2 features: [1,0] for skin pixel and [0,1] for non-skin pixel. These HDF5 files were used as input to train the ELM.

3. ELM training and classification

For classification we used the HP-ELM toolbox (https://github.com/akusok/hpelm). Each pixel in a picture was classified one row (from left to right) at a time except a 3 pixel margin around the picture. Since we used a 7x7 pixel block in the classification and that leaves a 3 pixel margin.

We created the ELM model with 147 input neurons (one input neuron for each feature) and two output neurons which gave us classification: [1,0] for skin pixel and [0,1] for non-skin pixel.

In the hidden layer there were 3072 hidden neurons in our model.

Output neurons needed to be biased, because there are just a few skin pixels in images and most pixels are non-skin.

The process:

1. We trained the ELM with a training set of 1.300 pictures and skin masks as a result we got the trained ELM model.
2. With the trained ELM model we used the validation set of 700 pictures to create as many skin masks.
3. Lastly we compared output masks from step 2 with given validation masks to get the accuracy of our trained ELM model.

4. File sizes

Training set file sizes:

- Training set original images: ~100 MB (1300 JPEG images)
- As pixel vectors in compressed HD5F file: ~9 GB
- Training set skin mask images: ~1 GB (1300 BMP images)
- As classification data in compressed HD5F file: 19 MB

Validation set file sizes:

- Validation set original images: ~85 MB (700 JPEG images)
- As pixel vectors in compressed HD5F file: ~4.9 GB
- Validation set skin mask images: ~500 MB (700 BMP images)
- As classification data in compressed HD5F file: ~10 MB

Test set file sizes:

- Test set original images: ~235 MB (2000 JPEG images)
- As pixel vectors in compressed HD5F file: ~15 GB

RESULTS

Number of pixels in each set if not counting the 3x3 image border

- Train: 342003478
- Validation: 183245863
- Test: 560606236

Time it took to process the images and create the HD5F files

- Train: 20 hrs 30 min
- Validation: ~11 hrs
- Test: ~30 hrs

Training was done with 3072 hidden neuron ELM using the training set data as input. This took approximately 10 hrs 45 minutes using the Arcada Big Data lab computers and GPU acceleration (GeForce GTX GPU).

Predicting the classifications for the validation set using the 3072 hidden neuron model took about 25 minutes using the Arcada Big Data lab computers and GPU acceleration.

Predicting classifications for the test set using the 3072 hidden neuron model took about 2 hrs 40 minutes using the Arcada Big Data lab computers and GPU acceleration.

CONFUSION MATRIX

Based on the real output data and predicted output data generated by our ELM model we calculated a confusion matrix (below table). With the confusion matrix we can calculate the accuracy of our model.

	actual true	actual false
predicted true	10,19%	3,62%
predicted false	8,93%	77,25%

Which gives us (18 681 377+141 558 322)/(18 681 377+141 558 322+6 649 637+16 356 527) = 160 239 699/183 245 863 = 87,4% accuracy rate.

MODEL SELECTION

For model selection we used 2765 hidden neurons in our ELM model and decreased the amount of hidden neurons by 10% in each step until we reached only 15 hidden neurons. Accuracy was calculated based on “true positive”, “false positive”, “false negative” and “true negative” with the same formula as we did in the confusion matrix calculation.

Table 1. Full table of outputs

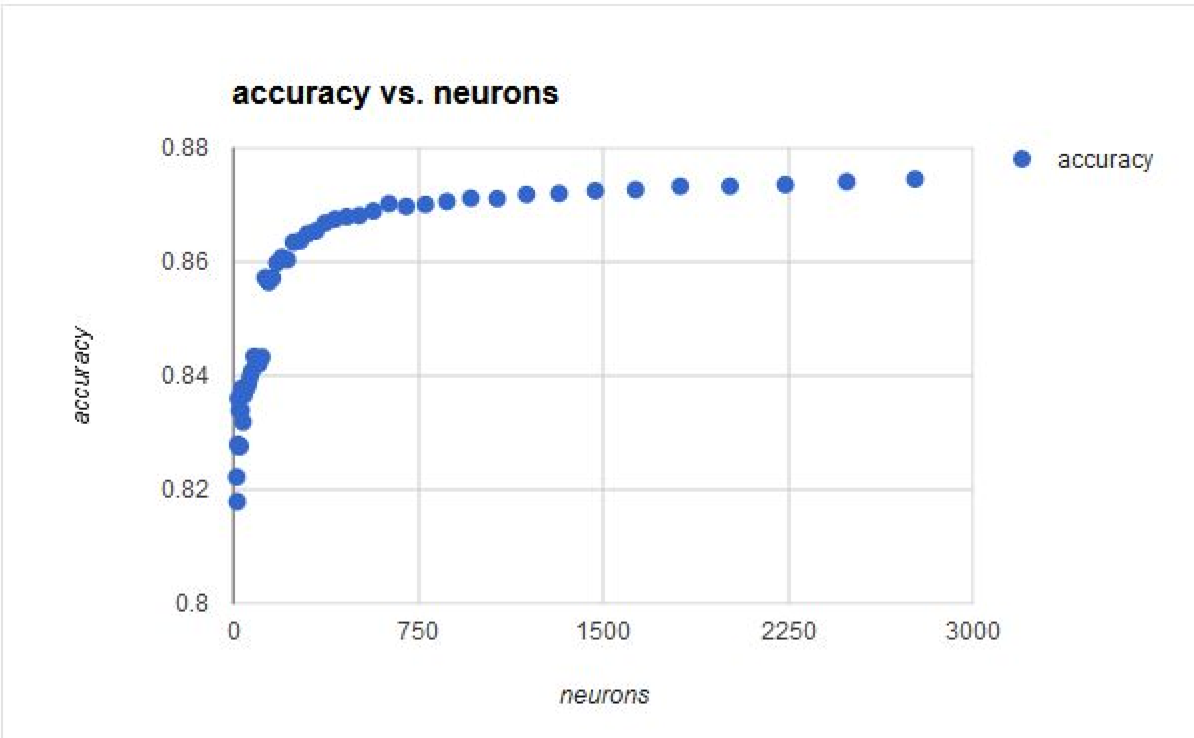
No. of neurons	Accuracy	True positive	False positive	False negative	True negative
2765	0.8745348428	18720199	6673266	16317705	141534693
2488	0.8740698719	18651945	6690216	16385959	141517743
2239	0.8735506405	18421674	6555092	16616230	141652867
2015	0.8732767189	18435055	6618668	16602849	141589291
1813	0.8732757749	18488235	6672021	16549669	141535938
1632	0.8726704078	18296781	6591498	16741123	141616461
1469	0.8724868403	18205542	6533897	16832362	141674062
1322	0.8720109987	18026052	6441603	17011852	141766356

1190	0.8718292047	18094210	6543074	16943694	141664885
1071	0.8710881457	17887104	6471764	17150800	141736195
964	0.8711648623	17917300	6487902	17120604	141720057
867	0.8705916433	17844747	6520389	17193157	141687570
780	0.8700855364	17670921	6439305	17366983	141768654
702	0.8697251845	17767154	6601571	17270750	141606388
632	0.8702117821	17727866	6473116	17310038	141734843
569	0.8689228799	17568962	6550398	17468942	141657561
512	0.8681303817	17131597	6258255	17906307	141949704
461	0.8679186007	17011329	6176795	18026575	142031164
414	0.867502373	17053231	6294969	17984673	141912990
373	0.8668278367	16598460	5963804	18439444	142244155
336	0.8654127488	16400693	6025346	18637211	142182613
302	0.8649374147	16279262	5991018	18758642	142216941
272	0.863683853	15898570	5840036	19139334	142367923
245	0.8634574795	15743357	5726305	19294547	142481654
220	0.8603662556	14934654	5484056	20103250	142723903
198	0.8608020635	15142221	5611763	19895683	142596196
178	0.8598178612	14721595	5371488	20316309	142836471
160	0.8571658559	14464097	5599959	20573807	142608000

144	0.8564026845	13886970	5162680	21150934	143045279
130	0.8572173332	14388550	5514979	20649354	142692980
117	0.843300959	10380402	4056949	24657502	144151010
105	0.8420909562	10234914	4133189	24802990	144074770
94	0.8424175775	9650642	3489065	25387262	144718894
85	0.8433989203	10763631	4422227	24274273	143785732
76	0.8408085098	9792067	3925345	25245837	144282614
69	0.8398692199	9682604	3988003	25355300	144219956
62	0.8386047657	9531152	4068257	25506752	144139702
56	0.8378767438	9497124	4167636	25540780	144040323
50	0.8376200777	9294518	4012063	25743386	144195896
45	0.8366368195	8918166	3815889	26119738	144392070
40	0.8318680733	8891384	4662960	26146520	143544999
36	0.8378251028	8958970	3638945	26078934	144569014
33	0.8338758349	8763676	4167338	26274228	144040621
29	0.8276796841	8758917	5297998	26278987	142909961
26	0.8339440165	9119200	4510368	25918704	143697591
24	0.8275001166	8215033	4787019	26822871	143420940
21	0.8360124015	8157133	3169278	26880771	145038681
19	0.827965857	8058211	4544852	26979693	143663107

17	0.8179117473	7257590	5586605	27780314	142621354
15	0.8222214599	5882108	3421386	29155796	144786573

Figure 1. Chart drawn from the neurons and accuracy columns of table 1



In figure 1 it can be seen that increasing the amount of neurons from 15 to ~130 hidden neurons increases the accuracy radically. After 130 neurons the accuracy is not increasing in the same pace anymore and the curve becomes almost flat. Already with 15 hidden neurons we reach more than 80% accuracy. With 130 hidden neurons we reach 86% accuracy and with 2765 only 1% more i.e. 87%. With 117 hidden neurons the accuracy is only 84% which makes the optimal amount of hidden neurons to be somewhere around 130 if we want to optimize for maximal accuracy with minimal hidden neurons. Accuracy seems to increase when the amount of hidden neurons is increased but using a big amount of hidden neurons consumes much more resources and time.

How amount of hidden neurons affects model computing time:

Models 15 to 373 Training times between 15min to 30min

Models 415 to 632 Training times between 30min to 1hrs

Models 702 to 1071 Training times between 1hrs to 2hrs

Models 1190 to 1469 Training times between 2hrs to 3hrs

Models 1632 3 hrs 22 min

Models 1813 4 hrs 6 min

Models 2015 4 hrs 48 min

Models 2239 5 hrs 55 min

Models 2488 7 hrs 17 min

Models 2765 9 hrs 30 min

PYTHON CODE

How we ran our code:

1) Start bash in screen.
   Run: screen bash
2) Execute python job in screen and append output to a logfile.
   Run: python hd5f-util.py | tee logfile.txt
3) Put screen in background.
   Run: ctrl+ad
4) Follow the processing by reading the logfile.
   Run: tail -f logfile.txt

Code for pre-processing the image files and storing them in compressed hdf5 -files.

hd5f-util.py

```
import os
import h5py
import numpy as np
from tables import open_file, Atom, Filters
from PIL import Image
import datetime

train_skinmask_path = r'/home/bdalab2/Desktop/Group3/Images/Skin/train'
train_pictures_path = r'/home/bdalab2/Desktop/Group3/Images/Original/train'

validation_skinmask_path = r'/home/bdalab2/Desktop/Group3/Images/Skin/val'
validation_pictures_path = r'/home/bdalab2/Desktop/Group3/Images/Original/val'

test_pictures_path = r'/home/bdalab2/Desktop/Group3/Images/Original/test'

output_train_skinmask_classes = r'/home/bdalab2/Desktop/Group3/input/train_skinmask_classes.h5'
output_train_pixel_vectors = r'/home/bdalab2/Desktop/Group3/input/train_pixel_vectors.h5'
output_train_image_shapes = r'/home/bdalab2/Desktop/Group3/input/train_shapes.h5'

output_validation_skinmask_classes = r'/home/bdalab2/Desktop/Group3/input/validation_skinmask_classes.h5'
output_validation_pixel_vectors = r'/home/bdalab2/Desktop/Group3/input/validation_pixel_vectors.h5'
output_validation_image_shapes = r'/home/bdalab2/Desktop/Group3/input/validation_shapes.h5'

output_test_pixel_vectors = r'/home/bdalab2/Desktop/Group3/input/test_pixel_vectors.h5'
output_test_image_shapes = r'/home/bdalab2/Desktop/Group3/input/test_shapes.h5'

def store_skinmask_classes_in_hd5_file( image_directory, outfile ):
  "Reads all image files from image_directory and stores the images pixel skinmask classes in compressed hdf5 file"

  # Open h5 outputfile for writing.
  # Note: "w"-mode overwrites h5 file contents.
  h5 = open_file(outfile, "w")

  # Process all files in image_directory.
  lst = os.listdir(image_directory)
  lst.sort()
  for file_name in lst:
      # Opens the image.
      im = Image.open(image_directory + '\\' + file_name)
      image_width = im.size[0]
      image_height = im.size[1]
      pixels = im.load()
      skinclass = 0
      # Skip 3 pixel image border..
      for y in range(3, image_height - 3, 1):
          # Skip 3 pixel image border..
          for x in range(3, image_width - 3, 1):
              # Get pixel RGB-values at coordinates x,y.
```

```
            pixel = pixels[x,y]
            if( (pixel[0] == 255) and (pixel[1] == 255) and (pixel[2] == 255) ):
                # Not classified as skin if R, G, B (pixel[0], pixel[1], pixel[2]) are set to 255 (white color).
                skinclass = [-1]
            else:
                # Anything else is classified as skin.
                skinclass = [+1]
            # Create node /classes in h5 file if it does not exist.
            if not h5.__contains__("/classes"):
                a = Atom.from_dtype(np.dtype(np.float64), dflt=0)
                flt = Filters(complevel=1, shuffle=True)
                h5classes = h5.create_earray(h5.root, "classes", a, (0, 2), "Output classes", filters=flt)
            # Append skinclass data to /classes node.
            if skinclass == [-1]:
                h5classes.append([[0, 1]])
            else:
                h5classes.append([[1, 0]])

  h5.flush()
  h5.close()
  return;

def store_image_rgb_values_in_hd5_file( image_directory, outfile_pixels, outfile_shapes ):
  "Processes all images in image_directory. Reads pixel RGB values in 7x7 area around the classified pixel. Normalizes the RGB
values and stores them in compressed hdf5 file. Image shapes are stored in separate hdf5 file."

  # Open h5 outputfile for writing.
  # Note: "w"-mode overwrites h5 file contents.
  h5_pixels = open_file(outfile_pixels, "w")
  h5_shapes = open_file(outfile_shapes, "w")

  # Process all files in image_directory.
  lst = os.listdir(image_directory)
  lst.sort()
  for file_name in lst:
      # Opens the image.
      im = Image.open(image_directory + '\\' + file_name)
      image_width = im.size[0]
      image_height = im.size[1]
      pixels = im.load()
      # todo: Maybe this code block could be optimized somehow.. -->
      # Skip 3 pixel image border..
      for y in range(3, image_height - 3, 1):
          # Skip 3 pixel image border..
          for x in range(3, image_width - 3, 1):
              rgb_values = None
              # We are now at the center of the 7x7 pixel area.
              # Start processing from top left corner of the 7x7 area.
              # Process the first 'row' of 7 pixels and then proceed to next 'row' in the 7x7 area.
              for yp in range(-3, +4, 1):
                  for xp in range (-3, +4, 1):
                      # Get pixel rgb values.
                      pixel = pixels[x + xp, y + yp]
                      # On first iteration set variable.
                      if rgb_values == None:
                          # Normalize data: subtract -128 from pixel rgb value and divide by 64.
                          rgb_values = [((pixel[0] - 128) / 64), ((pixel[1] - 128) / 64), ((pixel[2] - 128) / 64)]
                      else:
                          # If not first iteration, append to variable.
                          # Normalize data: subtract -128 from pixel rgb value and divide by 64.
                          rgb_values = np.append(rgb_values,[((pixel[0] - 128) / 64), ((pixel[1] - 128) / 64), ((pixel[2] -
128) / 64)])
              if not h5_pixels.__contains__("/rgb"):
                  a = Atom.from_dtype(np.dtype(np.float64), dflt=0)
                  flt = Filters(complevel=1, shuffle=True)
                  h5pxl = h5_pixels.create_earray(h5_pixels.root, "rgb", a,(0,147), "rgb-values", filters=flt)
              # Append rgb_values data to /rgb node.
              h5pxl.append(rgb_values[np.newaxis,:])
              # <-- todo: Maybe this code block could be optimized somehow..
      # Create node /shapes in h5 file if it does not exist.
      if not h5_shapes.__contains__("/shapes"):
          a = Atom.from_dtype(np.dtype(np.uint), dflt=0)
          flt = Filters(complevel=1, shuffle=True)
          h5shps = h5_shapes.create_earray(h5_shapes.root, "shapes", a, (0,2), "Output shapes", filters=flt)
      # Append shape data to /shapes node.
      h5shps.append([[image_width - 6, image_height - 6]])
  h5_pixels.flush()
  h5_shapes.flush()
  h5_pixels.close()
  h5_shapes.close()
  return;

## This will take a long while to run..
print("Starting.")
```

```
print(str(datetime.datetime.now()))
# Train
store_skinmask_classes_in_hd5_file(train_skinmask_path, output_train_skinmask_classes)
store_image_rgb_values_in_hd5_file(train_pictures_path, output_train_pixel_vectors, output_train_image_shapes)
# Validation
store_skinmask_classes_in_hd5_file(validation_skinmask_path, output_validation_skinmask_classes)
store_image_rgb_values_in_hd5_file(train_pictures_path, output_train_pixel_vectors, output_train_image_shapes)
# Test
store_image_rgb_values_in_hd5_file(test_pictures_path, output_test_pixel_vectors, output_test_image_shapes)
print("Done.")
print(str(datetime.datetime.now()))
```

Code for calculating the class weight percentage.

calculate_nonskin_pixels.py

```
import h5py
import numpy as np

input_train_skinmask_classes = r'/home/bdalab2/Desktop/Group3/input/validation_skinmask_classes.h5'

# Read the hdf5 file.
with h5py.File(input_train_skinmask_classes, 'r') as hf:
   # get /classes node
   data = hf.get('classes')
   # Store /classes node contents in numpy array
   np_data = np.array(data)

nonskin = 0
pixels = 0
for clazz in np_data:
   pixels = pixels + 1
   if clazz[0] == 0 and clazz[1] == 1:
       nonskin = nonskin + 1

print("Total number of classified pixels in training set:")
print(str(pixels))
print("Total number of pixels classified as non-skin in training set:")
print(str(nonskin))
percentage_nonskin = float(nonskin) / float(pixels)
print("Percentage of pixels classified as non-skin in training set:")
print(str(percentage_nonskin))
```

Code for batch processing class weighted GPU accelerated HPELMs.

batch_hpelm_gpu.py

```
import hpelm
import numpy as np
import h5py
from PIL import Image
import datetime
import os
from tables import open_file, Atom, Filters

input_train_pixel_vectors = r'/home/bdalab2/Desktop/Group3/input/train_pixel_vectors.h5'
input_validation_pixel_vectors = r'/home/bdalab2/Desktop/Group3/input/validation_pixel_vectors.h5'

input_train_skinmask_classes = r'/home/bdalab2/Desktop/Group3/input/train_skinmask_classes.h5'
input_validation_skinmask_classes = r'/home/bdalab2/Desktop/Group3/input/validation_skinmask_classes.h5'

output_trained_model = r'/home/bdalab2/Desktop/Group3/model/'
output_predicted_classes = r'/home/bdalab2/Desktop/Group3/predicted_classes/'


def run_elm( neurons ):

  "Creates HPELM which uses GPU Acceleration and Class Balancing. 'neurons' parameter sets number of neurons in ELM"
  ## Train set pixel classes are 84% non-skin.
  w = np.array([1 / 0.16 , 1 / 0.84])
```

```
  # Initialize HPELM
  model = hpelm.HPELM(inputs=147, outputs=2, accelerator="GPU", classification="wc", w=w)
  model.add_neurons(neurons,"sigm")

  print("Training model" + str(neurons))
  print(str(datetime.datetime.now()))
  # Train the model.
  model.train(input_train_pixel_vectors, input_train_skinmask_classes)
  print("Done training model" + str(neurons))
  print(str(datetime.datetime.now()))

  # Stores the calculated model in outfile_model
  # The model can be loaded using model.load()
  model.save(output_trained_model + str(neurons) + ".hd5")

  print("Predicting classes Validation" + str(neurons))
  print(str(datetime.datetime.now()))
  # Predict and save output classes (Validation)
  model.predict(input_validation_pixel_vectors, output_predicted_classes + str(neurons) + ".hd5")
  print("Done predicting classes Validation" + str(neurons))
  print(str(datetime.datetime.now()))

  with h5py.File(input_validation_skinmask_classes, 'r') as hf:
      # get /data node
      data = hf.get('classes')
      # Store /data node contents in numpy array
      correct_classes = np.array(data)

  with h5py.File(output_predicted_classes + str(neurons) + ".hd5", 'r') as hf:
      # get /data node
      data = hf.get('data')
      # Store /data node contents in numpy array
      predicted_classes = np.array(data)

  tp=0
  tn=0
  fp=0
  fn=0
  i=0

  for predictions in predicted_classes:
      # If predicted as skin
      if predicted_classes[i][0] > predicted_classes[i][1]:
     # And correct class is skin
     if correct_classes[i][0] == 1:
         tp = tp + 1
     else:
         fp = fp +1
      # Predicted as non-skin
      else:
     # And correct class is non-skin
     if correct_classes[i][1] == 1:
         tn = tn + 1
     else:
         fn = fn +1
      i = i +1

  print(str(neurons))
  print(str("True Positive:"))
  print(tp)
  print(str("True Negative:"))
  print(tn)
  print(str("False Positive:"))
  print(fp)
  print(str("False Negative:"))
  print(fn)

  print(str("Accuracy:"))
  accuracy = float(tp + tn) / float(tp + tn + fp + fn)
  print(accuracy)

  # Append results to CSV-file.
  with open("/home/bdalab2/Desktop/Group3/accuracy/accuracy.txt", "a") as myfile:
          myfile.write(str(neurons) + ";" + str(accuracy) + ";" + str(tp) + ";" + str(fp) + ";" + str(fn) + ";" + str(tn) +
"\n")
          myfile.close()

  return;

## Calculate models for different number of neurons.
## This will take a long time; Check back in a a couple of days. ;-)
run_elm(3072)
run_elm(2765)
```

```
run_elm(2488)
run_elm(2239)
run_elm(2015)
run_elm(1813)
run_elm(1632)
run_elm(1469)
run_elm(1322)
run_elm(1190)
run_elm(1071)
run_elm(964)
run_elm(867)
run_elm(780)
run_elm(702)
run_elm(632)
run_elm(569)
run_elm(512)
run_elm(461)
run_elm(414)
run_elm(373)
run_elm(336)
run_elm(302)
run_elm(272)
run_elm(245)
run_elm(220)
run_elm(198)
run_elm(178)
run_elm(160)
run_elm(144)
run_elm(130)
run_elm(117)
run_elm(105)
run_elm(94)
run_elm(85)
run_elm(76)
run_elm(69)
run_elm(62)
run_elm(56)
run_elm(50)
run_elm(45)
run_elm(40)
run_elm(36)
run_elm(33)
run_elm(29)
run_elm(26)
run_elm(24)
run_elm(21)
run_elm(19)
run_elm(17)
run_elm(15)
```

Snippet of code used to create black and white image of predicted output data.

```
import hpelm
import h5py
import numpy as np
from PIL import Image

outfile_model = r'/home/bdalab2/Desktop/Group3/output/hpelm_model_3072_sigm.bin'
outfile_benjamin = r'/home/bdalab2/Desktop/Group3/benjamin/benjamin.hd5'
benjamin_predictions = r'/home/bdalab2/Desktop/Group3/benjamin/predictions_benjamin.hd5'

model = hpelm.HPELM(inputs=147, outputs=2, accelerator="GPU")
model.load(outfile_model)
model.predict(outfile_benjamin, benjamin_predictions)

with h5py.File(benjamin_predictions,'r') as hf:
  data=hf.get('data')
  predicted_classes = np.array(data)

## image size without 3 pixel border:
## 528 - 6 = 522
## 960 -6 = 954
w, h = 522, 954

data = np.zeros((w, h, 3), dtype=np.uint8)
for y in range(0, h, 1):
   for x in range(0, w, 1):
       if predicted_classes[i][0] > predicted_classes[i][1]:
           data[x, y] = [0, 0, 0]
```

```
        else:
            data[x, y] = [255, 255, 255]
        i = i + 1
img = Image.fromarray(data, 'RGB')
img = img.transpose(Image.FLIP_LEFT_RIGHT)
img = img.transpose(Image.ROTATE_90)
img.save('benjamin.png')
img.show()
```