CS 5600/6600: F20: Intelligent Systems
Project 1: Image and Audio Classification with ANNs and ConvNets

Vladimir Kulyukin
Department of Computer Science
Utah State University

September 29, 2020

Learning Objectives

1. Artificial Neural Networks (ANNs)
2. Convolutional Neural Networks (ConvNets)
3. Image Classification
4. Audio Classification

Introduction

This project has two objectives: 1) design, train, evaluate, and persist ANNs and ConvNets for image classification and 2) design, train, evaluate, and persist ANNs and ConvNets for audio classification.

Let me say a few words about this project to give you a bigger picture. Back in 2013, I started seeing many stories about failing honeybee colonies all over the world. The high rates of colony loss threaten to disrupt our food supply. For example, for U.S. beekeepers, the average colony loss varies from 25% to 50% per year. The cost of equipment, bee packages, maintenance, and transportation is so high that profit margins for beekeepers are very small. Pursuant to my principal research criterion that my systems not only advance science, but also affect social change, I asked myself what I could do as a computer scientist and a beekeeper to improve the health of honeybee colonies.

In 2013–2014, after studying the apiary science literature, I discovered an emerging consensus among some entomologists, computer scientists, and engineers that electronic beehive monitoring (EBM) can help extract critical information on colony behavior and phenology without invasive inspections and significant transportation costs.

In 2014, I came across Michael Nielsen's book *Reinventing Discovery: The New Era of Networked Science* (Princeton University Press, 2011). The book had a most profound intellectual impact on me as a scientist and a researcher. I learned about *Polymath*, a crowdsourced mathematics project, where spontaneous virtual communities all over the world collaborate to solve previously unsolved problems, and about *Galaxy Zoo*, a crowdsourced astronomy project, where over 250,000 amateur astronomers work together to understand the structure of the Universe and discover new galaxy types. In my subsequent research, I learned about *Bee Informed*, an apiary science network, where thousands of beekeepers, from industry leaders to backyard sideliners, collaborate to improve beekeeping practices and make apiary science discoveries. I also learned that Georgia Tech launched its own research crowdfunding site, which I find to be a remarkable example of forward thinking. These are truly inspiring stories of how researchers, amateur and professional alike, are using computer networks to increase our combined brainpower.

In 2017, I created and ran my first science crowdfunding project *BeePi: A Multisensor Electronic Beehive Monitor* on Kickstarter to crowdfund my hardware needs for my electronic beehive monitoring (EBM) and biosensor research. Since this was my first crowdfunder, my target goal was a modest $1,000. I was genuinely amazed that within 60 days 61 backers pledged $2,940 to bring my project to life. In 2019, I ran another science crowdfunding project on Kickstarter *BeePi: Honeybees Meet AI: Stage 2* with the target goal of $5,000. I was again pleasantly surprised that 2 months later 59 backers pledged $5,753 to bring my project to life.

The wonderful datasets described in the next section we'll be working with in this project come courtesy of our generous Kickstarter backers whose donations helped us build the hardware to capture the data. My point is that research can (and should!) be crowdfunded and that science crowdfunding is possible. Random acts of kindness and generosity go a long way. Before we dive into them, I'd like to thank, from the bottom of my heart, all my wonderful graduate and undergraduate students who've been busy labeling, curating, and experimenting with these datasets.

Datasets

We've created a USU Box repo where you can download the datasets. Table 1 gives the details for each dataset. These datasets were obtained from the data captured by the BeePi monitors deployed on live beehives in Logan and North Logan, UT in 2017 – 2018.

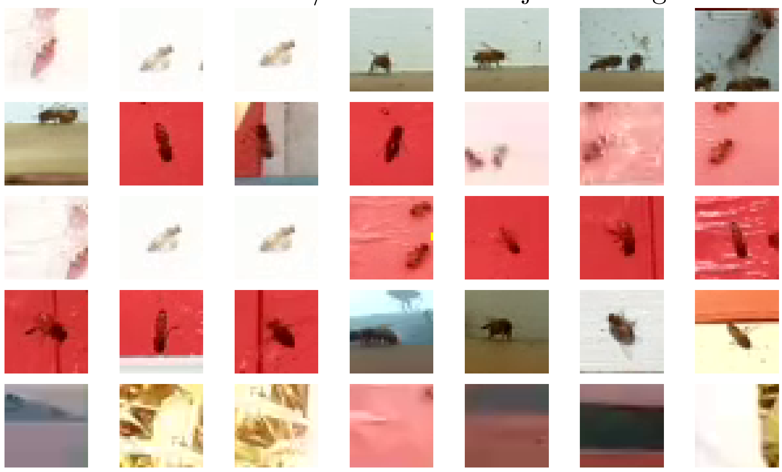
Table 1: CS5600/6600: F20: Project 1 Datasets

Dataset	Training	Testing	Validation
`BUZZ1`	7000	2110	1150
`BUZZ2`	7582	2332	3000
`BUZZ3`	9000	2746	2746
`BEE1`	38139	12724	3528
`BEE1_gray`	38139	12724	3528
`BEE2_1S`	35374	11863	10964
`BEE2_1S_gray`	35374	11863	10964
`BEE_4`	28965	9521	16192
`BEE_4_gray`	28965	9521	16192

There are 6 image datasets in the repo: `BEE1`, `BEE1_gray`, `BEE2_1S`, `BEE2_1S_gray`, `BEE_4`, `BEE_4_gray`. There are 3 audio datasets: `BUZZ1`, `BUZZ2`, `BUZZ3`. The suffix `_gray` in an image dataset's name means that the dataset is a grayscale version of the corresponding color dataset. For example, `BEE1_gray` is the grayscale version of `BEE1`. The 3 grayscale datasets (i.e., `BEE1_gray`, `BEE2_1S_gray`, `BEE_4_gray`) will be used in the training and testing of ANNs. The other 3 datasets (i.e., `BEE1`, `BEE2_1S`, `BEE_4`) will be used in the training and testing of ConvNets. The audio datasets can be used in the training and testing of both ANNs and ConvNets.

Each dataset consists of 6 pickle files: `train_X.pck`, `train_Y.pck`, `test_X.pck`, `test_Y.pck`, `valid_X.pck`, `valid_Y.pck`. The `_X` files include examples, the `_Y` files contain the corresponding targets (i.e., ground truth). The `train_` and `test_` files can be used in training and testing. The `valid_` files can be used in validation. Table 2 contains a sample of images out of which the numpy arrays in the pickle files have been created.

Table 2: CS5600/6600: F20: Project 1 Images



The project 1 zip contains 3 audio files (`buzz.wav`, `cricket.wav`, `noise.wav`) to give you examples of audio files out of which the numpy arrays of `BUZZ1`, `BUZZ2`, and `BUZZ3` were computed and normalized.

Loading Datasets

The zip archives contain the files `project1_audio_anns.py`, `project1_audio_cnns.py`, `project1_image_anns.py`, and `project1_image_cnns.py` that show you how to load the processed datasets, create ANNs and ConvNets with tflearn (you'll need to install this library for this project).

Let's run `project1_image_anns.py`. I skip the dimensions of each dataset in the output below.

```
>>> from project1_image_anns import *
loading datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BEE1_gray/...
...
datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BEE4_gray/ loaded...
```

Let's check the shape of the `BEE1_gray` training examples and targets.

```
>>> BEE1_gray_train_X.shape
(38139, 64, 64, 1)
>>> BEE1_gray_train_Y.shape
(38139, 2)
```

The above output shows that the training set consists of 38,139 64x64x1 numpy arrays. The last 1 means that the image has 1 channel (i.e., it is grayscale). The corresponding targets consists of 38,139 2-element numpy arrays. Let's print the first 10 targets.

```
>>> BEE1_gray_train_Y[:10]
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
```

```
        [0., 1.],
        [0., 1.],
        [1., 0.],
        [1., 0.],
        [0., 1.],
        [1., 0.]])
```

If a target is `[1., 0.]`, the corresponding example is classified as `BEE`. When a target is `[0., 1.]`, it means that the corresponding example is classified as `NO_BEE`. For example, since `BEE1_gray_train_Y[0]` is `[0., 1.]`, `BEE1_gray_train_X[0]` is classified as `NO_BEE`.

Let's play with the audio datasets and load `project1_audio_anns.py`. Again, I skip the dimensions of each dataset in the output below.

```
>>> from project1_audio_anns import *
loading datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ1/...
datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ1/ loaded...
loading datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ2/...
datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ2/ loaded...
loading datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ3/...
datasets from /home/vladimir/teaching/AI/F20/project_01/datasets/BUZZ3/ loaded...
```

Let's explore `BUZZ1`. The other two datasets (`BUZZ2` and `BUZZ3`) are similar.

```
>>> BUZZ1_train_X.shape
(7000, 4000, 1, 1)
>>> BUZZ1_train_Y.shape
(7000, 3)
>>> len(BUZZ1_train_X[0])
4000
>>> BUZZ1_train_Y[0]
array([1., 0., 0.])
>>> BUZZ1_train_Y[:10]
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

The above interaction shows that each audio example is 4000x1 numpy array. It's 4,000 amplitude readings. Each target is a 3-element numpy array (e.g., `[1., 0., 0.]`). This is a 3-way classification – `BEE`, `CRICKET`, and `NOISE`. When a target is `[1., 0., 0.]`, the corresponding example is classified as `BEE`; when a target is `[0., 1., 0.]`, the corresponding example is classified as `CRICKET`; and when a target is `[0., 0., 1.]`, the corresponding example is classified as `NOISE`. Some of you asked me which tools we used to process the audio files. We used `scipy.io.wavfile` and normalized the amplitudes to be between 0 and 1.

Training, Testing, and Validating ANN Image Models

The function `make_image_ann_model()` in `project1_image_anns.py` shows you an example of defining an ANN with `tflearn`. An ANN is just a set of fully connected layers. The network has an input layer that takes 64x64x1 inputs, the input layer is fully connected to `fc_layer_1` (the hidden layer) of 128 ReLU neurons. The hidden layer is connected to `fc_layer_2` (the output layer) of 2 softmax neurons. The call to the `regression` function defines the learning rate of the network, the stochastic gradient descent as the weight optimization function, and the categorical cross entropy as the loss function (i.e., the cost function).

```
def make_image_ann_model():
    input_layer = input_data(shape=[None, 64, 64, 1])
    fc_layer_1 = fully_connected(input_layer, 128,
                                 activation='relu',
                                 name='fc_layer_1')
    fc_layer_2 = fully_connected(fc_layer_1, 2,
                                 activation='softmax',
                                 name='fc_layer_2')
    network = regression(fc_layer_2, optimizer='sgd',
                         loss='categorical_crossentropy',
                         learning_rate=0.1)
    model = tflearn.DNN(network)
    return model
```

Here's how we can make an ANN model.

```
>>> ann_model = make_image_ann_model()
>>> ann_model
<tflearn.models.dnn.DNN object at 0x7faa39e90908>
```

Once we have a model, we can train it. The function `train_tfl_image_ann_model()` gives you an example of how to do it. Your output should look as follows.

```
>>> train_tfl_image_ann_model(ann_model,
                              BEE1_gray_train_X, BEE1_gray_train_Y,
                              BEE1_gray_test_X, BEE1_gray_test_Y)
---------------------------------
Run id: image_ann_model
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 38139
Validation samples: 12724
--
Training Step: 3814  | total loss: 0.66422 | time: 5.367s
| SGD | epoch: 001 | loss: 0.66422 - acc: 0.5912 | val_loss: 0.66742 -
                                                   val_acc: 0.5386 -- iter: 38139/38139
--
Training Step: 7628  | total loss: 0.65895 | time: 5.206s
| SGD | epoch: 002 | loss: 0.65895 - acc: 0.5784 | val_loss: 0.66352 -
                                                   val_acc: 0.5549 -- iter: 38139/38139
--
```

Our loss is pretty high and accuracy is low, but we've trained it only for 2 epochs. Once we have a trained model, we need to do 2 things with it: validate it on a dataset that was not used in training/testing and, if it's good enough, persist. The above model is not that great, but I'll perist it anyway to show you how to do id. Let's validate it first.

```
>>> validate_tfl_image_ann_model(ann_model, BEE1_gray_valid_X, BEE1_gray_valid_Y)
0.5702947845804989
```

Now we can persist it in an appropriate `tfl` file.

```
>>> ann_model.save('project_01/datasets/BEE1_gray/ann_model_bee1_gray.tfl')
```

Later on, if we want to use a persisted model or train it some more, we need to load it first and then use/train it.

```
>>> am = load_image_ann_model('project_01/datasets/BEE1_gray/ann_model_bee1_gray.tfl')
>>> test_tfl_image_ann_model(am, BEE2_1S_gray_valid_X, BEE2_1S_gray_valid_Y)
0.7147026632615834
```

Training, Testing, and Validating ConvNet Image Models

Let's go and train, test, and validate ConvNets on images. The function `make_image_cnn_model()` in `project1_image_cnns.py` shows you an example of defining an ANN with `tflearn`. This ConvNet contains a convolution layer followed by a max-pool layer and a fully connected hidden layer. The output layer has 2 output softmax neurons. Note that the shape of the input is 64x64x3, because these arrays were created from color (i.e., 3 channel) images.

```
def make_image_cnn_model():
    input_layer = input_data(shape=[None, 64, 64, 3])
    conv_layer_1 = conv_2d(input_layer,
                           nb_filter=8,
                           filter_size=3,
                           activation='relu',
                           name='conv_layer_1')
    pool_layer_1 = max_pool_2d(conv_layer_1, 2, name='pool_layer_1')
    fc_layer_1 = fully_connected(pool_layer_1, 128,
                                 activation='relu',
                                 name='fc_layer_1')
    fc_layer_2 = fully_connected(fc_layer_1, 2,
                                 activation='softmax',
                                 name='fc_layer_2')
    network = regression(fc_layer_2, optimizer='sgd',
                         loss='categorical_crossentropy',
                         learning_rate=0.1)
    model = tflearn.DNN(network)
    return model
```

Let's make a ConvNet model.

```
>>> cnn_model_bee1 = make_image_cnn_model()
>>> cnn_model_bee1
<tflearn.models.dnn.DNN object at 0x7f515df5c5f8>
```

Let's train this ConvNet. The function `train_tfl_image_cnn_model()` gives you an example of how to do it. Your output should look as follows.

```
>>> train_tfl_image_cnn_model(cnn_model_bee1, BEE1_train_X, BEE1_train_Y,
                              BEE1_test_X, BEE1_test_Y)
---------------------------------
Run id: image_cnn_model
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 38139
Validation samples: 12724
--
Training Step: 3814  | total loss: 0.13714 | time: 19.153s
| SGD | epoch: 001 | loss: 0.13714 - acc: 0.9503 | val_loss: 0.13552 -
                                                   val_acc: 0.9477 -- iter: 38139/38139
--
Training Step: 7628  | total loss: 0.13324 | time: 19.037s
| SGD | epoch: 002 | loss: 0.13324 - acc: 0.9591 | val_loss: 0.13543 -
                                                   val_acc: 0.9499 -- iter: 38139/38139
--
```

Our loss could be smaller and accuracy is 0.95. We need to train some more. Let's validate the model on a different dataset that was not used in training/testing and persist it.

```
>>> validate_tfl_image_cnn_model(cnn_model_bee1, BEE1_valid_X, BEE1_valid_Y)
0.8738662131519275
>>> validate_tfl_image_cnn_model(cnn_model_bee1, BEE2_1S_valid_X, BEE2_1S_valid_Y)
0.6159248449470996
>>> validate_tfl_image_cnn_model(cnn_model_bee1, BEE4_valid_X, BEE4_valid_Y)
0.5946763833992095
>>> cnn_model_bee1.save('AI/F20/project_01/datasets/BEE1/cnn_model_bee1.tfl')
```

We can now load the model into Python and use/train it. Here' how.

```
>>> cm = load_image_cnn_model('project_01/datasets/BEE1/cnn_model_bee1.tfl')
>>> validate_tfl_image_cnn_model(cm, BEE1_valid_X, BEE1_valid_Y)
0.8738662131519275
>>> validate_tfl_image_cnn_model(cm, BEE2_1S_valid_X, BEE2_1S_valid_Y)
0.6159248449470996
>>> validate_tfl_image_cnn_model(cm, BEE4_valid_X, BEE4_valid_Y)
0.5946763833992095
```

Training, Testing, and Validating ANN Audio Models

The file `project1_audio_anns.py` contains examples of how to construct and train, test, and validate ANNs on the 3 audio datasets, persist them, and load them into Python. The interaction below shows you how to do these steps. I won't comment the steps, because they're very similar to the steps discussed in the previous two sections.

```
>>> audio_model = make_audio_ann_model()
>>> train_tfl_audio_ann_model(audio_model, BUZZ1_train_X, BUZZ1_train_Y,
                              BUZZ1_test_X, BUZZ1_test_Y)
---------------------------------
Run id: audio_ann_model
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 7000
Validation samples: 2110
--
Training Step: 700  | total loss: 0.93983 | time: 1.877s
| SGD | epoch: 001 | loss: 0.93983 - acc: 0.5913 | val_loss: 0.97629 -
                                                   val_acc: 0.6526 -- iter: 7000/7000
--
Training Step: 1400  | total loss: 0.75596 | time: 1.890s
| SGD | epoch: 002 | loss: 0.75596 - acc: 0.6911 | val_loss: 0.91760 -
                                                   val_acc: 0.6991 -- iter: 7000/7000
--
>>> validate_tfl_audio_ann_model(audio_model, BUZZ1_valid_X, BUZZ1_valid_Y)
0.47043478260869565
>>> audio_model.save('project_01/datasets/BUZZ1/audio_model_buzz1.tfl')
>>> am = load_audio_ann_model('project_01/datasets/BUZZ1/audio_model_buzz1.tfl')
>>> validate_tfl_audio_ann_model(am, BUZZ1_valid_X, BUZZ1_valid_Y)
0.47043478260869565
>>> validate_tfl_audio_ann_model(am, BUZZ2_valid_X, BUZZ2_valid_Y)
0.519
>>> validate_tfl_audio_ann_model(am, BUZZ3_valid_X, BUZZ3_valid_Y)
0.19811858608893956
```

Training, Testing, and Validating ConvNet Audio Models

The file `project1_audio_cnns.py` contains examples of how to construct and train, test, and validate ConvNets on the 3 audio datasets, persist them, and load them into Python. The interaction below shows you how to do these steps.

```
>>> from project1_audio_cnns import *
>>> audio_model = make_audio_cnn_model()
>>> train_tfl_audio_cnn_model(audio_model, BUZZ1_train_X, BUZZ1_train_Y,
                              BUZZ1_test_X, BUZZ1_test_Y)
---------------------------------
Run id: audio_ann_model
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 7000
Validation samples: 2110
--
Training Step: 700  | total loss: 0.46220 | time: 5.340s
| SGD | epoch: 001 | loss: 0.46220 - acc: 0.8266 | val_loss: 0.55650 -
                                                   val_acc: 0.7986 -- iter: 7000/7000
--
Training Step: 1400  | total loss: 0.53039 | time: 5.370s
| SGD | epoch: 002 | loss: 0.53039 - acc: 0.8033 | val_loss: 0.52352 -
                                                   val_acc: 0.8038 -- iter: 7000/7000
```

```
--
>>> validate_tfl_audio_cnn_model(audio_model, BUZZ1_valid_X, BUZZ1_valid_Y)
0.4156521739130435
>>> audio_model.save('project_01/datasets/BUZZ1/audio_cnn_model_buzz1.tfl')
>>> cnn_model = load_audio_cnn_model('project_01/datasets/BUZZ1/audio_cnn_model_buzz1.tfl')
>>> validate_tfl_audio_cnn_model(cnn_model, BUZZ1_valid_X, BUZZ1_valid_Y)
0.4156521739130435
```

What You Need to Do

1. Train one ANN network for the following image datasets: `BEE1_gray`, `BEE2_1S_gray`, and `BEE_4_gray`. Persist your trained nets into `ann_image_model_bee1_gray.tfl`, `ann_image_model_bee2_1s_gray.tfl`, `ann_image_model_bee4_gray.tfl`, respectively.

2. Train one ConvNet for the following image datasets: `BEE1`, `BEE2_1S`, and `BEE_4`. Persist your trained nets into `cnn_image_model_bee1.tfl`, `cnn_image_model_bee2_1s.tfl`, `cnn_image_model_bee4.tfl`, respectively.

3. Train one ANN network for the 3 audio datasets: `BUZZ1`, `BUZZ2`, and `BUZZ3`. Persist your trained nets into `ann_audio_model_buzz1.tfl`, `ann_audio_model_buzz2.tfl`, `ann_audio_model_buzz3.tfl`, respectively.

4. Train one ConvNet network for the 3 audio datasets: `BUZZ1`, `BUZZ2`, and `BUZZ3`. Persist your trained nets into `cnn_audio_model_buzz1.tfl`, `cnn_audio_model_buzz2.tfl`, `cnn_audio_model_buzz3.tfl`, respectively.

5. Define the loader function for each of your trained nets in `load_nets.py`.

What to Submit

1. `load_nets.py` – this is the file where you need to define the loader functions for your nets; in the comments in these files, mention the performance of your best performing nets on the appropriate validation sets;
2. `ann_image_model_bee1_gray.tfl` – your best ANN for `BEE1_gray`;
3. `ann_image_model_bee2_1s_gray.tfl` – your best ANN for `BEE2_1S_gray`;
4. `ann_image_model_bee4_gray.tfl` – your best ANN for `BEE4_gray`;
5. `ann_audio_model_buzz1.tfl` – your best ANN for `BUZZ1`;
6. `ann_audio_model_buzz2.tfl` – your best ANN for `BUZZ2`;
7. `ann_audio_model_buzz3.tfl` – your ANN for `BUZZ3`;
8. `cnn_image_model_bee1.tfl` – your best ConvNet for `BEE1`;
9. `cnn_image_model_bee2_1s.tfl` – your best ConvNet for `BEE2_1S`;
10. `cnn_image_model_bee4.tfl` – your best ConvNet for `BEE4`;
11. `cnn_audio_model_buzz1.tfl` – your best ConvNet for `BUZZ1`;
12. `cnn_audio_model_buzz2.tfl` – your best ConvNet for `BUZZ2`;
13. `cnn_audio_model_buzz3.tfl` – your best ConvNet for `BUZZ3`.

How We'll Test Your Nets

Your nets will be tested on similar audio and image datasets that are not part of the datasets on which you'll be training, testing, and validating your nets. You'll receive the performance report of your nets each dataset on which we'll test them.

Happy Hacking!