PROJECT OF JESSICA ROY

Assignment 6

This is the final report on your implementation. Paste Assignments 1 - 5 at the head of this *as is*, except that in Assignment 5, respond to each of your facilitator’s comments within each of their comments. Again, retain these gray parts. Keep in mind the evaluation criteria (at the end) and that they are somewhat altered for this assignment.

For voluminous material, reference appendices (at the end). These will be read on an as-needed basis.

Excluding appendices and figures, this response should not exceed 6 pages—without prior permission

# ASSIGNMENT 1: PROJECT PROPOSAL DRAFT

Please use this template. Retain the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references and figures. Note the evaluation criteria, and leave plenty of time for editing that best responds. We recognize that you may alter your plans as the term progresses. That is to be expected—your changes or substitution will fit with this growing document.

## 1.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project.

The Caltech-UCSD Birds-200-2011 Dataset [8] has 11,788 annotated, labeled images of 200 species of birds. Each image has 15 parts (beak, tail, etc.), a bounding box around the bird, and 312 binary attributes, plus the speed and certainty with which the attribute was given (“Has\_Belly\_Color: Yellow (definitely, 4.4980sec)”). The annotations were made by multiple Mechanical Turk users per picture.

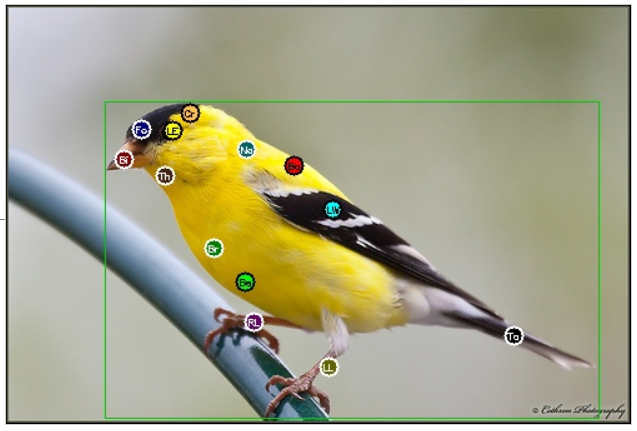


Image from <http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/American_Goldfinch.html>

I plan to use the attributes to predict the species of bird. Although this plan omits some potential data (such as the posture or environment of the bird), it does have an advantage over inspecting the image directly, as Chai, Lempitsky and Zisserman point out [1]: “With generic classification approaches these fine differences often get “swamped” by the bulk of the image… Once the discriminative parts are localized, they are encoded into separate parts of the visual signature, enabling the classifier to pick up on the fine differences in those parts.” My understanding of this: once the beak, tail, etc. have been identified, the classifier can more easily compare bird beaks without getting results muddled by background shapes near the beak. We’re also pointing out to the classifier that bird leg color is important, but the background color near the bird is not.

## 1.2 I/O EXAMPLES

At least two specific examples of projected output for designated input. You will not be held to this—it is just explanatory at this point.

The attribute data is provided as: <image\_id> <attribute\_id> <is\_present> <certainty\_id> <time>

For example: 2702 8 1 4 22.2660

2702 is the image shown above, an American Goldfinch, labeled as class 47. The 8 means this line represents the eighth of the 312 attributes. Attributes 1-9 all have to do with bill shape, and 8 is has\_bill\_shape::cone. Next, the 1 indicates this attribute was selected. The 4 is a certainty rating of “definitely”, and it took a little more than 22 seconds for the decision to be made.

My designated input would then be all 312 of the values for the American Goldfinch, and my expected output would be 47, the number representing that species. I would also like my algorithm to give a probability rating for the classification. I don’t yet know what that might look like, but I imagine it might be something like 0.89 if it is 89% likely to be an American Goldfinch.

Sparrows might be more difficult, as many look alike, and there are 21 species in the set (classes 113-133). For example, input would be the 312 values for the Grasshopper Sparrow, and the expected output would be 121. But I won’t be too surprised if the top probabilities for the classifications in the 113-133 range are lower, such as 68% Grasshopper Sparrow.

For full examples of the input values for these two birds, see:

<http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/American_Goldfinch.html>

http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/Grasshopper\_Sparrow.html

## 1.3 REQUIREMENTS

High-level requirements statement in 3 roughly equal numbered lists, organized by triage. Separate your requirements into thee approximately even categories using triage (select the two extreme categories—definite and nice-to-do—and then place the remainder in the middle category). State requirements in declarative language such as “The application will recognize numbers 0-9 from a 12 by 35 array of black-or-white pixels” (not “First I will build a neural net”).

### 1.3.1 Definite Requirements (first priority)

a. **Correctness: The application correctly classifies sets of attributes into 200 classes, with as few incorrect classifications as possible.** I don’t know if 100% accuracy is possible with this dataset – perhaps some birds simply can’t clearly be distinguished based on the data provided. But I would want to see if I can fine tune the algorithm in some way to improve the accuracy.

b. **Reasonable errors: all incorrect classifications are between birds with similar attributes** (that is, it might mix up two similar sparrows, but it should not mistakenly identify a goldfinch as a sparrow). I’m not sure this is quantifiable yet.

c. **Missing data is handled appropriately.** I should account somehow for any attributes where the certainty rating is “not visible” because this is not a valid feature of a bird. Just because a Blue Jay’s tail isn’t visible in one photo does not mean that it has something in common with a Herring Gull with its tail not visible in a different photo.

### 1.3.2 Not sure yet (second priority)

d. **Probability:** The algorithm provides a **probability that its classification is correct**.

e. **Certainty/Timing:** A second version of the algorithm attempts to **take the certainty or timing of the rating into account** somehow when determining the weight of a given feature. Perhaps “definitely” with a short time gets more weight than “guessing” with a longer time.

### 1.3.3 Nice-to-do (can dispense with if time does not allow; third priority)

f. **Explanations:** Perhaps the algorithm can be made to reveal **why it made the decisions it made,** in some way. For example, maybe it can explain what attributes contribute most to the decision of “47. American Goldfinch.” This may be more challenging than I think, as neural nets aren’t known for being able to explain themselves. “But neural nets are black boxes. After training, a network may be very good at classifying data, but even its creators will have no idea why.” [5]. On the other hand, I am not working with extremely complex data here such as images or sound files. I’m working with 312 binary attributes. Could I somehow get the algorithm to reveal which attributes wound up having the greatest influence in each final decision?

g. **Additional data:** Visual data isn’t the only key to good bird identification – if the pictures had somehow also been tagged with the location, habitat, sound, or behavior of the bird, I would expect that the identifications would be far more accurate. So, if possible: **Match this data set up with another data set, and see if that improves assessments.** See “Data Sources” below.

## 1.4 HOW SUCCESS WILL BE ASSESSED

Explain, as specifically as possible (quantification is ideal) how success of the project should be assessed.

For the criteria above:

1. **Correctness:** Comparing the predicted class value for each photograph with the actual class value for each photograph, for the testing set.
2. **Reasonable errors:** Review the (hopefully few) errors and confirm that they do in fact share many attributes in common.
3. **Missing data:** A comparison of the algorithm with and without accounting for the “not visible” rated attributes reveals whether or not taking these attributes into account is helpful.
4. **Probability:** A numeric value is output expressing the confidence in the algorithm’s categorization.
5. **Certainty/Timing:** Certainty and/or timing data is included in the processing in some way.
6. **Explanations:** The algorithm provides the top 3-5 attributes that were most important in making a given classification.
7. **Additional data:** Another data set is used to attempt to improve accuracy.

## 1.5 TECHNOLOGY EXPLANATION

Explain what two technologies you are seriously considering--and why you feel they apply. One may be emphasized as the implementation and the other as an alternative. If possible, show fragments of experimentation with these. An example is [here](https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.88592&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&co).

I am looking most closely at the **Multi-Layer Perceptron type of neural network**. This is a classification problem, but one with more than two categories – it is not a yes/no zero/one binary question – so the simple linear classification Perceptron won’t work here. We have labeled data we can use for training purposes. We have 312 inputs, and 200 possible categories as outputs. I might start with 10-15 markedly different bird species, then later use the entire dataset. At the moment, although I suspect it is an oversimplification, I’m thinking of the MLP as sort of 200 different Perceptrons combined: one that tells us 001.Black\_footed\_Albatross or not, one that tells us 002.Laysan\_Albatross or not, and so on. I’m wondering if the first output would be a list of 200 values: a 1 for the matching bird and a 0 for everything else. Later, perhaps a percentage output might be 0.89 for bird A, 0.09 for bird B, and 0.01 for bird C, and a 0 for everything else. Output could also be a sparse representation [4] of the data.

For training, Marsland [7] recommends a split for training, verification, and testing: “50:25:25 if you have plenty of data, and 60:20:20 if you don’t” (p. 21). However, the data set also comes with a file called train\_test\_split.txt which is worth investigating, because the data set shouldn’t be split up randomly. Each species should be represented equally in the training group; the application won’t be able to identify a species for which it wasn’t trained. Marsland [7] explains that “you should generally make sure that you have approximately the same number of each class in your training set” (p. 93).

A perhaps simpler algorithm would be a **decision tree**. This is how I think about bird classification in the field while birdwatching – I might notice the size, then shape, then behavior, then overall color, then wing markings… Here I might take advantage of the fact that the 312 attributes are divided into 28 categories such as size, bill shape, belly color. One problem, however: the categories can have more than one value. Here’s data for the same Goldfinch photo again, correctly indicating that the bird has a black and white tail:

2702 91 1 2 4.4240  
2702 92 1 2 4.4240

91 has\_upper\_tail\_color::black  
92 has\_upper\_tail\_color::white

I’m not sure yet if a decision tree can be made to handle when multiple options are selected.

## 1.6 DATA SOURCES

Explain whether or not your project requires data. If so, describe were you will obtain it.

The Caltech-UCSD Birds-200-2011 Dataset [8] is freely available. I have already downloaded it and started reviewing it.

The eBird Reference Dataset [2] is also freely available, and I have downloaded a sample. The full data set must be requested, but requests are granted for non-commercial purposes. It has a value called FRAGSTATS that may be usable as an approximation of habitat. (e.g., “Mixed Forest” or “Grasslands”). I will need to review this dataset to see if it is in sufficient use or not. If it is, I would attempt to match each photo with a random FRAGSTATS value for an eBird observation of the same species.

If FRAGSTATS is not sufficient, I might try something based on latitude and longitude. However, this itself might be a machine learning task to determine where to draw the boundaries!

Migration data would be another interesting choice – extensive work has already been done [6] to extract migration patterns from eBird data. Unfortunately, although this seems to have made a fascinating map [3], it doesn’t appear that the data is readily available online.

## 1.8 REFERENCES FOR PROPOSAL PHASE

Fill in and cite each of the following (e.g., “[2]“) within the text. References can include specific places in the notes and textbook.

[1] Chai, Y., Lempitsky, V., Zisserman, A. (2013). “Symbiotic Segmentation and Part Localization for Fine-Grained Categorization” , *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia. Retrieved from <http://www.robots.ox.ac.uk/~vgg/publications/2013/Chai13/chai13.pdf.pdf>

[2] Cornell Lab of Ornithology, eBird Data Access. Retrieved from <https://ebird.org/data/download>

[3] Cornell Lab of Ornithology (2016, 20 Jan). Mesmerizing migration map: Which species is which? Retrieved from <https://www.allaboutbirds.org/mesmerizing-migration-map-which-species-is-which/>

[4] Google, Machine learning glossary: sparse representation. Retrieved from <https://developers.google.com/machine-learning/glossary/#sparse_representation>

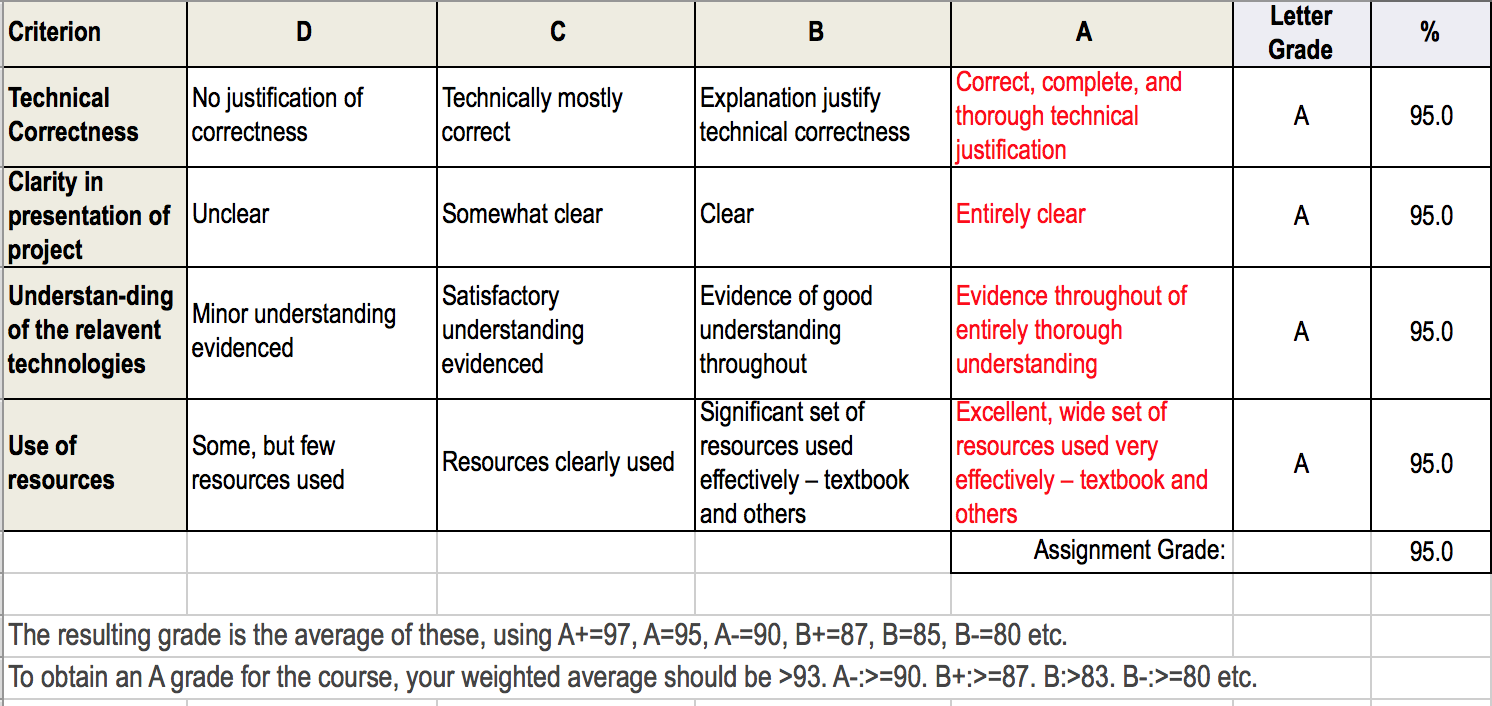
[5] Hardesty, L. (2016). “Making computers explain themselves.” *MIT News.* Retrieved from <http://news.mit.edu/2016/making-computers-explain-themselves-machine-learning-1028>

[6] LaSorte, F.A., Fink, D., Hochachka, W.M., Kellig, S. (2016, 20 Jan). *Convergence of broad-scale migration strategies in terrestrial birds.* Proceedings of the Royal Society B. Retrieved from <http://rspb.royalsocietypublishing.org/content/283/1823/20152588>

[7] Marsland, S. (2015). Machine learning: An algorithmic perspective. CRC Press (Taylor & Francis Group), Boca Raton, FL.

[8] Wah C., Branson S., Welinder P., Perona P., Belongie S. (2011). “The Caltech-UCSD Birds-200-2011 Dataset.” *Computation & Neural Systems Technical Report*, CNS-TR-2011-001. Retrieved from <http://www.vision.caltech.edu/visipedia/papers/CUB_200_2011.pdf>

## 1.7 Instructor’s Evaluation of Assignment 1



Responses to comments

I was not able to download the comments when I downloaded the Word document. Professor Braude advised me to add responses to the comments in a separate section between assignment 1 and assignment 2.

**(page 1) “Great description and write up!”**

Thanks! Since then I have been watching a number of videos online about artistic uses for machine learning, and I’m beginning to regret not having chosen something with a little more creative output than mere classification. I’ve been giving that some thought, and I have an idea, which I will propose below.

**(page 2) “Multi-class classification problems might be quite tricky as you have 200 classes and the distribution of classes in the training set is crucial. Say you have 1 species that dominates the training sample where as another species might have super small number of observations in there. Then your ML model might be biased towards the species with the most data. We just need to be careful about it :D”**

Indeed. Fortunately, this data set is fairly balanced across the species, with approximately 60 photos per species. It should be fairly straightforward to get an even distribution across the training set.

**(page 3) “great point!” (re: “Just because a Blue Jay’s tail isn’t visible in one photo does not mean that it has something in common with a Herring Gull with its tail not visible in a different photo.”)**

Thanks. I wonder what other similar factors I haven’t thought of yet…

**(page 3) “one thing you can look at is ‘partial dependence plot’ to see which features are more important than the others, and try to relate that with its biological explanation. Partial dependence plots are useful when black box type classifiers are used.”**

Thanks, that looks like exactly the sort of thing I’m looking for here. I found this article:

<https://arxiv.org/pdf/1309.6392.pdf>

Sometimes the most useful thing for me about an article like this is the review of the existing research often found at the beginning. Here, they offer a survey of various ways of visualizing how features influence a prediction, including PDP (which they are extending).

**(page 3) (re: Correctness) “Accuracy is a great measure to look at. You can also look at what is called ROC curve, and area under the ROC curve.”**

I’ve been reading about those measures on Google’s Machine Learning Crash Course: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

It’s fairly easy for me to conceive of how those work for a linear regression where you can make those four groupings: true positives, false positives, true negatives, false negatives. But what happens when you have 200 categories? Does it get evaluated 200 times? This will need more research.

**(page 4) You can also look at Random Forest. Which is essentially building multiple decision trees and combine the corresponding classifiers.**

This page says a lot of strongly-worded positive things about Random Forests… then again, it appears to be a page written by one or more of the people for whom Random Forests is a trademark, Leo Breiman and Adele Cutler:

<https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#overview>

It says things like “It is unexcelled in accuracy among current algorithms. It runs efficiently on large data bases. It can handle thousands of input variables without variable deletion. It gives estimates of what variables are important in the classification…” I’ll give this a try!

# ASSIGNMENT 2: PROJECT PROPOSAL PLUS

Please use this template. Retain the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 additional pages excluding references and figures. Note the evaluation criteria, and leave plenty of time for editing. There are two related aspects to your term project: it should have function and it should be an opportunity to learn. You can refer to both in writing this up.

## 2.0 WHAT’S CHANGED

Provide no more than a page of 12-point type explaining what has been changed or added since assignment 1. Include in this whether and how the material in module 2 influenced this. If it did not, refer to reading that you did in working on this assignment (#2).

2.2 I/O Examples: Included code for loading the data set as a Pandas data frame and manipulating it to form a table where each row is an image and the columns are the 312 attribute values for that image, each 0 or 1.

2.3 Requirements:

* Incorporated my facilitator’s suggestion of considering using ROC and area under ROC as accuracy measurements.
* Hesitantly moved “Explanations” to first priority. I’m more confident that there’s a way forward with this. I’m still hesitating because I’m not sure whether what I already had under requirements was already enough to fill the next few weeks
* “Certainty/Timing” is now second priority as it is less interesting to me than other goals.
* Removed: stretch goal of combining data sets. In its place, I have included a stretch goal that uses the images. This will give me a chance to take on an image processing challenge, as well as possibly looking into an unsupervised learning algorithm. It also allows me to focus more on trying out the tools and algorithms rather than hunting for a compatible second dataset and trying to get the two connected. It’s also fun.

2.4 How success will be assessed: Updated to match the new requirements.

2.5 Technology explanation: Updated to use as Random Forests as a second algorithm choice.

2.6 Data Sources: Removed possibilities for second data source.

2.7 References: Added new and removed unused. Also, Word now manages my reference list so it is now in Word’s best guess at IEEE format, numbered by appearance, not alphabetically.

I read module 2 and the textbook chapters on perceptrons and MLP prior to assignment 1.This week, I’ve been getting up to speed on NumPy, Pandas, matplotlib, and a little TensorFlow. I enrolled in this class last minute, so I’m working to catch up to my classmates who have had weeks to prepare.

The main influences this week have therefore been:

* The first 20 lectures of <https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/> and the first 40 lectures (no exercises) of <https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/>
* A number of machine learning + art websites and videos, most notably this one: “A visual overview examining the ability of neural networks to create abstract representations from collections of real world objects.” <https://medium.com/artists-and-machine-intelligence/perception-engines-8a46bc598d57>

## 2.1 V2 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project.

The Caltech-UCSD Birds-200-2011 Dataset [1] has 11,788 annotated, labeled images of 200 species of birds. Each image has 15 parts (beak, tail, etc.), a bounding box around the bird, and 312 binary attributes, plus the speed and certainty with which the attribute was given (“Has\_Belly\_Color: Yellow (definitely, 4.4980sec)”). The annotations were made by multiple Mechanical Turk users per picture.

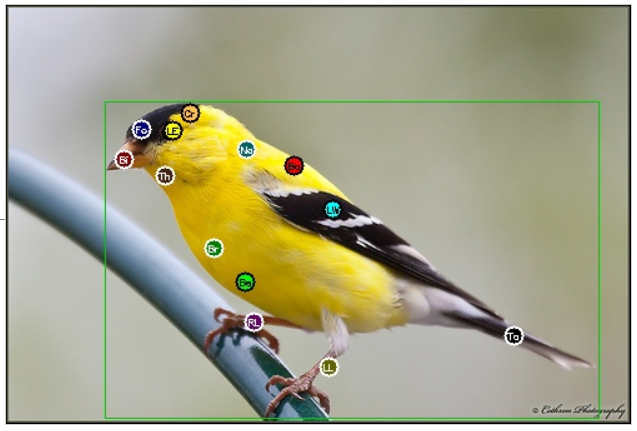


Image from <http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/American_Goldfinch.html>

I plan to use the attributes to predict the species of bird. Although this plan omits some potential data (such as the posture or environment of the bird), it does have an advantage over inspecting the image directly: “With generic classification approaches these fine differences often get “swamped” by the bulk of the image… Once the discriminative parts are localized, they are encoded into separate parts of the visual signature, enabling the classifier to pick up on the fine differences in those parts.” [2] My understanding of this: once the beak, tail, etc. have been identified, the classifier can more easily compare bird beaks without getting results muddled by background shapes near the beak. We’re also pointing out to the classifier that bird leg color is important, but the background color near the bird is not.

## 2.2 V2 I/O EXAMPLES

At least two specific examples of projected output for designated input. You will not be held to this—it is just explanatory at this point.

The attribute data is provided as: <image\_id> <attribute\_id> <is\_present> <certainty\_id> <time>

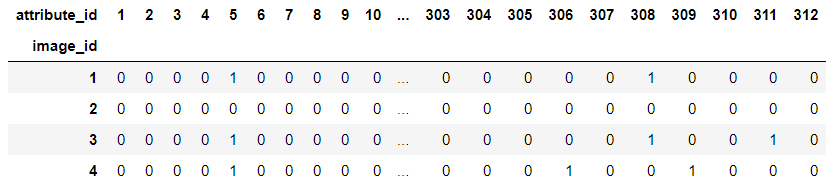
For example: 2702 8 1 4 22.2660

2702 is the image shown above, an American Goldfinch, labeled as class 47. The 8 means this line represents the eighth of the 312 attributes. Attributes 1-9 all have to do with bill shape, and 8 is has\_bill\_shape::cone. Next, the 1 indicates this attribute was selected. The 4 is a certainty rating of “definitely”, and it took a little more than 22 seconds for the decision to be made.

My designated input would then be all 312 of the values for the American Goldfinch, and my expected output would be 47, the number representing that species. Here is some code I’m playing with, using Pandas to read in the data file with the attributes and group it into one row per image\_id:

cols='image\_id attribute\_id is\_present certainty\_id time'.split()  
birds = pd.read\_csv("leaftest4/image\_attribute\_labels\_1.txt",' ',names=cols)  
by\_image = birds.pivot(index='image\_id',columns='attribute\_id',values='is\_present')

The resulting data frame looks like this:



Sparrows might be more difficult, as many look alike, and there are 21 species in the set. For example, input would be the 312 values for the Grasshopper Sparrow, and the expected output would be 121.

For full examples of the input values for these two birds, see:

<http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/American_Goldfinch.html>

http://www.vision.caltech.edu/visipedia-data/CUB-200-2011/browse/Grasshopper\_Sparrow.html

## 2.3 V2 REQUIREMENTS

High-level requirements statement in 3 roughly equal numbered lists, organized by triage. Separate your requirements into thee approximately even categories using triage (select the two extreme categories—definite and nice-to-do—and then place the remainder in the middle category). State requirements in declarative language such as “The application will recognize numbers 0-9 from a 12 by 35 array of black-or-white pixels” (not “First I will build a neural net”).

### 2.3.1 Definite Requirements (first priority)

a. **Correctness: The application correctly classifies sets of attributes into 200 classes, with as few incorrect classifications as possible.** 100% accuracy may not be possible with this dataset – perhaps some birds simply can’t clearly be distinguished based on the data provided. But I will attempt to fine tune the algorithm to improve the accuracy. At my facilitator’s suggestion, I will consider ROC and area under ROC as measurements of accuracy. I found a resource here: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

I see how those work for a linear regression where you can make those four groupings: true positives, false positives, true negatives, false negatives. But what happens when you have 200 categories? Does it get evaluated 200 times? This will need more research.

b. **Reasonable errors: all incorrect classifications are between birds with similar attributes** (that is, it might mix up two similar sparrows, but it should not mistakenly identify a goldfinch as a sparrow). I’m not sure this is quantifiable yet.

c. **Missing data is handled appropriately.** I should account somehow for any attributes where the certainty rating is “not visible” because this is not a valid feature of a bird. Just because a Blue Jay’s tail isn’t visible in one photo does not mean that it has something in common with a Herring Gull with its tail not visible in a different photo.

d. **Explanations:** Perhaps the algorithm can be made to reveal **why it made the decisions it made,** in some way. For example, maybe it can explain what attributes contribute most to the decision of “47. American Goldfinch.” Neural nets have a reputation as “…black boxes. After training, a network may be very good at classifying data, but even its creators will have no idea why.” [3]. But at my facilitator’s recommendation, I began reviewing Partial Dependence Plots and a number of other ways of visualizing how features influence a prediction. [4]

2.3.2 Nice-to-do (second priority)

e. **Certainty/Timing:** A second version of the algorithm attempts to **take the certainty or timing of the rating into account** somehow when determining the weight of a given feature. Perhaps “definitely” with a short time gets more weight than “guessing” with a longer time.

f. **Bird Art:** If time allows, I would like to experiment with using machine learning algorithms to generate art from the images of the birds. I envision this as an unsupervised learning task, where the algorithm is provided all of the images of each species of bird, and attempts to identify what the images for a given species have in common. What makes an American Goldfinch look like an American Goldfinch?

Each image would be restricted to the bounding box described above so that the algorithm is focusing primarily on the bird and not the background. The data points indicating the parts of the bird might be further useful for orienting the image – for example, all photos with only the right wing showing could be flipped to make it appear as though we were looking at the left side of the bird. We also know where the bird’s eye is (although there may be 0 or 2 eyes visible in the photo) so I could perhaps align the images around the eye in some way.

## 2.4 V2: HOW SUCCESS WILL BE ASSESSED

Explain, as specifically as possible (quantification is ideal) how success of the project should be assessed.

For the criteria above:

1. **Correctness:** Comparing the predicted class value for each photograph with the actual class value for each photograph, for the testing set.
2. **Reasonable errors:** Review the (hopefully few) errors and confirm that they do in fact share many attributes in common.
3. **Missing data:** A comparison of the algorithm with and without accounting for the “not visible” rated attributes reveals whether or not taking these attributes into account is helpful.
4. **Explanations:** The algorithm provides the top 3-5 attributes that were most important in making a given classification.
5. **Certainty/Timing:** Certainty and/or timing data is included in the processing in some way.
6. **Bird Art:** Given a species in the dataset, the algorithm provides a visual representation combining the image properties for that species. I expect that at least my initial attempts will just be a blur of color. I’m hoping that I can refine the algorithm so that it makes something that seems representative – in an abstract sort of way – of the particular species of bird. Perhaps, for a more concrete criterion: given the results for two different species of bird, someone who has seen the photos (or otherwise is familiar with the two species) could tell which species was represented by each resulting image.

## 2.5 V2 TECHNOLOGY EXPLANATION

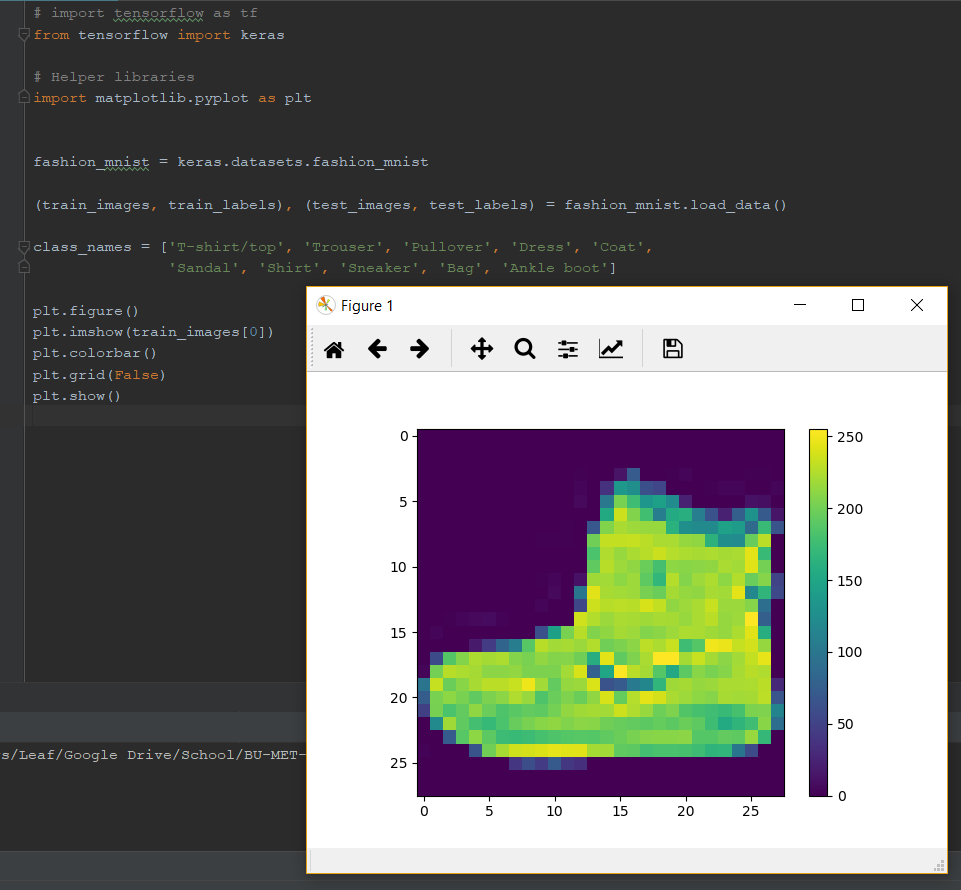
Explain what two technologies you intend to use--and why you feel they apply to your particular project. One of the two may be emphasized as the implementation and the other as an alternative or as a complement. If possible, show fragments of code execution with these. For example, if you are using TensorFlow, show that you have run some code. This can be simple.

I am looking most closely at the **Multi-Layer Perceptron type of neural network**. This is a classification problem, but one with more than two categories – it is not a yes/no zero/one binary question – so the simple linear classification Perceptron won’t work here. We have labeled data we can use for training purposes. We have 312 inputs, and 200 possible categories as outputs. I might start with 10-15 markedly different bird species, then later use the entire dataset. I’m thinking of the MLP as sort of 200 different Perceptrons combined: one that tells us 001.Black\_footed\_Albatross or not, one that tells us 002.Laysan\_Albatross or not, and so on. The first output might be a list of 200 values: a 1 for the matching bird and a 0 for everything else.

For training, Marsland [5] recommends a split for training, verification, and testing: “50:25:25 if you have plenty of data, and 60:20:20 if you don’t” (p. 21). However, the data set also comes with a file called train\_test\_split.txt which is worth investigating, because the data set shouldn’t be split up randomly. Each species should be represented equally in the training group; the application won’t be able to identify a species for which it wasn’t trained. Marsland [5] explains that “you should generally make sure that you have approximately the same number of each class in your training set” (p. 93).

I am also going to consider **Random Forest**. This appears to be a useful algorithm for large classification jobs. Breiman and Cutler, for whom “Random Forest” is a trademark, mention the ability for the algorithm to provide information about how the classifications are made: “It is unexcelled in accuracy among current algorithms. It runs efficiently on large data bases. It can handle thousands of input variables without variable deletion. *It gives estimates of what variables are important in the classification*…” [6]

And the following isn’t my code – it’s from the TensorFlow beginner’s tutorial on tensorflow.org [7] – but I did finally, after hours of frustration and confusion, get this running on my computer, in both Jupyter Notebook and PyCharm (shown below):



## 2.6 V2 DATA SOURCES

Explain whether or not your project requires data. If so, describe were you will obtain it.

The Caltech-UCSD Birds-200-2011 Dataset [1] is freely available. I have already downloaded it and started reviewing it. It contains not only all the attributes mentioned, but also all the photos, and all the data for identifying the parts of the bird in the photos.

## 2.8 REFERENCES FOR PROPOSAL V2

Fill in and cite each of the following (e.g., “[2]“) within the text. References can include specific places in the notes and textbook. You are free to include references from the prior assignment version.

[1] C. Wah, S. Branson, P. Welinder, P. Perona and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Computation & Neural Systems Technical Report, CNS-TR-2011-001., 2011. [Online]. Available: <http://www.vision.caltech.edu/visipedia/papers/CUB_200_2011.pdf>.

[2] Y. Chai, V. Lempitsky and A. Zisserman, "Symbiotic Segmentation and Part Localization for Fine-Grained Categorization," in *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, 2013.

[3] L. Hardesty, "Making computers explain themselves," MIT News, 2016. [Online]. Available: <http://news.mit.edu/2016/making-computers-explain-themselves-machine-learning-1028>.

[4] Goldstein, A. Kapelner, J. Bleich and E. Pitkin, "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation," The Wharton School of the University of Pennsylvania, 21 03 2014. [Online]. Available: <https://arxiv.org/pdf/1309.6392.pdf>.

[5] S. Marsland, Machine learning: An algorithmic perspective, Boca Raton, FL: CRC Press (Taylor & Francis Group), 2015.

[6] Breiman, L and A. Cutler, "Random Forests," [Online]. Available: <https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#overview>.

[7] TensorFlow, "Train your first neural network: basic classification," [Online]. Available: <https://www.tensorflow.org/tutorials/keras/basic_classification>.

[8] Google, "Machine learning crash course," [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

## 2.7 Instructor’s Evaluation of Assignment 2



# Assignment 3: Project Design, version 1

Keep in mind the evaluation matrix at the end as you do the work and use it to guide what you submit. Use no more than 6 pages of 12-point text excluding figures. You may include as many appendices as you wish for reference. Parts of these may be read as needed.

## 3.1 Final Requirements

List your final requirements, numbering them in the form DiX and NiX where:

D/N means “Definite” / “Nice to do” (two categories, not three)

i = 1, 2, 3, …

X=L and the goal is a *learning* goal – or – X=F and the goal is *functional*

You will reference these numbered requirements in the rest of the term, when you will be asked to show what the project accomplished.

**D1F. Correctness: The application correctly classifies sets of attributes into 200 classes, with as few incorrect classifications as possible.** 100% accuracy may not be possible with this dataset – perhaps some birds simply can’t clearly be distinguished based on the data provided. But I will attempt to fine tune the algorithm to improve the accuracy.

At my facilitator’s suggestion, I will consider the Receiver Operator Characteristic (ROC) curve and area under the ROC curve as measurements of accuracy. I found a resource here: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>. It is easy for me to see how those work for a linear regression with four groupings: true positives, false positives, true negatives, false negatives. Because I have 200 categories instead of 2, I will need to take a “one vs. rest” and/or “one vs. one” approach for calculating the ROC curve or the area under the ROC curve.

Precision and recall are also possibilities. Precision is the number of true positives divided by the number of *all* positives (both true and false). Recall is the number of true positives divided by the number of results that *should* have been positive (both true positives and false negatives). Marsland explains that precision and recall “are to some extent inversely related, in that if the number of false positives increases…then the number of false negatives often decreases, and vice versa. They can be combined to give a single measure, the F1 measure” (Marsland, 2015).

F1 = 2 \* ((precision \* recall) / (precision + recall))

= true positives / (true positives + (false negatives + false positives)/2)

**D2F.** **Explanations:** Perhaps the algorithm can be made to reveal **why it made the decisions it made,** in some way. For example, maybe it can explain what attributes contribute most to the decision of *47. American Goldfinch.* Neural nets have a reputation as “…black boxes. After training, a network may be very good at classifying data, but even its creators will have no idea why.” (Hardesty, 2016).

I began reviewing Partial Dependence Plots and a number of other ways of visualizing how features influence a prediction. “PDP plots the change in the average predicted value as specified features vary over their marginal distribution” (Goldstein, Kapelner, Bleich, & Pitkin, 2014). While this sounds interesting, I don’t really have time to delve into how to determine such a thing on my own in some way. Fortunately, it looks like scikit learn might have some means of doing a partial dependence plot. (scikit learn developers, 2017)

**D3F. Reasonable errors: all incorrect classifications are between birds with similar attributes** (that is, it might mix up two similar sparrows, but it should not mistakenly identify a goldfinch as a sparrow). I am hoping there will be few enough incorrect responses that I can just review them by hand to evaluate this. This may turn out to be a sub-goal of D2F, in that I might be evaluating the explanations provided to see if the characteristics used in making the classification were similar.

**D4F. “Not Visible” attributes are handled appropriately.** I should account somehow for any attributes where the certainty rating is “not visible” because this is not a valid feature of a bird. Just because a Blue Jay’s tail isn’t visible in one photo does not mean that it has something in common with a Herring Gull with its tail not visible in a different photo. I could try giving these attributes a weight of 0 to see if that improves the algorithm.

**N5F.** **Certainty/Timing:** A second version of the algorithm attempts to **take the certainty or timing of the rating into account** somehow when determining the weight of a given feature. Perhaps “definitely” with a short time gets more weight than “guessing” with a longer time.

**N6F.** **Bird Art:** If time allows, I would like to experiment with using machine learning algorithms to generate art from the images of the birds. I initially envisioned this as an unsupervised learning task, where the algorithm is provided all of the images of each species of bird, and attempts to identify what the images for a given species have in common. What makes an American Goldfinch look like an American Goldfinch?

Each image would be restricted to the bounding box described above to focus primarily on the bird and not the background. The data points indicating the parts of the bird might be further useful for orienting the image – for example, all photos with only the right wing showing could be flipped to make it appear as though we were looking at the left side of the bird. We also know where the bird’s eye is (although there may be 0 or 2 eyes visible in the photo) so I could perhaps align the images around the eye in some way.

**N7L. Learn about Generative Adversarial Networks.** In reading more about the possibilities for art, I found the article *Image Completion with Deep Learning in TensorFlow* (Amos, 2016). This article in turn quotes a Quora post from Yann LeCun:

“The idea is to simultaneously train two neural nets. The first one, called the Discriminator — let’s denote it D(Y) — takes an input (e.g. an image) and outputs a scalar that indicates whether the image Y looks “natural” or not. … The second network is called the generator, denoted G(Z), where Z is generally a vector randomly sampled in a simple distribution (e.g. Gaussian). The role of the generator is to produce images so as to train the D(Y) function to take the right shape (low values for real images, higher values for everything else). During training D is shown a real image, and adjusts its parameter to make its output lower. Then D is shown an image produced from G and adjusts its parameters to make its output D(G(Z)) larger (following the gradient of some objective predefined function). But G(Z) will train itself to produce images so as to fool D into thinking they are real. It does this by getting the gradient of D with respect to Y for each sample it produces. In other words, it’s trying to minimize the output of D while D is trying to maximize it. Hence the name adversarial training.” (LeCun, 2016)

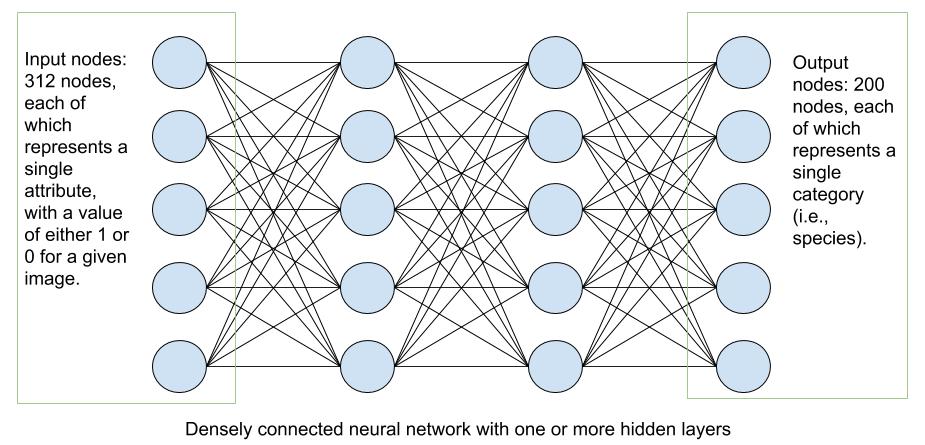
I figure that even if this is too ambitious of a project for me to implement in the time that I have, I will at least learn more about Generative Adversarial Networks and how they work.

**N8L. Learn about image processing.** It might be useful for me to have at least a passing familiarity with how image processing takes place in the world of machine learning. How do we have a machine learn from or about images without having humans encode the properties of the images? In exploring my “bird art” goal, I hope to at least become familiar with what tools and techniques are out there to aid in image processing. Convolutional Neural Networks (CNNs) have already been mentioned in class as being potentially useful for processing images, for example.

## 3.2 Design and Theory

Describe the design of your proposed system. Use annotated diagrams. Explain the theory behind your design. Explain how the two technologies will interface or compare. The reader should understand how you plan to fit the pieces together. Show this at a high level, as well as providing as much detail as you can at this point. Include at least one (meaningful) figure.

My initial implemented design will be a densely connected neural network with at least one hidden layer. I envision the 312 attributes for each image as the input nodes, and the 200 species as the categories would be the output nodes. The weights would be determined as the algorithm is trained and learns which attributes are most important in making the determination of a given species. The output should be expressed as a value between 0 and 1 indicating the conclusion that the algorithm has drawn. For example, the output might have a value of 0.73 at one node, 0.15 at another, 0.06 at another, 0.03 at a fourth node, and the rest perhaps all < 0.01, divided among the other nodes. The algorithm would then determine that the image should be categorized as the species represented by the node with the 0.73 value.



If possible, I would like to introduce a rule to the neural network to zero the weight of any attribute that is marked as “not visible” so that it is not taken into account one way or another in the final determination.

For the Random Forest algorithm, I would probably not have time to implement this, but I envision that it would be a bunch of randomly generated decision trees, each one determining based on randomly chosen features what species the bird is. The species chosen most often would be the final classification from the algorithm.

For the bird art – and given the risks identified below, this too may be more of a theoretical exercise rather than an implemented one – it would use the Generative Adversarial Network structure as described above. I envision that this would use a Convolutional Neural Network to sample grids of pixels, like a window shifting across the image to sample, say, 9 or 16 pixels at a time. This allows the network to not only consider the value for each pixel but to consider the relationships between adjacent pixels, including pixels that would not necessarily be in close proximity if the pixel values were to be listed in one long array from top to bottom and left to right. I would need to normalize the images somehow, making each image a comparable size. Perhaps there would be some fill around the edges, because if I am basing the image on the *bounding box* that contains the bird, the images are going to be all different shapes and sizes.

## 3.3 Tools

Describe the tool(s) you will probably use, or explain why you will build from scratch. It is OK if you say "I will use tool 1 or tool 2." Support the fact that you have reasonably investigated and tried out tools. Explain your choice. Show samples that make you and us reasonably confident of your choices. Show that you understand how the tools work.

**See 3.6.1 Appendix: screen shots below for examples.**

* I have been working with **NumPy** for generating arrays of random-ish numbers to use for learning purposes. I don’t yet see a need to use NumPy in my project. I will continue to use it as a learning tool, however.
* **Jupyter Notebook** has been indispensable so far in trying things out. I like to get a clear sense of what each bit of code in my program does, and it is very helpful in letting me inspect the data along the way.
* I will use either **pandas** or **TensorFlow** itself to read the data from the files and to set up the data for the algorithm. My main data set comes as multiple files, with keys that can be used to connect data in one file with data in another.
  + I have had some practice using **pandas** to read in the data, inspect it, understand it, and convert it into the form I need for my algorithm.
  + However, I also see that **TensorFlow Estimators** can also be used to read in data and manipulate it. I will try this to see if I find it even easier to use than pandas.
* I anticipate using **TensorFlow**, possibly the **Estimators** API, for designing and running the algorithm. I have several reasons:
  + TensorFlow appears to support neural networks and random forests, and it might even have support for the GAN.
  + I have found some pretty good training materials on TensorFlow so I am likely to have a lot of support, and
  + Unless TensorFlow turns out *not* to meet my needs, I’d rather put more energy into learning one tool well enough to complete the project, rather than putting the energy into researching other tools.
* I may need to delve into **scikit learn** in order to do the Partial Dependence Plot I mentioned earlier for determining the explanations for the algorithm’s decisions. I am also reading that it has
* I do intend to research **other image processing tools** if I have time.

## 3.4 Risk Retirement

Identify and prioritize the 5 top risks in carrying out the project. Try as best you can to retire the top two by the time you submit this, by means of experiments, prototypes, or work-arounds. Explain how you did this. Explain how you will retire the remaining ricks in advance.

1. **OUTSTANDING - Loading the data set.** I’ve probably spent about 15 hours in the past week and a half just trying to figure out how to get the data set arranged in such a way that TensorFlow can use it. I’ve been studying Jupyter Notebook, Python, NumPy, Pandas, and TensorFlow, looking for tools to let me inspect and manipulate the data. I’ve tried a number of experiments. I’ve emailed with my facilitator and I’ve posted to the discussion forum. I’ve succeeded in running machine learning algorithms on *other* data sets… just not on this one. If I can’t figure out how to load this data set soon, the entire project is in jeopardy. I can’t run algorithms, adjust parameters, or analyze errors if I can’t even load the data.
2. **OUTSTANDING – Short time frame – can I learn fast enough?** I am increasingly uncomfortable with my promise to analyze multiple different types of errors, using half a dozen tools I hadn’t even heard of three weeks ago, primarily learning on my own through resources found on the Internet. I’m working as hard as I can, but there’s only so much time in a day and only so much energy I can devote to coding (I’ve found that after about 10-12 hours of coding, quality really goes downhill.)
3. **ADDRESSED – Missing or malformed data.** Just in the process of attempting to manipulate the data, I found that Pandas was initially not able to read the data at all as a CSV file because some of the rows had extra columns. On inspecting these rows and spot-checking them against how they appear on the website (which also has an extraneous 0), it was evident to me that an additional 0 and space were inserted incorrectly into about 500 of the 3.6 million records in the main file linking images to their attributes. I manually removed
4. **ADDRESSED – Uneven distribution of birds in the set.** The model might not be properly trained if one species of bird dominates the set, or if some species are underrepresented in the training set. I was able to use Pandas
5. **OUTSTANDING –** **Data has no correlation.** The risk here is this: if my experiments don’t allow the algorithm to classify according to this data, how will I know whether the problem is with my algorithm or with the data? It’s possible that the attributes are too varied for the algorithm to be able to learn from them. But how will I know?

## 3.5 Schedule

Explain in outline the steps you intend to take to carry out the project. Show the completion of the stages. Include a schedule, as detailed as can be reasonably foreseen.

(A4 = Assignment 4, etc)

26-27 Sept Get TensorFlow working on bird data!

28-30 Sept Adjust parameters and experiment, begin to assess errors

1-2 Oct Update risk retirement & schedule to complete A4, review A5 requirements

3-4 Oct Lab 5 and A5 as much as possible, research Random Forests

5-8 Oct Vacation… go look at some actual birds

9 Oct Incorporate feedback into A5, continue w/errors, write about Random Forests

10-12 Oct Catch up, Lab 6, review Assignment 6

13-16 Oct OPEN: A6 + any lingering tasks; work on stretch goals if possible

## 3.6 References

Add to your references. Instructions as above.

Changed format. I find the APA citations easier to keep track of than numbers that change from assignment to assignment as new references are added and old ones removed. There are good reasons why humans use domain names and not IP addresses for websites…

# References

Amos, B. (2016, August 09). Image Completion with Deep Learning in TensorFlow. Retrieved from http://bamos.github.io/2016/08/09/deep-completion/

Breiman, L, & Cutler, A. (n.d.). Random Forests. Retrieved from https://www.stat.berkeley.edu/~breiman/RandomForests/cc\_home.htm#overview

Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2014, 03 21). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. (The Wharton School of the University of Pennsylvania) Retrieved from https://arxiv.org/pdf/1309.6392.pdf

Hardesty, L. (2016). Making computers explain themselves. (MIT News) Retrieved from http://news.mit.edu/2016/making-computers-explain-themselves-machine-learning-1028

LeCun, Y. (2016, July 28). What are some recent and potentially upcoming breakthroughs in deep learning? (Quora.com) Retrieved from https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning/answer/Yann-LeCun?srid=nZuy

Marsland, S. (2015). Machine learning: An algorithmic perspective. Boca Raton, FL: CRC Press (Taylor & Francis Group).

Portilla, J. M. (n.d.). Complete Guide to TensorFlow for Deep Learning with Python. (Udemy.com) Retrieved from https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

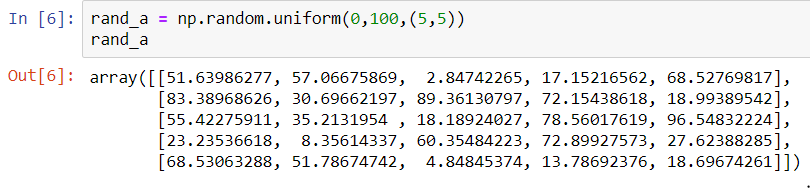
TensorFlow. (n.d.). Train your first neural network: basic classification. Retrieved from https://www.tensorflow.org/tutorials/keras/basic\_classification

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. (Computation & Neural Systems Technical Report, CNS-TR-2011-001.) Retrieved from http://www.vision.caltech.edu/visipedia/papers/CUB\_200\_2011.pdf

## 3.6.1 Appendix: screen shots

**Screen shot 1: Jupyter Notebook and NumPy**

Creating a 5x5 array of random numbers from a uniform distribution between 0 (inclusive) and 100 (exclusive). This code is not mine (it is from a Udemy class on TensorFlow (Portilla, n.d.)), but it is typical of the sorts of practice exercises I have been doing to get used to how Jupyter Notebook and NumPy work:



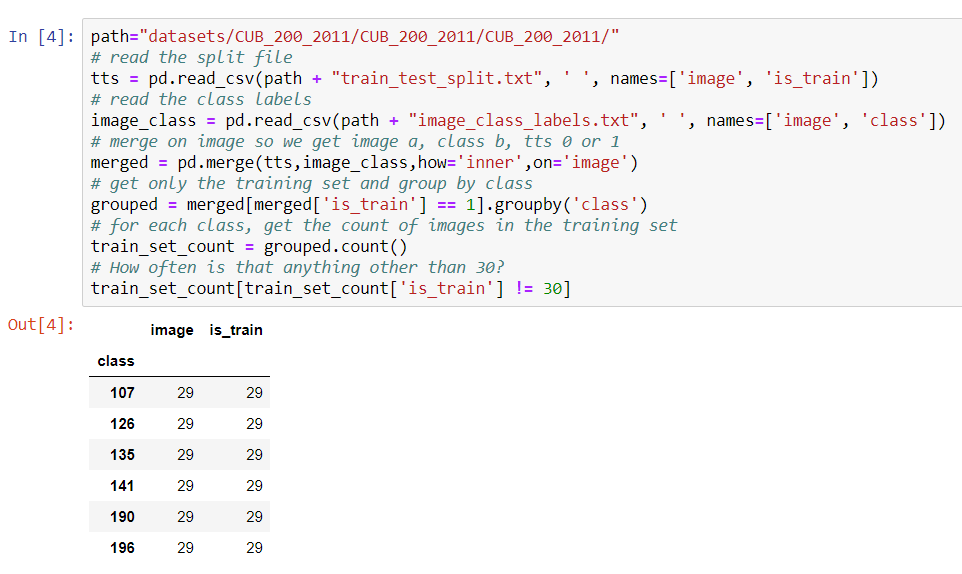
**Screen shot 2: Pandas**

Experimenting with processing the data. This code is mine:



**Screen shot 3: Pandas**

Pandas to determine how many species (“class”) in the training set (“is\_train”) found in the train\_test\_split.txt document (“tts”) have a count of images that is anything other than 30. The results suggest to me that the split suggested by the dataset authors is fairly balanced across species, with 194 species with 30 images in the set and only 6 species with 29 images.

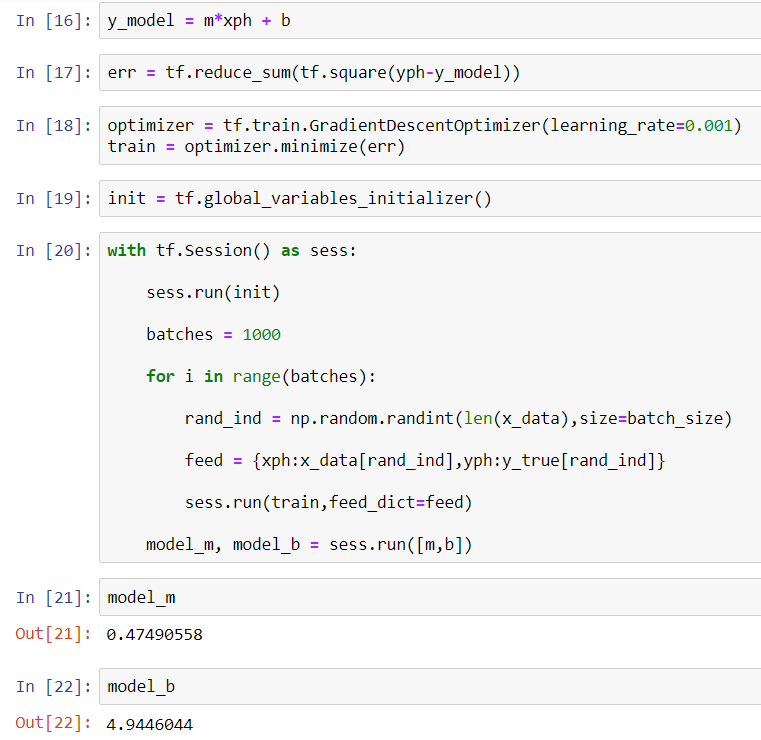


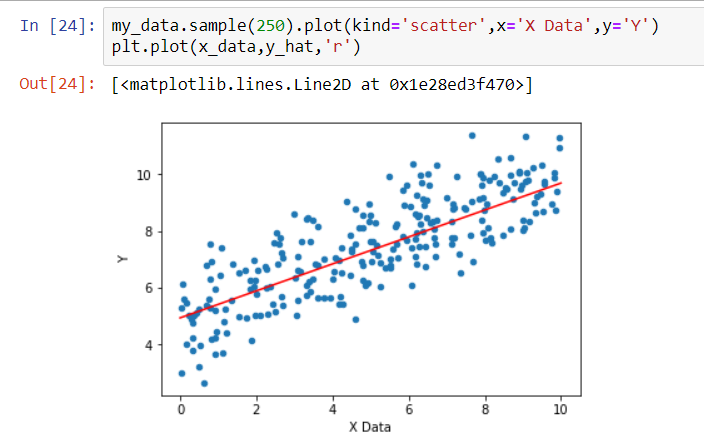
**Screen shots 5 and 6: Early TensorFlow practice, more pandas, and matplotlib**

I’ve been practicing with the code below from the Udemy class (Portilla, n.d.). I haven’t included the whole block of code because it is long. Prior to this code is setup for a fake linear regression scenario.

The first screen shot below is the key part where the error function is defined to be the sum of the squares of the differences, the optimizer is set up to use gradient descent with a specific learning rate, and the session is run. In other words, I’m starting to get a little guided hands-on experience with actually using some of the concepts we’ve discussed in class and in the labs.

The second screen shot below is what immediately follows the running of the training, the use of pandas (my\_data here is a pandas DataFrame) and matplotlib to show the results of the algorithm as a red line against a scatter plot of a random sample of 250 from the data.





## 3.7 Evaluation of Assignment 3



# Assignment 4: Project Design Plus, version 2

Keep in mind the evaluation matrix at the end as you do the work and use it to guide what you submit. Use no more than 6 pages of 12-point text excluding figures. You may include as many appendices as you wish for reference. Parts of these may be read as needed. This revision is your final view of the design prior to implementation; however, you will still be free to change it when you implement.

## 4.1 V2 Final Requirements

List your final requirements, refined again if necessary, numbering them in the form DiX and NiX where:

D/N means “Definite” / “Nice to do” (two categories, not three)

i = 1, 2, 3, …

X=L and the goal is a *learning* goal – or – X=F and the goal is *functional*

You will reference these numbered requirements in the rest of the term, when you will be asked to show what the project accomplished.

I have shifted focus away from the attributes list, and as such, the project has changed a lot.

**D1F. Image classification via convolutional neural network.** I plan to use the *images* (instead of the attributes) from the Caltech-UCSD Birds-200-2011 Dataset (Wah, Branson, Welinder, Perona, & Belongie, 2011) to train a convolutional neural network to recognize different species of birds.

**D2L. Understand different kinds of neural networks.** I plan to start with the overview of AlexNet, VGGNet, Inception, and ResNet at CV-tricks.com (Sinhal, 2018). I’d like to learn what challenges each of them set out to address and how the architectures differ. As my second “comparison” algorithm, I’ll look at the advantages that CNNs have over non-convolutional neural nets for an image classification task.

**N3F. Experiment with different kinds of neural networks.** I hope to experiment with different kinds of convolutional neural networks to actually understand the differences between them more fully. I may be limited by my computing power.

**D4F. The application correctly classifies sets of attributes into 200 classes**. I will attempt to fine tune the algorithm to improve the accuracy. I’m considering using precision and recall, and the F1 measure, as ways to assess the accuracy. Precision is the number of true positives divided by the number of all positives (both true and false). Recall is the number of true positives divided by the number of results that should have been positive (both true positives and false negatives). Marsland explains that precision and recall “are to some extent inversely related, in that if the number of false positives increases…then the number of false negatives often decreases, and vice versa. They can be combined to give a single measure, the F1 measure” (Marsland, 2015)

F1 = 2 \* ((precision \* recall) / (precision + recall))

= true positives / (true positives + (false negatives + false positives)/2)

**N5F. Implement a Generative Adversarial Network to generate images of birds.** I may not have enough data in this data set – or enough computing power on hand – to have the GAN attempt to generate specific species of birds. But I might be able to have it generate images of birds in general. I’m hoping I can reuse the convolutional neural network code above as the “discriminator” in the GAN and build a new “generator”.

**D6L. Learning about Generative Adversarial Networks.** Even if I can’t fully implement the GAN, I would like to come away with a better understanding of how these networks work, as well as their advantages and limitations or challenges.

**N7F. Image classification using attributes.** My original project goal. If I have time, I’d like to revisit this, using a regular (not convolutional) neural network.

## 4.2 V2 Design and Theory

Describe the final pre-implementation version of the design of your proposed system. Use annotated diagrams. Explain the theory behind your design. Explain how the two technologies will interface or compare. The reader should understand how you plan to fit the pieces together. Show this at a high level, as well as providing as much relevant detail as you can ~~at this point~~. Include at least one (meaningful) figure.

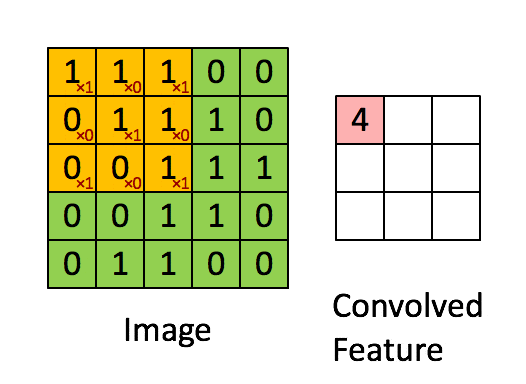
**Part I. Neural Networks**

Input will be the features (batches of arrays of RGB values for each 224x224 pixel image) and the labels (classes 1-200, possibly one-hot encoded). The input will be divided into a training set and testing set.

For initializing the weights, I’ll be looking into Xavier (or Glorot) Initialization, which Portilla mentions in the Udemy class on Tensorflow (Portilla, n.d.). This is a random initialization (normal or uniform), but where the weights have a mean of zero.

I’d want to start with a standard convolutional neural network architecture of some sort rather than trying to invent my own structure. Since I’m still learning how the various pieces fit together – filter size and padding, pooling, stride – I will probably just pick an example to start with, especially if I can find one recommended for use without a GPU. I might start by modeling something based on the example provided by Ankit Sachan in an excellent tutorial on using TensorFlow for CNNs (Sachan, n.d.)

Here’s an image from Sachan’s tutorial that I found incredibly helpful in understanding. It’s an animated image, so I recommend visiting the tutorial at <https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/> to view the animation:



The image is of a 3x3 filter being shifted across the rows of the 5x5 original. The filter (the numbers in red in the corners) is applied as a logical AND to each 3x3 area in turn. In the instance highlighted in this snapshot, four of the 1s in that area pass through the filter, so the final result is 4.

This reduces the nodes in the next layer, which may or may not be what one wants – this is where padding comes in. Sachan poses a question in that article: “Can you think of why 28\*28 from 32\*32 with the filter of 5\*5 and stride of 1” (Sachan, n.d.) – great question! Took me a while, but it’s because the 5x5 filter will bump into the edges of the 32x32 original. Put another way, there are only 28 ways you can position a 5 unit wide item in a 32 unit wide space.

Whatever I pick, it is likely to have an input layer that takes 4D tensors, one or more 2D convolution layers, and a “flattening” layer. I don’t quite understand how the flattening layer works yet, but I do understand that we need to get from this multidimensional input to our categories as output, so it makes sense. I’ll use a softmax activation function for the final output here – because it returns values 0 < x < 1 that add up to 1, it can be used as a measurement of “probability” for a given classification.

I’ve seen the softmax\_cross\_entropy\_with\_logits function used to calculate cost, I would try that. I’ll use backward propagation in the form of GradientDescent to adjust the weights as the algorithm learns. I’ve learned that the Adam optimizer helps by having the learning rate automatically slow down as we get closer to the target, allowing a bit of a balance between a large learning rate (which can learn faster but might overshoot), and a smaller learning rate (slower but more likely to converge).

**Part II. Generative Adversarial Networks**

I’m hoping I can reuse some of my convolutional neural network from Part I above as the basis for a discriminator. Instead of having output as a probability that a bird is of a specific species, the output would be the probability that an image is real or fake.

I may not have enough data to train a discriminator well enough on specific species of birds for the GAN, as it may need a more than just 30-60 pictures to determine what makes a real picture of an American Goldfinch specifically. But I might be able to train it on what real pictures of birds look like, using some mix of these photos and perhaps similarly sized photos from any dataset that doesn’t have birds in it.

Then, once the discriminator is trained, I would also try to train a generator function to create new images perhaps starting with random noise, and then using the discriminator to assess how close they are to “real” images.

## 4.3 V2 Tools

Describe the tool(s) you will definitely use, or explain why you will build from scratch. ~~It is OK if you say "I will use tool 1 or tool 2."~~ Support the fact that you have reasonably investigated and tried out tools. Explain your choice. Show samples that make you and us reasonably confident of your choices. Show clearly that you understand how the tools work (in this, the next section will help too).

**TensorFlow and TensorBoard**

*See the “Implementation Fragments” section below for more TensorFlow code.*

I have been able to run a Generative Adversarial Network on TensorFlow with code and data from the Udemy course on TensorFlow that I’ve been watching (Portilla, n.d.) and I have a model for how it can be used to build a Convolutional Neural Network. This is not my code, but here’s a snippet of Portilla’s GAN algorithm that I have tried:

def discriminator(X,reuse=None):

with tf.variable\_scope('dis',reuse=reuse):

hidden1 = tf.layers.dense(inputs=X,units=128)

# Leaky Relu

alpha = 0.01

hidden1 = tf.maximum(alpha\*hidden1,hidden1)

hidden2 = tf.layers.dense(inputs=hidden1,units=128)

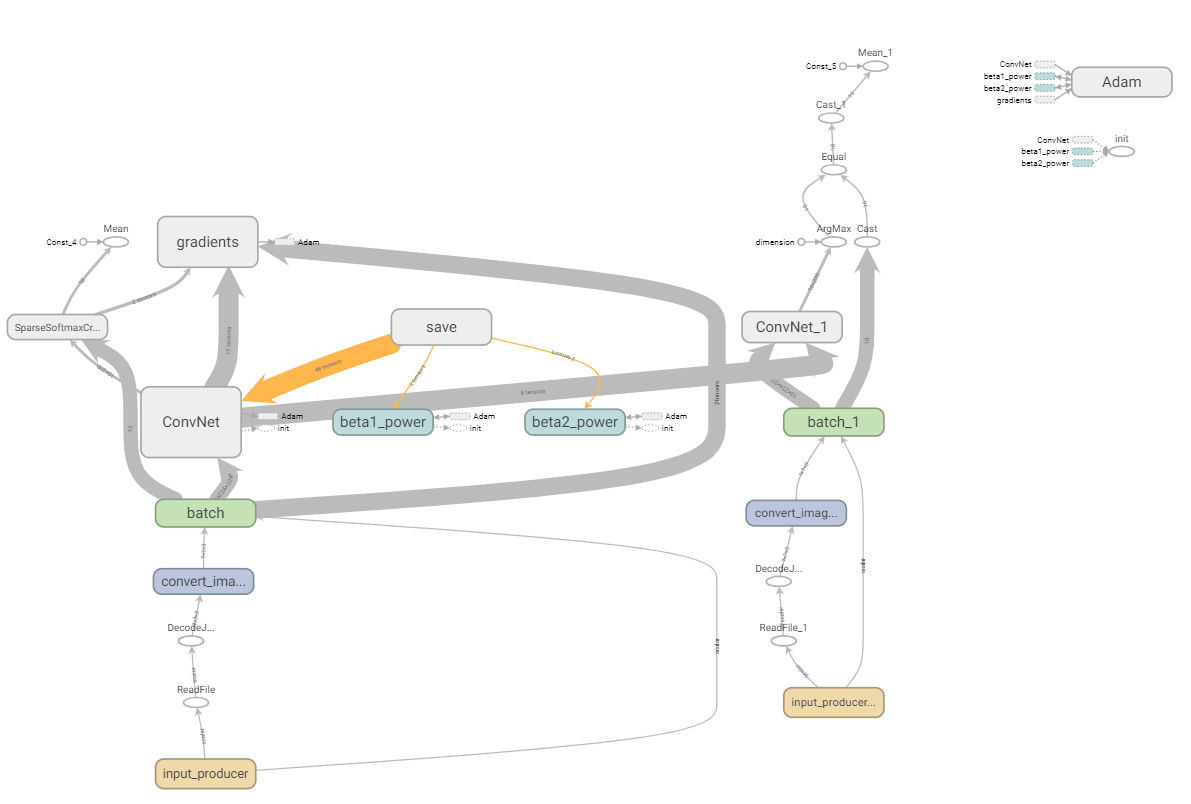
hidden2 = tf.maximum(alpha\*hidden2,hidden2)

logits = tf.layers.dense(hidden2,units=1)

output = tf.sigmoid(logits)

return output, logits

I’ll be using TensorBoard to help me understand and troubleshoot my networks. As you can see, I need to do some work on adding labels and cleaning up the graph:



*TensorBoard graph of my CNN. If you can’t read this, that’s okay. It doesn’t make sense yet anyway.*

**Pandas**

Pandas has been great for assembling, reviewing, and manipulating my dataset to get everything ready. Here’s my code for bringing several different files into one DataFrame:

# Read in the bounding boxes and image\_ids

boxes = pd.read\_csv(DATA\_PATH + "bounding\_boxes.txt",' ',names=['image\_id','left','upper','width','height'])

# Set up right and lower values, drop the width and height

boxes['right'] = boxes['left'] + boxes['width']

boxes['lower'] = boxes['upper'] + boxes['height']

boxes.drop(['width','height'], axis=1, inplace=True)

# Read in the class names, connect each to the image id

classes = pd.read\_csv(DATA\_PATH + "image\_class\_labels.txt",' ',names=['image\_id','class\_id'])

classes['class\_id'] = classes['class\_id'] - 1

images = pd.merge(boxes,classes,on='image\_id')

# Read in the names of the image files, connect each to its image\_id

files = pd.read\_csv(DATA\_PATH + "images.txt",' ',names=['image\_id','path'])

images = pd.merge(images,files,on='image\_id')

# Merge with the train-test split indicators

split = pd.read\_csv(DATA\_PATH + "train\_test\_split.txt",' ',names=['image\_id','is\_train'])

images = pd.merge(images,split,on='image\_id')

And so on, creating a train/test split.

**NumPy**

NumPy has come in handy for a few tasks I couldn’t handle through Pandas.

**Pillow** ([https://pillow.readthedocs.io/en/3.0.x/index.html#](https://pillow.readthedocs.io/en/3.0.x/index.html))

This Python image library tool has been fantastic. For image prep, I am using some code based on this: <https://www.kaggle.com/gauss256/preprocess-images> ("gauss32", Kaggle user, n.d.) and I started looking into what else it can do. It’s really quite elegant.

The open, crop, and save functions below are from pillow. The resize\_image and norm\_image functions are from https://www.kaggle.com/gauss256/preprocess-images

for item in image\_data.itertuples():

path = os.path.join(DATA\_PATH, 'images', item.path)

img = Image.open(path)

img = img.crop((item.left, item.upper, item.right, item.lower))

img = resize\_image(norm\_image(img), SIZE)

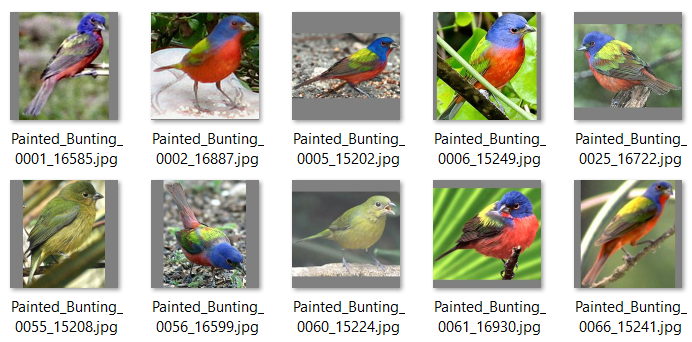
bird\_path = os.path.join(out\_dir,os.path.dirname(item.path))

os.makedirs(bird\_path, exist\_ok=True)

full\_image\_path = os.path.join(bird\_path, os.path.basename(item.path))

img.save(full\_image\_path)

Example of resulting images: they are now all cropped to the bounding box for the bird, and made square by adding gray to the edges to maintain the original aspect ratio.



## 4.4 Implementation Fragments

Show enough *parts* of an implementation—or a simplified form of it—to convince the reader that you will have the implementation of the definite requirements completed on time. These can be experimental or exploratory in nature. Your choices can coordinate with section 4.4 below. Cut and paste commented code below.

I’m experimenting right now with two different ways of setting up CNNs. Here’s the first one. This code is from (Damien, 2017).

def conv\_net(x, n\_classes, dropout, reuse, is\_training):

# Define a scope for reusing the variables

with tf.variable\_scope('ConvNet', reuse=reuse):

# Convolution Layer with 32 filters and a kernel size of 5

conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)

# Max Pooling (down-sampling) with strides of 2 and kernel size of 2

conv1 = tf.layers.max\_pooling2d(conv1, 2, 2)

# Convolution Layer with 32 filters and a kernel size of 5

conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)

# Max Pooling (down-sampling) with strides of 2 and kernel size of 2

conv2 = tf.layers.max\_pooling2d(conv2, 2, 2)

# Flatten the data to a 1-D vector for the fully connected layer

fc1 = tf.contrib.layers.flatten(conv2)

# Fully connected layer (in contrib folder for now)

fc1 = tf.layers.dense(fc1, 1024)

# Apply Dropout (if is\_training is False, dropout is not applied)

fc1 = tf.layers.dropout(fc1, rate=dropout, training=is\_training)

# Output layer, class prediction

out = tf.layers.dense(fc1, n\_classes)

# Because 'softmax\_cross\_entropy\_with\_logits' already applies softmax,

# we only apply softmax to testing network

out = tf.nn.softmax(out) if not is\_training else out

return out

TRAIN\_PATH = os.path.join(FLAGS.file\_dir, 'train/train\_data.txt')

TEST\_PATH = os.path.join(FLAGS.file\_dir, 'test/test\_data.txt')

x\_train, y\_train = read\_images(TRAIN\_PATH, FLAGS.batch\_size)

x\_test, y\_test = read\_images(TEST\_PATH, FLAGS.batch\_size)

# Because Dropout has different behavior at training and prediction time, we

# need to create 2 distinct computation graphs that share the same weights.

# Create a graph for training

logits\_train = conv\_net(x\_train, N\_CLASSES, FLAGS.dropout, reuse=False, is\_training=True)

# Create another graph for testing that reuses the same weights

logits\_test = conv\_net(x\_test, N\_CLASSES, FLAGS.dropout, reuse=True, is\_training=False)

# Define loss and optimizer (with train logits, for dropout to take effect)

loss\_op = tf.reduce\_mean(tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(

logits=logits\_train, labels=y\_train))

optimizer = tf.train.AdamOptimizer(learning\_rate=FLAGS.learning\_rate)

train\_op = optimizer.minimize(loss\_op)

# Evaluate model (with test logits, for dropout to be disabled)

correct\_pred = tf.equal(tf.argmax(logits\_test, 1), tf.cast(y\_test, tf.int64))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)

init = tf.global\_variables\_initializer()

# Saver object

saver = tf.train.Saver()

# Start training

print("Training with batch size {}, learning rate {}, dropout {}, steps {}".format(

FLAGS.batch\_size,

FLAGS.learning\_rate,

FLAGS.dropout,

FLAGS.max\_steps))

with tf.Session() as sess:

writer = tf.summary.FileWriter(os.path.join(FLAGS.file\_dir, FLAGS.log\_dir))

writer.add\_graph(sess.graph)

# Run the initializer

sess.run(init)

# Start the data queue

coord = tf.train.Coordinator()

threads = tf.train.start\_queue\_runners(coord=coord)

# Training cycle

for step in range(1, FLAGS.max\_steps + 1):

if step % FLAGS.display\_steps == 0:

# Run optimization and calculate batch loss and accuracy

\_, loss, acc = sess.run([train\_op, loss\_op, accuracy])

print("Step " + str(step) + ", Minibatch Loss= " + \

"{:.4f}".format(loss) + ", Training Accuracy= " + \

"{:.3f}".format(acc))

else:

# Only run the optimization op (backprop)

sess.run(train\_op)

# Let python shut down cleanly

coord.request\_stop()

coord.join(threads)

print("Optimization Finished!")

# Save your model

saver.save(sess, FLAGS.file\_dir)

I’m comparing it with code from (Sachan, n.d.) which is a similar pattern (two convolution layers with pooling, a flattening layer, a dense layer) but the actual TensorFlow is quite different.

## 4.4 V2 Risk Retirement

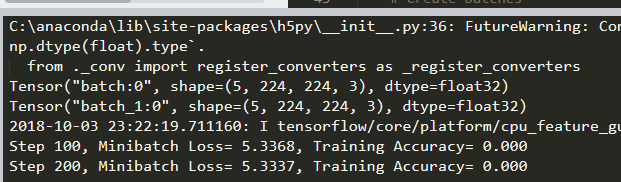
Identify and prioritize the 5 top risks in carrying out the project. Try as best you can to retire the top ~~two~~ four by the time you submit this, by means of experiments, prototypes, or work-arounds. Explain how you did this. Explain how you will retire the remaining risks in advance.

Previously addressed risks:

**Missing or malformed data.** As described previously.

**Uneven distribution of birds in the set.** As described previously.

New:

1. **ADDRESSED - Loading the data set.** I dodged this by using a different data set, which I have been able to load.
2. **ADDRESSED – Short time frame – can I learn fast enough?** By restructuring my project, I’m less concerned about whether I will be able to learn fast enough. As long as I learn enough and can demonstrate that, I think I’ll be fine.
3. **ADDRESSED (sort of) – Computing power.** My first run of a CNN algorithm on my real dataset was slow, so I let it run overnight… and my computer shut off. I’ve since set up tensorflow on a different computer – it isn’t any faster, but at least I can use my laptop to continue working while the algorithm is running!
4. **OUTSTANDING –** **Getting the algorithm to work.** I may have some CNN code, but my first round of results were not encouraging:  
     
     
     
   This went on for all 1000 steps, with the Minibatch loss hovering around 5-6 and the Training Accuracy remaining at a solid 0.000. The good news is that this is probably some simple error, maybe it is not successfully comparing the results with the labels.
5. **OUTSTANDING –** **Insufficient data.** By switching from attributes to images, I have eliminated the risk of there being no correlation in the data by switching from attributes to images. However, there are only 60 images per species. I’m not sure this is enough for the algorithm to make a useful correlation. I have several options: a) I could download more images, b) I could introduce variations on the images I already have (like flipping them, shifting them slightly, and/or rotating them slightly), or c) I could try reducing the number of categories, eliminating some similar-looking birds, which might make the task easier.

## 4.5 V2 Schedule

Explain in outline the updated steps you intend to take to carry out the project. Show the completion of the stages. Include a schedule, as detailed as can be reasonably foreseen.

3-4 Oct Get first convolutional neural net (CNN) working, or at least not crashing

5-8 Oct Vacation… go look at some actual birds… and Lab 5 ☹

9 Oct Get CNN working for real, change the settings (learning rate, etc.), Assignment 5

10-12 Oct Catch up, Lab 6, review Assignment 6

13-16 Oct OPEN: A6 + any lingering tasks; work on GAN if possible

## 4.6 V2 References

Add to your references. Instructions as above.

# Bibiliography

"gauss32", Kaggle user. (n.d.). Preprocess images. Retrieved from Kaggle.com: https://www.kaggle.com/gauss256/preprocess-images

Amos, B. (2016, August 09). Image Completion with Deep Learning in TensorFlow. Retrieved from http://bamos.github.io/2016/08/09/deep-completion/

Damien, A. (2017, 08 29). TensorFlow-Examples/examples/5\_DataManagement/build\_an\_image\_dataset.py. Retrieved from github.com: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/5\_DataManagement/build\_an\_image\_dataset.py

Marsland, S. (2015). Machine learning: An algorithmic perspective. Boca Raton, FL: CRC Press (Taylor & Francis Group).

Portilla, J. M. (n.d.). Complete Guide to TensorFlow for Deep Learning with Python. (Udemy.com) Retrieved from https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

Sachan, A. (n.d.). Tensorflow Tutorial 2: image classifier using convolutional neural network. Retrieved from CV-Tricks.com: https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/

Sinhal, K. (2018, 08 03). ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks. Retrieved from CV-Tricks.com: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. (Computation & Neural Systems Technical Report, CNS-TR-2011-001.) Retrieved from http://www.vision.caltech.edu/visipedia/papers/CUB\_200\_2011.pdf

## 4.7 Evaluation of Assignment 4



# Implementation v1

## 5.1 Summary v1

In a paragraph or two, summarize the outcome of your project functionally and learning-wise but avoid duplication with Section 5.3 below.

As of shortly before submitting assignment 5, I’ve rewritten my project from the ground up yet again, this time abandoning TensorFlow in favor of PyTorch. See the Tools section below.

I would say that I’ve achieved my goal of classifying my dataset with a CNN and learning more about neural networks in general and convolutional neural nets specifically. I haven’t achieved the goal of classifying it *well*, mostly due to the constant stream of aggravations in attempting to implement anything in TensorFlow by following the examples of others. I’m closer to that goal now that I’m not using TensorFlow anymore.

Mostly I’ve learned the perils of having insufficient data, insufficient time, insufficient computing power, and above all, insufficient (or outright conflicting) guidance from documentation and tutorials.

## 5.2. Report on Requirements v1

Explain the extent to which you accomplished each definite requirement "DiX" (X = F or L) so far as well as any other fulfilled requirements. For each, include 1-4 sentences and screenshot, as appropriate. Your effectiveness depends largely on how much you demonstrate that you learned, not necessarily that you implemented every requirement.

**D1F. Image classification via convolutional neural network.**

* In theory I have successfully trained the algorithm in TensorFlow.
* I’ve learned that convolutional neural nets are better than standard fully-connected neural networks for image processing. Two reasons:
  + The convolution restricts the connections for each neuron to its nearest neighbors, effectively mimicking the way vision works in humans. This allows the neural network to learn about data in context, for example to learn about a set of neighboring pixels together that might form a shape such as a curve to the left.
  + This also limits the number of weights that need to be calculated, in comparison with a fully-connected layer of the neural network. With the sheer amount of data in images (for a color image, width x height x 3 channels for RGB), this can become large quickly.

**D2L. Understand different kinds of neural networks.**

Sources:

Sinhal, K. <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>

Szegedy et al, https://arxiv.org/pdf/1512.00567.pdf

* AlexNet was a winning entry from Krizhevsky et al in a 2012 ImageNet competition, and others have sought to improve on it since.
* VGG was a deeper network with smaller stacked filters. This depth allowed learning of more complex features, but it had the disadvantage of requiring a lot of computation (138M parameters according to Sinhal from the link above!)
* The Inception architecture from Google has a much lower computational cost, making it more suitable for models to be used in environments such as mobile computing where resources are limited.
* ResNet turns out to be only one type of residual network, which is an architecture designed to improve training of deep networks by

Both Inception and ResNet use global average pooling instead of a fully connected layer at the end prior to the final classification layer. This turns out to reduce the number of parameters significantly while still being effective.

**N3F. Experiment with different kinds of neural networks.**

* It has been difficult to translate broad, generic descriptions of different neural network architectures into concrete implementations, especially in TensorFlow. There are some packages out there where you can choose a pre-made network.
* I did begin to experiment more however when I converted my neural network over to using PyTorch instead of TensorFlow. See more about this in the Tools section.

**D4F. The application correctly classifies images into 200 classes**.

* This dataset may not be large enough to correctly classify images. There are only approximately 60 images per species. I have identified two more datasets that could be used, but the process of either converting my existing code to use them or trying to integrate them in with the dataset I’ve been using has proven to be an enormous waste of time. I spent at least an hour matching the dataset classes up by hand before giving up.
* It’s hard to experiment with the parameters to try to improve the accuracy when every run of your algorithm takes several hours.
* The best reports I’ve had from my TensorFlow algorithm have been a loss of ~0.49 and an accuracy of 0.14. Underwhelming.
* My new PyTorch algorithm – still running as I type – has so far (after 4 of 6 epochs) achieved a loss of 2.14 and 54% accuracy. With 200 categories, that’s better than random chance, at least, and it is much better than I was achieving with TensorFlow code (assuming that either of these algorithms is correctly measuring its own accuracy, which I have yet to properly determine). I think the algorithm implementation is different, so I’ll need to review the parameters to see what I’ve changed in the process of changing tools.

**N5F. Implement a Generative Adversarial Network to generate images of birds. D6L. Learning about Generative Adversarial Networks.** **N7F. Image classification using attributes.** I haven’t attempted any of these three goals yet, as I’ve been so focused on just trying to get something useful in the form of a CNN. I’m still optimistic that I’ll be able to touch on these goals in the next week.

## 5.3 Report on Design v1

Describe the design that you have used so far. Indicate how, where, and why it has so far differed from your planned design. Describe its advantages and its shortcomings. Include a description of how the technologies you explored (not the tools—those are described below) leveraged each other. Include at least one diagram.

Currently, my PyTorch design is a convolutional neural network with two convolution layers, each followed by batch normalization, each using ReLU, and each followed by a max pool layer. This is more or less what I expected, but with the addition of batch normalization. Read an article today (https://towardsdatascience.com/dont-use-dropout-in-convolutional-networks-81486c823c16) suggesting that batch normalization was a better strategy than dropouts, which I had been using previously in my TensorFlow design.

PyTorch code sample, this part of my code was adapted from https://medium.com/ml2vec/intro-to-pytorch-with-image-classification-on-a-fashion-clothes-dataset-e589682df0c5:

def \_\_init\_\_(self):

super(CNN, self).\_\_init\_\_()

self.layer1 = nn.Sequential(

nn.Conv2d(3, 16, kernel\_size=5, padding=2),

nn.BatchNorm2d(16),

nn.ReLU(),

nn.MaxPool2d(2))

self.layer2 = nn.Sequential(

nn.Conv2d(16, 32, kernel\_size=5, padding=2),

nn.BatchNorm2d(32),

nn.ReLU(),

nn.MaxPool2d(2))

self.fc1 = nn.Linear(PIXELS\*PIXELS\*2, PIXELS\*2)

self.fc2 = nn.Linear(PIXELS\*2, 200)

I’m not thrilled with the second part of the above. There is a flattening (not shown in the above code), but I need to reimplement softmax in determining the results. Haven’t had the chance to do that yet as I’ve spent most of my time trying to find the magic combination of parameters for the convolution layers that would at least allow the network to run without crashing. I’ve learned that they need to match between layers in ways that are not always obvious.

This image isn’t mine but it is pretty much exactly what I’m aiming for:

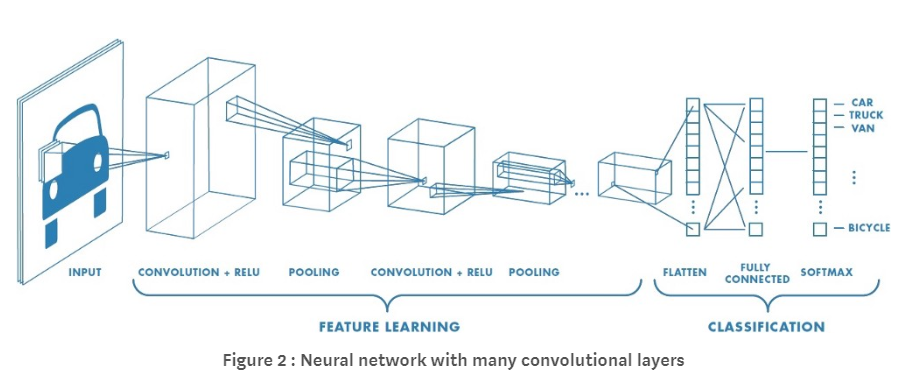


Image from https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

## 5.4 Tools v1

Describe the tool(s) that you are using. Show samples. Describe their advantages and their shortcomings. Limit: 1 page of 12-point Times New Roman.

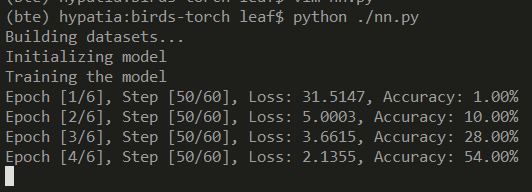
I got so fed up with TensorFlow that I switched to PyTorch, even though it meant starting over from scratch with less than a week remaining.

I started out following TensorFlow tutorials that used Estimators and Keras, higher level TensorFlow APIs. However, examples on the internet use a variety of TensorFlow techniques. Building something new becomes difficult, because these different pieces don’t always fit together.

For example, I spent a great deal of time learning to convert my data into a tf.data.Dataset, because that was recommended as a best practice and I was hoping it would let me use higher level APIs to simplify the coding process. However, so once I had that working for the data loading process, nothing else I was planning to use would work with it. None of the examples that I found for CNNs were using Datasets.

Theano came up in a few examples but official support for it has ended and it is no longer being actively developed. I tried PyTorch, and I was able to get up and running within 5-6 hours.

Code example from PyTorch is above, here’s an output example (I probably shouldn’t have had it displaying output every 50 when steps are set to 60… I’m letting it run anyway.)



# Instructor’s Evaluation

.

# Implementation v1

## 6.1 Summary v2

In a paragraph or two, summarize the outcome of your project functionally and learning-wise but avoid duplication with Section 5.3 below. Underline edited sentences and additions from v1, if any.

My project contains several pieces:

* A convolutional neural net that attempts to classify images from the CUB-200-2011 dataset into 200 species.
* A second neural net that classifies images from CUB-200-2011 and two other datasets (Birdsnap and NABirds) for 30 species of birds that don’t have much visual overlap
* A training program designed to help evaluate hyperparameters
* A Jupyter Notebook containing code I used to clean and prepare the three datasets to be compatible.

Mostly I’ve learned the perils of having insufficient data, time, computing power, and documentation. CUB-200-2011 (at ~60 images per species) is too small for the CNN, let alone my Generative Adversarial Network plan. The GAN was certainly not possible in this time frame, with the resources I have. Even the combined datasets only give me about 200-250 images per bird species. Getting more data may be better than any tuning of hyperparameters I could do.

## 6.2. Report on Requirements v2

Explain the extent to which you accomplished each definite requirement "DiX" (X = F or L) so far as well as any other fulfilled requirements. For each, include 1-4 sentences and screenshot, as appropriate. Your effectiveness depends largely on how much you demonstrate that you learned, not necessarily that you implemented every requirement. Underline edited sentences and additions from v1, if any.

**D1F. Image classification via convolutional neural network.**

* In theory I have successfully trained the algorithm in TensorFlow.
* I have had more success training two CNN algorithms in PyTorch.
* I’ve learned that convolutional neural nets are better than standard fully-connected neural networks for image processing. Two reasons:
  + The convolution restricts the connections for each neuron to its nearest neighbors, effectively mimicking the way vision works in humans. This allows the neural network to learn about data in context, for example to learn about a set of neighboring pixels together that might form a shape such as a curve to the left.
  + This also limits the number of weights that need to be calculated, in comparison with a fully-connected layer of the neural network. With the sheer amount of data in images (for a color image, width x height x 3 channels for RGB), this can become large quickly.

**D2L. Understand different kinds of neural networks.**

Sources:

Sinhal, K. <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>

Szegedy et al, https://arxiv.org/pdf/1512.00567.pdf

* AlexNet was a winning entry from Krizhevsky et al in a 2012 ImageNet competition, and others have sought to improve on it since.
* VGG was a deeper network with smaller stacked filters. This depth allowed learning of more complex features, but it had the disadvantage of requiring a lot of computation (138M parameters according to Sinhal from the link above!)
* The Inception architecture from Google has a much lower computational cost, making it more suitable for models to be used in environments such as mobile computing where resources are limited.
* ResNet turns out to be only one type of residual network, which is an architecture designed to improve training of deep networks by

Both Inception and ResNet use global average pooling instead of a fully connected layer at the end prior to the final classification layer. This turns out to reduce the number of parameters significantly while still being effective.

**N3F. Experiment with different kinds of neural networks.**

* It has been difficult to translate broad, generic descriptions of different neural network architectures into concrete implementations, especially in TensorFlow. There are some packages out there where you can choose a pre-made network.
* I did begin to experiment more however when I converted my neural network over to using PyTorch instead of TensorFlow. See more about this in the Tools section.

**D4F. The application correctly classifies images into 200 classes**.

* My original dataset may not be large enough to correctly classify images. There are only approximately 60 images per species. I identified two more datasets and created a new subset I’m calling “30 birds” – only 30 species of bird, each fairly easily distinguished from the others. I have had more luck with this dataset made artificially easier. I also augmented the dataset by adding “new” images that were the original images flipped left to right, rotated slightly, and with the bounding box slightly larger.
* It’s hard to experiment with the parameters to try to improve the accuracy when every run of your algorithm takes several hours. I was not able to find a prepackaged method of doing a grid search or random search in PyTorch – I found one that required knowledge of scikit learn that I didn’t have time to acquire, and another that did not support CNNs. I did however experiment with creating my own partial grid search – just an algorithm to loop through different learning rates and try them out in sequence, checking against a validation set. There isn’t enough time remaining for me to run a full grid search.
* My first reports were a loss of ~0.49 and an accuracy of 0.14. I realized my batch size of 3 was far too low to be of any use, and increased that to 100. But after 5 epochs I had 100.00% accuracy on the training data… and 5% accuracy on the test data, which sounds like overfitting. After augmenting as described above, I had 86% accuracy on training data (still suspiciously high for 200 species) but my program crashed when it came time to test.
* I then created the “30 birds” data set described above and (using a batch size of 80). I changed the algorithm somewhat too, adding batch normalization after the individual layers. And I changed kernel size to 3, which was probably too many changes at once to figure out what was most helpful… I got an accuracy of 73% in training matched by an accuracy of 79% on test images. Interesting. I removed one of the two linear layers, and both of those numbers decreased by a few percentage points.

**N5F. Implement a Generative Adversarial Network to generate images of birds. D6L. Learning about Generative Adversarial Networks.** **N7F. Image classification using attributes.** I ran out of time before I could attempt this goal, unfortunately. I did at least find a promising example that I would have tried to use as a starting point:

<https://github.com/last-one/DCGAN-Pytorch/blob/master/network.py>

However, as I mentioned above, the dataset may not be large enough for a GAN anyway.

## 6.3 Report on Design v2

Describe the design that you have used so far. Indicate how, where, and why it has so far differed from your planned design. Describe its advantages and its shortcomings. Include a description of how the technologies you explored (not the tools—those are described below) leveraged each other. Include at least one diagram. Underline edited sentences and additions from v1, if any.

I wound up with something similar to what I had planned in assignment 5: A convolutional neural net with two convolution layers, each with batch normalization applied, each using ReLU as the activation function, and each with pooling done after it. I tried three convolution layers on my “older” neural net (the 200 classes one) and my computer froze! The convolution layers use a kernel\_size of 3 (down from 5, I seem to remember reading that a smaller kernel was better for fine-grained feature recognition, which I thought might be useful for these bird images). They have a padding of 1 and stride of 1 – this may have come from the same source that recommended the kernel size of 3.

The convolution layers are followed by flattening, then by two fully connected layers. The hidden layer has an arbitrarily chosen width of 128. In none of my research could I find any suggestions for how to choose this width, other than “try using different widths”. It is my understanding that it should be somewhere in between the enormous width of the flattened layer and the output layer (which is the same as my number of classes).

No softmax – I never did figure out how to implement it exactly. I did at least learn that some consider the log\_softmax function superior to the softmax function, so I was planning on using log\_softmax. Maybe it is as simple as return F.log\_softmax(out, dim=1), I’m not sure.

I’m using an Adam optimizer, because most of the examples I found used that. I like the idea of changing the learning rate so that it is faster at the beginning and slower as one gets further along – you get where you’re going faster but cut back on your risk of overshooting the target.

This image isn’t mine but it is exactly what I wound up with, except for softmax:

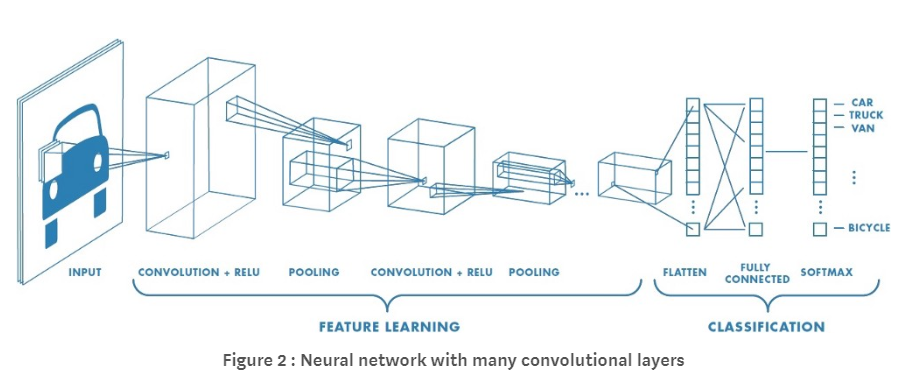


Image from https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

## 6.4 Tools v2

Describe the tool(s) that you are using. Show samples. Describe their advantages and their shortcomings. Limit: 1 page of 12-point Times New Roman. Underline edited sentences and additions from v1, if any.

I got so fed up with TensorFlow that I switched to **PyTorch**, even though it meant starting over from scratch with less than a week remaining. I continued to use **NumPy and Pandas** for data cleaning and manipulation. Samples:

def build\_datasets(path, batch\_size, validation\_mode):  
 train\_csv = os.path.join(path, 'train/train\_data.txt')  
 train\_dataset = BirdDataset(path + 'train/', csv\_file=train\_csv, transform=ToTensor())  
 train\_loader = DataLoader(dataset=train\_dataset, batch\_size=batch\_size, shuffle=True)  
*# omitted code for brevity…*

class BirdDataset(Dataset):  
*# omitted code for brevity…* def \_\_getitem\_\_(self, idx):  
 img\_name = self.birds.iloc[idx, 0]  
 # A print statement here helped me fish out a bad image file that was crashing my program!  
 # print(img\_name)  
 img = Image.open(os.path.join(self.image\_path, img\_name))  
 img = np.array(img)  
 label = self.birds.iloc[idx, 1]  
 sample = {'image': img, 'label': label}

if self.transform:  
 sample = self.transform(sample)

return sample

class ToTensor(object):  
 """Convert ndarrays in sample to Tensors."""  
 def \_\_call\_\_(self, sample):  
 image, label = sample['image'], sample['label']  
 # swap color axis because  
 # numpy image: H x W x C  
 # torch image: C X H X W  
 image = image.transpose((2, 0, 1))  
 return {'image': torch.from\_numpy(image).type(torch.FloatTensor),  
 'label': torch.tensor(int(label)).type(torch.FloatTensor)}

**Jupyter Notebook:** I wound up coding a lot of my data cleaning in a Jupyter Notebook because that made it easier to see what I was doing step by step.

## 6.5 Contrast between approaches

You were to include two approaches to your problem, and implement at least one. Contrast the two approaches as they specifically relate to your project.

As much as I would have liked to have researched Random Forests extensively, I wound up spending about two weeks just trying to get a neural network to actually use my data, let alone actually complete without crashing. Then there was a third week of just trying to get the neural network to run. In the process, I learned about different kinds of neural nets (convolutional and not) as described in my goals section in 6.2.

Perhaps in place of learning about Random Forests, I learned a bit about GANs. There have been some examples of neural nets being easily “fooled” by a GAN. Subtle perturbations can be applied to an image classified one way such that a human would not be able to tell the difference, but a neural network might classify the image incorrectly with a high degree of certainty. This has security implications as a malicious image could be created that would be misclassified.

GANs themselves have problems to watch out for, such as “mode collapse”. This is a scenario where the generator has discovered a successful example (or several), so instead of improving or trying something new, it just keeps submitting similar results repeatedly. There’s a more extensive article on this (and GANs in general and their difficulties) here: https://medium.com/@jonathan\_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

## 6.6 What did *not* work well

Explain the most important aspects of your project that fell short of your plans or desires.

As I understand it, a lot of machine learning is really about preparing your data, and (as we saw in class with the gray codes example) deciding how it should be represented. I would have liked to have downloaded a dataset, spent a day or two understanding it and perhaps cleaning it a little, and then moved on to training an algorithm. So my first and foremost disappointment was the amount of time I had to spend on doing things that weren’t directly machine learning. I guess that is the nature of the work, though.

I’m also disappointed by the output of these algorithms. Many of the articles I read had full color examples or at least interesting data. All I got from my algorithm was an accuracy measurement (and, if I wanted, I could get a list for each batch of which class was chosen and which was expected). I wish I could point to a picture and the algorithm could tell me what bird it thought that was, or even better, which birds were its top choices and how confident was it in those choices. But figuring out which photos were problematic wasn’t baked into the algorithm that I chose, and there was no time to explore it further. I was also hoping softmax would help with this at least somewhat, but applying softmax wasn’t as intuitive as it sounded from the reading.

## 6.7 What *did* work well

In paragraph form, explain the most important aspects of your project that met or exceeded your plans or desires.

I actually started to understand the math of the convolutional neural network, as the weeks went by. When I got errors in setting up my neural network, I found that I was increasingly able to understand them and resolve them. One of the most common errors resulted from having the tensor shapes of different layers not matching in some way. Took me a while to figure out what that error was really telling me was not that they should *match*, but that they should be compatible: the “out” of one should match the “in” of the next.

I was able to teach myself a handful of new tools in short order, while also teaching myself about the concepts behind what I was doing. I didn’t have the advantage of advance notice for setting up tools that my classmates had, so I was starting at zero with trying to get TensorFlow to work with at least one of my three preferred Python development environments (I was eventually successful with both PyCharm and SublimeText, couldn’t quite get it working in VS Code.) I supplemented my learning with hours of videos from Udemy.com, Google’s machine learning “crash course”, hunting through the official Pandas documentation, reading and rereading official and unofficial tutorials on both TensorFlow and PyTorch, and of course Stack Overflow.

I’m thrilled that I got to work with image data. I did not have much prior knowledge about working with images and it’s a topic I’ve always found sort of daunting. The “pillow” image library was a tremendous help. If only working with images were always that easy!

I’ve listed some additional things I’ve learned in Appendix A.

## 6.8 Sample Source

Supply up to 1 page of key excerpts from your source code—or what comes closest to “source code.” Limit: 2 pages of 12-point Times New Roman. Include an explanation of where the excerpts fit in your implementation.

This is the main class that defines the neural network:

class CNN(nn.Module):  
 """   
 based on   
 https://medium.com/ml2vec/intro-to-pytorch-with-image-classification-on-a-fashion-clothes-dataset-e589682df0c5   
 <https://cs230-stanford.github.io/pytorch-vision.html>  
 """  
 def \_\_init\_\_(self, image\_size, num\_classes):  
 self.image\_size = image\_size  
 self.num\_classes = num\_classes  
 self.hidden\_layer = 128  
 super(CNN, self).\_\_init\_\_()  
 self.layer1 = nn.Sequential(  
 nn.Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, padding=1, stride=1),  
 nn.BatchNorm2d(32),  
 nn.ReLU(),  
 nn.MaxPool2d(2))  
 self.layer2 = nn.Sequential(  
 nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, padding=1, stride=1),  
 nn.BatchNorm2d(64),  
 nn.ReLU(),  
 nn.MaxPool2d(2))  
 self.fc1 = nn.Linear(64\*32\*32, self.hidden\_layer)  
 self.fc2 = nn.Linear(self.hidden\_layer, self.num\_classes)  
   
 def forward(self, x):  
 out = self.layer1(x)  
 out = self.layer2(out)  
 out = out.view(out.size(0), -1)  
 out = self.fc1(out)  
 out = self.fc2(out)  
 # Apply softmax here?  
 # return F.log\_softmax(out, dim=1)  
 return out

Here’s some code I’m proud of for cleaning up the images and making the “new” images for augmenting the data set. “Image” here is a class from the pillow image library.

for item in image\_data.itertuples():  
 path = os.path.join(DATA\_PATH, "images/", item.path)  
 if os.path.exists(path):  
 img = Image.open(path)  
 img1 = img.crop((item.left, item.upper, item.right, item.lower))  
 img1 = resize\_image(norm\_image(img1), SIZE)  
 bird\_path = os.path.join(out\_dir,os.path.dirname(item.path))  
 os.makedirs(bird\_path, exist\_ok=True)  
 full\_image\_path = os.path.join(bird\_path, os.path.basename(item.path))  
 img1.save(full\_image\_path)  
 # flip the image and modify it a little  
 full\_image\_path2 = os.path.join(bird\_path, "2\_" + os.path.basename(item.path))  
 img2 = img.rotate(2).crop((item.left+3,item.upper+3,item.right+3,item.lower+3))  
 img2 = img2.transpose(Image.FLIP\_LEFT\_RIGHT)  
 img2 = resize\_image(norm\_image(img2), SIZE)  
 img2.save(full\_image\_path2)  
 else:  
 # Oh nooo, I prevent them from generating images but they still wind up in the data file!  
 print("Could not find {}".format(path))

## 6.9 Source

Refer the reader to your source code (or what comes closest to it) and input where possible.

* My code (and presentation): <https://github.com/bouncingleaf/birds>
* CUB-200-2011 dataset: <http://www.vision.caltech.edu/visipedia/CUB-200-2011.html>
* NABirds dataset: <http://dl.allaboutbirds.org/nabirds>
* The Birdsnap dataset has a python script that attempts to download each image, one at a time. A number of images that I wanted to use were simply unavailable. I also needed to edit the download program to focus on the ~40 birds I was considering for my subset, otherwise it was taking too long. More info about this data set is here: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.450.500&rep=rep1&type=pdf>
* If possible, I will also submit a zip file of my data folders, these include the cropped and normalized versions of the images including the augmented images, divided into test and train. There are text files in each folder, and the test folder has text files further dividing the test set into a test set and a validation set. I’ve only used that division to use the validation set for parameters testing. If I can’t submit this zip file for some reason, I will be happy to upload it to Google Drive and send a link, or whatever else might work. I was not able to upload it to github as it was too large, even zipped.

## 6.10 Presentation

Make a 3-5 minute video presentation of your results, including a demonstration.

jmroy\_project\_presentation.zip, I believe the zoom\_0.mp4 file is the one you want to select. If I can upload it in addition to this document, I will. If not, it is available on my github site. If you can’t retrieve it there either for some reason, let me know.

# Evaluation



## Appendix 1 – More things I learned from this project

For voluminous material, as needed—to be read on an as-needed basis only. (References in at least one place within the paper.)

In addition to what I described above:

* It’s good to build flexible code for breaking up your dataset, because you might need to do that more than once.
* Reading your files from a list vs. reading the files directly from the folders they are in… I figured the former would be easier, but the more I worked on this project, the more I regretted not having gone with the latter approach.
* Cropping images to try to make them easier for the algorithm to recognize may be both useful and not. On one hand, it probably does help to eliminate background noise. On the other hand, it would limit the usefulness of my algorithm in identifying birds in any images where bounding boxes have not been determined.
* The variation in the dataset was greater than I expected. Not only do many species vary by gender or age (females and juveniles are often less brightly colored), but the position of the bird varied a lot (perched, sitting, leaning, flying – or just cut off by the frame of the photo, some pictures were just the bird’s head).
* I read that batch normalization was a better strategy than dropout for preventing overfitting. If I had more time I would add that back in to see the difference for myself.
* Timestamps in my programs would be helpful to see how long it takes for each to run.
* Using named parameters is better than positional parameters even when you can use either. My understanding of how CNNs are shaped increased a lot when I added the names for the first two parameters: in\_channels and out\_channels. I could then see what these numbers did and how they needed to match one another. This was helpful in taking stabs in the dark as to what the widths of each layer should be.
* *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville looks like a *really* good book (2016, MIT Press). I wish I had picked it up more than one day before the end of the term.
* Being able to pass in hyperparameters as arguments to your program is a good idea, as opposed to hardcoding. I had to rework my program a bit to get the testing code to even be an option.
* Working on two different computing systems, using two different operating systems, gets confusing quickly. I spent too much time dealing with that as well. My laptop wasn’t really up for the task of handling a long training session.
* My laptop’s peculiar habit of shutting itself off at night did not help matters either, as I could not leave anything running on it overnight and still expect to see results in the morning unless I was redirecting output to a file. Learned that the hard way…
* Google’s machine learning service has a free trial – good thing it’s free as I never did figure out how to get it set up to run anything for me.
* Five minutes of video recording goes by fast!