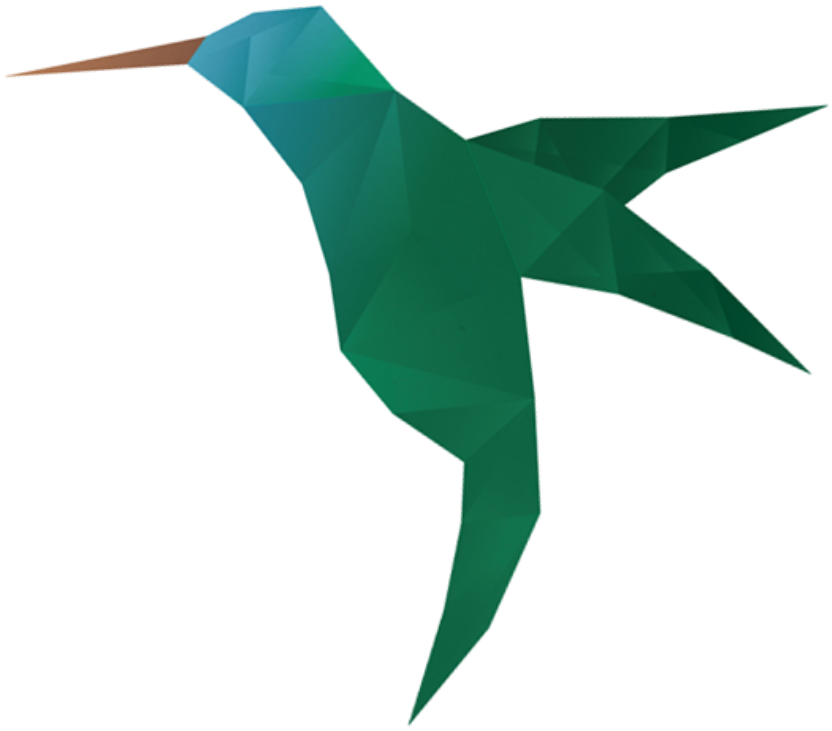


KNEST

Sponsored by:
Dr. Gary Leavens

Alexander Decurnou
Jonathon Rice
Joseph Leavitt
Nhi Nguyen

Table of Contents

Executive Summary

This is the design document to *Knest*, a project sponsored by Dr. Gary T. Leavens of the College of Engineering and Computer Science at the University of Central Florida. Knest. is a desktop application that leverages techniques from computer vision, deep learning, and machine learning in order to create a novel system for automatic image selection and enhancement. Given a directory of images as an input, the application will filter out any blurry images from the set. Then, for each image in the resultant subset, the system will detect whether a bird is present in the frame. All images that do not contain a bird are discarded from consideration. From this point, the system will attempt to locate an eye of the bird and whether the bird is in flight. Depending on the outputs from both of these tests, the image will be cropped in such a way that would be considered aesthetically-pleasing and will be stored in a subdirectory for viewing.

The main objective of the project is to reduce the amount of time that birders (and other nature photographers) spend checking their images for their desired subjects and enhancing said images. Thus, we aim to allow the user to process large amounts of photographs by filtering out blurry, unclear, and unhelpful images. This allows the user to progress straight into the identification of subjects, which is arguably more enjoyable than sorting through countless images.

Our technical approach blends together a number of technologies de jour in order to successfully accomplish our objectives. We make use of deep learning and machine learning techniques, specifically convolutional neural networks, in order to detect the presence of birds and related features in the images. We also use a popular open-source computer vision library, which helps us to determine the exact position of birds in the image frame and crop the image accordingly. Several web technologies and a user interface library are used in order to create a graphical user interface so that the user is not required to make use of the command line in order to use our application.

We believe our application has the potential to significantly affect the birding industry as it can greatly reduce the amount of time spent doing mundane photography manipulation tasks. This time can then be directed towards more enjoyable activities. Also, our application has been developed in such a way that we can add support for other animals or even human beings. This could be revolutionary for any users with image collections of any size as our application could then be used to process directories that may hold forgotten happy memories.

Introduction

Proposed Solution

Our proposed solution is a system that utilizes computer vision and machine learning in order to produce a selection of desirable photographs (henceforth referred to as files, photos, or pictures) of birds and possibly various other animals from a user's larger collection of original photographs. The system leverages computer vision techniques in order to calculate the blurriness of a photograph and rejects it if it does not meet a defined threshold. The file is then passed to a collection of neural networks which determine whether a bird is found in the picture, and if so, whether it is in-flight and/or facing the camera. Pictures that do not successfully pass through this collection of networks are discarded; the remainder of the collection is passed to the final stage for processing. This final stage utilizes other computer vision techniques in order to find the bird(s) in the photo and delineates a region containing the bird(s). This region is then sent to a method that programmatically crops the picture to that region, and the picture is then saved to a file in a new directory in which all other selected photographs will be saved.

Broader Impacts

Figure 1: Birding in action [1]

Currently, there are 18 million active bird watchers in the United States. This project not only has the potential to improve their lives but also of bird watchers all around the world. Imagine taking 3000 photos of birds throughout a long weekend of birding and then spending days filtering and enhancing these photos. The task is daunting and time-consuming, and hopefully our application can help. Having the ability to filter out thousands of images in a matter of seconds is an absolute game changer for people in the birding community. This could potentially spark a widespread interest in bird watching and bring people back into the hobby who simply didn't have the time to do it anymore. This is especially important as bird-watching alone is a lucrative source of income for many national and local wildlife preservation communities. More incoming traffic of enthusiastic birders can help support wildlife foundations and assist in the general preservation of local wildlife [2]. If our project gains traction then we could begin to support multi-animal detection and recognition to positively impact other animal watching communities as well.

Our project can potentially greatly benefit birds and other wildlife. Since wildlife observations are done within these animals' natural habitats, some wildlife observers risk disturbing their environments in order to take good images. With our project making the process of identification and image sorting faster and easier, they can continue taking hundreds of photos from safe distances with little to no more extra work. This impact is phenomenally significant as birds consume over 98% of certain pests, and pollinate many plant species, etc. Revolving interest and preservation of birds are essential to the economy, both financially and environmentally [2].

Lastly, the project's purpose can extend well beyond that of identification of birds and other animals. Families with newborn children take countless photographs in the hopes of catching an important moment or creating a timeless memory upon which they can fondly look back. Our project could have the potential to be modified in order to recognize human subjects in a picture and do much of the same work regarding cropping and blur detection. These parents could then look through a much smaller subset of pictures to find the preferred photographs, which would allow them to spend even more time with their children and create more memories.

In contrast, the project could be used as a supplement for security systems. Many businesses use systems in which a picture of an environment is periodically taken and stored on the system for later viewing. The project could be modified in such a way that prior to saving the pictures to disk, the system could process the files and only save the pictures that contain a human subject to the disk. Thus, in the possible case of an unfortunate event, one could find helpful information in much shorter span of time and taking advantage of that to spend more time rectifying the situation.

Motivations

Alexander Decurnou - I have two main motivations for selecting this project. The first is mainly academic, in that I wanted to work on a project that combined the greatest amount of material that I have learned into one thing, e.g. computer vision, artificial intelligence, etc. Regarding my second motivation, I used to do medical research on the human brain, and I have found it interesting ever since. So, it is exciting to be able to create something that models it, however small or rudimentary it may be.

Joseph Leavitt - My primary reason for choosing this project is to explore a topic in computer science that is outside the scope of my current strengths and skills. Machine learning and neural networks are very trendy topics in the computing industry and currently provide the best solutions to many problems in image recognition. With that said, I strongly believe that this project is well-suited for me and will help me dive deep into machine learning to learn something that has a high potential to be beneficial to me in the near future. I may consider going to graduate school to formally study machine learning if I become very interested in the subject matter.

Jonathon Rice - My motivation for this senior design project is to gain reputable experience in artificial intelligence. I have completed a couple of personal projects on deep learning and computer vision. However, as much as I learned from those projects, neither of them were serious or large. My personal goal is to pursue a masters in the field of artificial intelligence, if not further, depending on the academic experience. I believe this project will help me get there. Currently, I am part of the graduate computer vision course, learning relevant material to the project. My experiences will complement well with this project's goals.

Nhi Nguyen - Prior to this semester, I have had relatively little computer science experience outside of classroom settings and small personal projects. For senior design, I wanted to explore an interesting topic that was outside of my current technical knowledge. Dr. Leaven's bird photo project proposal combined topics and skills that I wanted to learn and gain experience with, namely artificial intelligence and computer vision. Before it was pitched, I had heard a little of neural networking, specifically TensorFlow object detection API using Python. Despite little to no personal experience with the scope of this project, I am confident that I can complete the project and gain valuable experience and knowledge with a field of study that I find interesting.

Technical Objectives

Goals and Objectives

Wildlife observation is a popular activity, both recreational and professional. Our project will primarily be aimed towards birders, although a desired aspect would be to include a diverse amount of species. Although contemporary digital cameras make taking hundreds of photos insanely easy, the most difficult and time-consuming aspect of bird-watching is having to sort through a large directory with only a select few number of actually usable images.

Our main goal is to give bird watchers a way to expedite the selection and enhancement of their bird photographs. Because birders observe birds in their natural habitat, they often take enormous amounts of photos upon sight, most of which are blurry and unhelpful due to the elusive and active nature of birds. Our objective is to make bird-watching and identification easier and speedier by eliminating the time birders take to sort through large sets of photographs to remove blurry, unclear and unhelpful photos. This project will allow birders to skip straight to the identification of bird species and save valuable time. We hope that our application will help birders simplify the process and allow them to spend more time actually observing birds and identifying species.

Function of the project

The primary function of our project is a desktop application - utilizing artificial intelligence, computer vision and machine learning - that will take a large directory of bird photographs, filter out blurry or out-of-focus images and create a subdirectory of bird photos that are clear, cropped and primarily centered. The user of the application need only input their directory of photographs, and our design project should automatically do the rest. The application will not delete or alter the original directory of photos in any way but will store a copy of the enhanced photo in a subdirectory within the original.

The design of our project must support the identification and cropping of not just a single bird but potentially a large flock of birds. The application will determine the number of birds within a flock to be a part of the final photo, potentially a few or all of them. Additionally, the application will sort through both resting and airborne birds through eye identification, flight detection, etc. As of present, the project will only need to support JPEG images, but will most likely be able to support other image formats. Moreover, the surrounding environment around the birds should also be considered by the application when enhancing and cropping the photo.

Specifications and Requirements

Requirements

1. The system shall function as a desktop application.
2. The system shall be compatible with the JPEG image format.
3. The system shall take a directory of images as an input.
4. The system shall create a subdirectory to store processed images.
5. The system shall remove images that match any of the following criteria:
    - Image is predominantly blurry
    - Image does not have a bird at any location in the viewpoint
    - Bird(s) in image are facing away from the viewpoint
6. The system shall be able to handle multiple subjects.
7. The system shall center the frame of the output image upon the eye of the bird or the most equidistant point between birds, if there are multiple subjects.
8. The system shall crop the image in such a way that the subject(s) of the image fill the frame.
9. The system shall store the processed images in the corresponding subdirectory.

Optional Functionality

There are a number of optional functionalities that may be added at the request of the sponsor or at the discretion of the group in the event that all the requirements are finished ahead of the proposed schedule.

A possible component that we can add at a later time would be the ability to recognize other animals in addition to birds. This is not an infeasible addition to the code of the project, as it would require only a few tweaks that allow for the detection of other classes and a final adjustment at the softmax layer since it returns a vector of all the calculated probabilities. The main difficulty with this component would be creating the training dataset. We were provided a large dataset for birds that contains thousands of images; in order to achieve the same accuracy for other animals, e.g. butterflies or dragonflies, we would need to either find a comparable dataset or construct our own image set. The latter option would take an exorbitant amount of time that may be better directed towards improving the primary objective of bird detection.

The suggestion of species detection has come up at various times during the project. While the “cool factor” associated with implementing this suggestion would be desirable, it is most certainly out of the realm of possibility when we consider the time and resources available to complete the original objective of this project. If we were to implement the suggestion for only bird species, it would require a massive effort to implement a class for each species as a common estimate puts the amount of bird species at ten thousand.

Even if we were to do this, there are two other reasons why this component would be infeasible to add. A neural network model with ten thousand classes would be extremely memory-intensive and would certainly exceed the amount of memory that we are able to use by an order of magnitude. Also, in order to properly train the network to correctly predict individual species of birds, we would need to have a labeled dataset that contains multiple unique images for each species; this would take a significant amount of time that could be used elsewhere. We would then have to create subfolders for every species detected in the original folder; this could lead to filesystem fragmentation, as there may be a lot of folders with only one image file in them.

We are developing this project as a standalone desktop application. However, the notion of photography software suite integration has been suggested at multiple times. This is a feasible implementation as it would mainly depend on the amount of time remaining when we finish the project. For integration into major suites, such as GIMP or Photoshop, we would use the corresponding standard development kit (SDK) and most likely use system calls to have the suite run our program. We could also create a view inside the suite(s) that would show the collection of images outputted by the application.

One of the main requirements is that the application works with the JPEG image format. However, there are many image formats currently in use for virtually every purpose, regardless of their initial use case. Thus, it would be desirable to have the application support multiple image formats as users have the ability to save their images in any format they choose. This suggestion would be almost trivial to implement as one of the modules that is being used (Pillow, a fork of the Python Image Library) supports virtually every image format in use today.

Research

Machine Learning

In order for a computer or program to carry out a given task, they require all the data necessary to fulfill the parameters and requirements governing the operation. However, computers cannot collect and interpret the data themselves, and humans are required to process the data in order for a system to consume it. This process has the potential to be extremely time-consuming as the system would only be able to process as much data as humans can find and present it. Machine learning is essentially about constructing computers and programs in a way that allow them to make inferences about observed data and determine the validity of said inferences. One could say that this mimics the act of learning found in animals, especially that of humans. Thus, it is said that the computer, or machine, is learning as it adds to its store of knowledge when it creates observations and corrects them, all without the intervention of a human.

A number of recent advancements have made this process technologically feasible. One of them is the steady improvement in processing power and speed over the last two decades. Processor speeds have significantly increased year-after-year for quite some time and when coupled with the advent of multi-core chip designs, calculations can be completed with an alacrity that would have seemed impossible a decade ago. These improvements bode well for advancement in machine learning as it is virtually comprised of only mathematical operations.

An additional change in society that has aided the development of machine learning is the sheer ubiquity of the internet in recent years. With the explosion in the amount of easily accessible data available to developers and research scientists, training and tweaking a machine learning algorithm is much easier than it was in the past. This allows for more rapid improvement in machine learning models, which can be transferred to other models in completely separate domains.

Computer Vision

Within the field of computer vision, scientists are interested in seeing how computers can be trained to understand digital imagery and videos. As a result, they can describe the physical world using discrete quantitative information in digital media. From sampling analog data, it is clear that images are composed of pixels. These pixels represent intensity values laid out in a graph locally connected to their neighbors. Figure 2 below demonstrates how pixels can be laid out to represent the physical world.

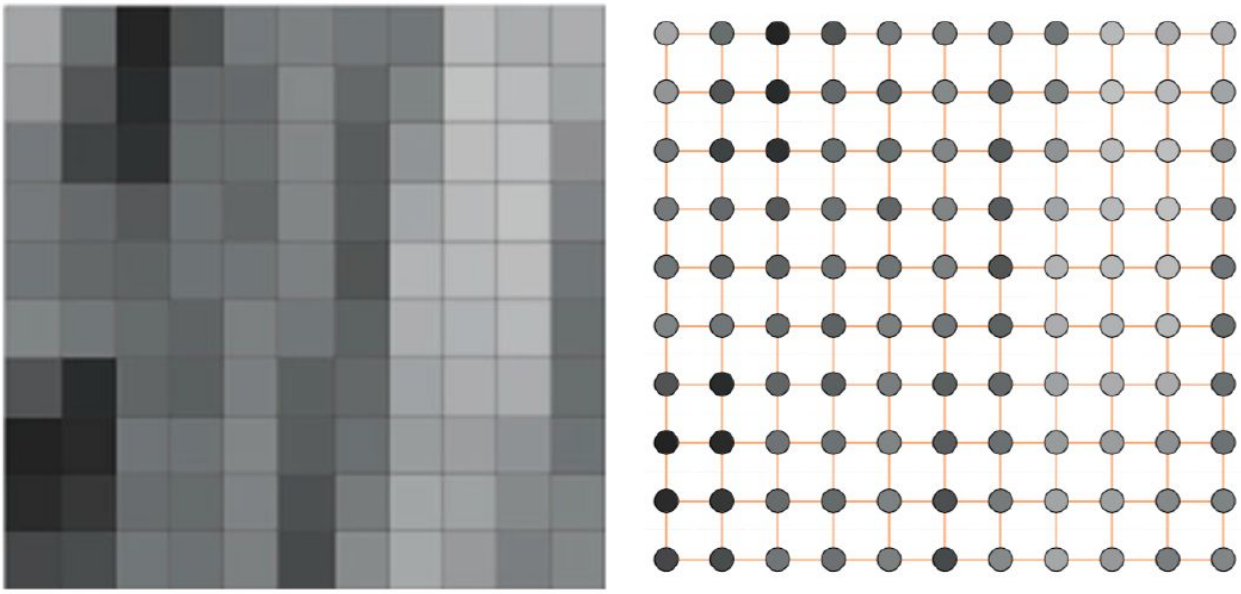


Figure 2 : An example showing the graph structure of pixels.

To create a color image, three channels are required; these channels represent the primary colors: red, blue and green. When these channels are stacked, a color representation similar to what the human eye sees can be created. By increasing the resolution of the image (the number of pixels in an image), clear and detailed images can be produced that appear in likeness to the physical world. A video is simply a sequence of images shown in a period of time that give the appearance of motion to the human eye.

By using the structure and information in images and videos, computers are tasked with extracting high-level understanding. This high-level understanding can then be applied as solutions to tasks such as object recognition, localization, tracking and segmentation.

Feature Extraction

In computer vision, features are used to recognize objects in images. These features are qualities of the object that should have correspondence across multiple views. Images of the same object rarely look the same as they can contain different properties, such as lighting and angles. In this sense, it is important to have invariant local features. These are features that do not change from translation, rotation, scale and other imaging qualities. In machine learning, feature engineering is an important step for the application of classification, which are used to identify objects and images. However, coming up with features is time consuming and requires expert knowledge.

Object Detection

Object detection is the process of examining features of an image to determine if an object is present in the image. Before the widespread use of neural networks, one of the most difficult aspects of object detection was feature engineering. This process is used to determine good quantitative data that could be used to evaluate if an object is present in the image. Examples of these features include the number of corners in an object, its probability density function, orientation and magnitude of pixels in an image. Quality features are affine invariant, resistant to movements, scale and rotation. However, neural networks learn their own features specific to the object. This aspect of neural networks removed a time consuming process and generalized the process to be easily applied to a variety of objects with minimal to no change.

Neural Networks

It is helpful to have a biological analogue for neural networks so that their artificial counterparts may be more easily explained. The brains of virtually every creature on Earth are comprised of cells called neurons, which process and transmit information from one another. Information is passed to the neuron through the use of electric and chemical signals. Neurons can be specialized to certain types of information, such as sensory information or the position of the body's extremities. All neurons are electrically excitable, and each neuron maintains a voltage gradient across its membrane that is influenced by ions of different charges. If and only if that voltage gradient passes a certain limit called the threshold potential, an electrochemical pulse called an action potential is propagated down the length of the neuron, which then influences other neurons that may be connected to the propagating neuron. Multiple neurons connected together form neural networks, and it is neural networks that allow animals, specifically humans, to learn from their environments and form knowledge.

Artificial neural networks attempt to emulate their biological counterparts in order to form inferences on data which can then be used in different ways, e.g. prediction, detection, etc. In this paradigm, neurons are represented by mathematical functions. These functions can take multiple inputs, each of which represent an action potential carrying information from another neuron in the network. Many of these inputs have separate and unique weights, which emulates the differing connection strengths between neurons. Artificial neurons simulate a voltage gradient by computing a summation of all inputs to the function. The sum is then passed to an activation function and will set the function's output to a certain number if and only if the sum has reached a certain threshold; otherwise, the output stays the same [3]. Figure 3.1 shows an example of the summation and activation; this characteristic is intended to mimic the all-or-nothing property of the action potential of a biological neuron. Figure 3.2 shows the relationship between the biological neuron and artificial neuron; the output of this neuron is then used as the input to another neuron, and the process repeats itself.

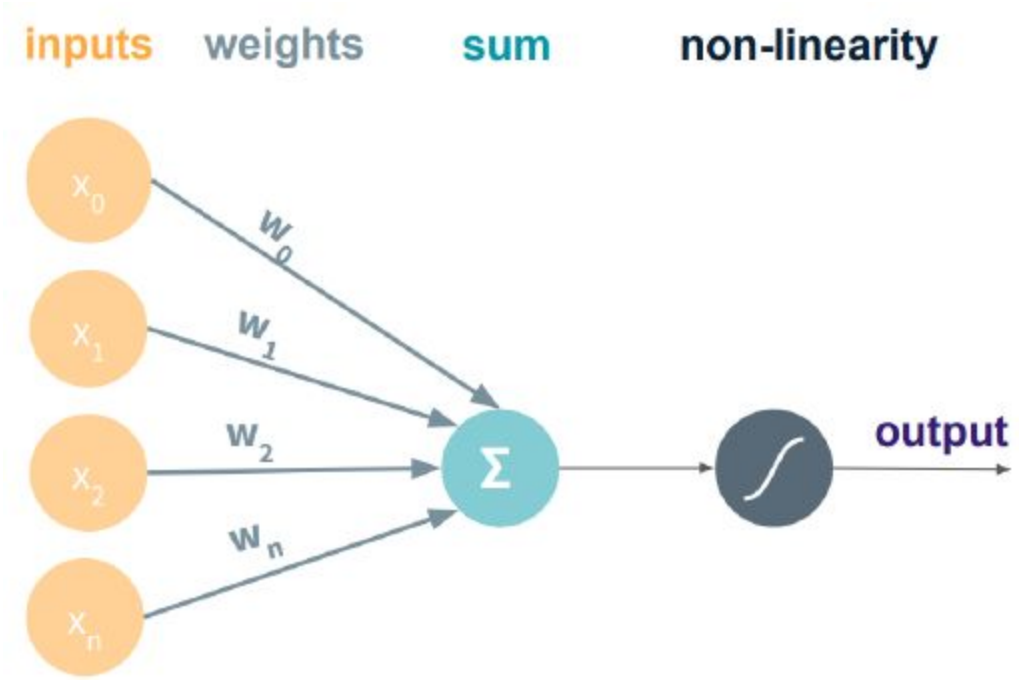


Figure 3.1: This shows how previous neurons are summed into a single neuron which feeds into an activation function to give it a non-linear representation.

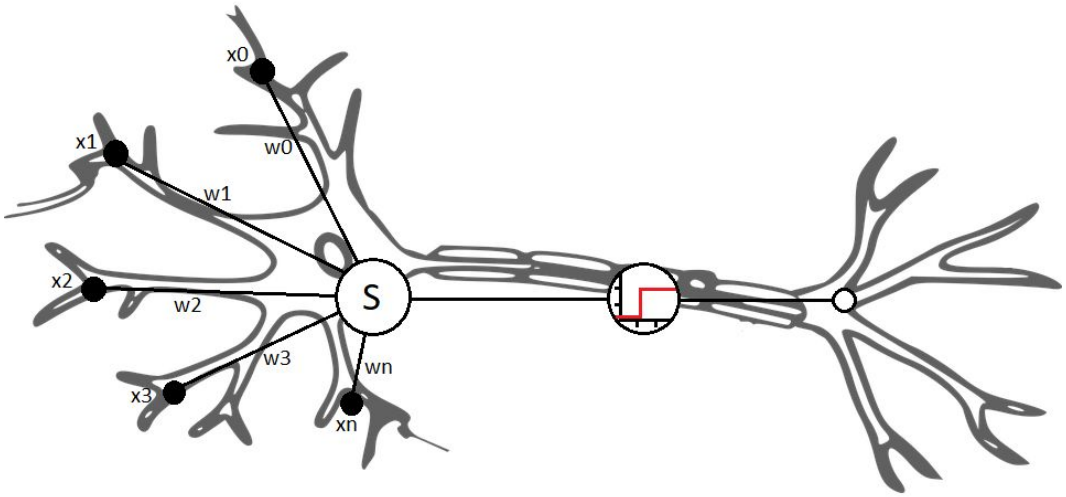


Figure 3.2: This illustration shows the relationship between a real neuron and an artificial neuron.

Biological brains are three-dimensional systems comprised of layers upon layers of neurons connected to each other. Artificial neural networks replicate this property by layering groups of functions together that perform different transformations upon the input information, e.g. convolution, pooling, etc. The first few layers of a network may detect primitive information about the input data. The outputs from these layers are then passed to the later layers, which may use the prior information to detect higher-level

information and make more complex inferences. Many times, a neural network will make incorrect inferences about the input data it is given, which is why the network must be trained. In this situation, a process called backpropagation occurs in which the error contribution for each neuron is calculated and then the weights for each respective neuron is updated accordingly. This enables the network to become more accurate over time as it seeks to minimize the system's total error in comparison to the training data [3].

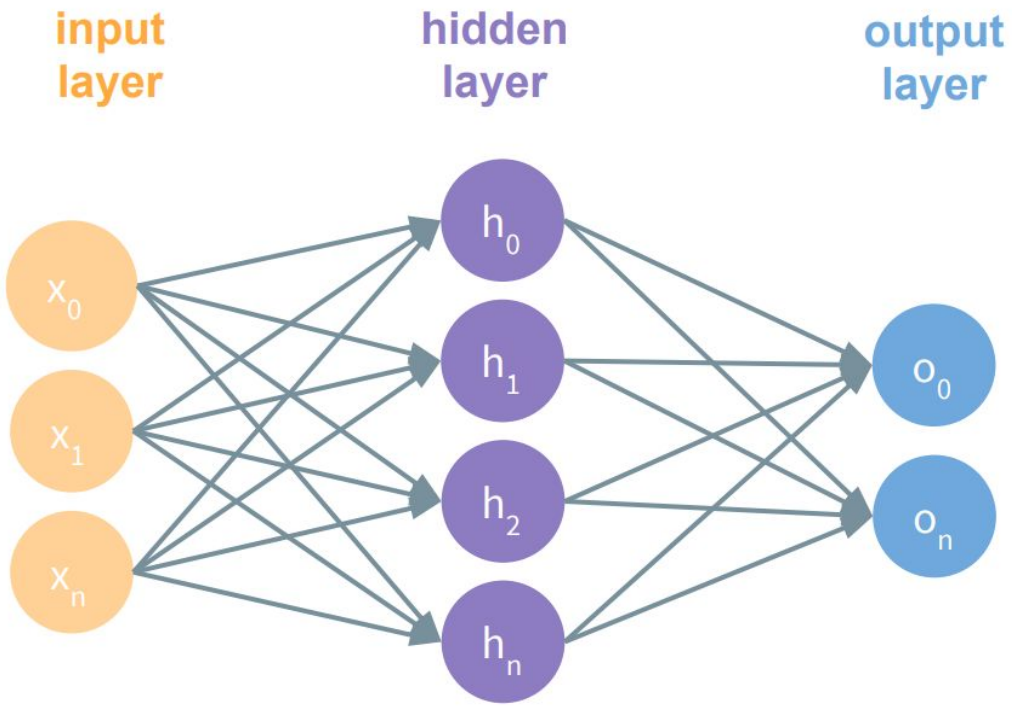


Figure 3.3: High-level photo of neural network connections.

Activation Functions

The activation function of a neural network allows the network to create non-linear representations rather than the weights of neurons creating a linear combination of weights. If the network were comprised of only linear combinations, it would be limited in representations it could create. Figure 4 shows three of the most popular activation functions for neurons.

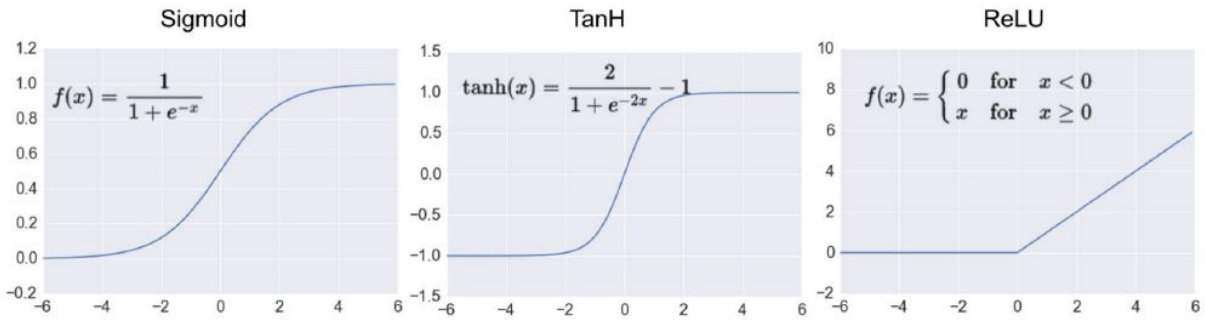


Figure 4: Graphs for the Sigmoid, TanH, and ReLU activation functions

ReLUs, rectified linear units, have become incredibly popular as they has a low computational cost as well as the fact that older activation functions have become less efficient because of an issue called the vanishing gradient. As Figure 4 shows, both the sigmoid and tanh functions become flat as they approach positive and negative infinity; this effectively removes the gradient as there is no change in the output of the function. In contrast, ReLUs allows faster convergence of gradient descent. ReLUs are a piecewise linear function, so it might not appear to be a good choice at first glance. However, in a deep network of many layers, the piecewise function can still create non-linear representations.

Backpropagation

The overall accuracy of predictions made by neural networks can be verified by calculating loss, a value that measures how well a model fits a training set. By minimizing loss, the network can increase its ability to classify data since the model is closer to the ground truths of supervised learning. This can be achieved through computation of the gradient; by using the chain-rule to follow the gradients of all the layers in reverse order, we can diminish the loss. This process is called backpropagation and is essentially how the neural network learns and improves.

Gradient Descent

Gradient descent is a process in which the parameters of a function are adjusted in such a way that the loss is minimized. However, when gradient values are calculated, they are minimized with a value called the learning rate. This reduced gradient allows the neural network to update its values in smaller increments as opposed to a single large jump. The purpose of this practice is to prevent overfitting and to generalize data rather than memorizing it.

Figure 5 shows an example of a loss function descending with the gradient. The goal of the backpropagation is to reach the global minimum of the loss. In neural networks and machine learning, there are many inputs and variables that can lead to making the space complex and high-dimensional. This leads to the problem of missing the global

minimum; the gradient may get stuck in a local minima if the learning rate is too low while the gradient may completely miss the global minimum if the rate is too high.

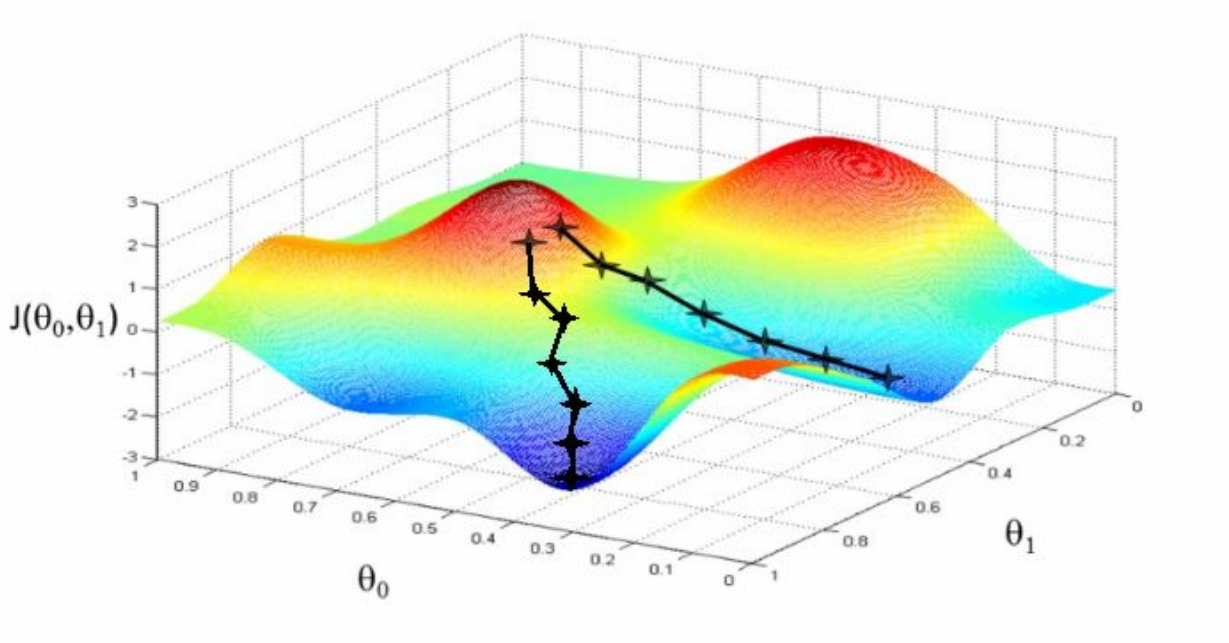


Figure 5

There are two solutions to help with gradient descent. The first is to use minibatches. This allows for parallel computation which gives a significant speed boost to the computation. The other is an adaptive learning rate, such as Adam optimizer. These methods change the learning rate based on mean average and gradients that allow for the learning rate to adjust itself so it isn't too high or too low [4].

Neural Network Competitor: Support Vector Machines

Support vector machines (SVMs) are a supervised form of classification and regression of data. Achieving common use in the 1990s, they are the other premier method of classifying images. SVMs classify data by moving it to a higher dimension and separating the data using a hyperplane. The location of the hyperplane is calculated to maximize the margins between the different classes or features in a dataset by using support vectors. Figure 6 shows what these support vectors and hyperplane look like in a low dimension.

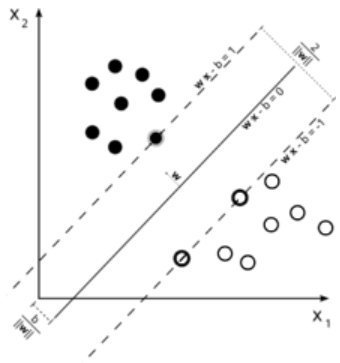


Figure 6: A picture of a hyperplane and its margins

Support vector machines have two major issues when compared to convolutional neural networks. For complex data sets, vector machines are slow with evaluation. The second issue with the vector machines are that features must be found and created for the data set. Convolutional neural networks have the advantage of being able to extract their own features while being as efficient, if not more so, in image classification.

Convolutional Neural Networks

Convolutional neural networks (CNNs) have recently led to incredible breakthroughs on a variety of classification problems that involve spatial data. The simplicity and elegance of the convolutional filtering process makes them perfect for handling problems with any form of spatial 2D or 3D data, e.g. images, sound, video, or even text. A simple way to check this type of spatial data is to verify whether a dataset is equally as useful when the rows and columns of its matrices are swapped. In such cases, a convolutional neural network should not be used.

Convolutional neural networks are similar to standard neural networks; however, CNNs' architecture assumes that the input are images, which helps network process large matrices of spatial data. The CNN creates feature maps and shrinks the data to retain only the most important regions, as to exponentially speed up the training and testing periods. This increase in performance is due to what is known as locally connected layers, where each neuron is only connected to a local group in the previous layer; both convolutional and pooling layers are set up this way. This is in contrast to the fully connected layer, where every neuron is connected to another in the next layer [3, 5, 6].

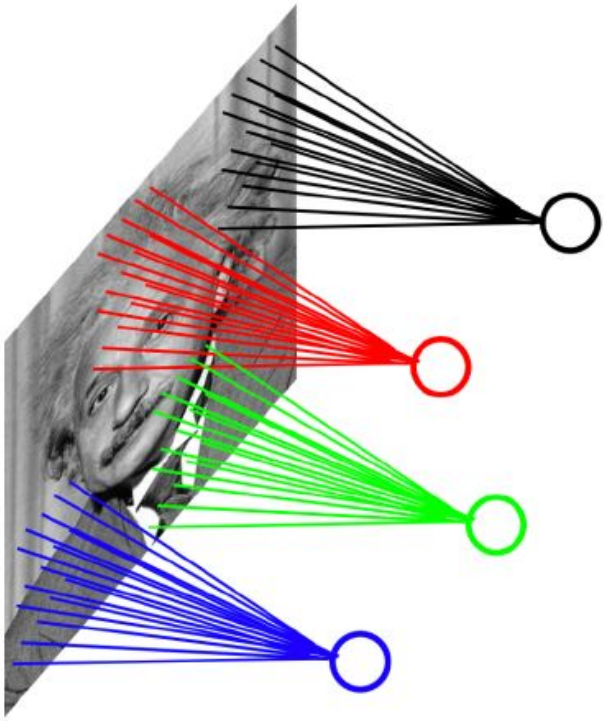


Figure 7: Filters convolving over an image

Convolutional Layer

The convolutional layer is responsible for the creating feature maps and calculating a neighborhood of pixels within a single pixel, also known as a filter. Each filter is responsible for extracting different features of an image and thus, each perform slightly different calculations. These feature maps are usually passed into a couple of convolutional layers before being fed into the fully connected neural network to create more complex feature maps.

Pooling Layer

The pooling layer is responsible for decreasing space while maintaining extracted features within data. This is accomplished by acquiring a neighborhood of pixels and mapping them to a single pixel in the next layer. One of two methods are usually used: max pooling or average pooling. Max pooling is accomplished by selecting the pixel with the highest intensity, whereas average pooling is accomplished by averaging a group of pixels and passing that value to a single pixel in the next layer.

CNN Architecture

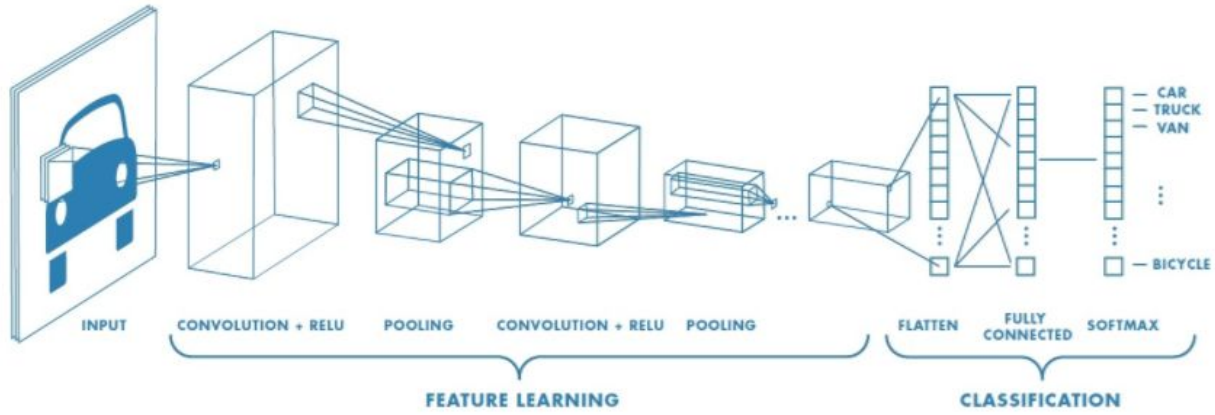


Figure 8: Convolutional Neural Network Architecture [5].

Input Layer

- When a color image is passed into the neural network, it is represented as a three-dimensional array of pixel values whose length and height dimensions match the corresponding dimensions of the image. The third dimension of the array is the depth, which contains the RGB values of the pixel located at that point in the original image [5].

Convolution + ReLU

- A filter (an array that contains numbers called weights) is convolved around the input layer; this is essentially a combination of adding and doing element-wise multiplication between the filter values and the pixel values of the original image.
- The ReLU component is a unit in which the rectifier is used. The rectifier is an activation function defined as the positive part of its argument: $f(x) = x^+ = max(0,x)$, where x is the input to a neuron.
- In a typical CNN setup, a layer with an activation function, such as an ReLU, will always follow a convolutional layer. This helps the network build up from low to high level features in the image as the input to each successive convolutional layer will be the activation map from the preceding layer [4].

Pooling

- A pooling layer will use a filter of a certain size and a stride of the same length; stride is the amount of units that the filter will shift at a time. It convolves around the input and outputs the corresponding value in each subregion through which the filter travels, e.g. maximum value for a max pooling layer. This is done to reduce the spatial dimensions of the input volume, which will decrease the cost of resources used by the network.

Dropout or Regularization

- A dropout layer will randomly disable neurons by setting their activations to zero, This forces the network to be redundant and aims to prevent overfitting by forcing the network to learn many variations of the training data.

Flattening

- The flattening action will turn a multidimensional array into a long one-dimensional array. It does this by starting at the first row, selecting the next row, and appending that to the end of the first row. It continues until the entire array has been transformed.

Fully Connected Layer

- The fully connected layer receives an input volume and outputs a vector of *N* dimensions where *N* is the number of classes from which the model can choose to classify the original input. The layer aims to determine which higher level features correlate to a certain class and also has weights to help ensure that different classes receive the correct probabilities.

Softmax Function

- The softmax function is a form of the logistic function in which a *K*-dimensional vector of arbitrary real values is “squashed” into an equal length vector of real values that range between 0 and 1. This can be used to show a probability distribution across *K* different possible classes. This function is typically used as the final layer of the network as it returns a summation of all elements at each *K* elements.

Notable Network Architectures vs. Standard CNN

There have been a number of notable breakthroughs regarding the efficacy of convolutional neural networks in the last few years. In this section, some of these breakthroughs are listed along with the characteristics that differentiate them from standard convolutional neural networks.

AlexNet

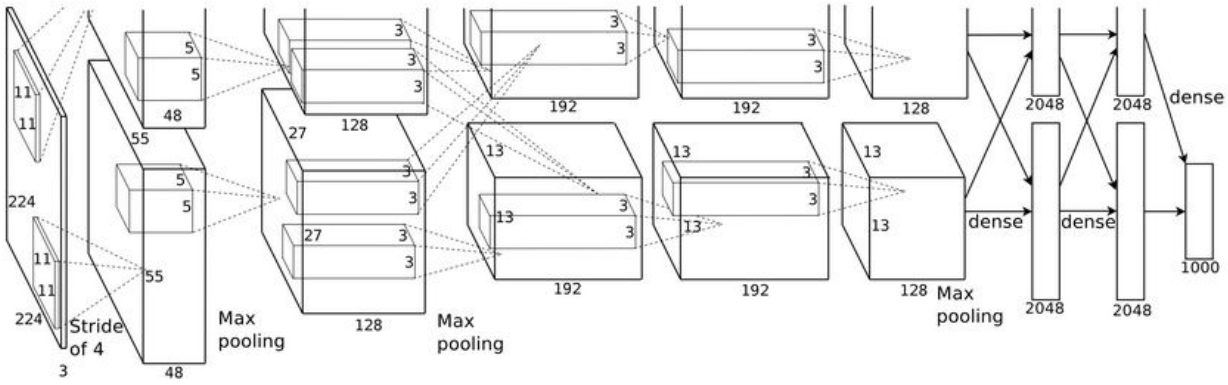


Figure 9: 16+ Layer AlexNet Architecture[7].

- Alex Krizhevsky at Google Brain introduced better nonlinearity in the network with the ReLU activation function; this proved to be efficient for gradient propagation.
- Introduced the concept of dropout as regularization: from a representation point of view, if the network is forced to “forget” things at random, it is forced to become redundant. This prevents overfitting in the network.
- Introduced data augmentation: when fed to the network, images are shown after being affected with random operations, e.g. translation, rotation, crop. This forces the network to be more aware of the attributes of the images rather than the images themselves.
- AlexNets went deeper. They stacked more convolutional layers before pooling operations. Consequently, the representation captures consequently finer features that reveal to be useful for classification [7].

input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 10: Top-down VGG network architecture (very deep).

VGG networks are simply convolutional neural networks that go *very* deep. VGG was researched and developed to solve the problem of designing convolutional neural networks and defining their hyperparameters. VGG shows that deeper networks give better results, and simple filter sizes for max pooling and convolution can achieve very good accuracy and performance.

Google Net

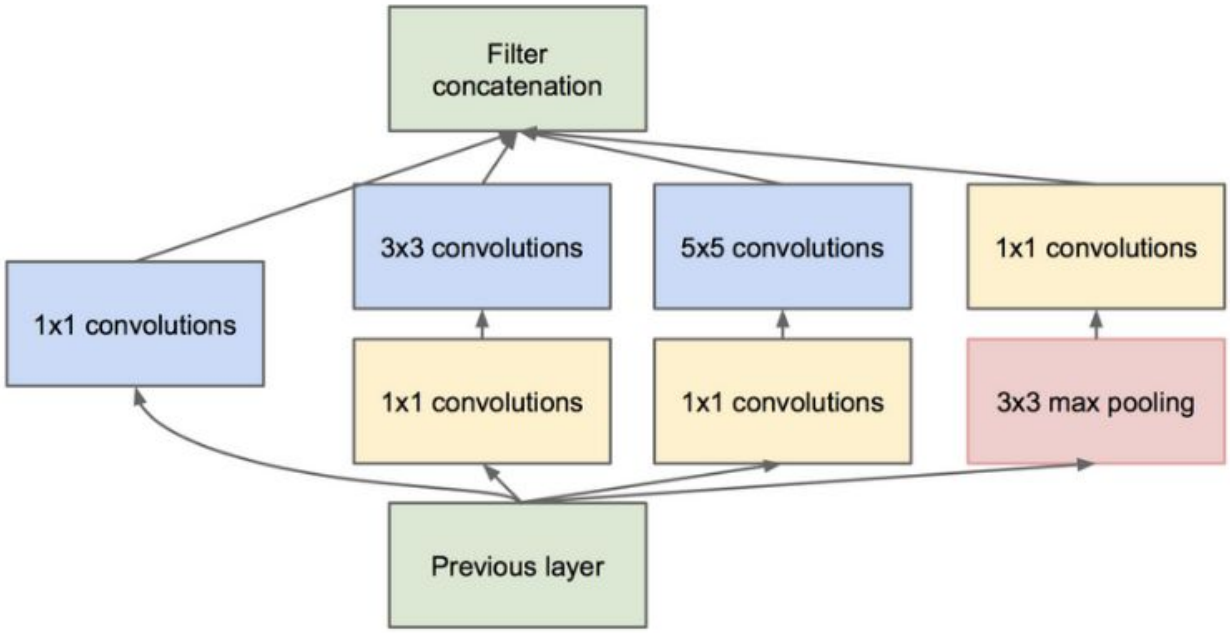


Figure 11: Multiple convolutional and stacking for GoogleNets [8].

Google Nets ‘go deeper with convolution’ by convolving the same input with different filter sizes and then concatenating them together. This allows the network to take advantage of multi-level feature extraction at each step. For example, if given an input containing human subjects, general features like faces and hands can be extracted with the 5x5 filters and local features like teeth and eyes can be extracted with the 3x3 filters [8].

ResNet

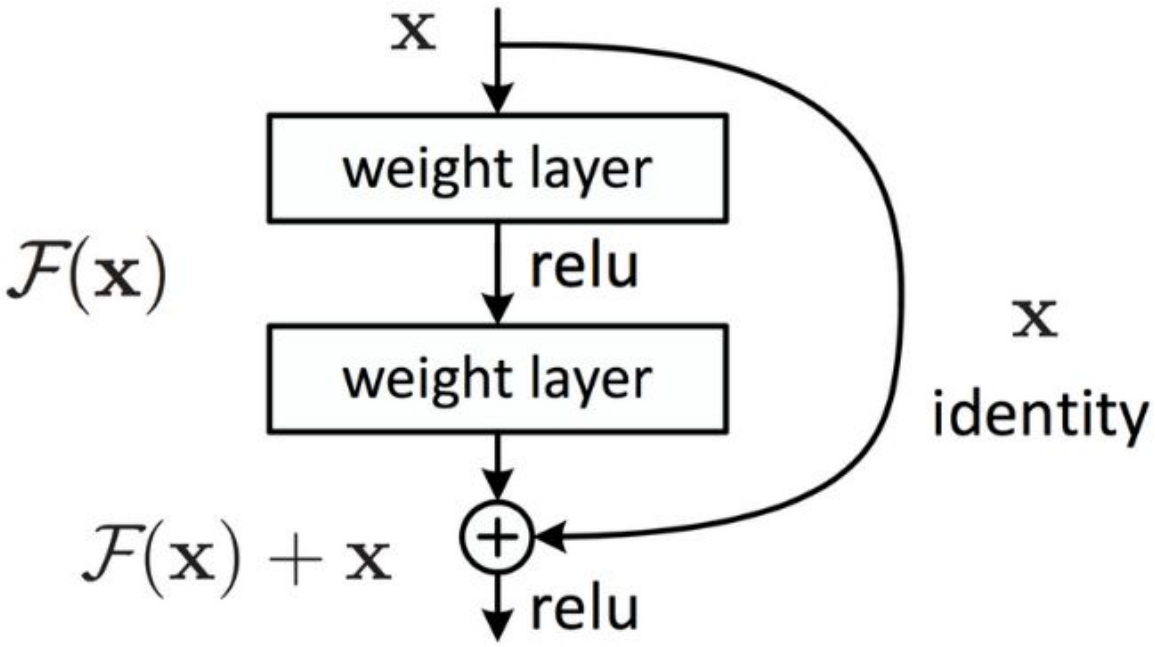


Figure 12: Element-wise matrix addition on any weighted layer, i.e 2d convolution or fully connected layers.

- At some point, we realize that stacking more layers does not lead to better performance. In fact, the exact opposite occurs. But why is that? The Gradient.
- Every two layers, there is an identity mapping via an element-wise addition. This proved to be very helpful for gradient propagation, as the error can be back-propagated through multiple paths.
- This helps to combine different levels of features at each step of the network.

Dense Net

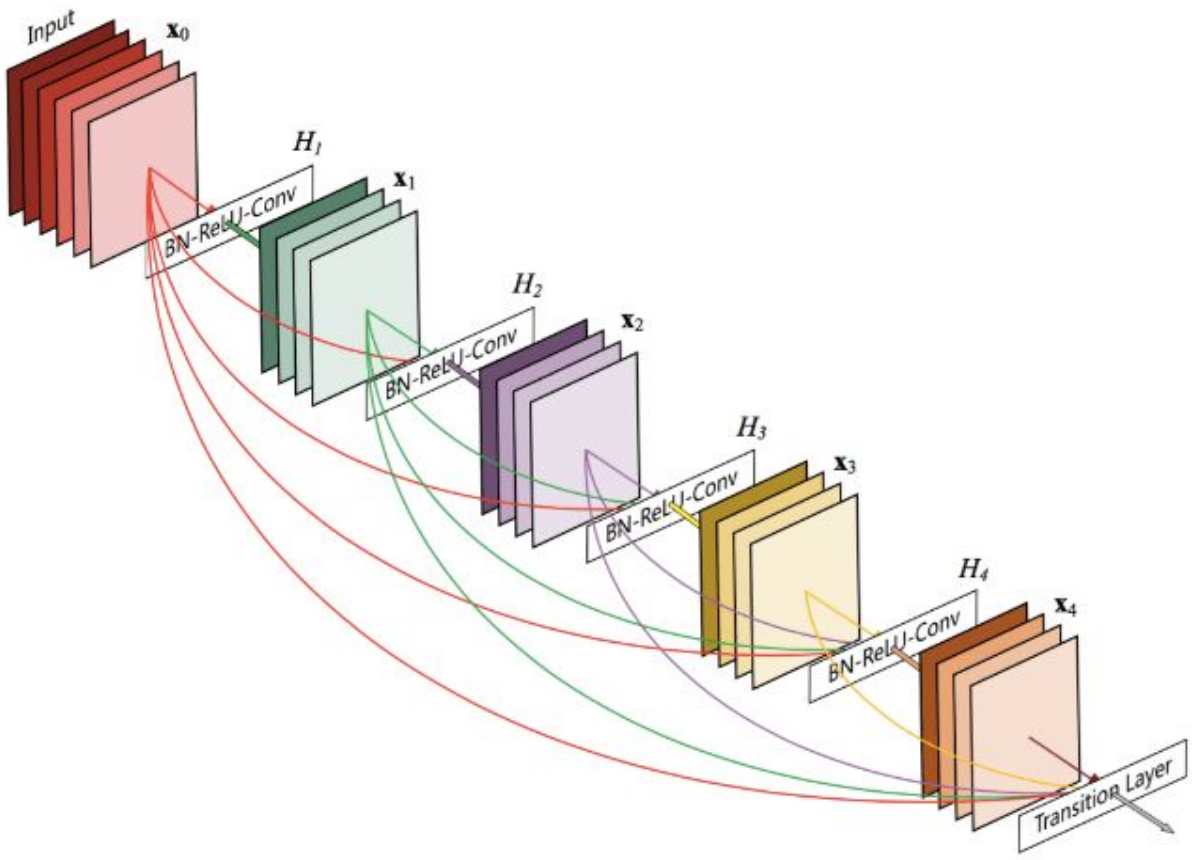


Figure 13: DenseNet architecture with an emphasis on blocks of layers [9].

DenseNets are proposed as networks in which entire blocks of layers connected to one another. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters [9].

Patterns for CNN optimizations

1. Networks are designed to be deeper and deeper.
2. Computational tricks (ReLU, dropout, batch normalization) have been also introduced alongside them and had a significant impact in improving performance.
3. Increasing use of connections between the layers of the network, which helps for producing diverse features and revealed to be useful for gradient propagation.

Drawbacks of CNNs

Geoffrey Hinton, who popularized backpropagation for multi-layer neural networks and created capsule networks, outlined two major reasons why convolutional neural networks are doomed during a talk at Massachusetts Institute of Technology in 2014.

According to Hinton, the reasons are as follows:

1) Sub-sampling loses the precise spatial relationships between higher-level parts, such as a nose and a mouth. The precise spatial relationships are needed for object classifications. However, overlapping the sub-sampling pools mitigates this.

2) They cannot extrapolate their understanding of geometric relationships to radically new viewpoints, which is why it's common practice to flip and rotate images when training CNNs [8].

Capsule Networks - The most cutting edge solution to the CNN problem

On 7 November 2017, Geoffrey Hinton published his paper on capsule networks, which have provided the highest classification accuracy to date on the MNIST dataset [7]. However, since stability analysis and further testing hasn't been done, there is no way to know how the asymptotic the behaviour of the layers will be over *n* iterations and how stable capsule networks will be for attacking difficult learning problems.

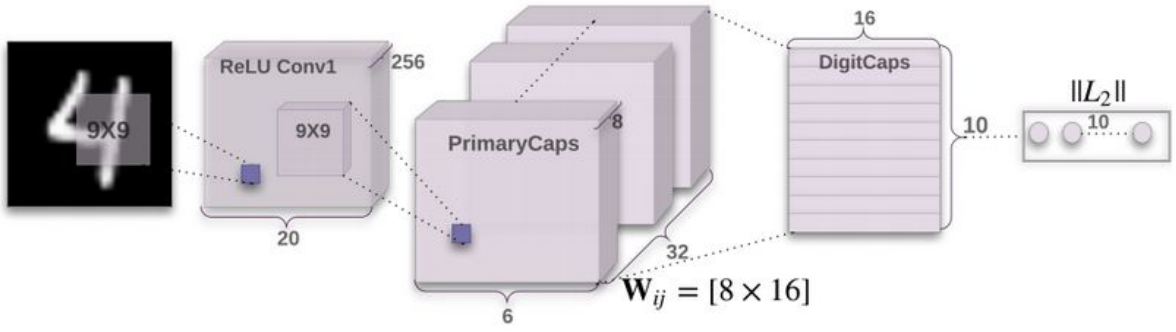


Figure 14: Capsule Network Architecture [10].

What are capsules?

- The capsule is a neural network architecture where a typical layer is modified to contain sub-structures [10, 11].
- The typical layer of units becomes a layer of capsules.
- Instead of making a neural network "deeper" in height, it makes it deeper in nesting or inner structure [11, 12].
- The model is robust for training images in different affine transformations e.g. reflection, rotation, scaling and translation.

How does it differ from traditional convolutional neural networks?

- CNNs cannot handle rotation at all; if they are trained on objects in one orientation, they will have trouble when the orientation is changed.
- Pooling gives some translational invariance in much deeper layers but only in a crude way.
- The human brain must achieve translational invariance in a much better way
- Hinton posits that the brain has modules he calls *capsules*, which are particularly good at handling different types of visual stimulus and encoding things like pose. The brain must have a mechanism for “routing” low level visual information to what it believes is the best capsule for handling it [7].
- According to Hinton, CNNs do routing by pooling. Pooling was introduced to reduce redundancy of representation and reduce the number of parameters, recognizing that precise location is not important for object detection [7].
- But pooling does routing in a very crude way. For instance, max pooling just picks the neuron with the highest activation, not the one that is most likely relevant to the task at hand.

Siamese Neural Networks

Siamese neural networks are two conjoined neural networks.These networks are specifically conjoined at their outputs and separated at every other point. The purpose of this structure is to allow parallel computation of classification. As such, one can use a pair of siamese neural networks to make two separate inferences on the same input and treat the input according to the outputs of these networks.

Region Proposal Network

Region proposal networks are responsible for estimating areas of an image that may contain objects. It serves as a form of subject localization that can help improve the recognition and classification of objects. They are another form of fully connected layers, similar to the hidden layers in a standard neural network [13].

Once the convolutional layer has produced feature maps, a spatial window, or a kernel, is used to slide across the image and create a set number of different regions. These regions are called anchors since it is locked within the center of the spatial window.

Faster Region-Based Convolutional Neural Network

Faster R-CNNs is currently the state-of-the-art method for classifying and detecting multiple objects within the same image. It is an improved implementation of Fast R-CNN. These region-based convolutional neural networks have an additional network that specializes in the localization of an object. This allows for a network to classify multiple objects in a given image. The advantage of Faster R-CNN over Fast R-CNN is that it improves the region proposal method, which was previously the bottleneck for performance on Fast R-CNN [13, 14].

In this process, the R-CNN begins by processing the image with convolutions from a convolutional layer. This produces a feature map that is then reshaped into a standard size and fed into the region proposal network. The network will then give proposals on possible locations of objects, known as region of interest pooling. These regions are then fed into the fully connected hidden layers of the neural network to classify the given regions [14]. The regions can be immediately put into the hidden layers because they have already been filtered and pooled by the convolutional layers.

Image Segmentation

In essence, image segmentation is the process of dividing an image into multiple sets of pixels (super-pixels) in order to change the representation of the image into something that is easier to analyze. More concisely, every pixel in an image is labeled as one of the sets such that all pixels with the same label are said to have the same properties. After the process finishes, the result is an image that is collectively covered by super-pixels [15]. Image segmentation is usually used to determine boundaries in an image, which in turn helps to locate objects in the image. There are a number of ways to complete image segmentation; however, the most common methods are thresholding, clustering, active contours, and energy minimization.

Thresholding is the simplest method of image segmentation. It often involves replacing any pixel of image intensity I with a black pixel if I is less than a threshold T ($I < T$), and a white pixel if I is greater than T ($I > T$). The user can set this threshold, but in order to achieve the optimal results, the program should automatically determine the threshold value to use. Many of the most popular thresholding methods are based on entropy, histogram shape, or background/foreground clustering [14, 15].

Clustering does not make use of one specific algorithm as the appropriate clustering algorithm depends on the original input (or dataset) and how the output of the cluster analysis will be used. Thus, it is said that clustering is an iterative process in which the model parameters are adjusted until the result reaches the desired characteristics. There are many different cluster models, each of which have their own unique algorithms. This is because there are several unique definitions of what constitutes a cluster. Common cluster models include centroid and density models. In a centroid-based model, clusters are represented by a central vector. K-means clustering is a common algorithm used in this model, which states one should find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized. In a density-based model, clusters are defined as areas of higher density than the rest of the data set. In the most popular method in this model (DBSCAN), clusters are created by connecting points within certain distance thresholds, but only when they have a minimum number of other points within the threshold [16].

Active contour models are designed to solve problems in which the approximate shape of the boundary of an object is known. A deformable, energy-minimizing spline (piecewise polynomial function) known as a snake is used to delineate an outline of an object. Since snakes are deformable, they can adapt to noise and differences in motion tracking. They also adaptively search for the minimum state automatically and are sensitive to scale as Gaussian smoothing is incorporated into the image energy function. Additionally, active contour models can find subjective contours by adapting to missing boundary information [16].

Mask Region-Based Convolutional Neural Network

Fast R-CNN introduced bounding boxes and object localization to convolutional neural networks. With Mask Region-Based Convolutional Neural Networks, or Mask R-CNN, semantic segmentation is also brought into the network. The goal is to not only label a region that surround detected objects but also to label each individual pixel in the image based off the objects. Figure 15 shows an example of the outputs of the network [14].

Segmentation is brought into the system by adding an additional branch into the region proposal network. This network already contained two branches, one for outputting bounding box coordinates and the other predicting the likelihood of an object in that region. The third branch, a segmentation proposal system, is another fully connected network. It outputs a binary mask for each proposed region, denoting whether or not a pixel is part of an object. The labels come from the classification branch which will let the segmentation branch know which labels to use and how many. This helps prevent issues when multiple objects are in a single region of interest.

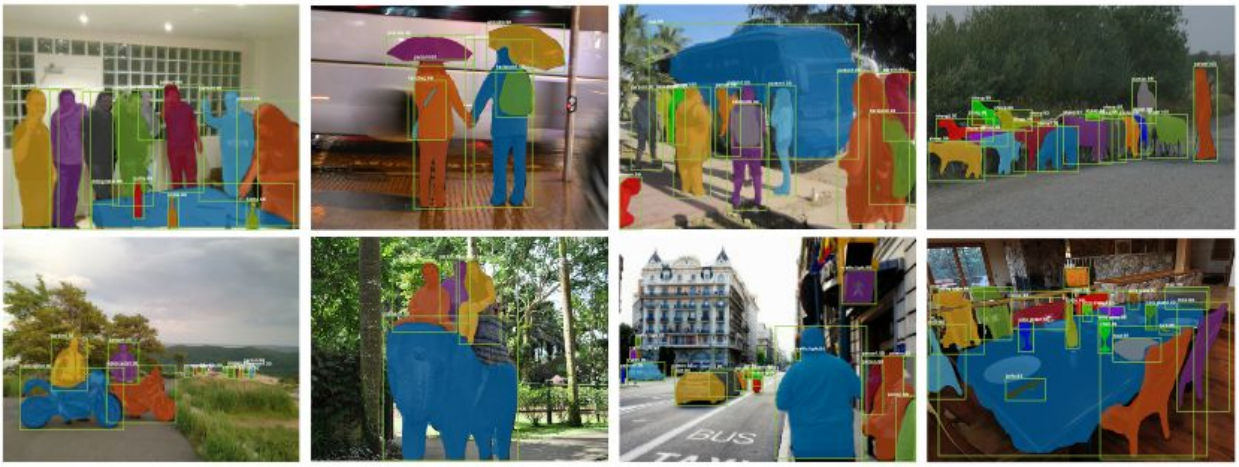


Figure 15: Mask R-CNN, image segmentation with region proposals

Supervised Learning and Transfer Learning

Transfer learning, also known as inductive transfer, is the act of using stored knowledge that was created in the solution of one problem to solve a different but related problem. In order to better understand this process, it may be beneficial to learn about supervised learning. This will help the reader better understand the significance of being able to use transfer learning [17].

In supervised learning, a function is inferred from labelled training data, which typically contains sets of training data. Given a pair of a training example and a desired output value, a supervised learning algorithm can create inferences about relationships between that particular example and the value. When this process is completed over a large number of examples, an optimal algorithm should be able to generate the correct output value for unique, unseen values. In order for all of this to be done correctly, a general process is typically followed.

First, one must decide what kind of training examples to be used as it has to relate to the domain of the problem, e.g. bird pictures for bird detection. Following this step, a training set must be procured, whether that be through creation of one's own dataset or finding a dataset that matches with the type of problem to be solved. Then, one must determine how to represent the input of the learned function; care must be taken to ensure that the number of features contains enough information for the algorithm to correctly predict the output, but not so many features that the information becomes sparsely distributed throughout memory. After that, the algorithm itself is designed in order to optimally use as much information as possible that is offered by the training data and then the algorithm is trained on the data. Finally, the accuracy of the learned function is assessed and adjustments are made if needed. It is after this point that the supervised learning model is able to be used on non-training data.

This approach works quite well when there is an abundance of data to properly train the model. However, there are many obstacles to procuring enough data to do such a thing. There may be no available dataset that correctly fit the problem, or if there are datasets available, there may not be enough data inside the sets to train the model to the desired accuracy. Also, there may be a temporal or financial obstacle to collecting training examples as it may take years to collect enough data to properly train some models, and the average person will be unable to spend all of their time collecting data. In these situations, transfer learning would be extremely beneficial, given the accuracy and efficacy of some models, such as ImageNet.

A simple definition on the theory of transfer learning may be beneficial. The basis of the theory relies on domains and tasks both inside and outside of the domain. A domain D consists of a feature space and a probability distribution over the feature space. Given a domain, a task T consists of a label space and a conditional probability distribution that is usually learned from the training data. Thus, given a source domain D_S and a corresponding source task T_S along with a target domain D_T and task T_T, one can use

the information gained from D_S or T_S (where $D_S \neq D_T$ or $T_S \neq T_T$) to determine the target conditional probability distribution in the target domain D_T [17].

There are a number of ways that transfer learning can be applied to augment or improve a model; the main ones at the time of this writing are using pre-trained convolutional neural network (CNN) features, learning domain-independent representations, making representations more similar, and confusing domains.

The most commonly used method of transfer learning is using features calculated in a high-accuracy CNN and including them in a new model, which would enable training using those high-accuracy features. In this process, the pre-trained parameters are usually frozen (kept constant) so that further training on the new model does not change the weights of the preceding layers, which prevents the loss of knowledge from the higher-accuracy network [17]. Doing this allows the model to start with virtually the same accuracy of the pre-trained model and allows for improvements upon the viability of the model instead of spending time and resources to get it to a minimally viable level.

The next two main methods for doing transfer learning on models involve representations. The first method is to learn domain-independent representations of data and its features. Instead of creating representations of data that are unique to a particular domain and are unusable for solving problems in other domains, one can create representations that do not vary between domains. This is more efficient as this type of representation will enable re-use through many domains and this is much less expensive than creating representations that cover all possible domains. The second method involves making the representations of data more similar to each other. When comparing two domains, it would be beneficial to make representations as similar as possible so that the model can focus on the commonalities between domains instead of the domain-dependent aspects of the data. This can be done by applying a pre-processing step to the data representation(s) and then using the processed representation(s) to train the model.

The final and perhaps the newest way to apply transfer learning to a model is through domain confusion. This process involves augmenting the model with a loss function (domain confusion loss) that encourages the model to “confuse” or minimize domain-specific information between the two domains. The domain confusion loss function is similar to a regular classification loss in that the model is still attempting to determine the domain of the input example. However, it is different in that gradients flow from the network to the loss as opposed to the regular version in which they flow from the loss to the network.

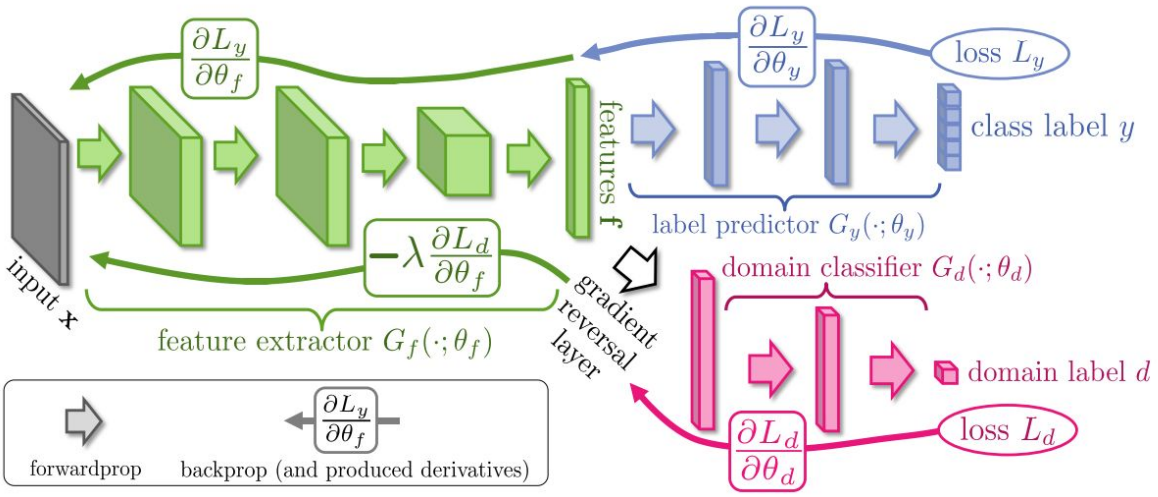


Figure 16: Transfer learning domain confusion

This causes the model to maximize the error in differentiating between domains instead of minimizing it, which in turn allows the model to learn representations that are less likely to be unique to each domain and minimize the error in the original objective of providing class labels. A comparison of the domain classifier score between models with and without an implementation of domain confusion is show below [17].

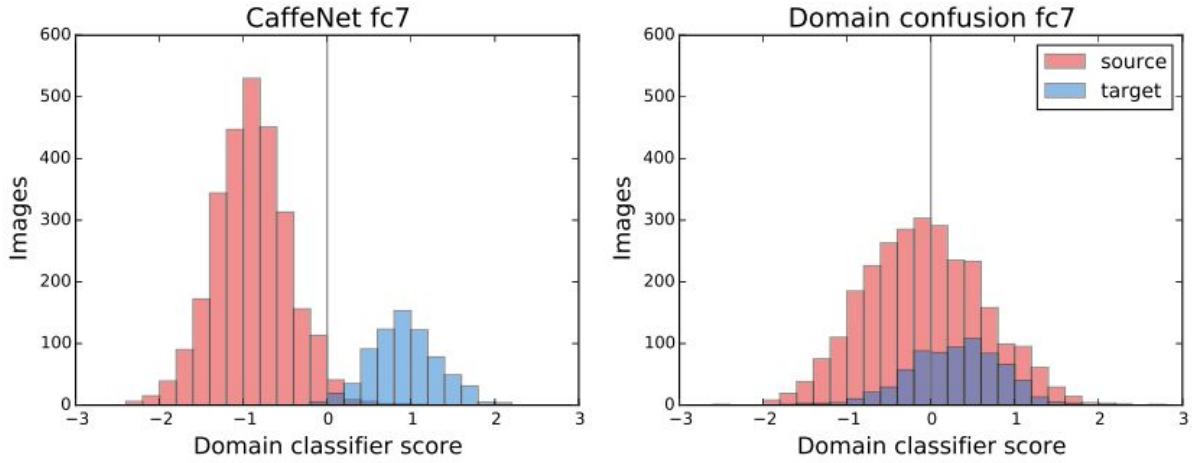


Figure 17: The effects of domain confusion

Generative Adversarial Network

The generative adversarial network is a model of two neural networks, each with their own task and that compete against one another. There is the discriminator, which is responsible for classifying whether an image is real or fake. A real image would be considered an image that was taken of the physical world while a fake image would be one created from scratch that does not exist in the real world. The other network is the generator, which is responsible for creating images intended to fool the discriminator. This competition is why the network is known as an adversarial one. Figure 18 shows how a standard GAN is set up [18].

The generator uses random noise to sample a probability density function to create a single instance of a class' image. These probability density functions are simply the frequency at which pixel intensity values occur for given object classes [18]. The generator's architecture is usually the inverse of the discriminators.

During training, only one network is updated at a time: first, the discriminator, and then the generator. The goal of training is to reach the Nash Equilibrium, where each network is in their optimal state for fooling and detecting one another. However, gradient descent does not guarantee that the Nash Equilibrium will be reached. However, an issue that affects generators is that they could have a mode collapse in which the output of the produced images lack diversity with different random inputs [19].

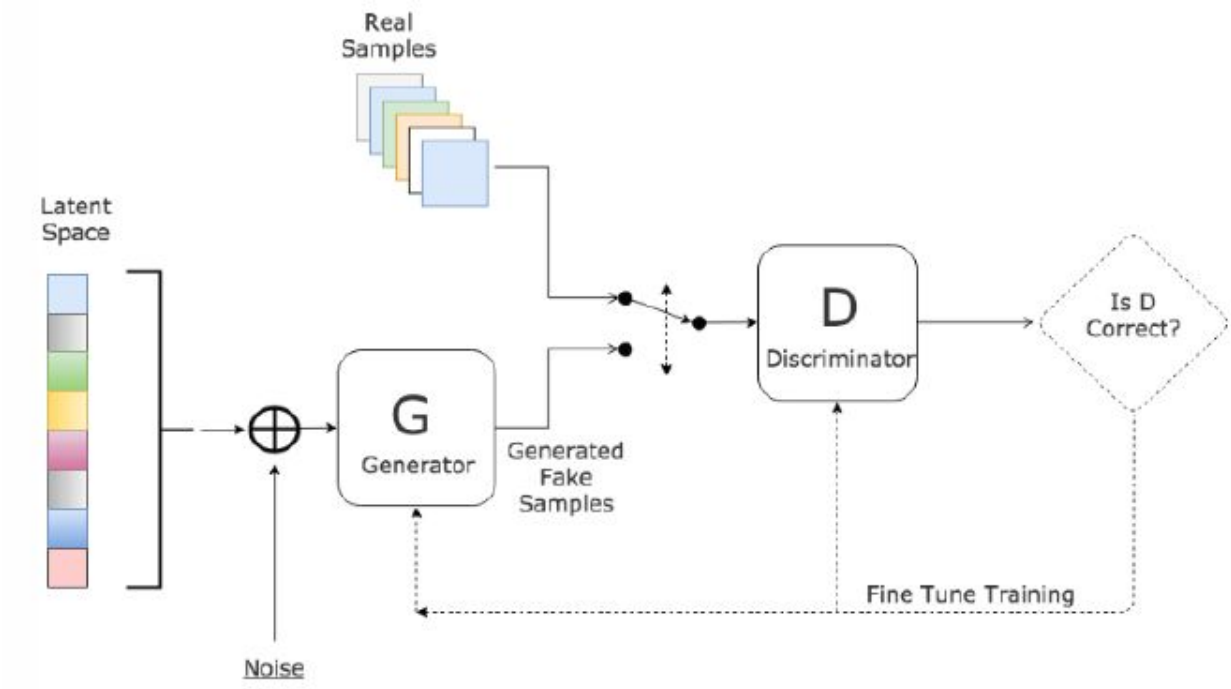


Figure 18: Architecture of a standard generative adversarial network

Unsupervised Learning and Overfitting

GANs are not only used for image generation but also for improving image classification. Since a generator never sees the original training data, it is robust against overfitting. This is considered a form of unsupervised training, where the network doesn't have labeled images fed to it. If a GAN is trained well enough, then the images it produces can be used as actually training data for a classification network.

Generator Architecture

The generator's architecture is related to convolutional neural networks. However, to build an image, the generator works in reverse compared to an image classifier, which does not. As shown in figure 19, the generator begins with random noise input and then feeds it into a fully connected layer. Afterwards, upsampling and convolutions are performed. Upsampling increases the dimensions of the image until the desired dimensions are reached while maintaining features.

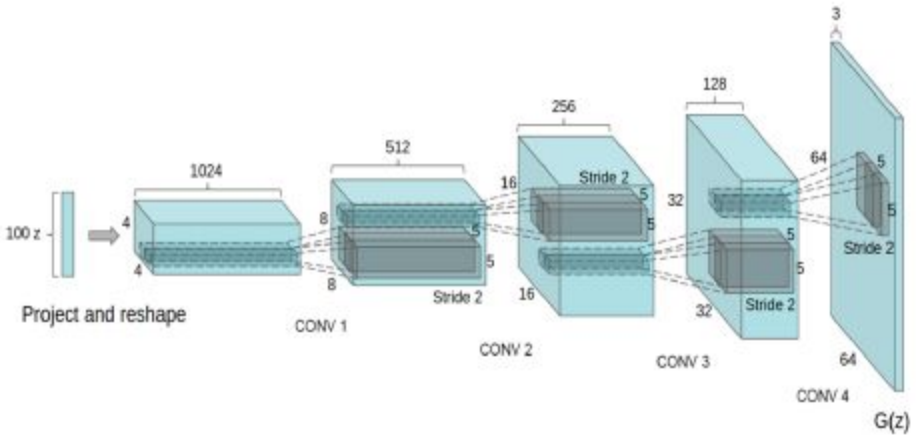
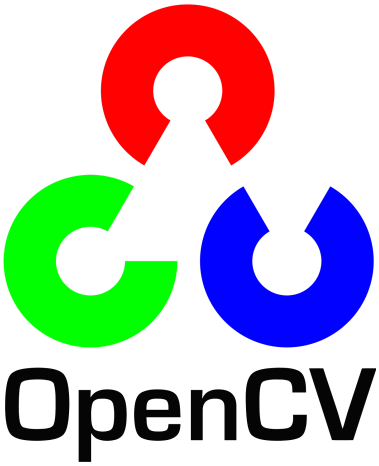


Figure 19: Generator

OpenCV



OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library for extracting and processing meaningful data from images. It boasts more than 2500 algorithms that are designed for efficient computation and implementation. OpenCV's comprehensive set of optimized computer vision algorithms can be used for a number of tasks, including finding, recognizing and classifying all or parts of an object, performing blur detection, creating a region for automated cropping, tracking moving objects, and recognizing scenery, etc [20].

Originally developed and released by Intel, it is now supported by Willow Garage and Itseez and regularly updated and maintained. Since OpenCV is released under a BSD license with purposes of making computer vision easily accessible for programmers and users, the library and all of its components are free-to-use with well-documented source-code and hand-tuned assembly language binaries [20]. This makes it incredibly easy to understand, analyze and make use of its performance.

While the library is written in C/C++, wrappers to many other languages- of which all of the documentation can be found online- have been developed and maintained to encourage use by a wider audience. Additionally, OpenCV has extensive OS support and is able to run on Windows, MacOs, Linux, Android, and iOS. For software developers that lack a specific operating system in mind, they can take advantage of its cross-platform and cross-language library [21].

OpenCV is a well-established computer vision and machine learning library and a de facto standard in the computer science industry. Big companies such as Google, Yahoo and Microsoft make extensive use of the library in addition to countless others, both new and well-established. Moreover, it's free to use, cross-platform, cross-language with ample online source-code and documentation, making it a fine choice to use when developing computer-vision and machine learning intensive software applications.

TensorFlow

TensorFlow is an open source software library for machine intelligence that uses data flow graphs for numerical computation. Developed by the Google Brain Team for deep learning research, it boasts extensive built-in support for machine learning, from which we can use to build neural networks [22, 23]. While image recognition comes easily to our brains, it's an incredibly difficult task for computers. Recently, the deep learning community has made significant progress in computer vision and machine learning with the introduction of optimally performing neural network models. These models have performed relatively well at visual recognition, often succeeding to mimic or even exceed human visual recognition. Not only does such progress provide a relatively extensive pool of models to choose from, we can implement them using TensorFlow. The standalone library is general and flexible enough for such tasks; any computation that can be presented as a data flow graph can be done through TensorFlow.

The library's primary unit of data are *tensors*, which are essentially multidimensional arrays - each dimension referred to as a *rank* - that are represented as data flow graph edges. Numerical and mathematical operations are represented as data flow graph nodes. This architecture allows for computation to be distributed across multiple clients with a single API. Below are a few examples of tensors and ranks [22]:

```
3 # a rank 0 tensor; a scalar with shape []
[1, 2, 3] # a rank 1 tensor; a vector with shape [3]
[[1, 2, 3], [4, 5, 6]] # a rank 2 tensor; a matrix with shape [2,3]
[[[1, 2, 3]], [[7, 8, 9]]] # a rank 3 tensor with shape [2, 1, 3]
```

Figure 20: Example of tensors and ranks, as used in TensorFlow

TensorFlow provides multiple APIs, can run on multiple CPUs and GPUs, and is available on Windows, Linux, MacOS, iOS, Android and more. More specifically, TensorFlow offers an incredibly flexible Python interface. Popularly, most users opt for Python to be their primary programming language for back-end development. Importing the library in Python is very simple, as shown below. The import statement gives Python access to all of the library's classes, methods and symbols.

```
import tensorflow as tf
```

Figure 21: Importing Tensorflow in Python

TensorFlow also offers a convenient and useful utility called TensorBoard, which depicts an image of the computational graph. The more operations, or nodes, added

results in a higher complexity computational graph [23]. In order to make these models trainable, the graph will need to be modified to support different outputs for the same inputs. The graph can be constructed with operations that can read in images for evaluation, perform classification and determine the loss, gradients, etc. Training a network will require a large amount of data [23, 25].

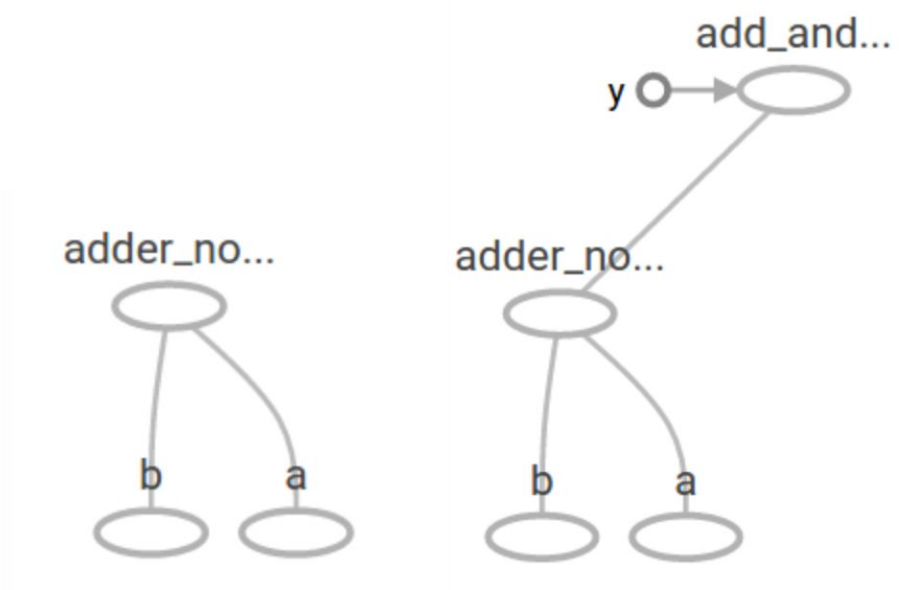


Figure 22: Example of a computation graph via TensorBoard. The right image is a more complex, extension of the left image.

As opposed to its competing libraries like Torch and Theano which also use computational dataflow graphs, TensorFlow is better at solving complex problems, thanks to the support of distributed computing. The library provides better computational graph visualizations and has an extensive circle of deep learning users and contributors.

TensorFlow does have its own limitations. Although powerful, TensorFlow is still a low-level library. Defining models in TensorFlow can be quite tedious, repetitive and impractical [24]. For added modularity, ease-of-use and flexibility, a high level interface that can be implemented alongside TensorFlow is desirable.

Tensorboard

Tensorflow has a visual tool, called Tensorboard, that allows the user to understand the networks that they build, making it easier to debug and optimize programs. Figure 23 show training results overtime of layers in the neural network and provides visualizations that help the user understand what is happening inside the network. It also provides a graph of the structure that was built, allowing the user to see whether the layers are connected to the correct regions, whether it outputs the correct tensors and what shape the tensors are taking. Other items, such as distributions, plot the values of the neurons in a network to show any large patterns that could be produced [23, 24, 25].

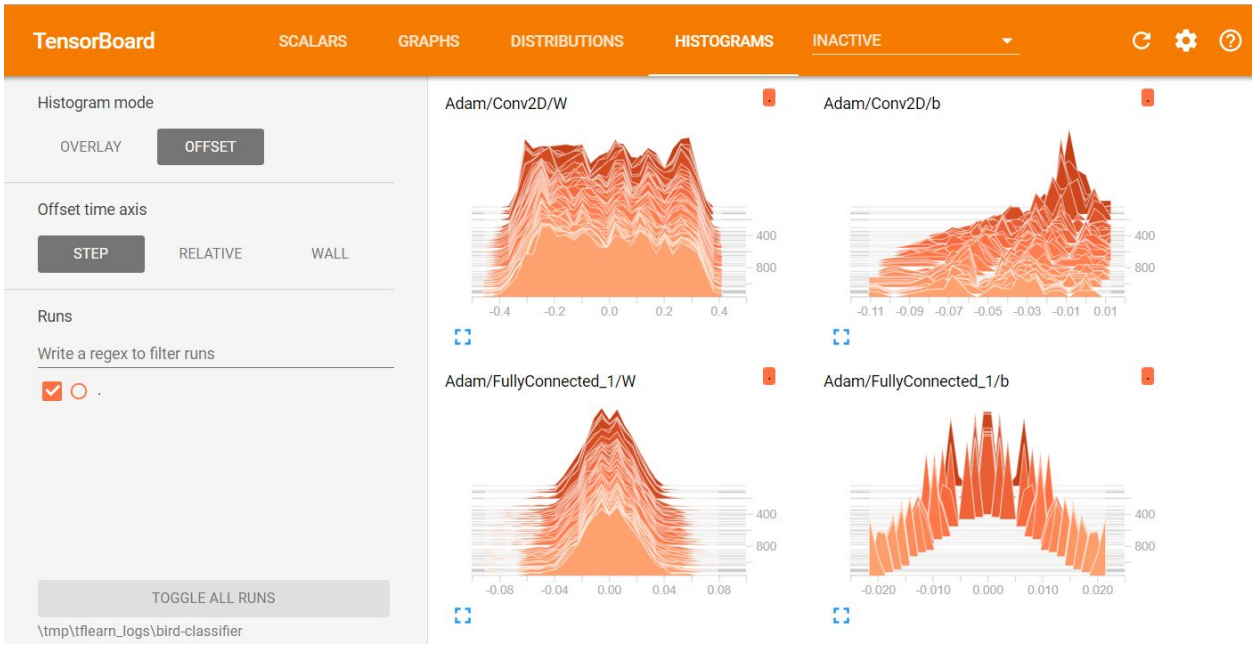


Figure 23: Training of a neural network overtime

TFLearn

One of the limitations of Tensorflow is that it is a low-level library. A common characteristic of low-level languages and libraries is that they are extremely verbose in their syntax. There are some users that think that this verbosity harms productivity as they feel that they have to write more code than necessary in order to implement a certain part of a model or network. Thus, an API called TFLearn was created to mitigate this issue. It is a library built on top of Tensorflow, meaning that all of the functions and structures in TFLearn are abstractions that call the underlying code. The library is designed to be modular and transparent, which allows for faster prototyping and experimentation [25, 26].

The API supports virtually all of the type of networks commonly used in modern deep learning models, such as convolutional neural networks, long short-term memory networks, etc. Each type of network is built into easily-instantiated modules and can be used with any combination of other components necessary for a neural network, e.g. optimizers, activation functions, etc. TFLearn also provides visualizations of the network models in the form of a graph and includes additional details about the layers, such as weights and activations. The user has the ability to create custom layers if there is not a built-in layer that provides the functionality necessary for their model. The library also works seamlessly with multiple CPUs or GPUs without requiring much user configuration.

```
with tf.name_scope('conv1'):
    W = tf.Variable(tf.random_normal([5, 5, 1, 32]), dtype=tf.float32, name='Weights')
    b = tf.Variable(tf.random_normal([32]), dtype=tf.float32, name='biases')
    x = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
    x = tf.add_bias(x, b)
    x = tf.nn.relu(x)
```

Figure 24.1: An example of creating a convolutional layer with Tensorflow

```
tflearn.conv_2d(x, 32, 5, activation='relu', name='conv1')
```

Figure 24.2: The same convolutional layer can be created with a single line of code in TFLearn

Keras

Like TFLearn, Keras is a high-level neural network API written in Python. It is able to run on top of Tensorflow, CNTK, or Theano. The main purpose of its development was to enable fast experimentation. This is a beneficial aspect when testing the feasibility of different models as one should want to write the least amount of code possible in order for a model to attain efficiency and feasibility. It is also similar to TFLearn in its modularity as the user can mix different neural network layers with different cost functions, optimizers, etc. It is similar still in the fact that one can define custom layers if

necessary. However, a key difference between Keras and TFLearn is that the former makes “user friendliness” a guiding principle. In this respect, the API tries to reduce cognitive load by offering simple functions for common use cases, such as instantiating an activation layer. Keras does have an objectively simpler syntax; however, when desiring control over the minutiae of a model, simplicity may hide key parameters that one may need to adjust in order to determine areas of improvement [26].

Blur Detection

Laplacian Operator

When analyzing an image to determine its quality and user appeal, a preliminary step is classifying whether or not the image is blurry. Analyzing the contents of an image is meaningless if the image itself is generally out-of-focus and unclear. As such, a method to recognize a certain level of blurriness is required.

A relatively simple implementation of blur detection can be done by manipulating the Laplacian operator to perform edge detection. OpenCV contains a built-in method that computes the Laplacian, or the second derivative, of an input image [20, 27]. For accuracy, this input image should be converted to grayscale beforehand, so the method can convolve it with the 3 X 3 Laplacian kernel, shown below, to return a floating-point value. The Laplacian operator is used because it effectively and quickly highlights areas that contain rapid changes in intensity. By computing the statistical variance- available through NumPy- of those intensity changes, the user can examine what is essentially a distribution of edges and responses [28]. A blurry image makes it difficult to clearly perceive the contents of a photograph and thus, can be expected to contain a very small number of edges and consequently, has a very low variance. A normal, non-blurry image, on the other hand, has a high variance because there is a wide spread of responses and a large number of edges.

0	1	0
1	- 4	1
0	1	0

Figure 25: The Laplacian kernel

Possessing a set of Laplacian variances is not enough; users must evaluate those variances to determine just how much blurriness or clarity is acceptable. That can be narrowed down to a general floating-point value that can serve as a threshold for acceptable, non-blurry images. In other words, users should define just how low of a variance they are willing to accept. This number is a very critical parameter as users should avoid mislabeling images. A program threshold set too low could include unwanted blurry images and one set too high could remove desirable images. Once a threshold is successfully determined, any images with variance below that will be disregarded and labeled as blurry, while images with variances at or above that can be further analyzed [27].

Below are classifications of two images, one blurry and the other non-blurry, with a threshold of 100.00 (reference source of images here). The image labeled as “blurry” falls well below the threshold, and it’s clearly fuzzy and out-of-focus. On the other hand, the second image’s Laplacian variance is well above the threshold value and is clearly in-focus.

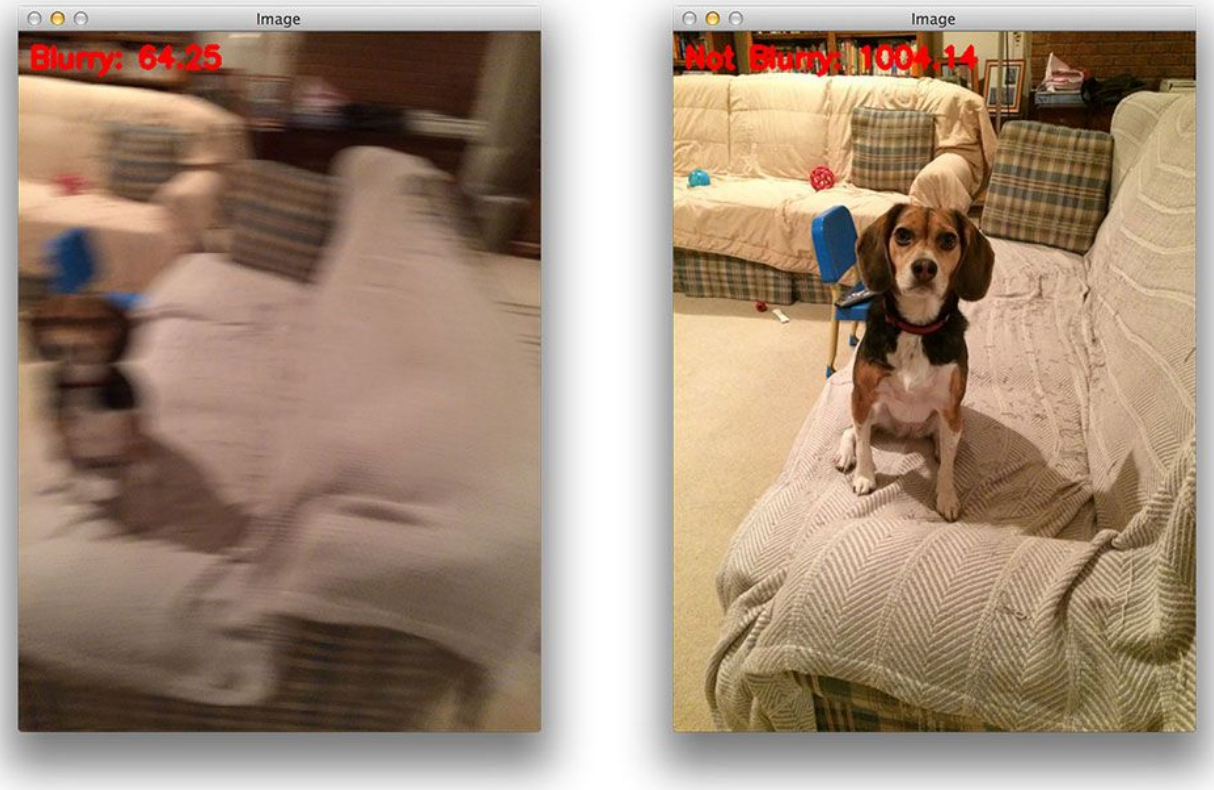


Figure 26: Example of images with labeled variances with a threshold value of 100.00. The image on the left has a low variance and is thus “Blurry,” while the image on the right has a high variance and is thus labeled “Not Blurry” [27].

Haar Wavelet Transforms

Blur detection via the Laplacian operator and variance has a number of limitations; results vary with image size and do not account for uniform backgrounds, focal blur, etc. This scheme will only identify the distribution and frequency of edges in a single image and, as such, may not provide enough inference for an accurate classification of blur. However, edges remain to be useful discriminative features that could be further analyzed to determine overall image quality.

As an alternative and more conclusive method of blur detection to the Laplacian operator, wavelet transforms can be utilized to not only determine whether an image is blurry but also to what extent an image is blurred. Blur extent is especially significant for images with uniform backgrounds, e.g. those with predominant regions of sky or water, and with focal blur, i.e. images where the background is blurred to emphasize the subject(s). Haar wavelet transforms, specifically, are ideal for edge analysis because its local maxima can detect locations of structure irregularity and recover the sharpness of blurred edges [36].

Most natural images contain all types of edges. Different edges are generally classified into three types: Dirac-Structure, Step-Structure and Roof-Structure. Step-Structure edges can be further decomposed into Astep-Structure and Gstep-Structure according to whether changes in intensity are gradual. When blur is present in an image, both Dirac-Structure and Astep-Structure edge points disappear and both Gstep-Structure and Roof-Structure edge points tend to lose their sharpness. In the blur detection scheme proposed by researchers from Tsinghua University, images were classified as blurry according to whether they contained Dirac-Structure or Astep-Structure edge points and blur extent was determined by calculating the percentage of Gstep-Structure and Roof-Structure which are blurred, which are more likely to exist in a blurry image [36].

The proposed algorithm for edge detection is comprised of three steps [36].

1. Perform Haar wavelet transform on the original image and iteratively on its approximations, for a decomposition level of three, as illustrated below.

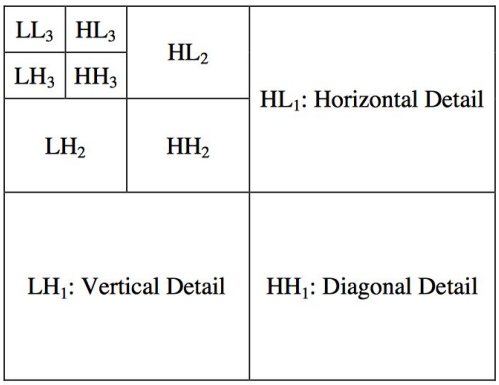


Figure 27: Hierarchical representation of Haar wavelet transform decomposition

Figure 28: A visual representation of three iterations of Haar wavelet transform. The left figure is the original image and the right figure depicts the decomposition and filters [37]

2. For each level of decomposition, construct an edge map from the vertical, horizontal and diagonal details of each level. It can be calculated as follows:

$$Emap_i\left(k,l\right) = \sqrt{LH_i^2 + HL_i^2 + HH_i^2} \quad (i = 1,2,3)$$

3. Partition each edge map into windows and determine the local maxima of each. $Emap_1$ is divided evenly into windows sized 8X8, $Emap_2$ into windows sized 4X4, and $Emap_3$ into windows sized 2X2. The results are denoted as follows:

$$E\max_i \ (i = 1,2,3)$$

Each maximum in $Emax_i$ represents the intensity of the edge; the greater the value, the more intense the edge. For any user-defined threshold and point *(k, l)*, if $Emax_i$ *(k, l)* > *threshold*, then *(k, l)* is considered an edge point and a non-edge point otherwise.

Once edge points can be successfully identified, they can then be classified into edge types. The rules for edge type identification and classification are as follows [36]:

1. If $Emax_1$ *(k, l)* > *threshold* or $Emax_2$ *(k, l)* > *threshold* or $Emax_3$ *(k, l)* > *threshold* then *(k, l)* is considered an edge point.

2. For any edge point *(k, l)*, if $Emax_1$ *(k, l)* > $Emax_2$ *(k, l)* > $Emax_3$ *(k, l)* then *(k, l)* can be classified as a Dirac-Structure or Astep-Structure edge point.

3. For any edge point *(k, l)*, if $Emax_1$ *(k, l)* < $Emax_2$ *(k, l)* < $Emax_3$ *(k, l)* then *(k, l)* can be classified as a Roof-Structure or Gstep-Structure edge point.

4. For any edge point *(k, l)*, if $Emax_2$ *(k, l)* > $Emax_1$ *(k, l)* and $Emax_2$ *(k, l)* > $Emax_3$ *(k, l)* then *(k, l)* is definitively classified as a Roof-Structure edge point.

5. For any Gstep-Structure or Roof-Structure edge point *(k, l)*, if $Emax_1$ *(k, l)* < *threshold* then *(k, l)* is more likely to be an edge point in a blurry image.

Once all edge points have been successfully classified, we can utilize them to gather significant information about and form an accurate inference on the blurriness of an image. In practice, judging the image quality and degradation of an image is subjective and entirely dependent on what the user is looking for. Thus, before an image can be conclusively classified as blurry or not blurry, two parameters need to be defined by the user, denoted as *threshold* and *MinZero*. The parameter *threshold* represents the desired edge intensity, while the parameter *MinZero* represents the desired ratio of Dirac-Structure and Astep-Structure edge points to the total number of edge points in the image. In their experiment, the developers of the proposed blur detection scheme suggested that *threshold* be set to a value at or above thirty, as humans are typically not sensible to intensity below that; likewise they suggested *MinZero* be set to a positive real number very close to zero.

To analyze the edge points for accurate blur detection, variables need to be instantiated as such:

- N_{edge} : the total number of edge points
- N_{da} : the total number of Dirac-Structure and Astep-Structure edge points
- N_{rg} : the total number of Roof-Structure and Gstep-Structure edge points
- N_{brg} : the total number of Roof-Structure and Gstep-Structure edge points that have lost their sharpness, which can be determined from whether $Emax_1$ *(k, l)* < *threshold* for a given edge point *(k, l)*
- *Per* : the ratio of Dirac-Structure and Astep-Structure edge points to all edge points, which can be calculated as *Per* = N_{da} / N_{edge}
- *BlurExtent* : the ratio of blurred Roof-Structure and Gstep-Structure edge points to all Roof-Structure and Gstep-Structure edge points, which can be calculated as *BlurExtent* = N_{brg} / N_{rg}

The latter two variables will determine the final classification of blurriness in an image. If *Per* > *MinZero*, the image is judged as non-blurry. *BlurExtent* is the blur confidence coefficient for the image; it is a significant ratio value that can offset the bias of *Per* by accounting for uniform backgrounds or focal blur, which will likely have a small ratio of Dirac-Structure and Astep-Structure edge points but also a low number of blurred Roof-Structure and Gstep-Structure edge points.

The final structure of the entire blur detection scheme is shown below.

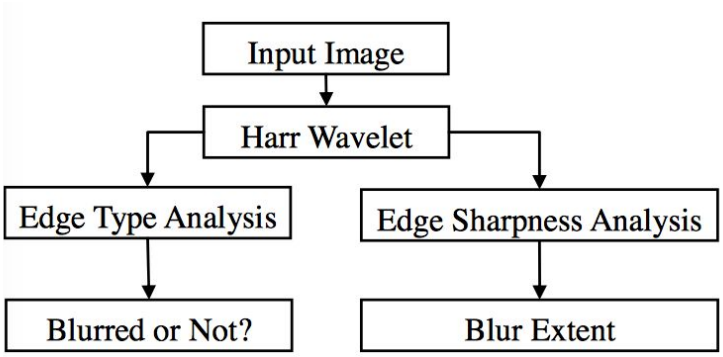


Figure 29: The structure of the blur detection scheme of digital images using Haar wavelet transforms

The developers' experimental results yield high accuracy and will be ideal for comprehensive classification of blur:

Test Image Set	Number	Accuracy (%)
Un-blurred	1398	98.21
Motion Blur	479	98.33
Out-of-focus Blur	458	100
Total	2355	98.60

Figure 30: Experimental results

Image Comparison

A large dataset of digital photographs that capture the same environments and subjects over a short period of time tend to contain an abundance of similar images. Furthermore, such a dataset can potentially contain a large amount of desirable images that meet the user's defined criteria. As such, an excessive number of related and similar images will likely be exhaustive to keep and recreationally sift through. It is heavily advisable to have a measure of image similarity in order to reduce the number of similar images in a final subdirectory.

There are currently several de facto standards for image analysis and comparison, including histograms, image hashing, structural similarity measures, etc. These measures typically work best on identical or nearly identical images with small differences in coloring, contrast, small shifts, etc. In that respect, in order to achieve relatively satisfactory results in image comparison for images that contain similar subjects in similar environments, the parameters and thresholds for classification of similarity must be quite generous.

Image hashing provides the best implementation of comparing images. Other measures like calculating histograms or structural similarity index are limited to identical images or produce inaccurate results. Image hashing, specifically wavelet hashing, is more likely to classify images with similar subjects and environments as similar. There exists a standardized image hash library that can be installed and imported into Python. This library boasts several methods of image hashing; however wavelet hashing yields more accurate results by comparing high level edge frequencies in any given image.

Figure 31: The hash values of the two images on the left will be similar, while that of those on the right will be largely different [40]

User Interface

Electron

Electron is an open source software framework developed by GitHub for the purpose of building cross-platform desktop applications using JavaScript, HTML and CSS by combining Chromium and Node.js. It essentially provides a runtime to build desktop apps with just JavaScript. Desktop applications built using Electron are automatically compatible with MacOs, Windows and Linux operating systems. A significant number of popular desktop applications have been developed with Electron, such as Slack, Discord, Atom, GitHub Desktop, etc. As such, the framework is regularly updated and maintained by GitHub and an extensive amount of users and contributors [29].

It boasts several impressive built-in features that make it user-friendly and easily maintainable: automatic updates, native menus and notifications, application crash reporting, debugging and profiling and Windows installer. For maintainability, automatic updates allows applications to automatically update themselves, while crash reporting enables users and programmers to submit crash reports to a remote server and Chromium's content module detects slow operations and performances. For ease of use, Electron creates native application and context menus and enables the fast and simple generation of Windows installers using their package.

A typical Electron application requires three files: *index.html*, *main.js* and *package.json*. The file *index.html* is essentially a blank application canvas rendered by Electron by default. The file *main.js*, or main process, starts the application and creates a browser window containing rendered web pages that is compatible with the native graphic user interface of the working operating system. The file *package.json* is essentially the most important because it contains the majority of the application's user interface contents, such as its dependencies, metadata and other necessary files, including the application's main process, *main.js* [30].

The bulk of the application is handled by the main process, which in addition to interacting with the operating system and starting and terminating the application window, opens dialogs and creates render processes. Render processes are browser windows and while there can only be a single main process, there can be multiple render processes, each independent to one another and can run both separately and

concurrently. Each process performs a specific task and communicates to both the main process and to each other. The hierarchy can be exemplified simply:

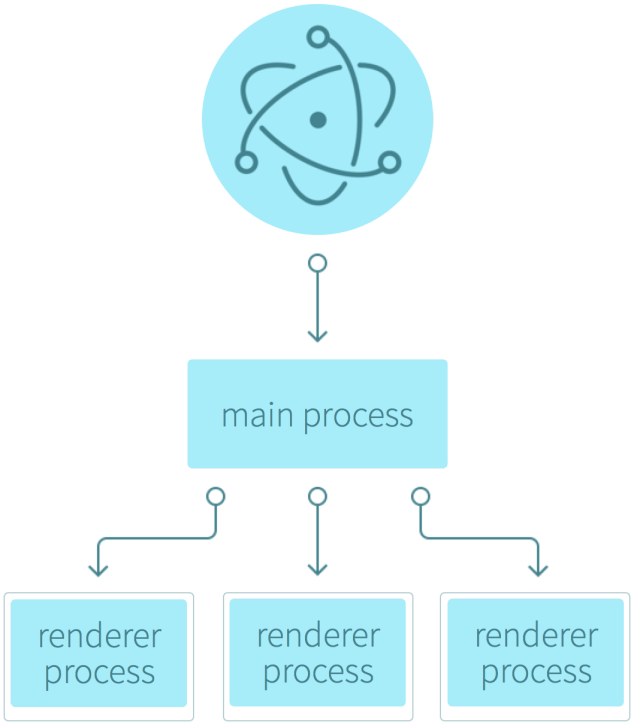


Figure 32: A simple depiction of how the main process is related to render processes in Electron [30]

Once a user interface is created using Electron, functionality must be added. By taking advantage of a Python subprocess, the application can react to all user inputs in JavaScript and reciprocate with an appropriate response the user can see and further interact with [29].

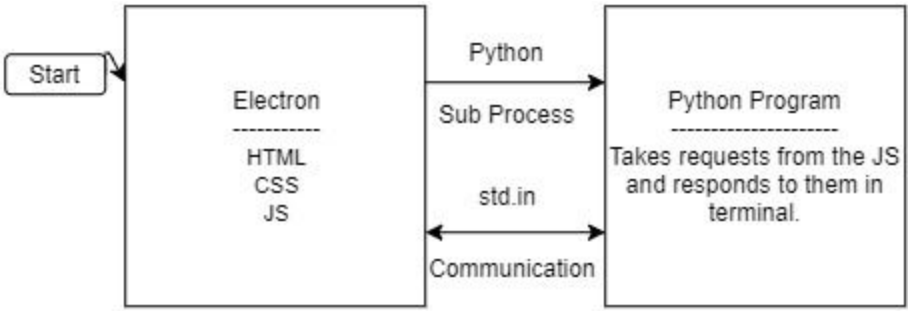


Figure 33: Relationship between an interface built with Electron and a program built in Python.

Kivy

Kivy is an open source Python user interface for rapid development of applications. The library supports cross-platform development and usage; it runs on Windows, Linux, macOS, Android, iOS and Raspberry Pi. Additionally, the same Python script written with Kivy can be run on all the aforementioned supported platforms.

Kivy also provides extensive documentation that new users can reference to understand the library better. It offers interactive examples of screens, layouts, buttons, images, etc. from which new users can reference and take inspiration from. The user interface elements native to Kivy contain many, if not all, standardized UI formats, making it easy to understand, interpret and use. Kivy can also be relatively easy to convert into a single executable for standalone application development [39].

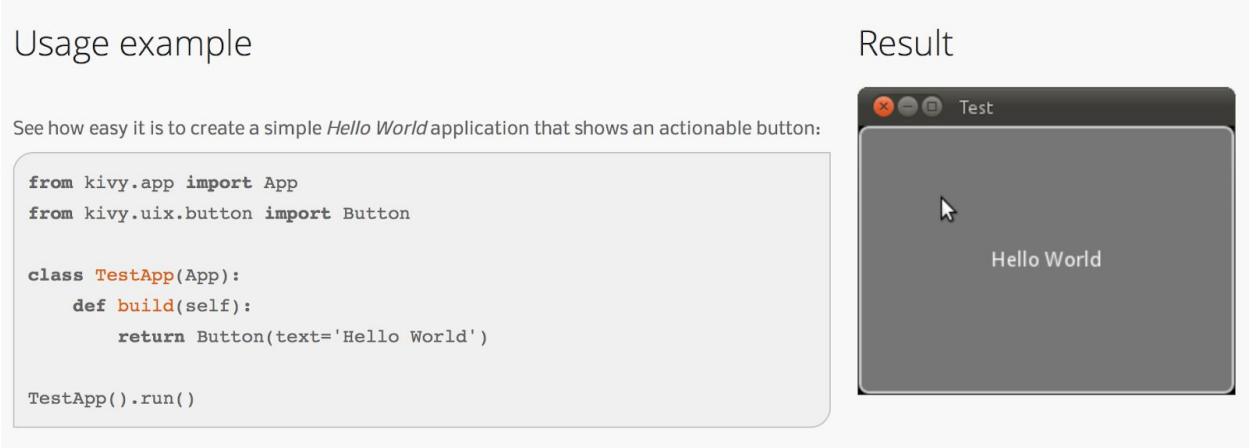


Figure 34: A simple “Hello World” usage example of Kivy’s user interface library [39]

```
MyRootWidget:
    BoxLayout:
        Button:
        Button:
```

```
root = MyRootWidget()
box = BoxLayout()
box.add_widget(Button())
box.add_widget(Button())
root.add_widget(box)
```

Figure 35: User interface elements can be defined in two ways: in a *.kv* file using Kivy's custom language (left) or in a Python script (right) [39]

`AnchorLayout`:
    Widgets can be anchored to the 'top', 'bottom', 'left', 'right' or 'center'.
`BoxLayout`:
    Widgets are arranged sequentially, in either a 'vertical' or a 'horizontal' orientation.
`FloatLayout`:
    Widgets are essentially unrestricted.
`RelativeLayout`:
    Child widgets are positioned relative to the layout.
`GridLayout`:
    Widgets are arranged in a grid defined by the *rows* and *cols* properties.
`PageLayout`:
    Used to create simple multi-page layouts, in a way that allows easy flipping from one page to another using borders.
`ScatterLayout`:
    Widgets are positioned similarly to a RelativeLayout, but they can be translated, rotate and scaled.
`StackLayout`:
    Widgets are stacked in a *lr-tb* (left to right then top to bottom) or *tb-lr* order.

Figure 36: Kivy's native layout options

Color Schemes

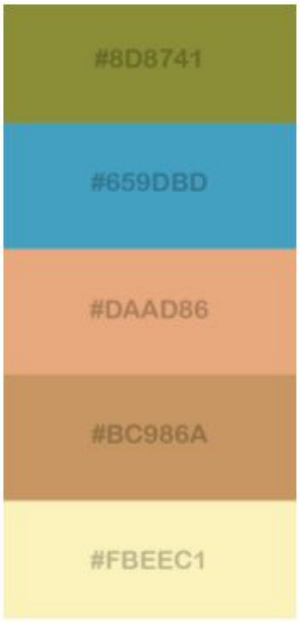


Figure 37.1

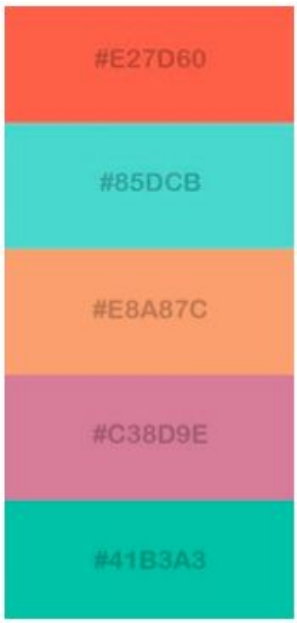


Figure 37.2

Figure 37.3

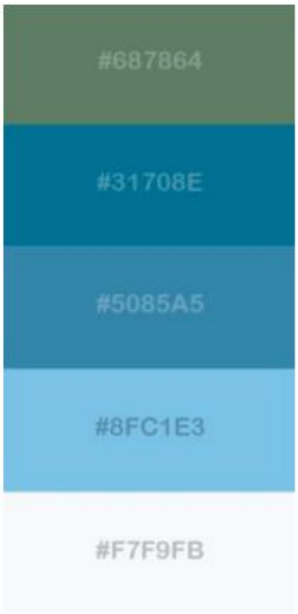


Figure 37.4

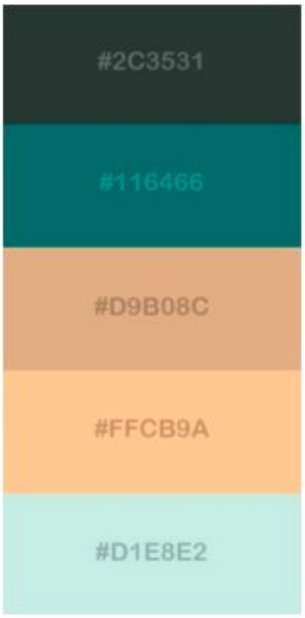


Figure 37.5

Figure 37.6

User Experience

By nature, motion holds the greatest level of significance for users, especially in a user interface where users often expect at least a certain level of interactive and aesthetic design. Animations in a user interface can enhance user experience as animated interface elements reveal the process and functionality of a user interface in ways that neither static text nor static images can. Similar to any other element of good design, these user interface animations should serve some type of functional purpose in order to maximize usability, utility and desirability of the application [31, 32]. They are quite often used to demonstrate to application users that either an action has been performed or an action is currently in progress. Most interfaces also include animated timelines, progress and loading bars, etc. to let users know how much longer an action will take to complete. It's much more than just visual representation; these small details of interactive design make a fundamental difference on modern applications.

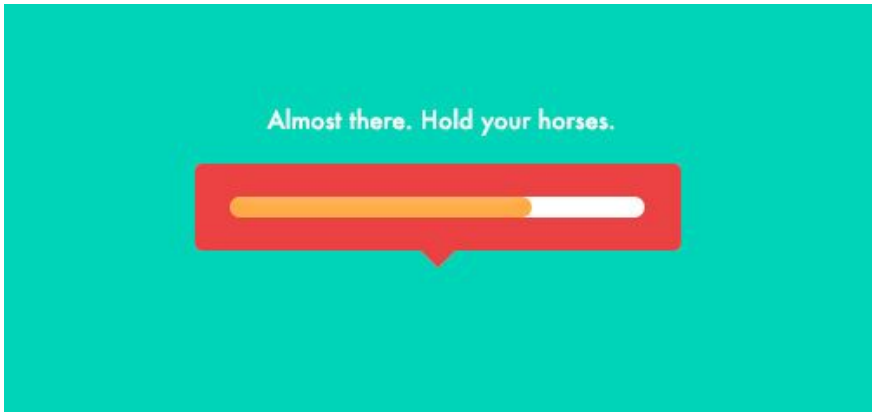


Figure 38: Example of a loading screen [33].

Interface development and design is typically directed towards users for ease-of-use and navigation. The central functionality and performance of all applications are hidden from the general user, leaving them with only the interface to interact with until the application completes its tasks or stumbles upon faults and errors. Depending on the application, this may result in a relatively long time frame where users are simply waiting for the application to perform its intended duties. In these cases, if there are no physical or visual indications of progress or performance, users are left waiting in uncertainty. Incorporating animated elements in a user interface not only blunts the negative experience of waiting, it can also indicate to the developer whether an error has occurred. If an interface is not performing or displaying as intended, developers can assume that an error or failure has occurred.

For a graphic user interface, all user inputs and selections should be followed with some sort of visual change in order to demonstrate to the user that their input or selection has been accepted by the application and is being processed according to the application's specifications and requirements. Without any visual indication, it's impossible to

determine whether or not an action will be performed at all. Consider the loading frames below; displaying this animation in an application demonstrates to the user that their input has been accepted and the application is subsequently and currently performing its required tasks. Similarly, once the software has successfully completed its tasks, the loading animation should terminate and another visual element should appear to indicate that the task has been completed. For example, if an online video is buffering, the user should see a visual representation of the stream loading and once it's finished, the video should begin playing.

Incorporating such movements and animated elements into the user interface will certainly enhance user experience. When users interact with any type of digital product, they expect to know what exactly is happening every step of the way. When users are informed, their opinion of this application is much more positive [32, 33].

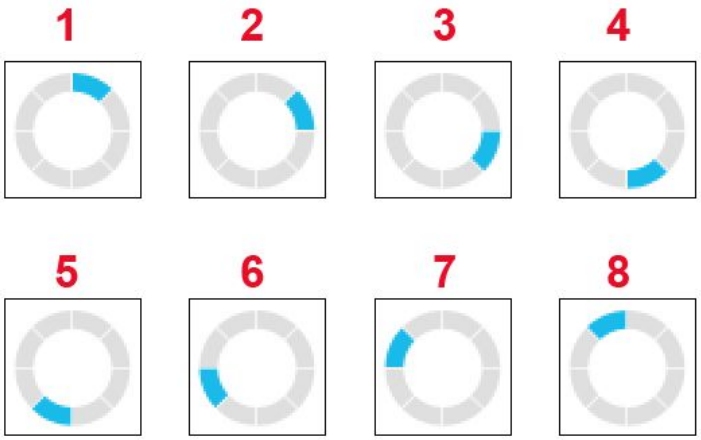


Figure 39: Frame-by-frame progression of a loader.

Detailed Design

Dataset Preparation

Training Data

In order to properly train the network, a large and diverse training dataset is required. We have gathered approximately five distinct datasets related to birds at this point. Our sponsor, Dr. Leavens, has provided us with approximately 22,000 images, most of which are birds. They are all unlabeled and will need to be manually labeled if we want to use them for supervised training. Instead, they could be used for the generative adversarial network's real images to compare to generated images. This will provide more robust training in images similar to our sponsor's style.

The first dataset we started working with was the CIFAR-10 dataset. As shown in figure 32 below, it is comprised of ten classes, each containing six thousand images at 32x32 resolution. This was a good dataset with which to begin our training because it is small in size, contains a variety of birds, and diverse data to help the network understand what is not a bird image. The small size provided quick training time for testing different network structures. This allowed us to find better layer specifications without waiting hours or days for results [34].

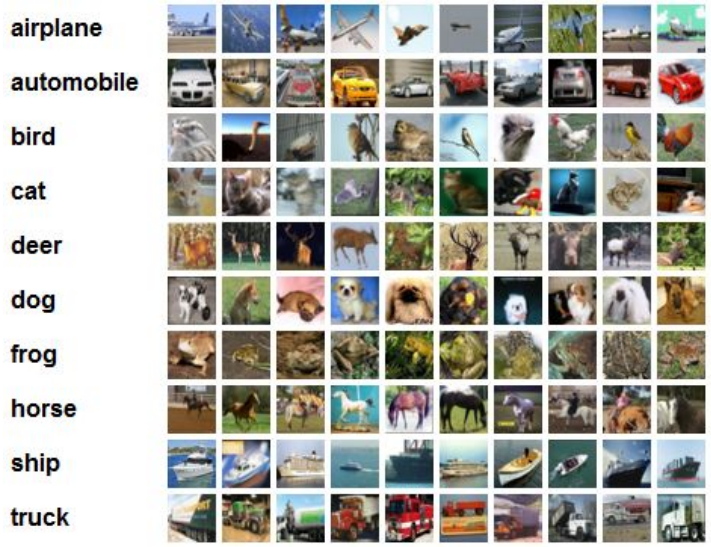


Figure 40: CIFAR-10 dataset

We will work with Caltech-UCSD as our next dataset which contains around twelve thousand bird images. The dataset also contains bounding boxes and segmentation for locations of the birds. This will allow us to begin testing the region proposal network [35].

UCLA Nature dataset is another dataset similar to Caltech, containing approximately eight thousand bird images and also contains bounding boxes. One of the last image sets that we will use will be ImageNet as it contains over thirteen million images, many with bounding boxes and specific labels.

Figure 41: Classes of birds separated by species

Face Dataset

From the NABirds dataset, we constructed our own dataset of bird faces. Of the 48,562 images, 46,825 were used to train for face detection. The excluded images either did not contain enough facial features or were considered too small. During creation of the dataset approximately 334 labels needed to be corrected from the given dataset, from either mislabeled parts or incorrect location information.

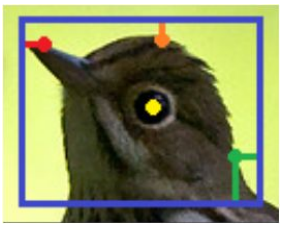


Figure 42: a demonstration of head features being used to construct a face bounding box

Using five of the eleven parts annotations, we were able to create bounding boxes around the head. The five parts included the bill, crown, nape and both eyes. These parts were used to determine the orientation of the bird or which direction it was facing. Approximately ten scenarios were considered from parts being at different heights or features not being present to help determine the orientation. After the orientation of the bird was establish a bounding box would be place around the present head parts. The distance from the parts depended on the orientation of the bird, some had constant distance from the parts while others used the ratio distance between parts. This helped prevent the bounding box from becoming just a cross section of the face

Afterwards, testing was done to make sure boxes didn't have errors, exceed the size of the image and accurately portrayed the bird's face. This included cropping all bounding boxes from the images to confirm there were no issues. Manual inspection of all cropped faces were used to help pick out bad faces. These images were split into 38,824 training images and 8,000 test images.

TensorFlow with TFLearn

Out of TensorFlow, Theano, Caffe, and SciKit Learn, TensorFlow was the clear winner; TensorFlow offers neural network and convolutional layer APIs, easy-to-use and optimized GPU support, optimizers straight-out-of-the-box (such as Adam and gradient descent), model checkpointing for dynamic training evaluation and model visualization through TensorBoard. TensorFlow is also under active development, whereas its top competitor Theano has recently announced end of development.

The top contenders for high-level TensorFlow APIs are TFLearn and Keras. As a team we decided to use TFLearn as TFLearn is meant for building industry-grade machine learning solutions and is optimized to function on top of TensorFlow. In contrast, Keras works best when used for small projects and is optimized for Theano, not TensorFlow.

1) ConvNet - TFLearn input layer
shape=[None, 32, 32, 3]
placeholder=None
dtype=tf.float32
data_preprocessing=preprocess
data_augmentation=augment
name='InputData'

The first layer of the convolutional neural network stores the raw pixel values of each image. In this example, the image has a width and height of 32 with 3 color channels: red, green, and blue.

Data preprocessing methods are applied in both training and testing time. Our preprocessing step zero-centers and normalizes the data dimensions of the image so that every image is approximately of the same scale. This is useful for ReLU because the important pixel values will be centered around the normalized region, and ReLu will get rid of negative values, which is the outputs of filters not responding well to the input.

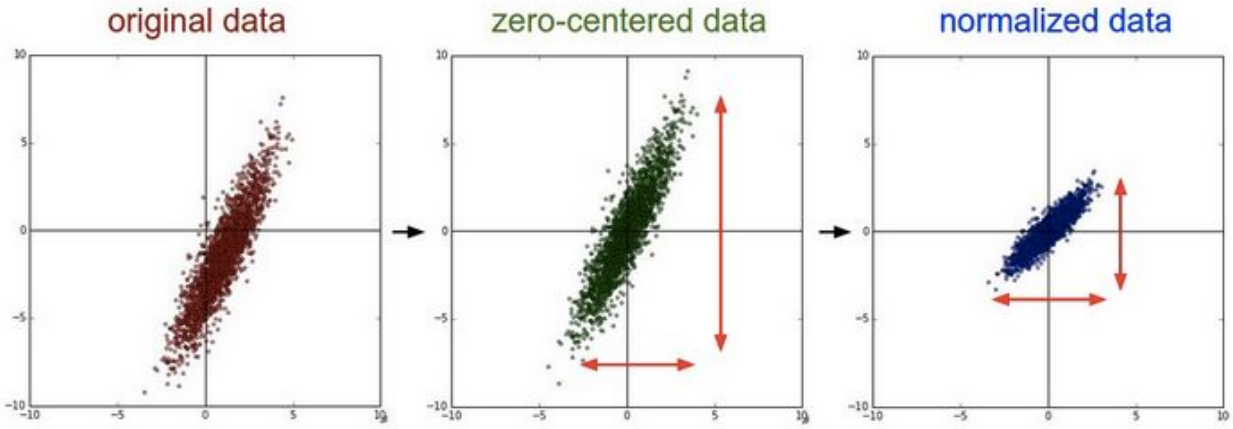


Figure 43: The process of zero-centering and normalization.

Data augmentation methods are applied only during training time. Our augmentation step flips the image left and right (along the x-axis) and randomly rotates the image from 0 to 25 degrees. This is important for properly training the CNN because CNNs cannot handle rotation at all - if they are trained on objects in one orientation, they will have trouble when the orientation is changed.

2) ConvNet - TFLearn 2d Convolution
incoming 4-D Tensor
nb_filter=32
filter_size=3
strides=1
padding='same'
activation='relu'
bias=True
weights_init='uniform_scaling"
bias_init='zeros"
regularizer=None
weight_decary=.001
trainable=True
restore=True
reuse=False
scope=None
name='Conv2D'

The second layer of the convolutional neural network does most of the computational heavy lifting for the CNN. This layer's parameters consist of a set of learnable filters, defined as *nb_filter* in TFLearn. Every filter is small in relation to the width and height of the input and extends through the full depth of the input volume. In our first 2D convolution layer, our filter size is 3x3x3 (i.e. 3 pixels width and height, and 3 for the red, green, and blue color channels). Convolution works by sliding each filter across the input data and computing the dot product between the filter entries and the input at the filter position. From these dot products, we create a 2D activation map that gives the responses of that filter at every spatial position. We apply this process for all 32 filters and produce 32 activation maps that we squash together to produce the output filter.

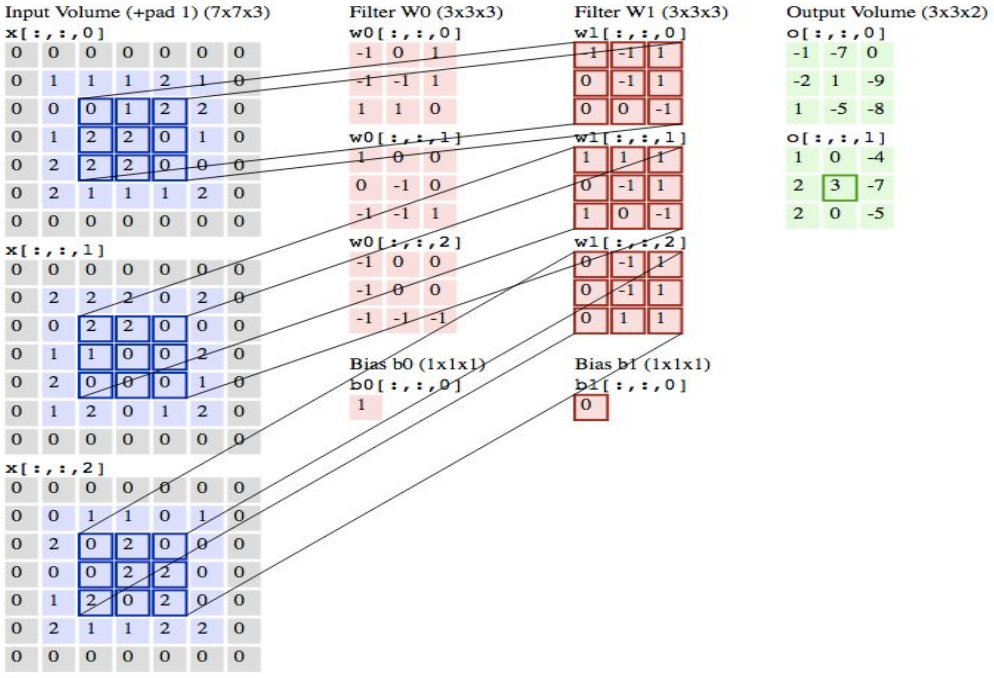


Figure 44: 2D Convolution

3) ConvNet - TFLearn max pooling
Incoming 4-D Layer
kernel_size=2
strides=None
padding='same'
name='MaxPool2D'

The third layer of the convolutional neural network acts as a process of downsampling by reducing the size, width and height of the input data for the next convolutional layer. Max pooling does not affect the color channels.

Similar to convolution, the max pooling process takes a small window and slides it across the input data. Instead of doing some sort of matrix operation, we simply take the max value within the observed region.

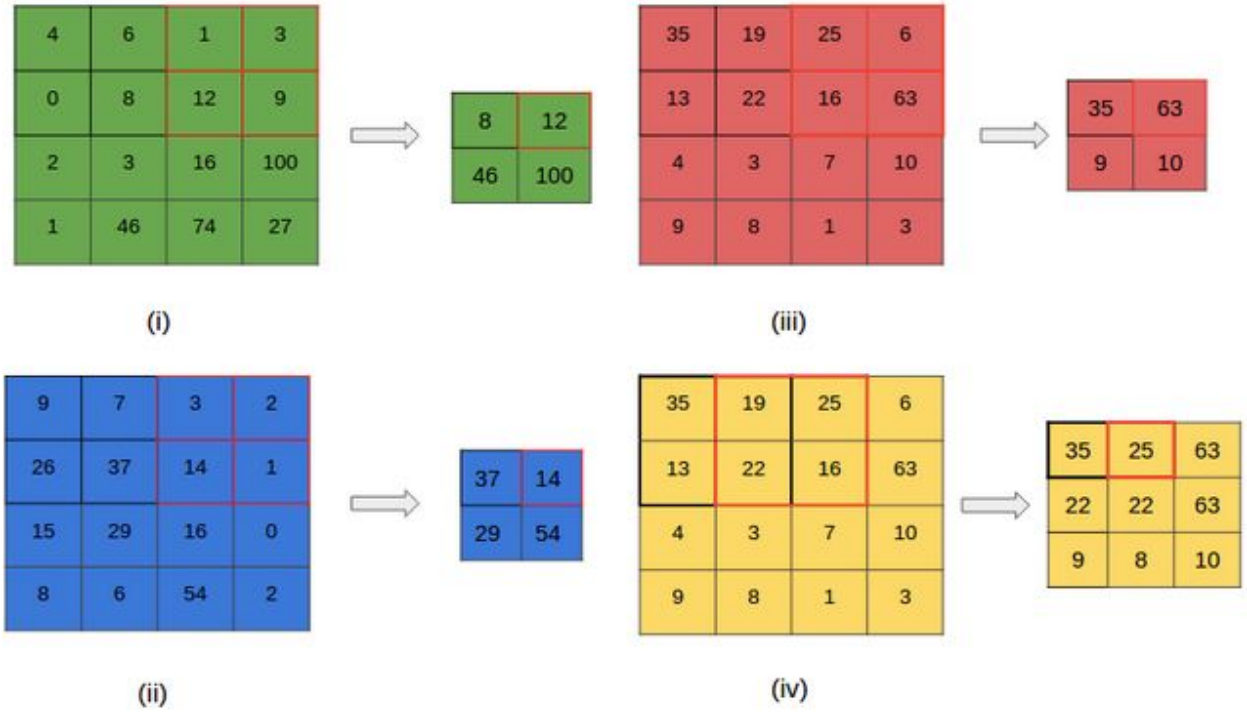


Figure 45: Max Pooling

Tensorflow Object Detection API

Tensorflow has developed a variety of object detection models that are open-source and available for widespread use. We opted to utilized their detection models because of their effectiveness and extensive training and testing. We tested a number of models, including Faster R-CNN and Single Shot Multibox Detector.

The provided models have been pre-trained using Google's extensive datasets. For the purposes of our application, we selected the models trained on the COCO dataset, as it boasts over 330,000 images and 80 classes, among which are bird classes. We trained and tested ResNet-50, ResNet-101 and Inception V2, toggling between preferring speed over accuracy, or vice versa. Additionally, we considered the API's Neural Architecture Search because of its reported accuracy; however, the runtime was approximately eighteen times slower than its contemporaries. Thus, we proceeded with the ResNet models instead.

We performed transfer learning using these pretrained models; the API's weights are frozen up until the feature maps, which allow ease of retraining with custom datasets. We trained the frozen weights further using Tensorflow's base configurations and hyperparameters, e.g. two thousand steps for training and a batch size of one.

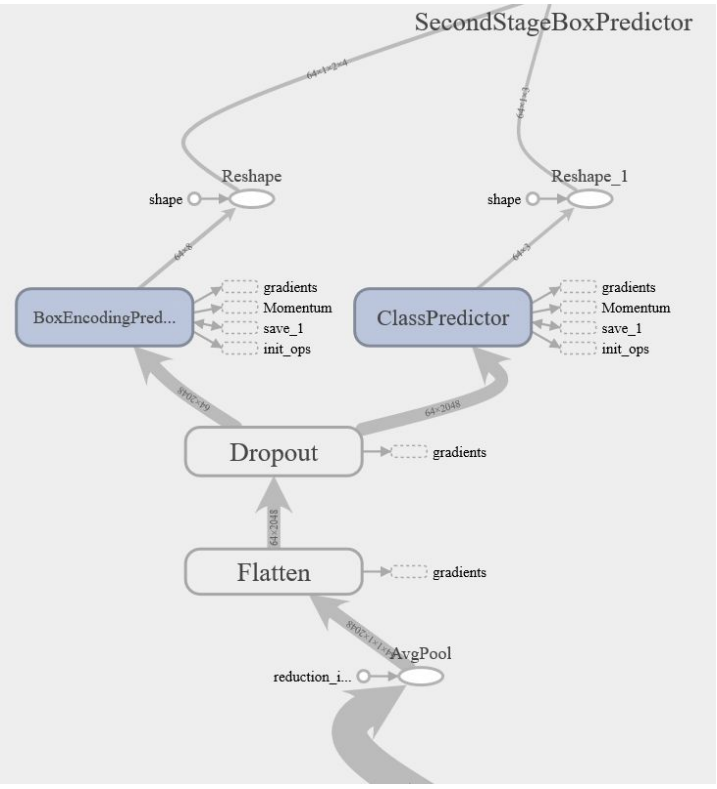


Figure 46: a graph of the API model's region proposal network, section we retrain.

Blur Detection

Following extensive research and preliminary implementations and evaluations of various standardized blur detection methods, e.g. using the Laplacian operator, fast Fourier transforms, Tenegrad, etc., we concluded that simple computations for blur detection did not provide satisfactory results. While most of the blur algorithms, including those mentioned above, fared relatively well in determining overall blur of an image, the experimental results left a lot to be desired. There were several significant limitations to these computations; image size affected results, focal blur was not taken into consideration, images with uniform backgrounds were wrongly classified and blur classifications were largely inconsistent.

Figure 47 and 48: Examples of images wrongly classified as blurry; the figure on the left contains a large amount of purposeful focal blur, while the figure on the right has a largely uniform background, i.e. the sky

The major limitations of these methods can be attributed to superficial edge detection and analysis, which provides inadequate information for accurate blur estimation. By thoroughly identifying, classifying and analyzing image edge points, we can not only judge whether or not a given image is blurred but also to what extent the given image is blurred. The method outlined in the blur detection scheme proposal by Tsinghua University researchers provide effective and accurate results [36].

In their scheme, they proposed taking advantage of Haar wavelet transforms to discriminate between different types of edges and how they respond to blur. Their algorithm involves calculating three iterations of Haar wavelet transforms on a single given image, constructing edge maps for each iteration, partitioning the edge maps into windows and finding the local maximum within each window. Each of the maximums represent edge intensities, which can be classified as one of the three major types of edges: Dirac-Structure, Step-Structure and Roof-Structure.

```
def calc_intensities(img):
    """
    Calculate haar wavelet transforms, edge maps and maxima
        img: (ndarray) grayscale image file
    """
    # i = 1
    LL_1, LH_1, HL_1, HH_1 = haar_wavelet_transform(img)
    emap1 = calc_emap(LH_1, HL_1, HH_1)
    emax1 = calc_emax(emap1, 8)

    # i = 2
    LL_2, LH_2, HL_2, HH_2 = haar_wavelet_transform(LL_1)
    emap2 = calc_emap(LH_2, HL_2, HH_2)
    emax2 = calc_emax(emap2, 4)

    # i = 3
    LL_3, LH_3, HL_3, HH_3 = haar_wavelet_transform(LL_2)
    emap3 = calc_emap(LH_3, HL_3, HH_3)
    emax3 = calc_emax(emap3, 2)

    return emax1, emax2, emax3
```

Figure 49: Method representation of proposed algorithm

```
def calc_emap(LH, HL, HH):
    """
    Construct edge map
        LH: (array) vertical detail
        HL: (array) horizontal detail
        HH: (array) diagonal detail
    """
    # square root of LH^2 + HL^2 + HH^2
    return np.sqrt((np.power(LH, 2) + np.power(HL, 2) + np.power(HH, 2)))


def calc_emax(emap, window_size):
    """
    Calculate local maxima of each edgemap partition
        emap: (array) edge map
        window_size: (int) dimensions of each partition
    """
    # split edge map into partitions sized (window_sized X window_sized)
    result = partition(emap, window_size, window_size)
    max_points = []

    # from 0 to the total number of partitions
    for i in range(0, len(result)):
        # find the max of each partition and
        # add to total list of local maxes
        max_points.append(np.max(result[i]))

    # return list of all local maxes
    return max_points
```

Figure 50: Method representations of emap and emax calculations

To achieve project-specific desirable results, we had to define three parameters: *EDGE_THRESH*, *MIN_ZERO* and *FIXED_SIZE*. The former two are renamed parameters, *threshold and MinZero*, from the original paper, while *FIXED_SIZE* is the standard resizing width for all input images. From extensive testing and tweaking, we settled on the following values:

- *EDGE_THRESH*: 35
- *MIN_ZERO*: 0
- *FIXED_SIZE*: 1024

By setting blur classification requirements for the purposes of our project, we were able to achieve the desired experimental results. Because images are removed from our application process if they are deemed blurry, we opted to be lenient with our specifications. For large images that were resized to the original *FIXED_SIZED* parameter, if the calculated *per* was equal to *MIN_ZERO* with a calculated *blur_extent* greater than 82.5% then the image was classified as blurry. For images resized to smaller dimensions due to small original input image dimensions, if the calculated *per* was equal to *MIN_ZERO* with a calculated *blur_extent* greater than 75% then the image was classified as blurry.

```
def blur_result(size, per, blur_extent):
    """
    Classify whether or not an image is blurry. Blur metrics vary based
    on resize dimensions
        size: (int) resized dimension
        per: (float) ratio of Dirac- and Astep-Structure to all edges
        blur_extent: (float) blur confident coefficient;
                             how many Roof- and Gstep-Structure edges are
    """
    if size == FIXED_SIZE:
        if (per <= MIN_ZERO and blur_extent > .825) or blur_extent >= .95:
            # blurry
            return 0
        else:
            # not blurry
            return 1
    else:
        if (per <= MIN_ZERO and blur_extent > .75) or blur_extent >= .9:
            # blurry
            return 0
        else:
            # not blurry
            return 1

    # preventive measure
    # default to not blurry result
    return 1
```

Figure 51: Method to determine whether or not an image is blurry based on calculated values

Figure 52.1: Example of an image of undesirable quality

Figure 52.2: Example of an image that would be accepted

```python
def calc_hash(img):
    """
    Calculate the wavelet hash of the image
        img: (ndarray) image file
    """
    # resize image if height > 1000
    img = resize(img)
    return imagehash.whash(Image.fromarray(img))


def compare(hash1, hash2):
    """
    Calculate the difference between two images
        hash1: (array) first wavelet hash
        hash2: (array) second wavelet hash
    """
    return hash1 - hash2
```

Figure 53: Methods utilizing imagehash library to implement image comparison

```python
def limit(img, std_hash, count):
    """
    Determine whether image should be removed from image dictionary in main.py
        img: (ndarray) image file
        std_hash: (array) wavelet hash of comparison standard
        count: (int) global count of images similar to comparison standard
    """
    # calculate hash for given image
    cmp_hash = calc_hash(img)

    # compare to standard
    diff = compare(std_hash, cmp_hash)

    # image is similar to standard
    if diff <= DIFF_THRES:
        # if there are 3 similar images already, remove image
        if count >= LIMIT:
            return 'remove'

    # non-similar image found
    else:
        # update comparison standard
        return 'update_std'

    # else continue reading images with same standard
    return 'continue'
```

Figure 54: The actual limit method of our application to reduce similar images

Cropping

The final stage of the application is the image manipulation state in which filtered images are cropped in such a way that emphasizes any birds detected. This is done using the bounding boxes outputted by the object detection stage.

The algorithm first calculates the center of the image from its dimensions. Then, given a bounding box for a bird from the set of bird face boxes returned from the detection stage, determine the center of that particular box and calculate the distance between the box center and the center of the image. If the center of this box is closer to the image center than all preceding boxes, then store the box center as the most central face box. Then, determine the extrema of all bird bounding boxes. From those extrema, calculate the difference between the largest and smallest extrema for both the x- and y-components. These differences are then multiplied by the square root of a scaling factor in order to enforce the inclusion of some of the surrounding environment in the final image. The resultant height and width are then divided in half to calculate the amount that needs to be added and subtracted from the most central face coordinates to determine the final crop area dimensions. Before cropping the image, the crop dimensions are checked to ensure that they are within the bounds of the original image. Finally, the image is cropped and written to disk.

Through the use of this algorithm, we enforce an aesthetic style in final images that centers the viewpoint around the eye of a bird, regardless of the amount of birds in the original image. In images with a single bird, the algorithm will use the center of the face bounding box, which serves as an approximation for the eye location, as the center for the final image. It then scales the area of bird bounding box by a scaling factor in order to preserve an area of the environment and uses this new scaled area as the viewpoint of the final image. In images with multiple birds, the algorithm selects the bird bounding box coordinates that are closest to each of the original image boundaries as the dimensions of the salient area. After the determination of this area, the image is manipulated using the same process as single-bird images.

```python
# UCF Senior Design 2017-18
# Group 38

from PIL import Image
import math
import numpy as np

SCALING_FACTOR = 3


def man(boxes, image_array, landscape=True, scaling_factor=SCALING_FACTOR):
    """
        Crop and manipulate the image for final output.
            image_array:    (Array) array representation of the image
            boxes:          (Dict) bounding box around subject;
                            form: (ymin, xmin, ymax, xmax)
            scaling_factor  (Integer) the amount by which to scale the
                            bounding box of the object; this is done as a way
                            to include some of the environment in the final image
    """
    image = Image.fromarray(image_array)
    width, height = image.size
    img_center_x = width / 2

    dist_face_center_x = math.inf
    central_face_x, central_face_y = 0, 0

    for i in boxes['faces']:

        # Get the bounding box coordinates of the face.
        fb_xmin, fb_ymin, fb_xmax, fb_ymax = i

        # Calculate the center of the face bounding box.
        fb_center_x, fb_center_y = (
            fb_xmax + fb_xmin) / 2, (fb_ymax + fb_ymin) / 2

        # Calculate the distance of the x-component of the
        # the face box centerfrom the center of the image.
        delta_center_x = math.fabs(fb_center_x - img_center_x)

        # Get the closest face box center coordinates over all face boxes.
        if delta_center_x < dist_face_center_x:
            dist_face_center_x = delta_center_x
            central_face_x, central_face_y = fb_center_x, fb_center_y

    # Initialize bounding box extrema.
    sm_xmin, sm_ymin, lar_xmax, lar_ymax = math.inf, math.inf, 0, 0

    # Calculate the factor by which we multiply the width and height of box.
    factor = math.sqrt(scaling_factor)

    # Get the extrema of the bounding boxes returned from the detection graph.
    for i in boxes['birds']:
        sm_xmin = min(sm_xmin, i[0])
        sm_ymin = min(sm_ymin, i[1])
        lar_xmax = max(lar_xmax, i[2])
        lar_ymax = max(lar_ymax, i[3])

    # Calculate the width and height of the final crop area.
    bb_width, bb_height = lar_xmax - sm_xmin, lar_ymax - sm_ymin
    new_width, new_height = round(
        bb_width * factor, 0), round(bb_height * factor, 0)

    if landscape:
        if new_width * 1.5 < new_height or math.fabs(new_width - new_height) < new_height * .5:
            new_width = new_height * 1.5

    # Calculate the amounts by which to adjust the face_box coordinates.
    width_diff, height_diff = new_width / 2, new_height / 2

    # Set the new dimensions for the final crop box.
    final_xmin, final_xmax = central_face_x - \
        width_diff, central_face_x + width_diff
    final_ymin, final_ymax = central_face_y - \
        height_diff, central_face_y + height_diff

    # Edge case handling.
    if final_xmin < 0: final_xmin = 0
    if final_xmax > width: final_xmax = width
    if final_ymin < 0: final_ymin = 0
    if final_ymax > height: final_ymax = height

    # Crop and attempt to save image.
    cropped_area = image.crop((final_xmin, final_ymin, final_xmax, final_ymax))

    try:
        final_image = np.asarray(cropped_area)
        return final_image, True
    except IOError:
        print("File could not be written properly.")
        return False
```

Figure 55: The crop method from our image class

User Interface

Kivy

For the purposes of our application, we opted to create our graphic user interface (GUI) by using Kivy's user interface (UI) library. While Electron provides an effective framework for UI development, it's more suitable for the creation of web applications and requires the knowledge and use of Javascript, HTML and CSS. Given that our backend functionality is written entirely in Python and intended for offline access and use, Kivy is an attractive choice; the library is suitable for standalone applications, compatible with Python, easily packaged into a single executable for multiple platforms, e.g. Windows, macOS and Linux. User interface elements can be declared with the main driver of the program or a separate *.kv* file in Kivy's custom language very similar to Python. User interface screens are declared as classes and can be manipulated through different methods and attributes in the same manner as any other Python script.

```
# config.kv should not implement any screen manager stuff as it
# overrides any definitions in this file, and cause a lot of strife
Builder.load_file("/Users/ayylmao/Desktop/knest/assets/config.kv")

# Create the screen manager
sm = ScreenManager(transition=FadeTransition())
sm.add_widget(LandingScreen(name='landing'))
sm.add_widget(FolderSelectScreen(name='folder_select'))
sm.add_widget(BlackScreen1(name='black1'))
sm.add_widget(BlackScreen2(name='black2'))
sm.add_widget(ProgressScreen(name='progress'))
sm.add_widget(CompareScreen(name='compare'))
sm.add_widget(BlackScreen3(name='black3'))
sm.add_widget(WriteScreen(name='write'))
sm.add_widget(BlackScreen4(name='black4'))
sm.add_widget(ProcessScreen(name='process'))
sm.add_widget(EndScreen(name='end'))
```

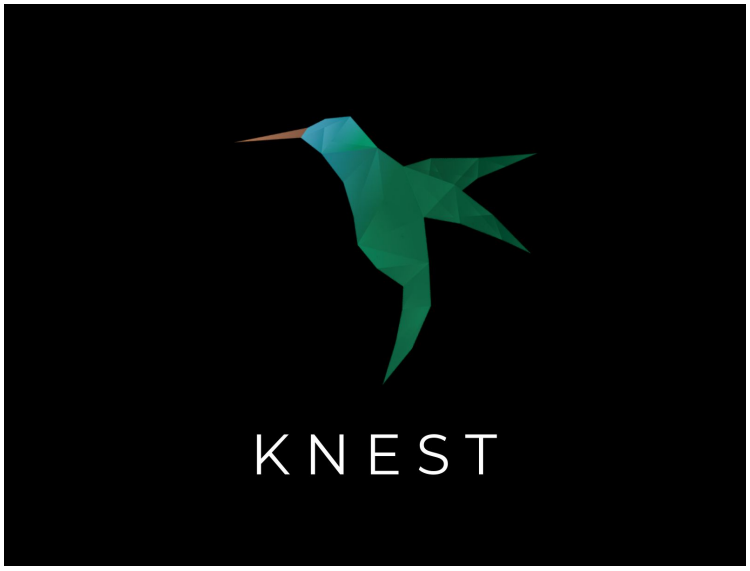
Figure 56: The .kv, which holds interface elements, and interface screens/transitions can be declared in the main driver

```
if __name__ == '__main__':
    BirdApp().run()
```

Figure 57: The application can be initiated from a single line of code

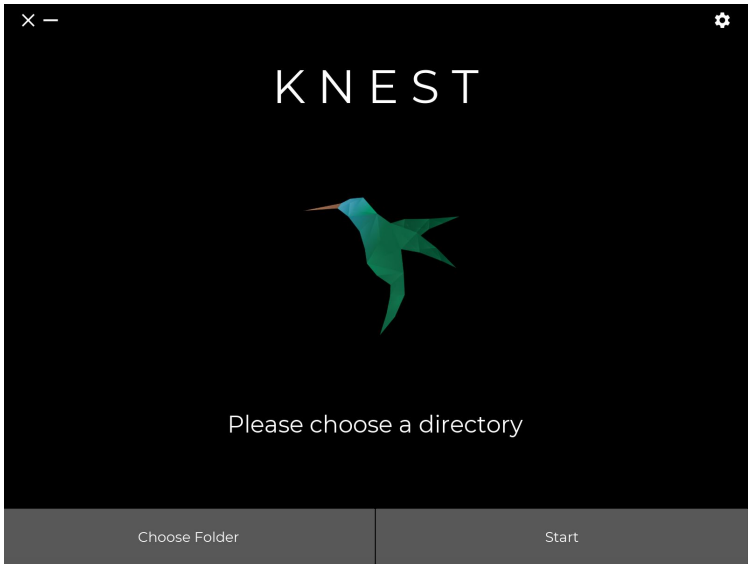
```
#:import Utils kivy.utils
#:import gv utils.global_var
#:import os os
#:import copy copy

<LandingScreen>:
    Button:
        background_color: (0.0, 0.0, 0.0, 0.0)
        on_release: app.root.current = 'folder_select'

    Image:
        allow_stretch: True
        keep_ratio: True
        pos_hint: {'center_x': 0.5, 'center_y': 0.60}
        size_hint_y: None
        height: dp(350)
        source: '/Users/ayylmao/Desktop/knest/assets/color_bird.png'

    Label:
        pos_hint: {'center_x': 0.5, 'center_y': 0.2}
        font_size: dp(60)
        font_name: '/Users/ayylmao/Desktop/knest/assets/Montserrat-Regular'
        text: 'K N E S T'

<FolderSelectScreen>:
    on_enter: gv.fs = self
    on_leave: self.ids.path.text = "Please choose a directory"

    Button:
        size_hint: (.025, .035)
        pos_hint: {'center_x': .965, 'center_y': .967}
        background_normal: ''
        background_color: (0, 0, 0, 0)
        on_press: Factory.AdvancedSettings().open()

        Image:
            source: '/Users/ayylmao/Desktop/knest/assets/settings.png'
            allow_stretch: False
            keep_ratio: True
            size: self.parent.size
            pos: self.parent.pos


    Button:
        size_hint: (.03, .04)
        pos_hint: {'center_x': .035, 'center_y': .967}
        background_normal: ''
        background_color: (0, 0, 0, 0)
        on_press: app.stop()
```

Figure 58: An example of Kivy's custom language to declare user interface elements



The figure on the left displays the splash screen of our final application. It boasts a minimalistic design and simply displays the application name and logo. The screen is set to appear for three seconds as a swift introductory transition to our base application. However, the user may circumvent this period of idleness by simply clicking anywhere on the screen; this will transition the application instantly to the next screen, i.e. the primary user screen.

Figure 59: The landing/splash screen of our application



The figure on the left displays the application's main user screen, where the user may select which folders they wish to process, modify the settings of their final processed subdirectory and start the actual application process. Visually, this screen is an extension of the landing screen; the application name and logo is centralized within the page. Additionally, the application prompts the user to choose a directory. Once the user has finished selecting and loading the folders they wish to process, this

Figure 60: The main user screen of our application

string display will update with their list of chosen folders, like the figure on the left below. Otherwise, if the user selects the button to start the application without selecting any folders, the string display will update to let the user know that no directory path was given like the figure on the right below.

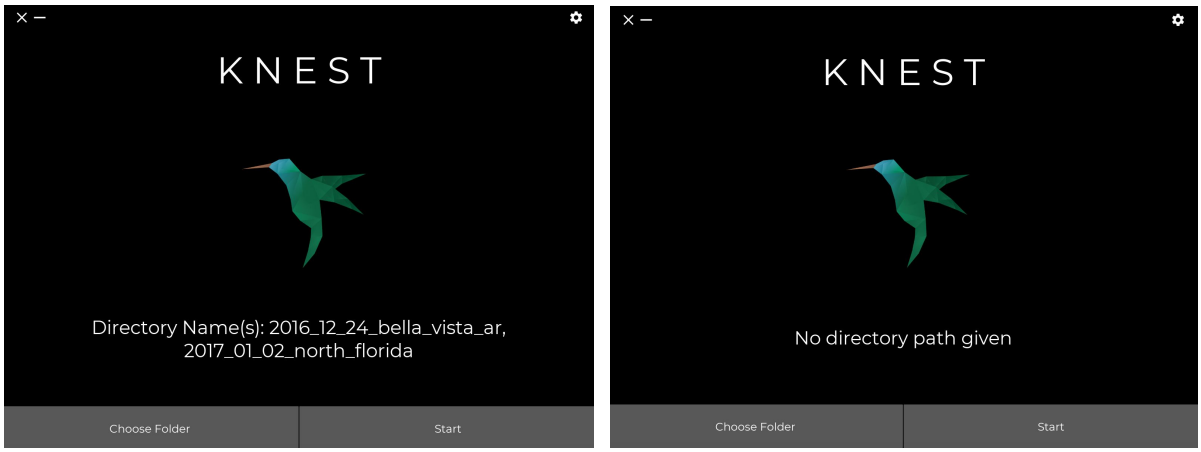


Figure 61: The main user screen updates the string display based on user actions; on the left, the figure displays the list of selected folders and on the right, the figure displays that no folder has been selected

At the top left of the window is the window adjustment and termination options; these options will not disappear and allow the user to exit the application (by pressing the x icon) and minimize the window (by pressing the minimize icon).

The user may modify certain settings of the application to specify the format of their final subdirectory of processed images. They may do so by selecting the gear icon at the top right of the screen. From the popup that appears, the user may toggle whether or not they wish the application to reduce the amount of similar images, crop the images and if so, crop in landscape orientation. If the switch for cropping is turned off, the switch for landscape orientation will be disabled. These options are all enabled at start of application.

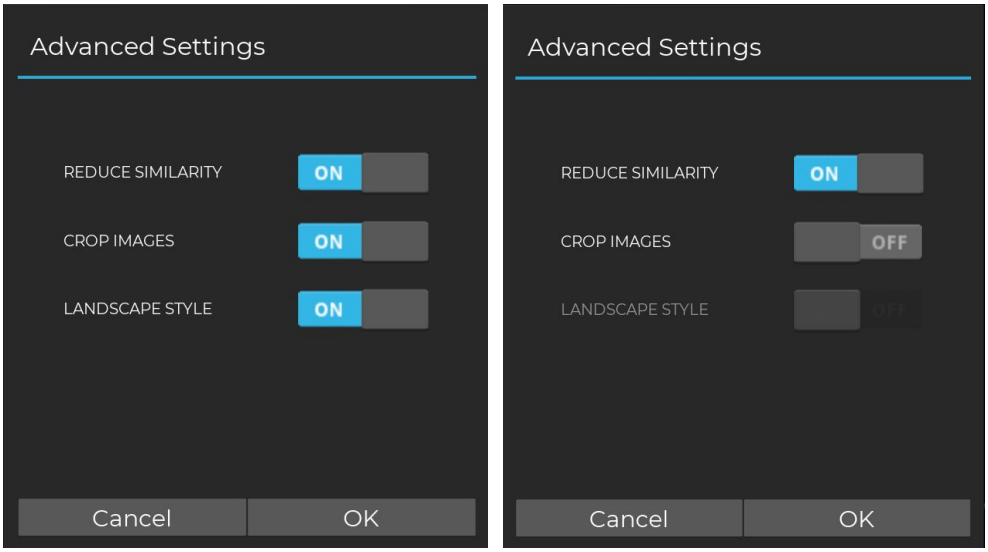


Figure 62: Popup that allows user to modify certain advanced settings

The user may begin selecting which folders they wish to process by selecting the “Choose Folder” button. On press, a popup will appear on the screen that lists the user’s entire folder selection. Users may select multiple folders, up to a maximum of eight and may only exit from the popup by pressing “Cancel” or “Load.” If no folders are selected, the “Load” button is disabled; likewise, if eight folders have been selected, the “Add” button is disabled.

Additionally, to connect user action to application performance, buttons are automatically set to change color on press. The initial and default color of a button is gray and is changed to blue upon selection.

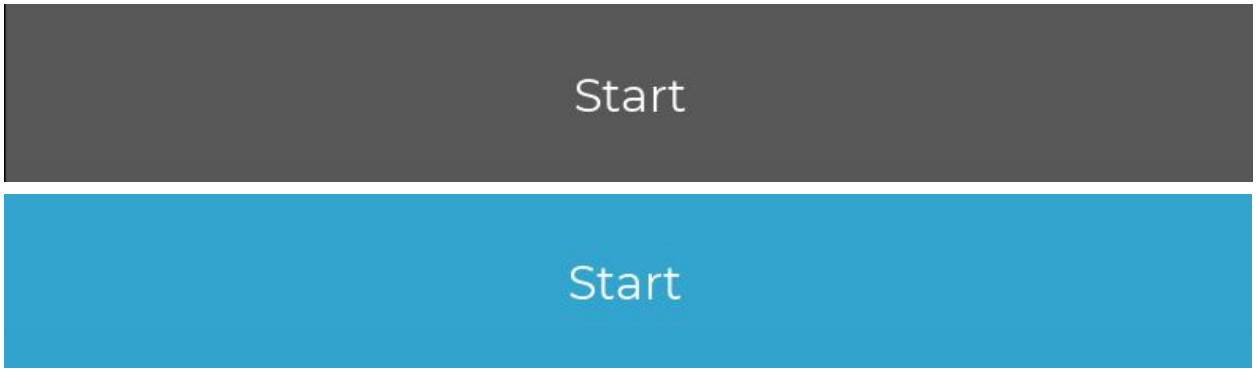


Figure 63: Inactive button (top) and pressed button (bottom)

The selections are displayed for the user on the right of the popup in the form of a checklist. If the user wishes to remove a folder, they can simply uncheck it and it will be promptly removed.

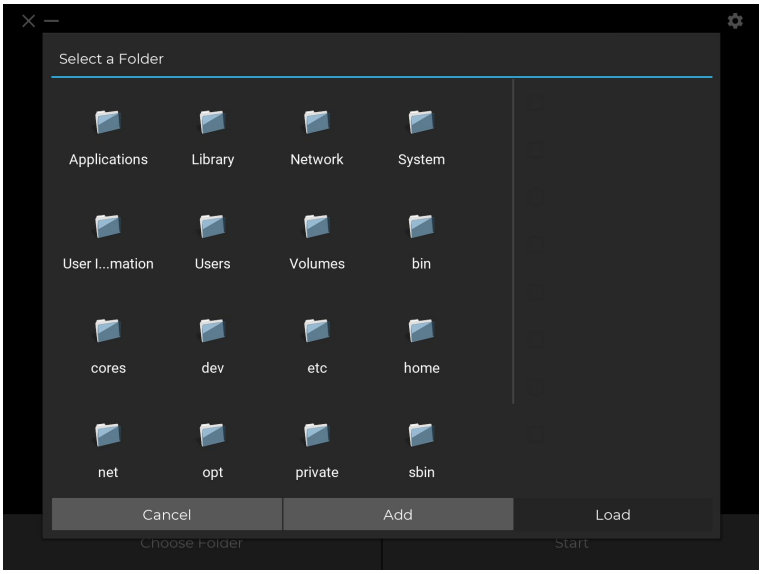


Figure 64: The folder selection popup with no queue of folders added

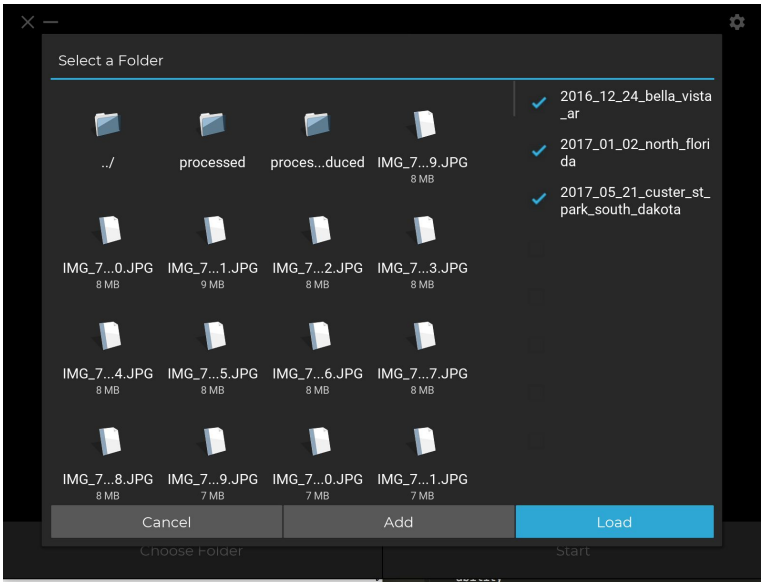


Figure 65: The folder selection folder with a three user-selected folders displayed

Exception handling is built throughout the application and error messages will be displayed for the user upon unexpected or unsupported behavior. This will prevent the application from quitting unexpectedly and alleviate user uncertainty about issues that arise while running it. The application will respond to different errors in different manners, i.e. for invalid folder selection, the user will be prompted to choose another folder and for an out-of-index exception, the user will be notified that all processes were aborted and prompted to select their folders again.

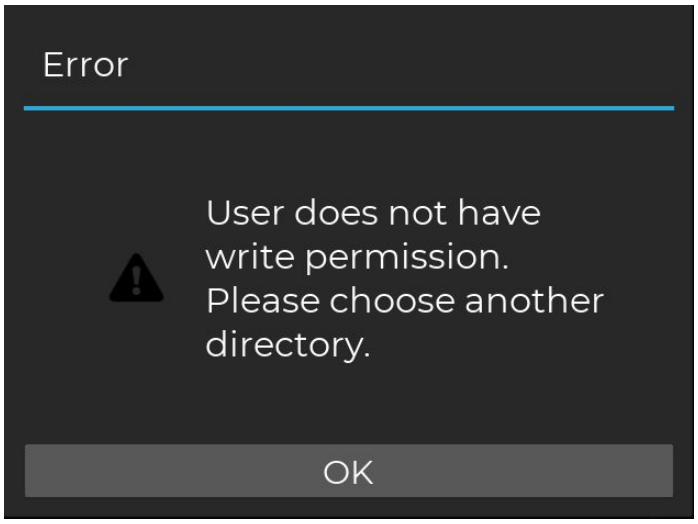


Figure 66: *Popup message of a write permission denied error*

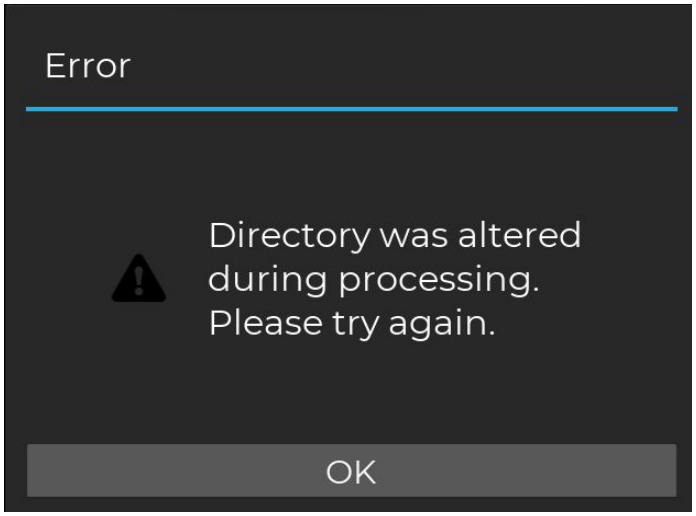


Figure 67: Popup message for an out-of-index error

After the user selects the folders they wish to process, pressing the “Start” button will initiate the primary functionality of the application. At the start of the process, we need to allow sufficient time for the object classification and object detection models to load. To alleviate user uncertainty during this stage of the application, a screen will appear that informs the user that the models are being initialized. The screen is minimalistic and only displays the application logo and message. If there are multiple folders in the queue to be processed, this screen is reused and appears in between folders. The message, “Initializing model,” however, is only displayed once when the models are being loaded. Thus, in other appearances of this screen, only the application logo is displayed.

Figure 68: Transition screen while models are being loaded

Our entire application process is comprised of multiple steps: blur detection, bird classification, bird localization, then optionally image comparison and manipulation and image copying. For a large dataset of images, this process could potentially take hours to complete. To maximize user experience, we must implement some type of intermediate processing animation or display that will alleviate user uncertainty about whether the application is running, etc. As such, we opted to display real-time image processing to the screen for the user to see as the application runs. The user may then see their input directory, its contents and status through every step of the process. For blur detection, bird

The initial processing step of the application is blur detection; images are read in from the user-selected directory and added to memory based on whether or not it is blurry. In our UI, we display the image as it's being processed and the final result once blur detection is successful. If an image is blurry, a red 'x' will be displayed in the center of the screen, otherwise, if it is non-blurry, a green check will be displayed.

Figure 69: Display example of an image classified as blurry

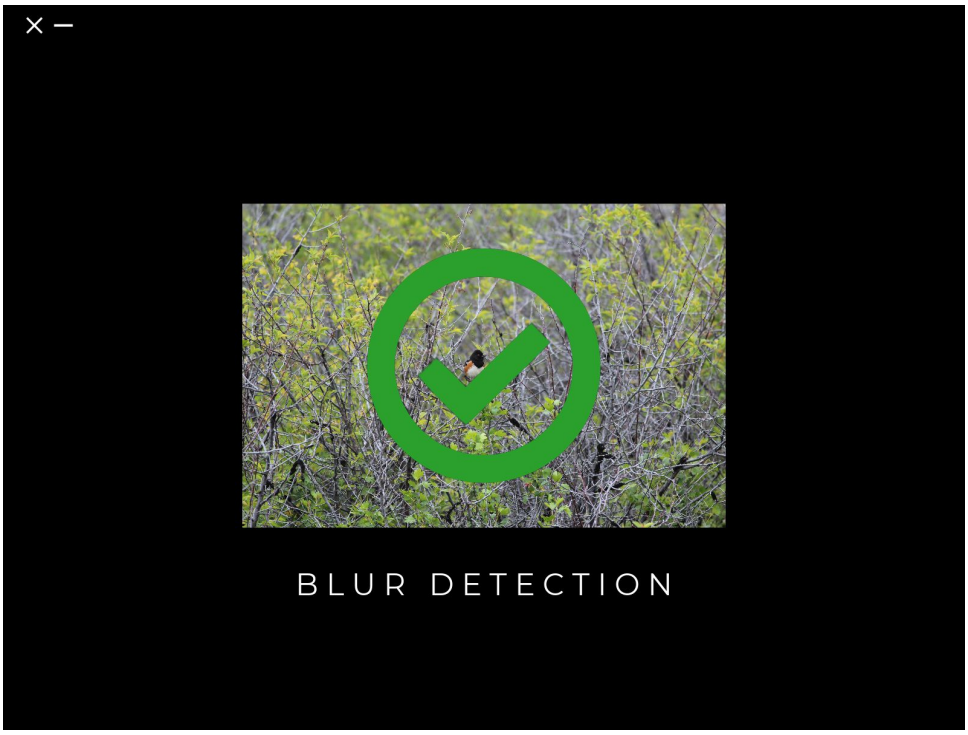


Figure 70: Display example of an image classified as non-blurry

The user interface display for bird classification is done in a similar manner. In our UI, if an image does not contain a bird, a red ‘x’ will be displayed in the center of the screen, otherwise, if it does contain a bird, a green checkmark will be displayed.

Figure 71: Display example of an image determined to contain a bird

The processing animation for bird localization is slightly different from blur detection and blur classification. The UI display is split into two sections: “Last Processed” on the left and “Currently Processing” on the right. As denoted, the last image processed will be displayed on the left with its detection results, which will either have bounding boxes if birds or bird faces are detected or a red ‘x’ if neither could be found.

Figure 72 and 73: Examples of bird localization UI display

Once blur detection, bird classification and bird localization are complete, depending on user preference for reducing similar images, the current screen should fade to display image comparison progress. The screen is minimalistic and displays a thin progress bar in the center of the screen. Once an image has been processed, the bar will update and the percentage complete will be displayed beneath.

Figure 74: The image comparison UI screen

The following screen (or immediate screen after blur detection, bird classification and bird localization if the comparison option was switched off by the user) would be the writing progress screen. This screen is very similar to the image comparison progress screen; the label, progress bar and percent complete information is central to the window. As each image accepted by our application is copied and written into a subfolder called “processed,” the bar will update and the percentage complete will be displayed beneath.

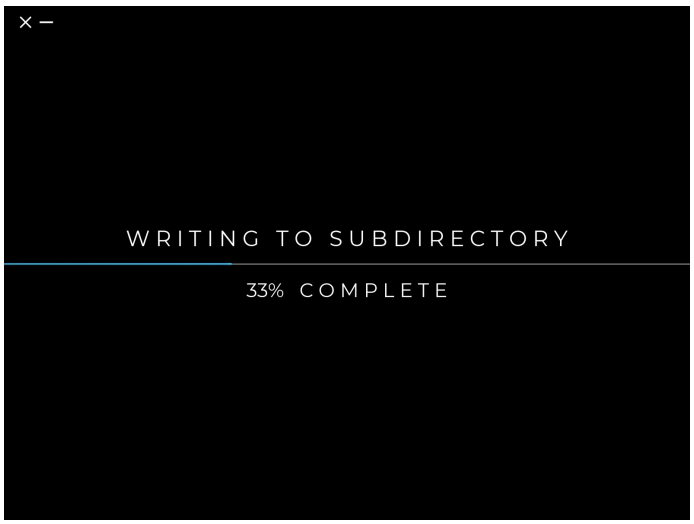


Figure 75: The writing UI screen

Once our application has successfully processed every user-selected directory in the queue, the user will be notified that the process is complete. For our user interface, this notification is in the form of a final screen, which displays a grayscale version of the application logo and a “Process complete” message. The user may process more folders again by clicking anywhere on the screen and will be promptly directed back to the folder selection screen.

Figure 76: Interactive screen where user is notified that the application is finished processing their images

Version Control

The group is using Git in order to track changes in files and code structure. It also allows for multiple people to contribute to a project by allowing for nonlinear development through branching and merging. Each user in a project is given a copy of the full development history and any changes (commits) are given cryptographic hashes. The hashing serves two purposes: it enables a user to easily refer back to a prior version of a file in the event of an unrecoverable issue, and it prevents undetectable changes to the code as any change to the source would cause the new hash to be different than the old hash.

We are using Github as the implementation of Git in this project. Github was chosen for its ease of use and its visualization of many core Git functions. It allows us to easily view differences in code and improves branch tracking by creating graphs that show where a branch was created and merged. It also adds a social aspect to issues and pull requests by allowing for extended discussion of contributions without the need to set up a mailing list. The code is currently stored in a free private repository that belongs to the project manager. It may be publicly available after the conclusion of the project.

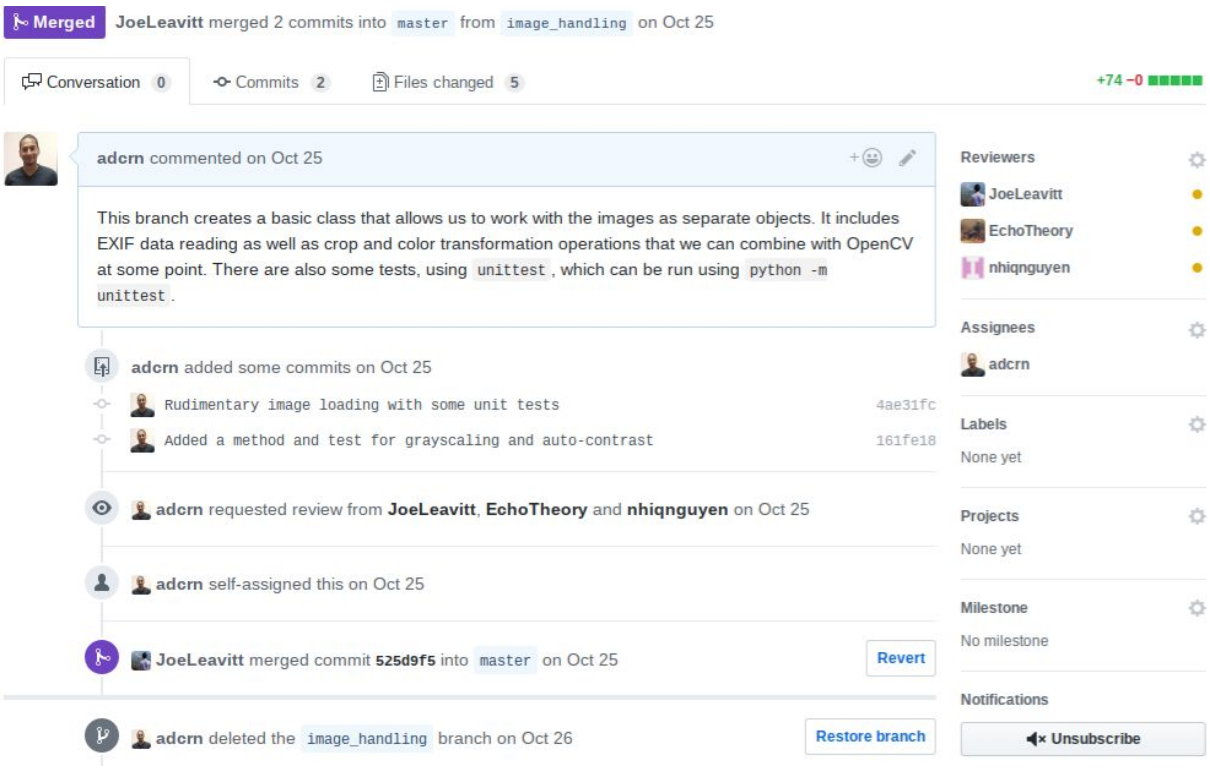


Figure 77: Example of a pull request from our repository

Testing

Unit Testing

We make use of unit testing to ensure the proper completion of the algorithm and to make sure that our code works properly at both pre- and post-commit to our repository. For any of the methods that are not already provided by one of the modules that we are using, a unit test is created that ensures the method is returning data correctly. The unit tests are written in the syntax required by the built-in Python *unittest* module. Prior to adding more code to any of the modules, all unit tests are run to make sure that all methods are working properly. Furthermore, after the code has been added, all unit tests are run once more in order to ensure that prior functionality has not been harmed. If a test case does not return the expected output, we can return to the precise method that failed the test and fix the issue as opposed to aimlessly changing methods until all the methods work properly.

```
class ImageTestCase(unittest.TestCase):
    """Tests for `pill.py`."""

    def setUp(self):
        self.img = Pill(TEST_PHOTO)

    def tearDown(self):
        self.img.pill.close()

    def test_pil_image_object_creation(self):
        """Is a test photo successfully opened as an image file?"""
        self.assertTrue(isinstance(self.img.pill, Image.Image))

    def test_existence_of_exif_data(self):
        """Is EXIF metadata being pulled out correctly?"""
        self.assertTrue(isinstance(self.img.exif_tags, dict))

    def test_cropping(self):
        box = (0, 0, 100, 100)
        self.img.crop_to_subject(box)
        self.assertEqual(self.img.pill.size, (100, 100))

    def test_gray_contrast(self):
        self.img.gray_contrast()
        self.assertTrue(isinstance(self.img.gray_con, Image.Image))
```

Figure 78: Test cases for our image handling module

Model Validation

Dr. Leavens provided a significant amount of images, both with and without birds, for use in training our model. A large part of this collection of images will serve as the training set for the neural network. The images in this set (along with their class labels) are used by the network to adjust the weights of certain neurons and features. The Leavens dataset was given to us without labels, so we will spend a fair amount of time preparing a separate file with the names and class labels (0 - not bird, 1 - bird) of each of the images. This ensures that we have the expected output for each training image, which is imperative for training the model.

During training, it is useful to be able to determine how well a model is predicting inferences. For this purpose, a validation set should be constructed for use. This can be done manually or done programmatically by randomizing a subset of the overall image set before training. This subset is then used to minimize overfitting of the model. The weights of the network are not being adjusted using this subset; the purpose is to verify that any increase in accuracy in the training set also leads to an increase in accuracy in a set of images that the network has not yet seen. If there is an increase in accuracy in regards to the training set, but the accuracy stays the same or decreases, then the network is actually overfitting. Once this starts to happen, training should be terminated and adjustments may be needed to the network in order to prevent overfitting in the future.

Explicit Design Summary

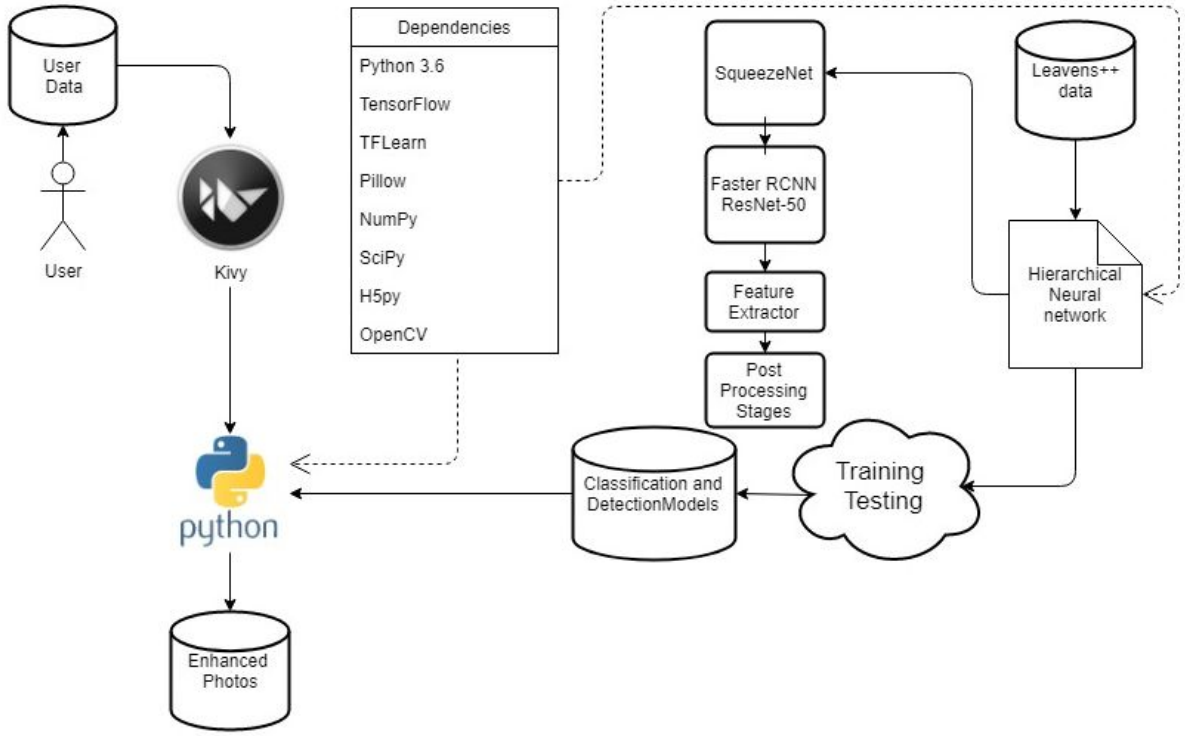


Figure 79: High level design.

Diagram outline:

- User interaction with the Electron app.
- Two-way communication between the Electron app and the JavaScript.
- Two-way communication between the Python code and the JavaScript.
- The Electron app running the Python code as a sub-process.
- The hierarchical neural network is encapsulated within the Python code in the Electron app.
- The dependencies for the hierarchical neural network.
- Feeding the training and testing data into the hierarchical neural network .
- The convolutional neural network for bird classification.
- The faster region-based convolutional neural network for detecting multiple birds.
- The siamese neural networks that run in parallel to detect if the eye is visible and if the bird is in-flight.
- Training and testing the neural network and producing the classification model.
- Loading the model into an identical hierarchical neural network on the user's machine.
- Selecting and enhancing good photos and producing the final picture.

The diagrams below were created from Tensorflow's Tensorboard that can generate graphs of the layers created with both Tensorflow and TFLearn. These diagrams help debug and understand what processes are taking place at every step of the program. Everything passed along these graphs are in tensors, which are multidimensional arrays that can be of varying sizes and reshaped to different lengths. These diagrams show the standard process for each layer, which each can be modified in their own way.

The design of the convolutional layer begins with the convolution operation by applying the weights to the input of a 4D tensor as shown in figure 47. After the bias is added to the tensor with its own weights and they are passed to the activation function, that levels off low values in the tensor. This tensor is then outputted to the next layer.

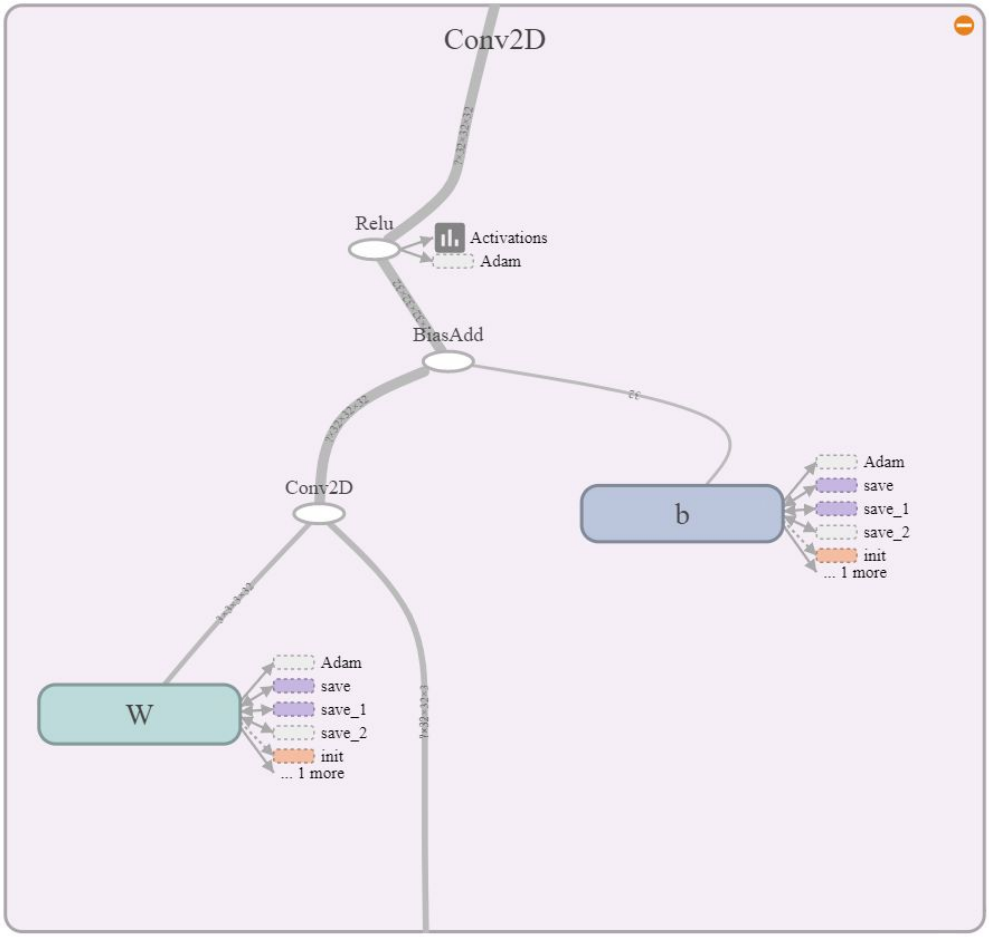


Figure 80: Example of a Conv2D tensor

The next diagram shows the standard process for fully connected layers in Tensorflow. As figure 48 shows, this layer begins by reshaping the input to a 2D tensor to prepare it for matrix multiplication with the weights of the neural network. Next, the bias is added to the tensor and this value is passed to the activation function. The output is also a 2D tensor.

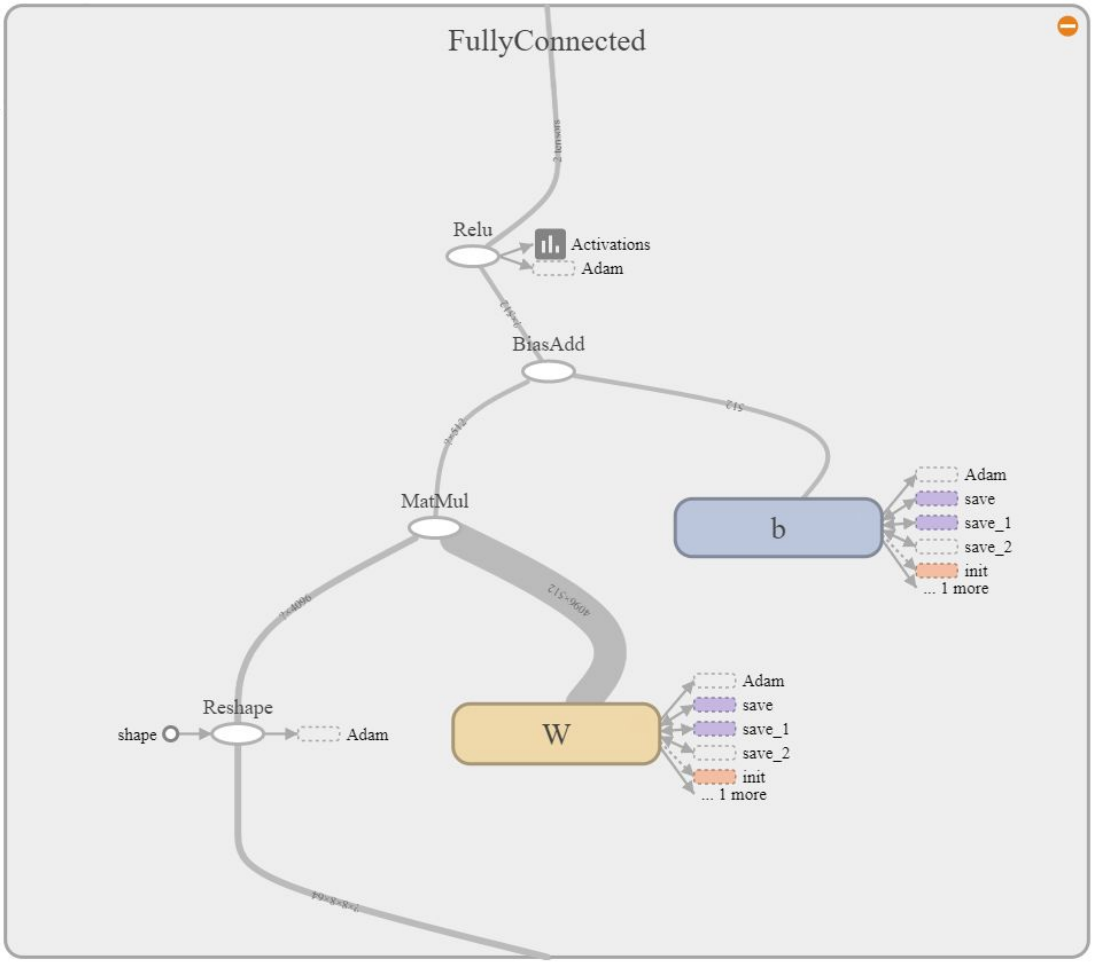


Figure 81: Example of a fully connected layer

The cross-entropy layer is the calculation of the cost function or loss. There are two inputs coming into this layer, the labels of the training data and the classification predictions from the neural network.

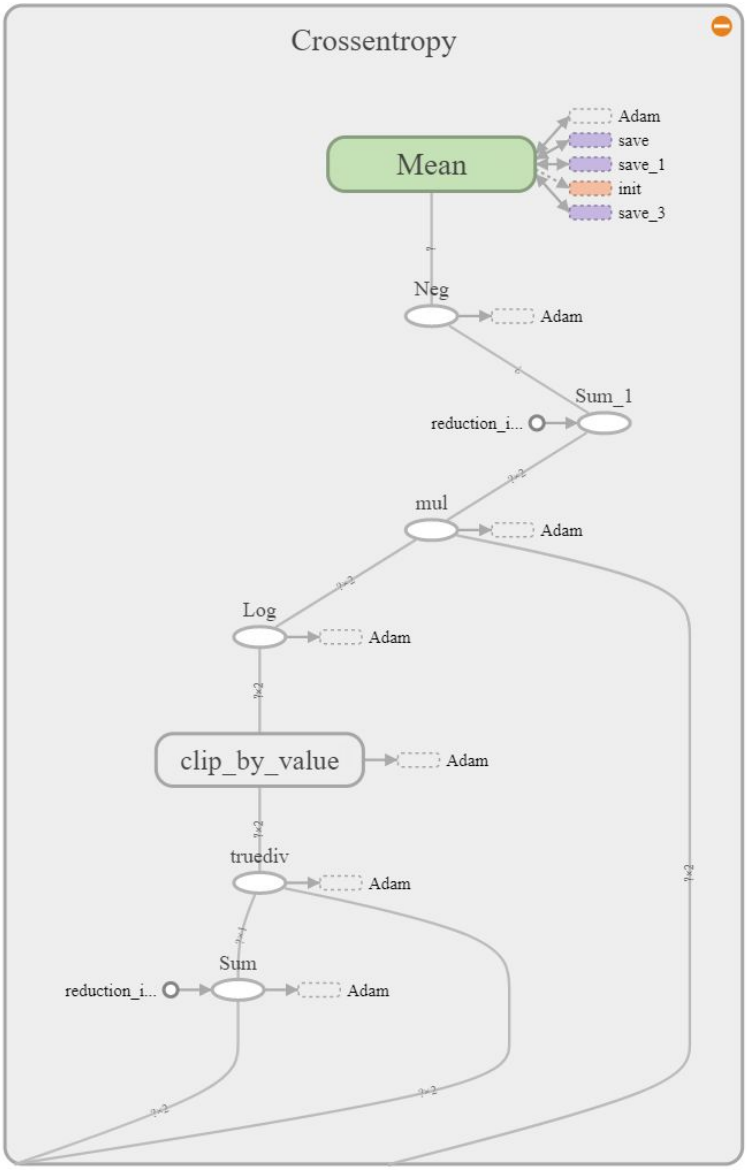


Figure 82: Example of a cross-entropy layer

The accuracy layer works in parallel with the cross-entropy layer, and has the same inputs of the classification predictions from the neural network and the labels from the training data. These values are selected to determine the highest value for each prediction to return what inference the network has made for the object. These values are then compared to see how many are equal and averaging the amount correct gets us our accuracy value.

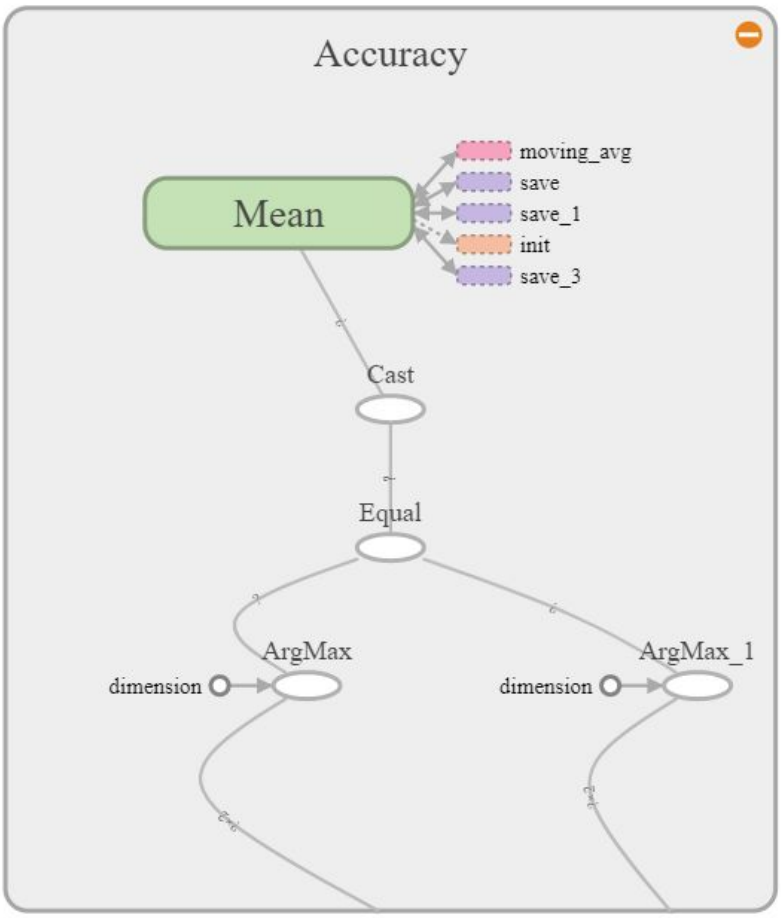


Figure 83: Example of an accuracy layer

Initial Activity Diagram

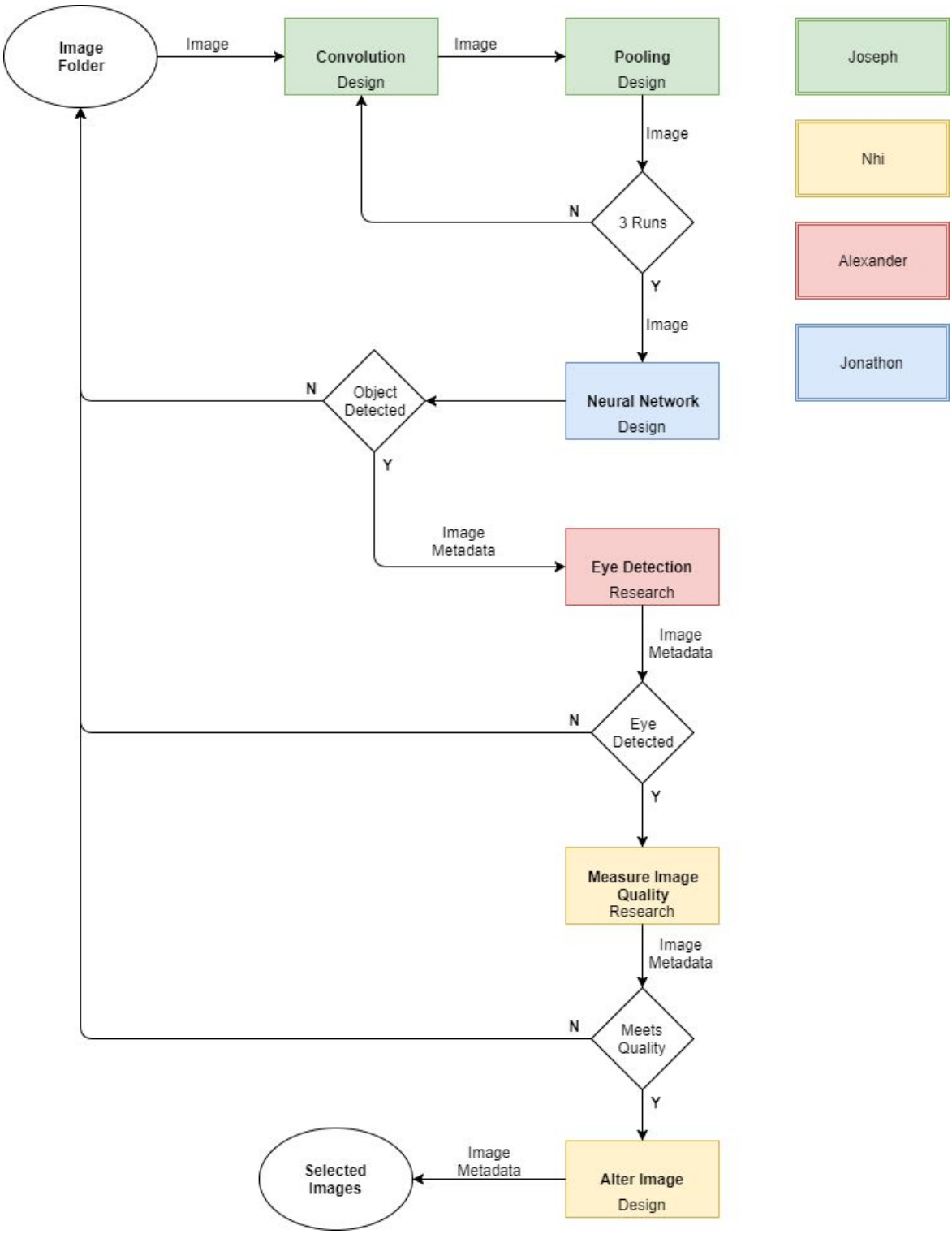


Figure 84: Activity diagram

Prototype

Below in figure 52 is a graph of a prototype we have already built for image classification. Figure 53 shows the code corresponding to the graph using TFLearn. This prototype is already able to classify many of Dr. Leavens's bird images but lacks the capacity to find the location of the birds. With a current accuracy of around ninety-two percent, it shows how powerful neural networks are in even a short time of training. This prototype shrinks images down to 32x32 pixels, the size of CIFAR-10 data, and then processes them.

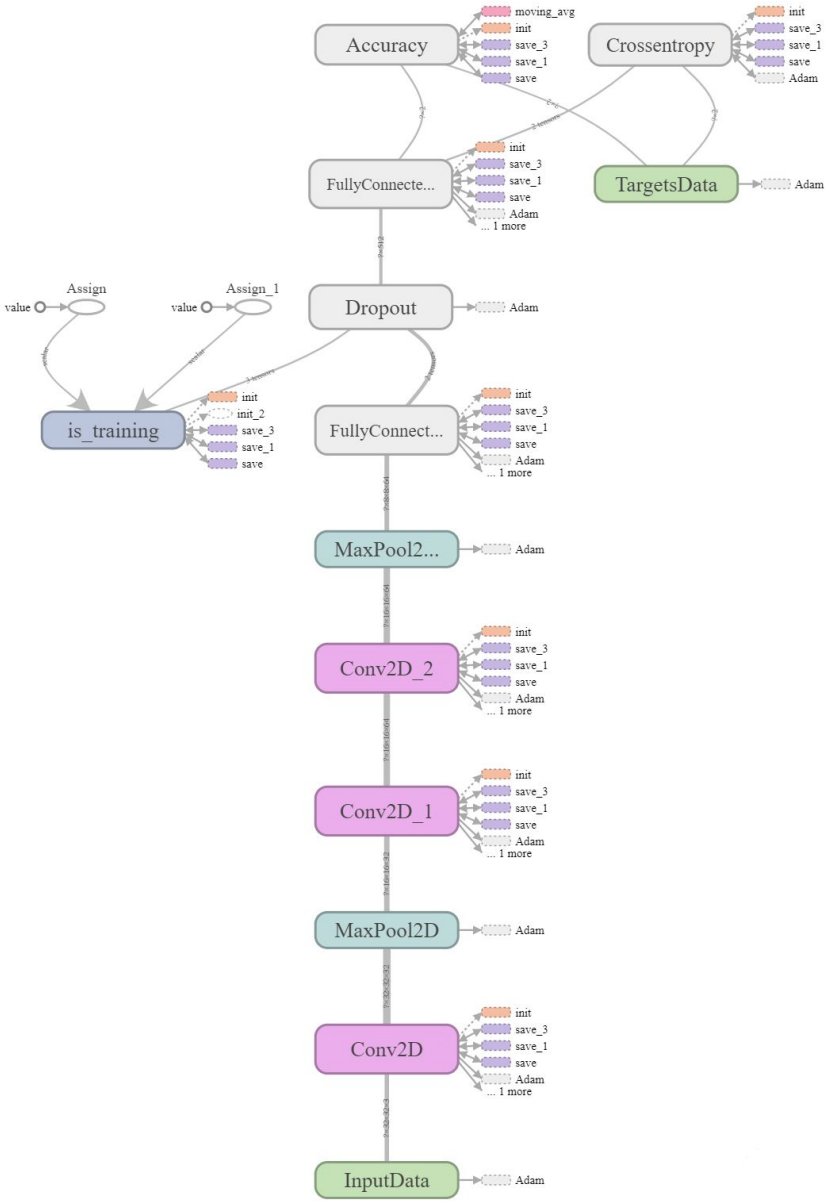


Figure 85 : Graph of prototype code, this shows the sequence of layers.

```
preprocess = ImagePreprocessing()
preprocess.add_featurewise_zero_center()
preprocess.add_featurewise_stdnorm()
augment = ImageAugmentation()
augment.add_random_flip_leftright()
augment.add_random_rotation(max_angle=25)

cnet = input_data(shape=[None, 32, 32, 3],
                  data_preprocessing=preprocess, data_augmentation=augment)
cnet = conv_2d(cnet, 32, 3, activation='relu')
cnet = max_pool_2d(cnet, 2)
cnet = conv_2d(cnet, 64, 3, activation='relu')
cnet = conv_2d(cnet, 64, 3, activation='relu')
cnet = max_pool_2d(cnet, 2)
cnet = fully_connected(cnet, 512, activation='relu')
cnet = dropout(cnet, 0.4)

cnet = fully_connected(cnet, 2, activation='softmax')
cnet = regression(cnet, optimizer='adam',
                  loss='categorical_crossentropy',
                  learning_rate=0.001)

model = tflearn.DNN(cnet, tensorboard_verbose=3,
      checkpoint_path='/training6_epoch_5/bird-classifier 5.tfl')
model.load("training6_epoch_5/bird-classifier 5.tfl")
```

Figure 86 : Code of our prototype bird classifier. It currently reaches ~90% validation accuracy.

Results

In figure 54.1 and 54.2 are accuracy results from when we first made this prototype to accuracy results after modifications, respectively. In our first results, we achieved an accuracy of around 82%, as shown in figure 54.1. After adjustments, we were able to reach an accuracy of 92%, as displayed in figure 54.2. The second run was trained over a longer period of time, which is why it appears more flat and stable. These graphs are smoothed and their actual representation is shown in the faded lines behind.

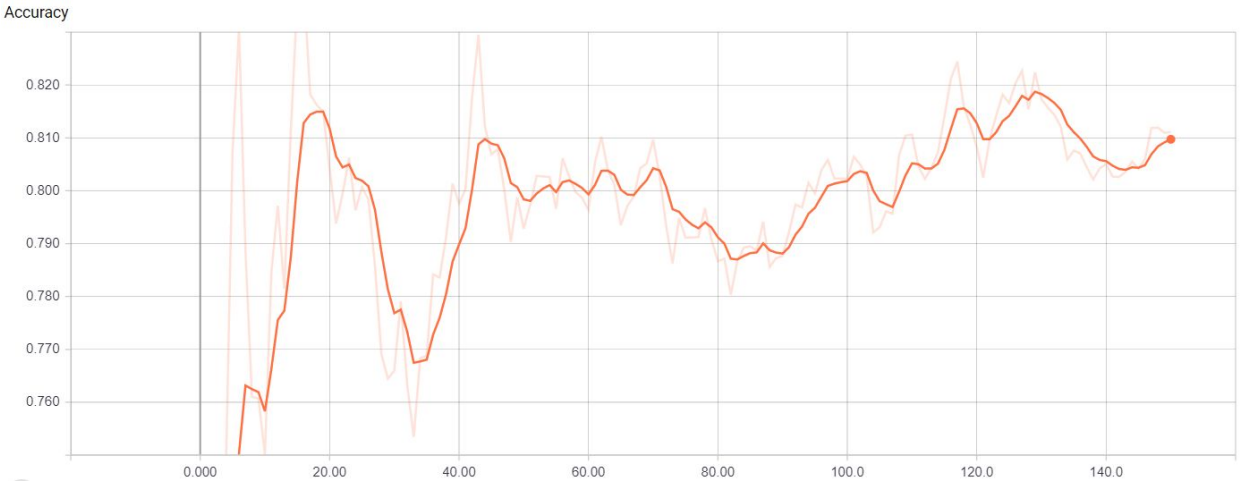


Figure 87.1: First results of prototype

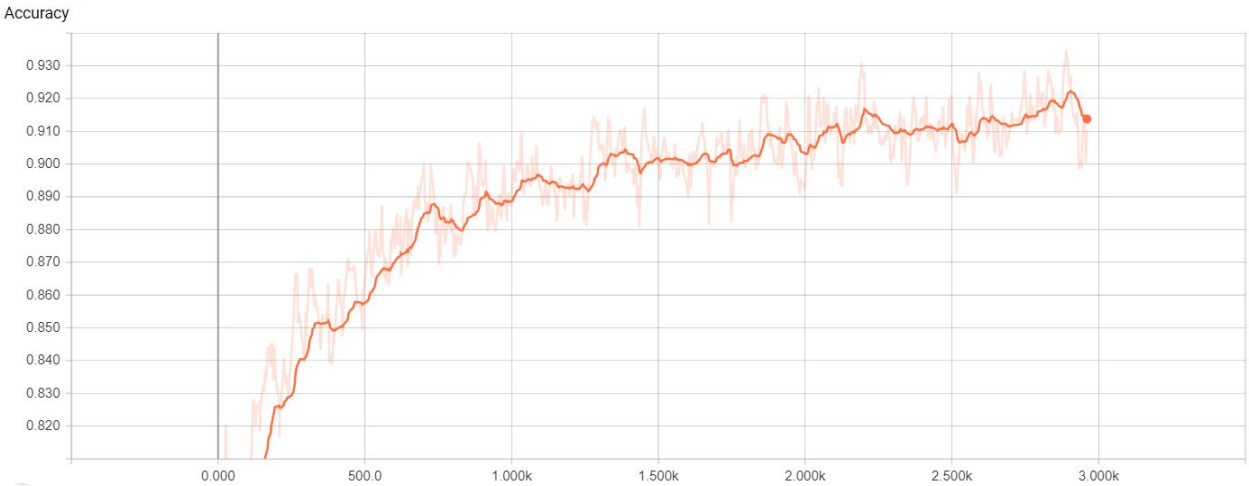


Figure 87.2: Results of prototype after adjustments

Shown in the figures below is validation data. Although similar to accuracy results, it measures the accuracy of test data that the network never trains on. Validation accuracy is important as it shows how much the network is overfitting and how accurate the network really is. In figure 55.1, the first training run, accuracy reached about 88% on the testing data and had a steady linear climb. In comparison, the second run, which reached 92%, had a piecewise climb. In the beginning, the data inclined slowly and then increased its rate of accuracy gain, until it started getting diminishing returns towards the end.

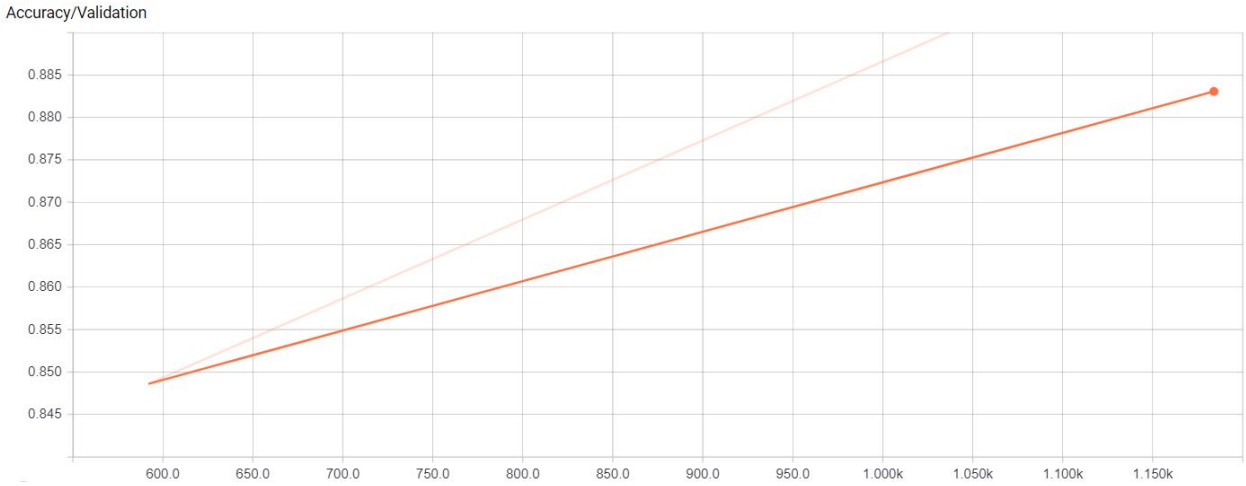


Figure 88.1: First training run data; its accuracy is around 88%

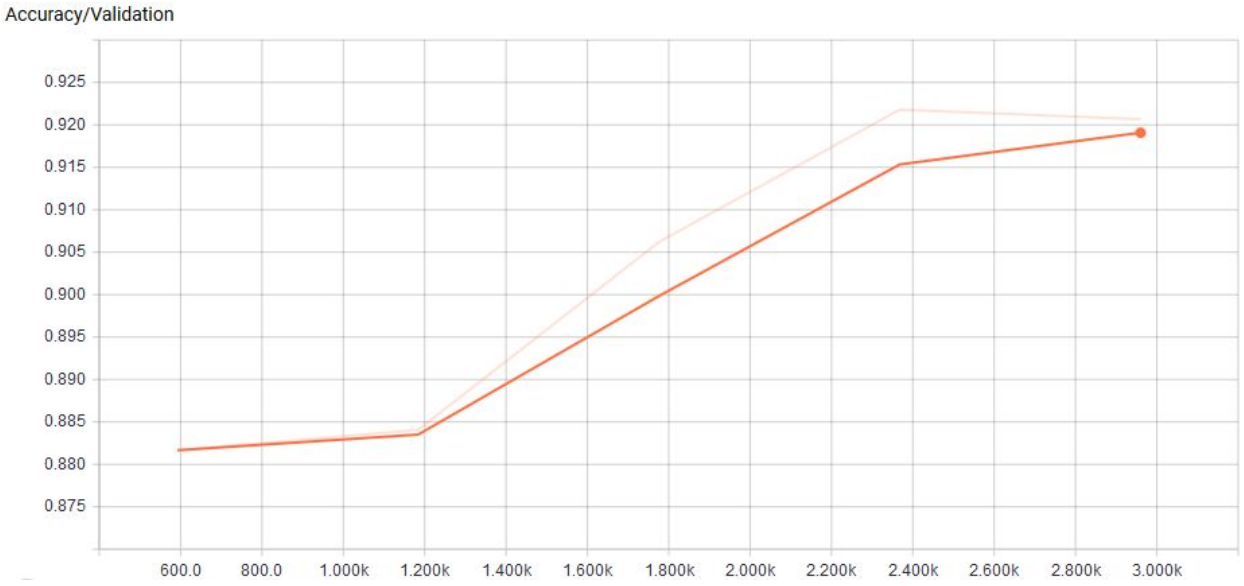


Figure 88.2: Second training run data

In figures 56.1 and 56.2, the loss of the networks are plotted. These show how well the network fits the dataset and sets the amount the network should change. In this case, we want to minimize the loss as much as possible. Overall, both runs have similar loss plots, with the second run showing a steeper decline in the beginning. The first run reached around .29 loss, while the second run reached around .20 loss.

Figure 89.1: First run's loss

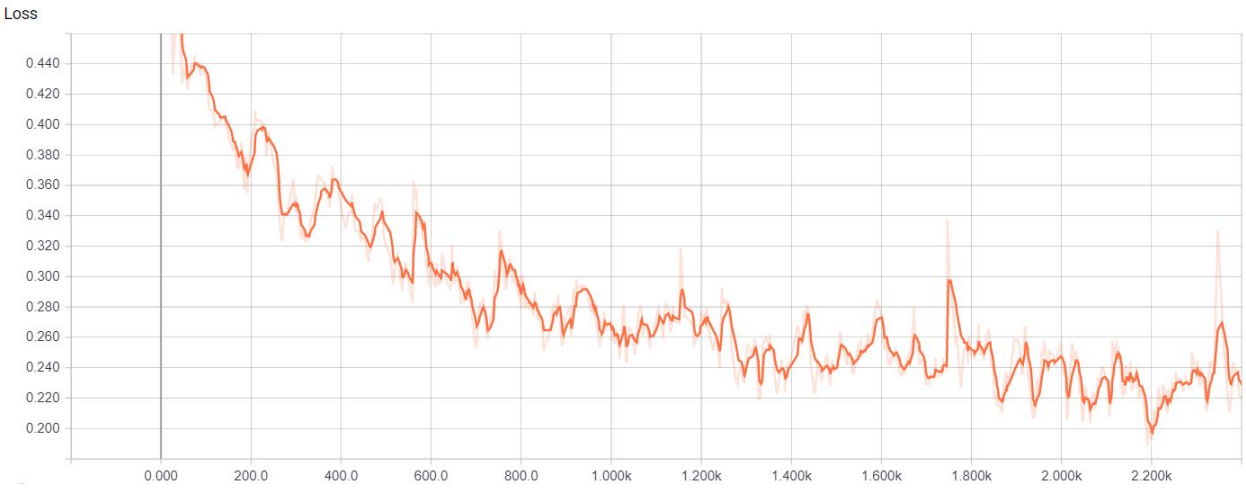


Figure 89.2: Second run's loss

Below, figure 57 shows a histogram of the weights in a convolutional layer over a period of time during training. The x-axis represents the values of the weights in the convolutional network, which can be either negative or positive, whereas y-axis represents time. This layer is one of the last convolutional layers before the fully connected layer. There is some movement in frequency but overall, it looks like the network retains its shape.

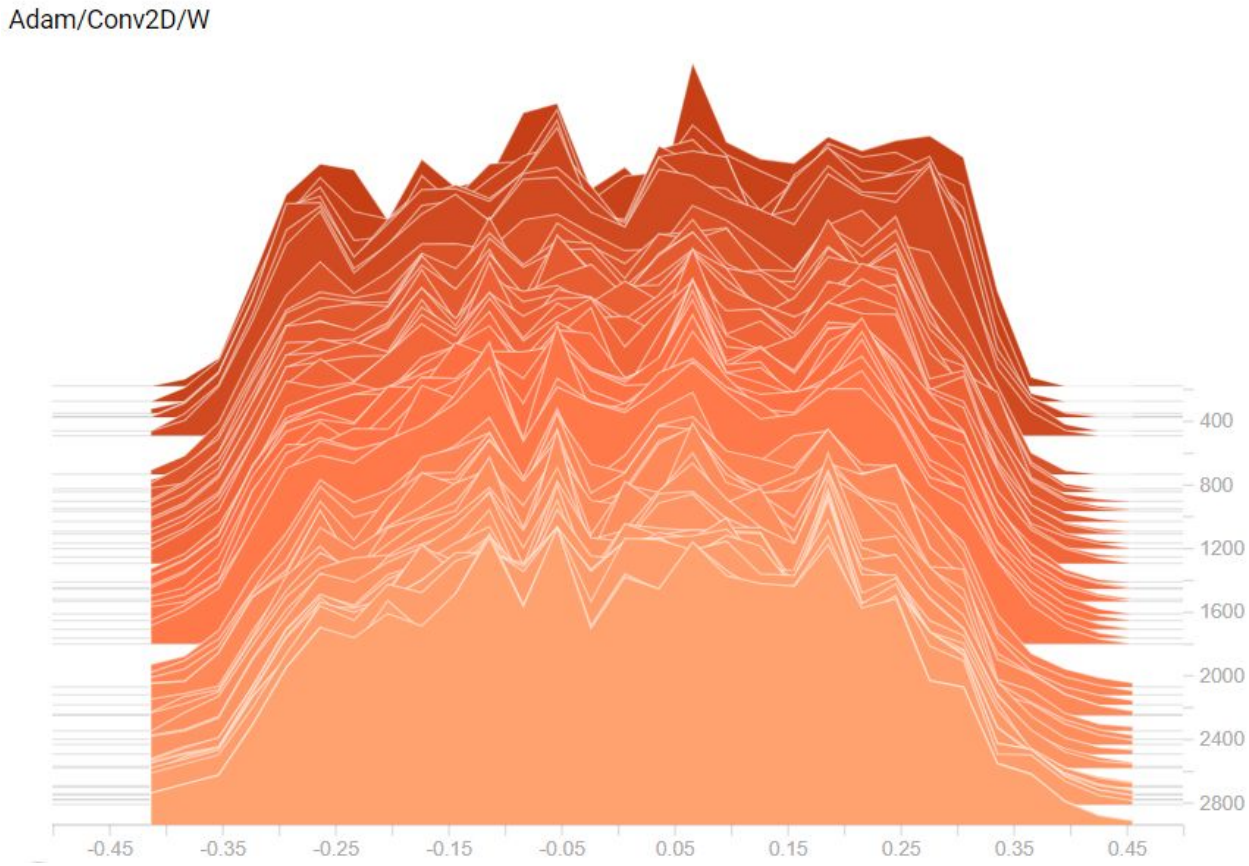


Figure 90: Histogram of the weights in a convolutional layer over a period of time

In figure 58, the biases of the same convolutional layer is shown. There appears to be a lot more movements in bias node values than in the weights of neurons in the layers. In the beginning, we see a high frequency of low negative numbers with a few in the positive range. However as the training progresses, the biases slowly spread out into the negative direction. There were slight climbs in the positive range as well but not nearly as significant.

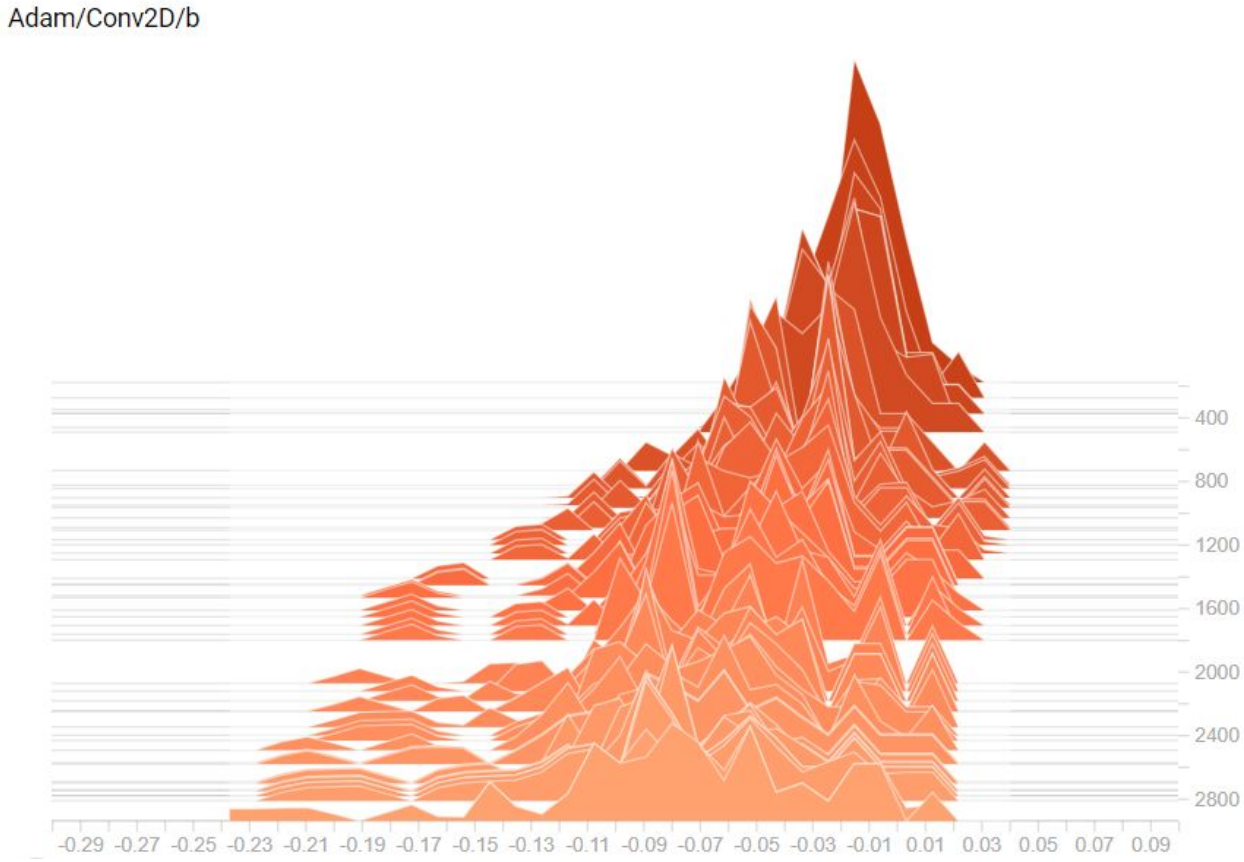


Figure 91: The biases of the same convolutional layer

Shown below in figure 59 is a fully connected layer. The chart shows a histogram of the weights for the fully connected layer. It is interesting to note that the frequency of weights undergoes very little change throughout the entire training process. To note, this is the network that reached 92% accuracy on the training data.

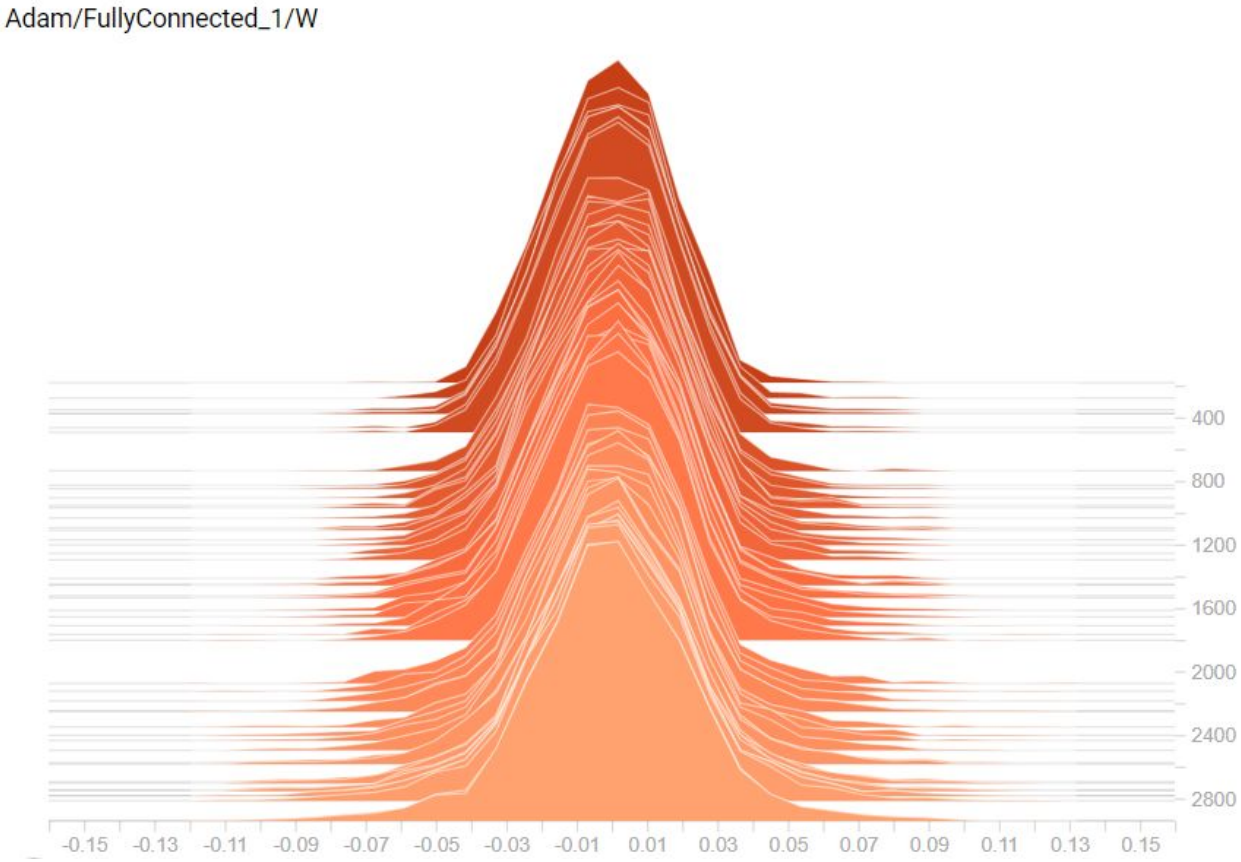


Figure 92: Fully connected layer

Shown below in figure 60 is a histogram of the bias nodes in the fully connected layer. As the chart shows, there is, again, a lot more movement in the biases than the weights of the layers. This layer in particular shows interesting behavior, in that it splits its nodes right down the origin, and the two equal groups move away from each other while maintaining equal distance away from the origin. The exact meaning of all these behaviors is still not immediately clear and will require more experience and research.

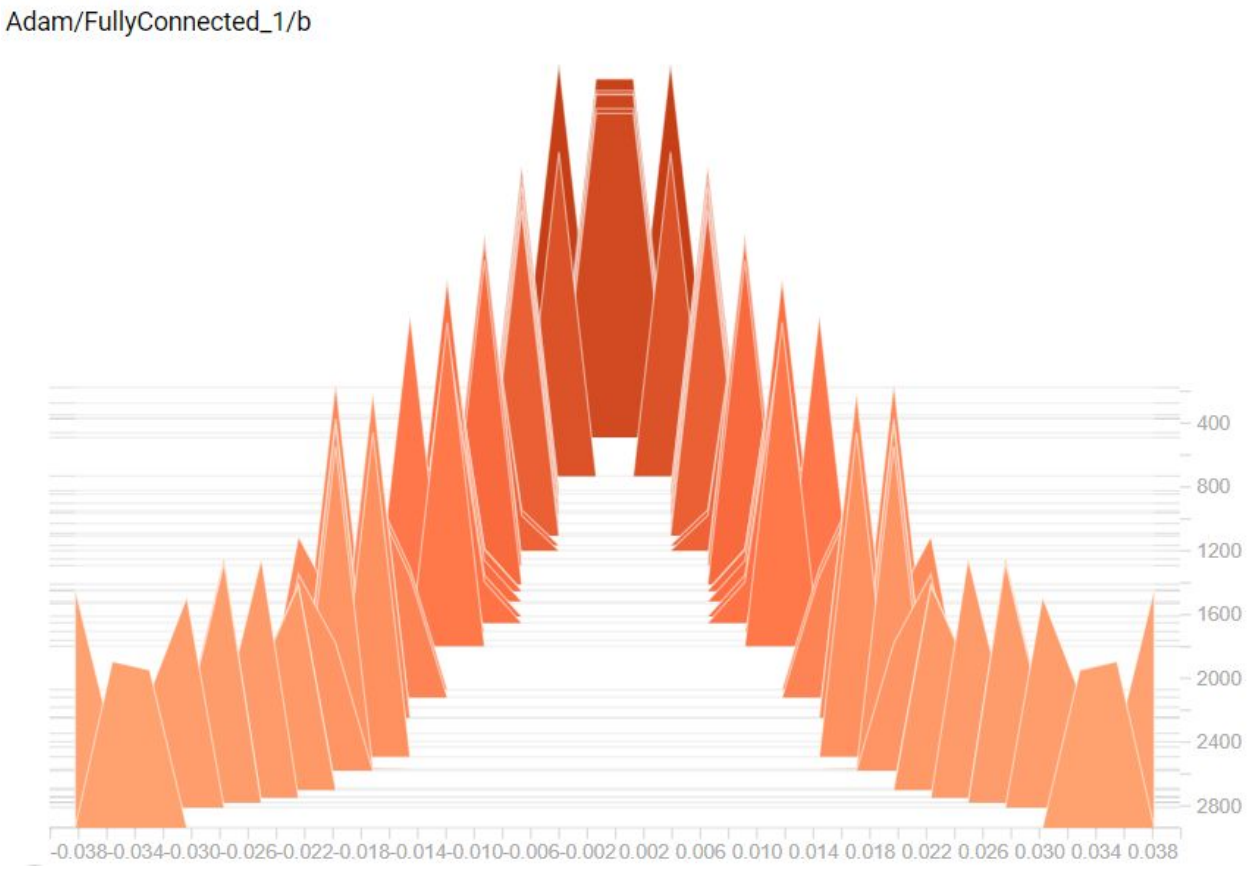


Figure 93: Histogram of the bias nodes in the fully connected layer

Testing

After training and testing our data, we gathered a small sample of Dr. Leavens's bird photographs and collected a few photographs ourselves. We included an image of an airplane and of a butterfly to see how the network performed against images that were relatively similar to birds. In this example, the prototype correctly classified whether a bird was in the image. The images used for testing the network is shown below in figure 61; the results printed to the console is shown in figure 61.

Impressively, the prototype was able to distinguish between the first two objects in images that were relatively high-resolution. Both images contain similar backgrounds, shapes and colors; however, the network still successfully and relatively quickly determine that airplane was not a bird.

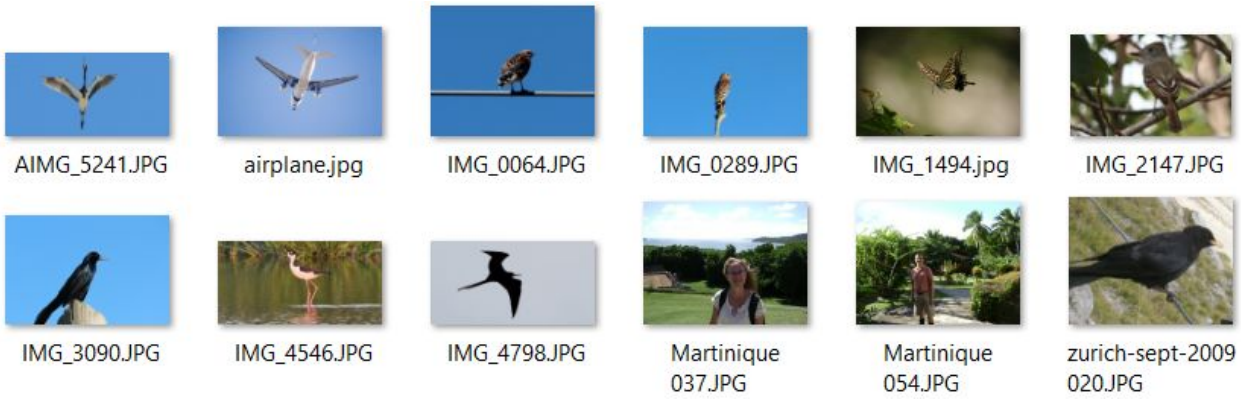


Figure 94: Photos used for testing

```
Instructions for updating:
Use tf.initializers.variance_scaling instead with distribution=uniform to get equivalent behavior.
2017-12-02 20:56:42.786023: I C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_featu
re_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
found bird!
bird not detected!
found bird!
found bird!
bird not detected!
found bird!
found bird!
found bird!
found bird!
bird not detected!
bird not detected!
found bird!
```

Figure 94: Prototype results printed to the console

Final Results

After training various detection models, we tested each set of weights against two different sets of data: a set of 8,000 images from the NABirds dataset we withheld from the training process and a set of 128 hand-labeled sponsor-provided bird images. The latter yields less accurate results due to large environmental backgrounds and multiple subjects.

To evaluate our network's performance, we utilized Tensorflow-provided PASCAL metrics, which uses the mean Average Precision (mAP) of the number of predicted bounding boxes that intersect with the ground truth boxes above a threshold of 0.5. This metric is known as Intersection over Union (IoU). The results of the various models for both sets of data are shown below:

Test set

Model	Bird	Face	Both	Steps
Resnet50	.9794	.9429	.9611	220k
Resnet50	.9871	.9561	.9716	440k
Resnet50	.9831	.9579	.9705	880k
Resnet101 (untuned)	.9320	.5923	.7622	200k
Resnet101	.9868	.9632	.9750	200k
Inceptionv2	.2569	.1744	.2156	200k

Figure 95: The accuracy mAP @ 0.5 IoU results for the test set

Dr. Leaven's Set

Model	Bird	Face	Both	Steps
Resnet50	.8066	.4778	.6422	220k
Resnet50	.8594	.4239	.6416	440k
Resnet50	.8484	.5163	.6823	880k
Resnet101 (untuned)	.5354	.1279	.3317	200k
Resnet101	.8434	.4228	.6331	200k

Figure 96: The accuracy mAP @ 0.5 IoU results for hand labeled images from Dr. Leaven's photos

In the first test set, the fine-tuned ResNet-101 came out on top with a mAP of .975. We can see that as we progressed training for ResNet-50, it performance improved in the first two sets of training and then began to drop after a third set. The untuned ResNet-101 was one of our first relatively successful working models; however, the standard configuration that came with it was not suitable for our training. The second model was ResNet-50 after making necessary adjustments. We also tried the Inception V2 model, a SSD implementation that boasted a significantly faster speed. However, its performance did not match our expectations given the mAP values Tensorflow had reported on the COCO dataset. Without further analysis, ResNet-101 would seem like the best choice.

Once we examined the second test set, we see that ResNet-101 actually underperforms all of the ResNet-50 models. For this set, ResNet-50 at either 220,000 steps or 880,000 steps were the best choices. However, only using mAP as a metric for determining the best models is not good enough for our application. False positives are not taken into account, so manual inspection was used to also evaluate these models. We found the best performing model on Dr. Leaven's photos to be ResNet-50 at 220,000 steps, and thus, settled on it for our application.

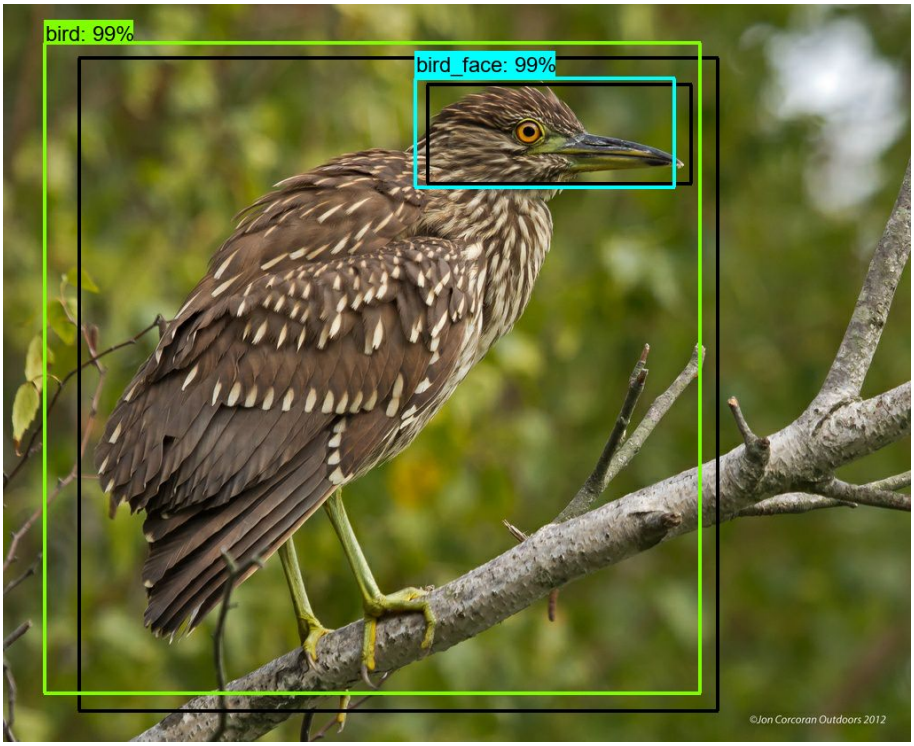


Figure 97: Network's predictions are the colored boxes, black represents the groundtruth boxes.

Below, figures 98-104 show the loss and weights of the model that our application will be utilizing. In figure 98, the total loss is shown. The loss quickly declines in the first couple of hours but then alternates between high and low from a certain period onward. If we break the loss up into the two outputs- the bounding box location and the classification of boxes-, as in figures 99 and 100, the localization loss is what seems to be leveling off. Classification seems to continue to improve.

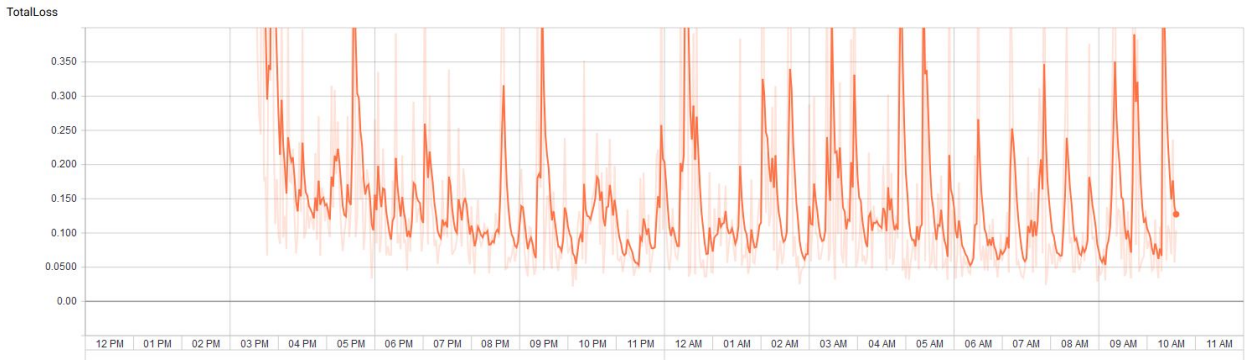


Figure 98: The total loss of our selected model through training.

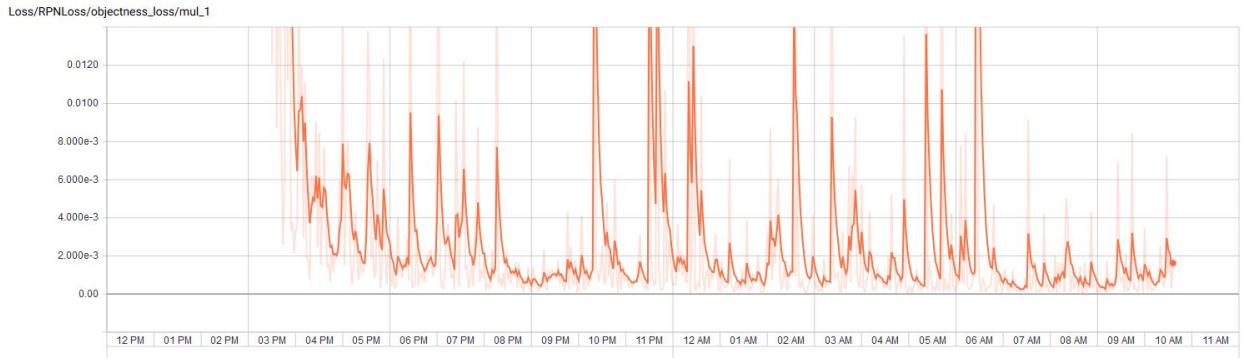


Figure 99: The classification loss of the network, easier of the two tasks to perform.

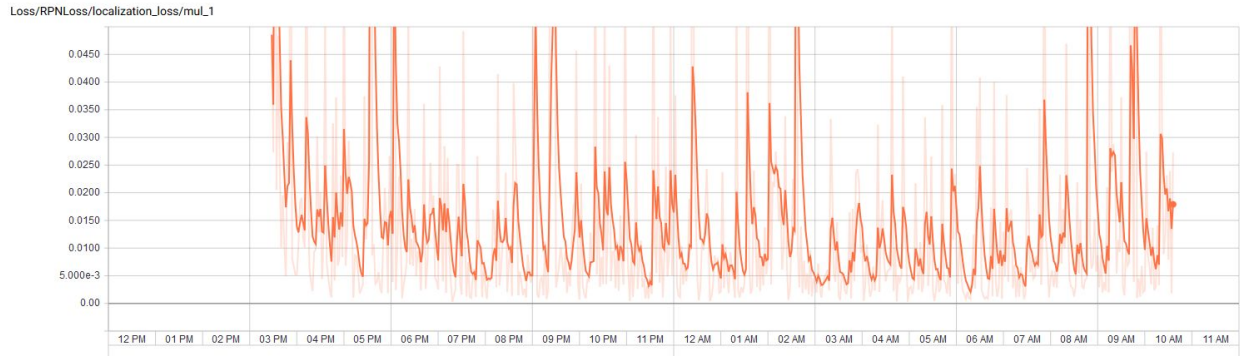


Figure 100: The localization loss of the network. The more difficult of the two tasks.

In the figures 101-104, weights of the networks changing as training progresses are shown. These are explicitly bias weights, as the other parts of the layers are mostly constant. These are the weights of the region proposal section of the network that we retrained. Other parts of the network maintain constant weights, and these can be seen in their histograms as well.

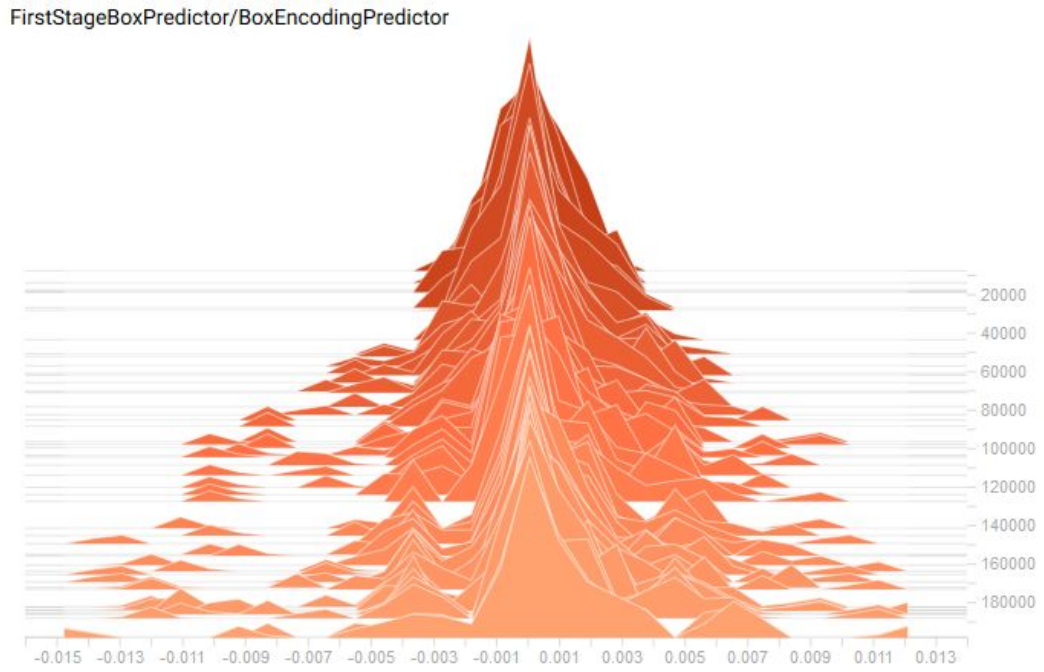


Figure 101: The weights of the 1st box predictor as training progresses.

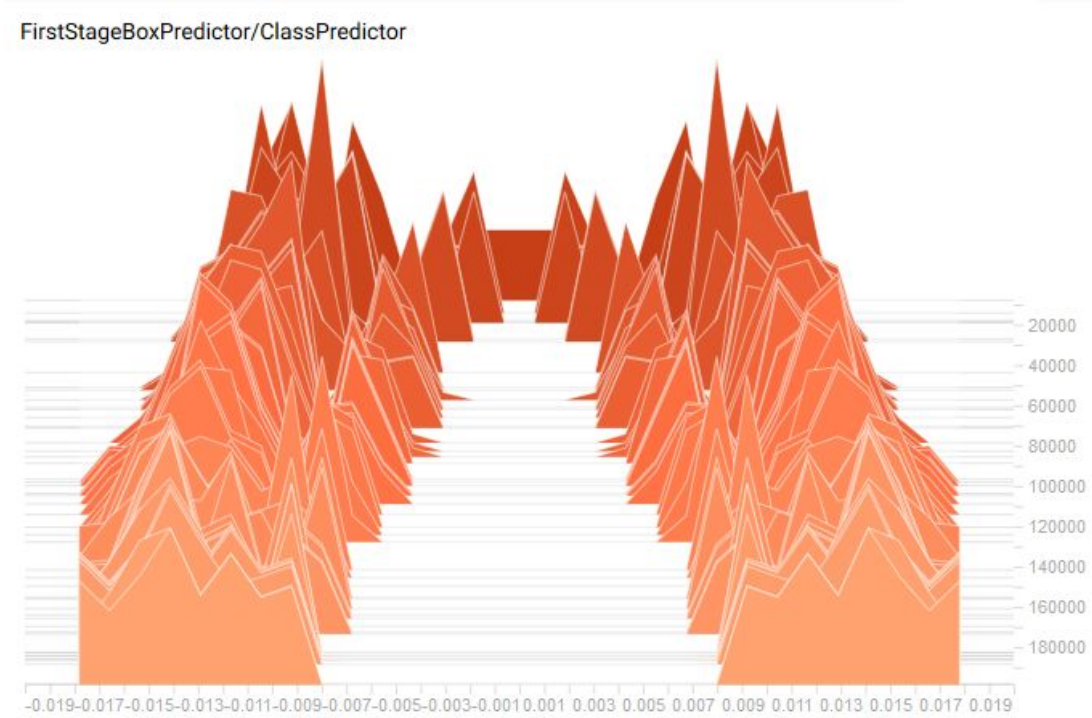


Figure 102: The weights of the 1st class predictor as training progresses.

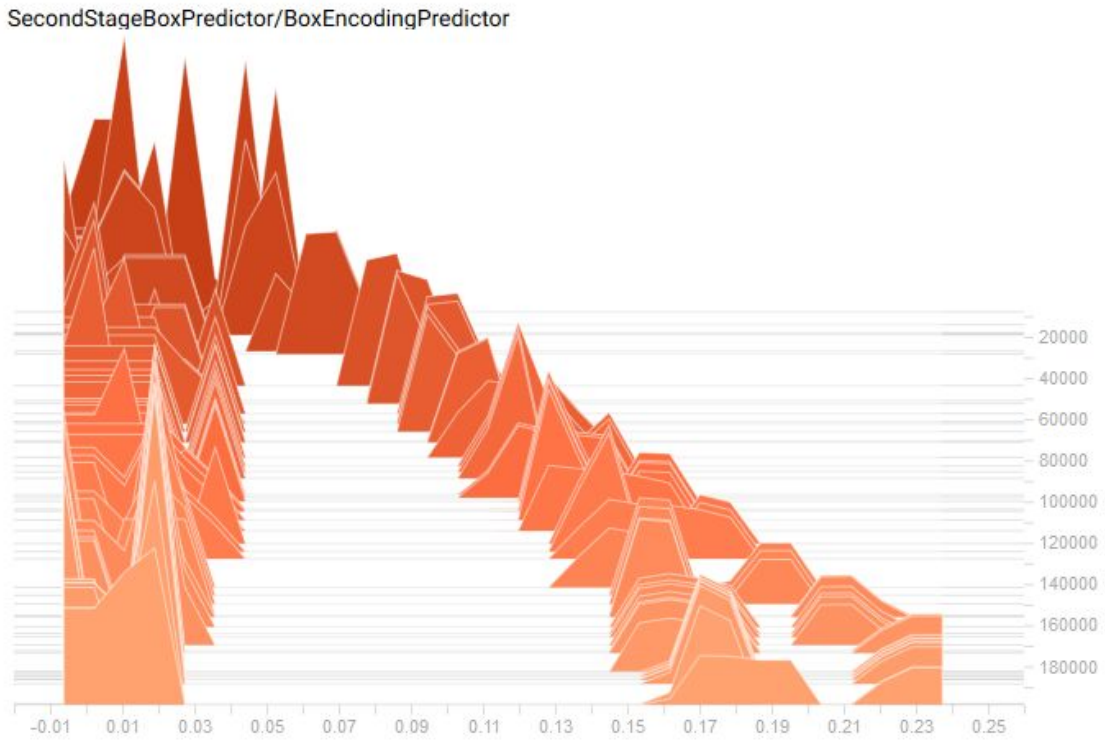


Figure 103: The weights of the 2nd box predictor as training progresses.

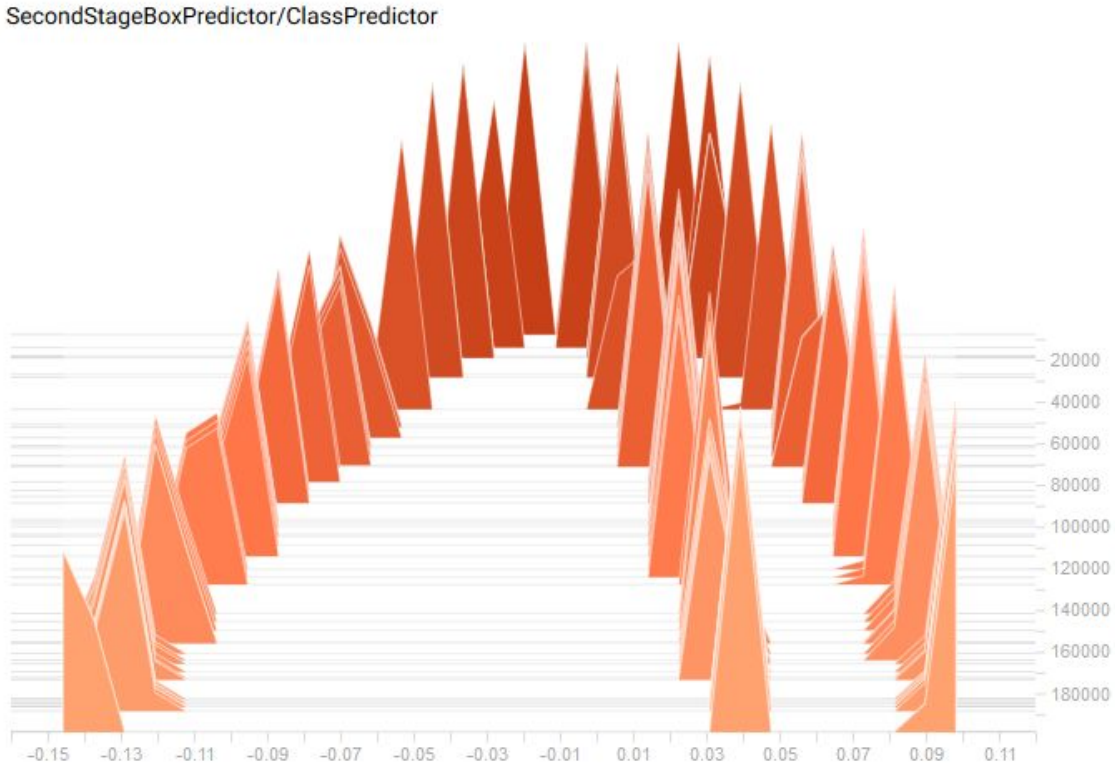


Figure 104: The weights of the 2nd class predictor as training progresses.

Consultants

Dr. Ulas Bagci

Dr. Ulas Bagci is an assistant professor at the University of Central Florida and is a faculty member of the Center for Research in Computer Vision for UCF. His research is involved with image processing and statistical machine learning and their applications. We have already sat down and discussed with Dr. Bagci about various aspects of our project on how to approach some of the issues. He pointed us in the direction of Faster R-CNN and Mask R-CNN to help us find the position of multiple birds in an image. He also suggested for us to implement a siamese neural network to identify if the bird has eyes and if it is in flight. We will continue to meet with Dr. Bagci later in the project for continued advice.

Dr. Gary Leavens

Dr. Gary Leavens, the UCF Computer Science Department Chair, is our sponsor for this project. His interest in Knest stems from him and his wife being bird-watchers themselves, and he hopes to use the application to save him countless hours of time searching through and editing bird photos. Dr. Leavens is only interested in his personal use of Knest and has decided to grant our team with the intellectual property (IP) of the project.

As a team, we have held two meetings with Dr. Leavens during the Fall 2017 semester. During the first meeting, we primarily focused on fleshing out the mandatory and optional requirements of the project; during the last hour of the meeting, we watched Dr. Leavens manually select and enhance bird photos from one of his recent bird watching escapades. During the second meeting, Dr. Leavens walked us through his 500GB repository of bird images, which was very well organized and labeled. We also briefly discussed our bird classification prototype, and at the end of the meeting, he left us with a 1TB external hard drive with all of his bird photos in it.

Administrative Content

Budget

Our costs totaled zero dollars ($0) and we did not require a budget. Each of the components of the main algorithm (TFLearn, Tensorflow, and OpenCV) are open-source and have permissive licenses that permit potential commercial use. Our code is being stored in a free private Github repository acquired by one of our team members. Our project uses Python, a language that is available for free on all platforms, which freed the team of having to pay for a developer program membership on certain platforms.

Milestones

Research and Implementation Design

Date	Objective/Goal	Status
09/21/17	Contact sponsor and discuss technical objectives, goals, specifications and requirements	✓
10/03/17	Discuss and research proposed solutions and possible optional functionalities	✓
10/06/17	Determine best platforms, tools and programming languages and analyze budget and project resources	✓
10/09/17	Complete initial project design proposal	✓
10/20/17	Research and analyze neural networks and apply to project design	✓
10/23/17	Consult university faculty about neural network and its implementations	✓
10/25/17	Research transfer learning, siamese neural networks and faster RCNNs	✓
11/03/17	Assign member tasks and responsibilities for final design document	✓
11/10/17	Research blur, object and bird/bird face detection methods, tools and possibilities	✓
11/13/17	Begin writing final design document	✓
11/15/17	Determine tentative neural network framework and tools	✓
11/20/17	Determine tentative testing evaluations and plans	✓
12/03/17	Finalize design document for delivery	✓
01/09/17	Assign implementation tasks and responsibilities	✓
03/12/17	Begin planning project presentation, documentation and critical design review	✓
04/13/17	Finalize project presentation, documentation and critical design review	✓

Algorithmic Implementation

Date	Objective	Status
10/25/17	Develop tentative unit test methods	✓
11/14/17	Implement blur detection	✓
11/17/17	Begin implementing basic bird detection with TensorFlow and TFLearn	✓
11/22/17	Revise and improve bird detection prototype to increase accuracy and performance	✓
01/15/17	Complete neural network implementation	✓
01/22/17	Begin implementing face detection	✓
02/02/17	Revise and improve quality of overall bird and bird face detection	✓
02/26/17	Complete implementation of bird recognition and detection	✓
03/09/17	Unify back-end implementation and run tentatively	✓
03/16/17	Run unit tests and evaluations and resolve any faults or errors that we come across	✓
04/06/17	Debugging	✓
04/13/17	Finalize software application	✓

User Interface

Date	Objective	Status
09/21/17	Discuss user interface requirements, specifications and functionalities	✓
10/20/17	Determine graphic user interface design and research Electron, Polymer and Kivy	✓
11/08/17	Begin graphic user interface design and determine animation elements to maximize user experience	✓
01/22/17	Begin creating the graphic user interface with Kivy	✓
02/09/17	Complete graphic user interface	✓
03/09/17	Integrate back-end application with graphic user interface	✓
04/09/17	Finalize graphic user interface	✓

Project Summary

In summary, our group created an application that will be able to assist photographers in selecting their best images in an automated fashion. Using the latest research, we implemented new machine learning techniques that will be able to handle a dynamic range of images. The application is centered around images of birds, but can be extended to any other object with the right dataset and enough time.

The platforms we utilized are Python, along with APIs such as Tensorflow, TFLearn, and Kivy. Python was selected for the wide range of research purposes that it fulfills in machine learning as well as the fact that it has options such as Tensorflow and TFLearn available.

Knest was created with an implementation that uses a combination of object classification and detection as well as several computer vision techniques in order to enable the application to identify multiple birds in an image and identify their location.

References

[1] “DALYAN BIRD WATCHING & CARETTA BOAT TOUR”, *Greece Turkey Tours,* http://www.greeceturkeytours.com/dalyan-bird-watching-caretta-tour.html

[2] U.S. Fish & Wildlife Services. “Economic Impact - Birds, Birdwatching and the U.S. Economy”, https://www.fws.gov/birds/bird-enthusiasts/bird-watching/valuing-birds.php

[3] Adit Deshpande. “A Beginner's Guide To Understanding Convolutional Neural Networks” 20 July 2016, https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. “Generative Adversarial Nets” 10 June 2014, https://arxiv.org/pdf/1406.2661.pdf

[5] Siraj Raval. “Convolutional Network Tutorial.” 10 July 2017, https://github.com/llSourcell/Convolutional_neural_network/blob/master/convolutional_network_tutorial.ipynb

[6] “A Guide to TF Layers: Building a Convolutional Neural Network”, *TensorFlow, https://www.tensorflow.org/tutorials/layers*

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks” 2012, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott, Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. “Going Deeper with Convolutions” 2015, https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten. “Densely Connected Convolutional Networks” 27 August 2017, https://arxiv.org/pdf/1608.06993.pdf

[10] “Matrix Capsules with EM Routing.” In ICLR 2018, https://openreview.net/pdf?id=HJWLfGWRb

[11] Siraj Raval. “Capsule Networks What Comes after Convolutional Networks?” 31 October 2017, https://github.com/llSourcell/capsule_networks

[12] Geoffrey E. Hinton, Sara Sabour, Nicholas Frost. “Dynamic Routing Between Capsules.” 7 November 2017, https://arxiv.org/pdf/1710.09829.pdf

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” 2015, http://www.cvlibs.net/projects/autonomous_vision_survey/literature/Ren2015NIPS.pdf

[14] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. “Mask R-CNN.” 5 April 2017, https://arxiv.org/pdf/1703.06870.pdf

[15] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. “One pixel attack for fooling deep neural networks.”, 16 November 2017, https://arxiv.org/pdf/1710.08864.pdf

[16] David MacKay. “Information Theory, Inference, and Learning Algorithms“ September 2003, http://www.inference.org.uk/mackay/itila/book.html

[17] Sebastian Ruder. “Transfer Learning - Machine Learning’s Next Frontier.” 21 March 2017, http://ruder.io/transfer-learning/

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,
Sherjil Ozair, Aaron Courville, Yoshua Bengio. “Generative Adversarial Nets” 10 June 2014, https://arxiv.org/pdf/1406.2661.pdf

[19] Nasim Souly, Concetto Spampinato, and Mubarak Shah. “Semi and Weakly Supervised Semantic Segmentation Using Generative Adversarial Network” 28 March 2017, https://arxiv.org/pdf/1703.09695.pdf

[20] Hanghang Tong, Mingjing Li, Hongjiang Zhang, Changshui Zhang. “Blur Detection for Digital Images Using Wavelet Transform”, https://arxiv.org/pdf/1406.2661.pdf

[21] “Haar Wavelet Transformation”, Why Do Math, https://www.whydomath.org/node/wavlets/hwt.html

[22] “About OpenCV.” *OpenCV,* https://opencv.org/about.html

[23] jenssengers. “imagehash”, https://github.com/jenssegers/imagehash

[24] theskumar. “imagehash”, https://github.com/theskumar/imagehash

[25] The Python Software Foundation. “About Python.” *Python,* www.python.org/about/

[26] Jeff Dean and Rajat Monga. “ TensorFlow - Google’s latest machine learning system, open sourced for everyone” 09 November 2015, https://research.googleblog.com/2015/11/tensorflow-googles-latest-machine_9.html

[27] “An open-source software library for Machine Intelligence." *TensorFlow*, www.tensorflow.org/

[28] “Image Recognition”, *TensorFlow,* https://www.tensorflow.org/tutorials/image_recognition

[29] Faizan Shaikh. “An Introduction to Implementing Neural networks using TensorFlow” 03 October 2016, https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow/

[30] “TFLearn: Deep Learning library featuring a higher-level API for TensorFlow.” *TFLearn,* http://tflearn.org/

[31] Adrian Rosebrock. “Blur Detection With OpenCV.” 7 September 2015, https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/

[32] “About NumPy.” *NumPy,* http://www.numpy.org/

[33] “About Electron.” *Electron, http://*www.electron.atom.io/docs/tutorial/about/.

[34] “Kivy: Cross-platform Python Framework for NUI.” Kivy, https://kivy.org/#home

[35] Kristian Poslek. “Building a desktop application with Electrion” 10 August 2015, https://medium.com/developers-writing/building-a-desktop-application-with-electron-204203eeb658

[36] “UI Animation: Eye-Pleasing, Problem-Solving.”, *Tubik Studio,* https://uxplanet.org/ui-animation-eye-pleasing-problem-solving-a8b27013f55c

[37] “36 Brilliant User Interface Animations”, *Digital Synopsis,* https://digitalsynopsis.com/design/gif-icons-menu-transition-animations/

[38] Nick Babich. “4 Ways to Use Functional Animation in UI Design” 24 May 2017, https://www.webdesignerdepot.com/2017/05/4-ways-use-functional-animation-in-ui-design/

[39] Krizhevsky, Alex. “Learning Multiple Layers of Features from Tiny Images.” *The CIFAR-10 dataset*, 8 Apr. 2009, www.cs.toronto.edu/~kriz/cifar.html

[40] Wah C., Branson S., Welinder P., Perona P., Belongie S. “The Caltech-UCSD Birds-200-2011 Dataset.” *Computation & Neural Systems Technical Report*, CNS-TR-2011-001.

Appendices

Copyright

OpenCV and NumPy are licensed under the 3-clause BSD License agreement, which permits redistributions of the source code or distributions of the code in binary form, with and without modification, as long as said distributions retain the aforementioned agreement, and the names or likenesses of the contributors or copyright holders cannot be used as endorsement or promotion material of any products without prior permission from said persons. The software (OpenCV, NumPy) itself is provided “as is” with no warranties extended unto the user.

Tensorflow is licensed under the Apache License agreement, which grants a “perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable” copyright license and patent license, although there are some stipulations regarding revocations should an individual bring forth litigation against the software, e.g. a patent claim. It also allows for the reproduction or redistribution of the software or any derivative works of said software, with or without modifications, as long as the software or derivative software carries a copy of this license. The Apache License does allow for different license terms to be applied to the derivative work of the user, provided it complies with the conditions stated in the license. The software (Tensorflow) itself is provided “as is” with no warranties extended unto the user.

Electron and TFLearn are licensed under the MIT License agreement, which grants permission to any person to deal with the software “without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies” of said software, as long as the license agreement appears in all copies or substantial portions of the software. The software (Electron, TFLearn) itself is provided “as is” with no warranties extended unto the user.

Pillow, a fork of the Python Image Library (PIL), is licensed under the PIL Software License, which allows one to “use, copy, modify, and distribute” the software and its documentation for any purpose as long as the copyright and permission notice appear in all copies and supporting documentation. Furthermore, the names of the creator or contributors may not be used in advertising or publicity relating to the distribution of the software. The software (Pillow) itself is provided “as is” with no warranties extended unto the user.

.License

Electron

NumPy

OpenCV

License Agreement

For Open Source Computer Vision Library

(3-clause BSD License)

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Pillow

The Python Imaging Library (PIL) is

    Copyright © 1997-2011 by Secret Labs AB
    Copyright © 1995-2011 by Fredrik Lundh

Pillow is the friendly PIL fork. It is

    Copyright © 2010-2017 by Alex Clark and contributors

Like PIL, Pillow is licensed under the open source PIL Software License:

By obtaining, using, and/or copying this software and/or its associated documentation, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its associated documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Secret Labs AB or the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Tensorflow

Copyright 2015 The TensorFlow Authors. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

TFLearn

Figures

26	48	Blurr and variance
27	49	Haar wavelet transform decomposition
28	50	Haar wavelet transform
29	52	tructure of the blur detection
30	52	Experimental results
31	53	Hash values
32	55	Electron
33	55	Electron
34	56	Kivy interface library
35	57	Kivy interface elements
36	57	Kivy native layout
37.1	58	Color Scheme
37.2	58	Color Scheme
37.3	58	Color Scheme
37.4	58	Color Scheme
37.5	58	Color Scheme
37.6	58	Color Scheme
38	59	Loading screen
39	60	Progression of a loader
40	61	CIFAR-10 dataset
41	62	Classes of birds
42	63	Face bounding box
43	65	Zero-centering
44	66	2D Convolution
45	67	Max Pooling
46	68	API model
47	69	Wrongly classified as blurry
48	69	Wrongly classified as blurry
49	70	Method representation
50	70	Method representation

51	71	Calculate blurry images
52.1	72	Undesirable quality
52.2	72	Accepted image
53	73	imagehash library
54	73	Limit to reduce similar images
55	74	Crop method
56	75	Kivy file
57	75	Kivy initiate code
58	76	Kivy’s custom language
59	77	Splash screen
60	77	Main user screen
61	78	Main user screen
62	78	Advanced settings
63	79	Inactive button
64	79	Folder selection
65	80	Folder selection
66	80	Popup message
67	81	Popup message
68	81	Transition screen
69	82	Display example
70	83	Display example
71	83	Display example
72	84	Bird localization UI display
73	84	Bird localization UI display
74	85	Image comparison UI screen
75	85	Writing UI screen
76	86	Interactive screen
77	87	Pull request
78	88	Test cases
79	90	High level design

80	91	Conv2D tensor
81	92	Fully connected layer
82	93	Cross-entropy layer
83	94	Accuracy layer
84	95	Activity diagram
85	96	Graph of prototype code
86	97	Prototype bird classifier
87.1	98	First results of prototype
87.2	98	Results after adjustments
88.1	99	First training run data
88.2	99	Second training run data
89.1	100	First run’s loss
89.2	100	Second run’s loss
90	101	Histogram of the weights
91	102	Histogram of the weights
92	103	Histogram of the weights
93	104	Histogram of the weights
94	105	Prototype results
95	106	Accuracy mAP
96	107	Accuracy mAP
97	108	Network’s predictions and groundtruth
98	108	Total loss
99	109	Classification loss
100	109	Localization loss
101	109	Histogram of the weights
102	110	Histogram of the weights
103	110	Histogram of the weights
104	111	Histogram of the weights

Software

Kivy - Kivy is an open-source user interface library that assists in the development of standalone desktop applications. It supports Python and can be connected to other foreign Python scripts natively. User interface elements can be defined within classes and methods in Python module or in a separate *.kv* file written in custom Kivy language similar to Python.

Python - Virtually all of the code produced for this project is written in Python. Python is a cross-platform, general-purpose programming language that aims to have clearer syntax and improved readability in relation to other general-purpose languages. It boasts a large standard library as well as a significant number of specialized modules, e.g. data science and numerical computing.

Tensorflow - Tensorflow is a software library that aims to use graphs for numerical computation. Its main use can be found in machine learning in which it can be used to build neural networks, a type of network that mimics the human brain in learning and reasoning ability. Tensorflow employs the use of tensors (multidimensional arrays) as the graph edges and mathematical operations as the graph nodes. This allows for computation to be distributed across multiple clients, if necessary.

TFLearn - TFLearn is a Python library that is built atop Tensorflow that provides a high-level API that is fully compatible with the underlying framework. The syntax of Tensorflow itself can be confusing to many users. As such, the library aims to provide an easy-to-use syntax along with helper functions that allow for faster prototyping and experimentation compared to Tensorflow itself. TFLearn is used in the project as a way to speed up the development process while also leveraging the computing power of Tensorflow proper.

OpenCV - OpenCV (Open Source Computer Vision Library) is a cross-platform, cross-language library containing over two thousand (2000) computer vision and machine learning algorithms that allow users to do many tasks, such as image classification, 3D model extraction, and reality augmentation. The library is used by companies both big and small as well as academic research groups. OpenCV is used in this project in order to recognize birds in a photograph and create a region for automated cropping.

NumPy - NumPy is a Python package that is mainly used for scientific computing. It allows one to use linear algebra and Fourier transform techniques, and contains the capability to easily create an *N*-dimensional array object (ndarray) that can then be passed to other functions. NumPy, as part of the larger project SciPy, is used by thousands of group and individual developers, ranging from hobbyists to multinational companies. In this project, images are represented as NumPy's ndarrays, and then passed to the neural network and OpenCV.

Pillow - Pillow is a Python image processing library used for image manipulation supporting a range of image file formats such as: PPM, PNG, JPEG, GIF, TIFF and BMP. Pillow provides a large kit of image processing APIs such as: changing file extensions, resizing, cropping, changing colors, blurring, image enhancement, pixel based manipulations, etc.

H5py - H5py interfaces with the HDF5 binary data format. It allows one to store extremely large amounts of numerical data in a single file, which can then be interpreted through the use of NumPy. H5py uses common Python and NumPy metaphors such as Dictionary and NumPy Array, which makes usage easy on users without special knowledge of HDF5. To allow for seamless collaboration and exchange of data H5py creates files in a widely-used standard binary format.