Slekit:
A Deep Learning Photo-Selection Application

Victor Querecuto

Sulaiman Hamouda

Camilo Andres Gomez

Jasper Chong

Sponsored by Dr. Gary T. Leavens

Table of Contents

Executive Summary

Modern advancements in storage technology make taking many photographs a convenient feat. So convenient, in fact, that people find themselves with folders full of hundreds or thousands of snapshots. Most photographers, amateur or professional, do not want to retain *all* the pictures they have taken. Even with the advancements in storage, our devices cannot hold an infinite amount of memory. As a result, people must go through the wearisome process of deleting the pictures they dislike. This can be particularly frustrating, especially if many of the photographs (photos) are nearly identical, or the list is seemingly endless.

The goal of this project is to create a program that will learn each user's taste in photos. In doing so, the program will become a "trusted assistant". Given a directory of photos, it will be responsible for selecting the photos it "thinks" the user would select. Users will be provided the opportunity to give feedback on the assistant's performance by specifying photos they do and do not like. This feedback will, in turn, fuel the training process. We selected a deep learning framework as the basis for this endeavor.

The program will be built as a Windows 10 desktop application. Performance concerns dissuaded us from supporting a client-server architecture. Our target users are photographers with thousands of photos; uploading their directories to a web server for processing would place a serious toll on runtime.

Broader Impacts

Photo Selection: A Reinforcement Learning Approach has many features which could be applied to various environments, recreational or work related. Professional photographers need to capture an image in the right moment so that the photo encompasses the emotion and detail the artist was searching for. Most photographers' approach would be to take as many photos as possible to have a likelier chance of obtaining an optimal image. However, the tedious task of going through similar photos and choosing which looks the least blurry or picture with the best lighting may take a long time. Our software will eliminate the time spent searching for the "right" image so the user will be able to continue taking photos. With an easy-to-use interface, people who take pictures casually will be able to run the application with no hassle.

This application could potentially be used in other fields. An example of this would be to use this application within the medical field, specifically in identifying better quality x-rays images from x-ray images that are distorted or defected. Implementing this application into the medical field could help radiologist be more efficient and have to spend less time manually processing each image in hopes to get more insight into the subject within the x-ray image.

Another sector our project could impact would be the environmental sector, specifically the agricultural industry. Many crop fields are monitored by video and images taken by aircraft. These videos and images are then processed and analyzed to find any crops that could be at risk. Our application could potentially make the process of identifying the best photos much easier which can later make the analysis process easier and more efficient than looking through thousands of images. Our application has the potential to improve many areas where large amounts of images need to be processed and separated into sets of good and bad quality images.

Requirements

1. Support Windows 10.
2. Support the JPEG image format.
3. For input, take image directories with as many as 2000 images.
4. The application should select less than 20% of the images in a directory and place these photos (copies of the originals) in a separate directory.
5. The application should learn to predict which photos the user would select from their directories.
6. The application should select photos the user would select, or images of high quality if the program hasn’t been sufficiently trained.

Motivations

Victor Querecuto

This project was of interest to me because it requires tools and algorithms that I am personally interested in and would like to potentially work with in the future. Specifically, with machine learning and deep learning. The foundation of the entire project is built on deep learning and neural networks, which has been a topic I have been interested in for most of my college career. I feel like this project would also allow me to understand and the entire process of development from idea to final working product. Another motivation is being able to build something that I am proud of and could potentially build on later.

Sulaiman Hamouda

I have multiple reasons on why I decided to participate in this project. Firstly, I do not have much experience in Machine Learning. So, being able to be part of a project that deals with Machine Learning seems like the best way to start learning about it and also gain new skills. Thus, one of the motivating factors for me is to expand my knowledge of Machine Learning. This is a growing field within Computer Science and this is my opportunity to acquire new skills in order to apply it to future projects. Secondly, being someone who takes many outdoor pictures, the idea of not having to go back to your camera in order to perform the tedious task of deleting the pictures you do not want would not only save me a lot of time but would also save me a lot of space on my camera. Being able to work on a project which will provide me with an indispensable set of skills and also solve a problem excites me and also makes my life easier.

Camilo Andres Gomez

My main motivation for working on this project is to have a stronger grasp on deep learning and machine learning as well as understand the fundamentals of vision processing. I have done projects in the past that incorporated these topics and developed a strong interest in pursuing them further. I plan on acquiring a masters in artificial intelligence so this project will be a good foundation to begin with.

Jasper Chong

Curiosity, personal gain, and utility - these are the driving factors that led me to choose this project. So far in my journey as a computer science undergraduate, I have not chanced upon the opportunity to dive too deeply in machine learning, to learn the inner mechanics and mathematics, therefore I am excited to be involved in a project that implements deep learning. The second factor may be considered slightly less virtuous; I hope to become a more effective and versatile software developer by learning these new skills. Lastly, I have friends and family who suffer from the exact problem this project aims to solve. By helping build this application, I will be helping those I care for.

Machine Learning

Machine learning, a subset of artificial intelligence, is the study of how algorithms and mathematical models are utilized by computers in order to productively and effectively perform tasks without having to provide it any specific instructions. Instead, it recognizes patterns and inferences in order to produce a meaningful output. However, what does a machine need for it to produce meaningful output, or in other words, what does it need for it to “learn?” Given a task, training experience, and a performance measure, a computer program is said to learn if its performance at the task improves with experience [1]. This is very similar to how humans and other living things learn.

The difference between traditional programming and machine learning is that in traditional programming, you write a program that you input to a computer. It then takes data and produces some output. In a machine learning approach, the idea is to have your output and data run on a computer. The computer will be given examples, data, of what the program should do. The idea is for it to learn from those examples and experiences, through recognizing the different trends and patterns. After it has “learned” a program will be produced. This program will then be used to infer new information about things. You can have the machine learning algorithm learn about those examples and experiences, which you can then use to solve other similar problems. After the machine learning aspect has been executed and a model has been created and fitted to some data, you can then use future instances of new data in order to predict some output relating to it. That is how a program can then be implemented in a traditional programming manner. Figure 1 illustrates how machine learning differs from traditional programming. Some applications of machine learning are consisted of image classification, facial recognition, recommendation systems, character recognition, assisted driving, and much more.

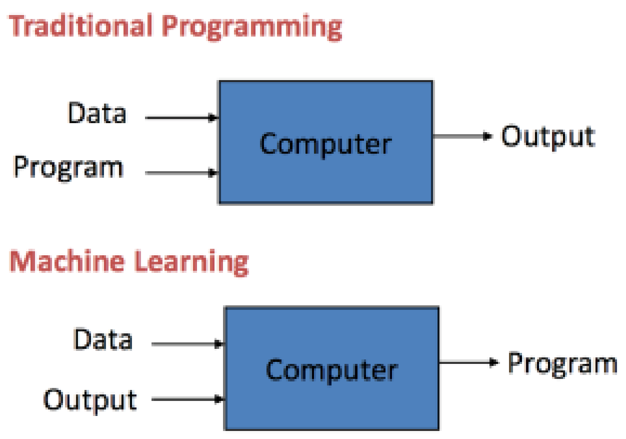


Figure 1: Illustration of a Biological Neural Network. Created by Professor Loc Vu-Quoc

In Tom M. Mitchell's book "Machine Learning," [1] he goes over an example of how machine learning can be implemented by teaching a program to learn how to play a game of checkers. He goes over the basic design issues and approaches to machine learning. He described how the program will train, what the data set is consisted of, and how choosing the right training experience determines whether it will be successful or not. He also determined what the program will learn and how the program will use those learning materials and experiences to give the best results.

In order to build an effective checker's machine learning program, he states that the program must know which legal moves it can take, and amongst those moves which are the best move for any given board state. For each board state, there was a numerical score which was assigned to it. That score represents how "good" the current board state is. The higher the score, the better the board state. This way, every move it takes will be the most optimal move. He also talks about how to estimate the training values for which we assign specific scores to specific board states. He shows the different techniques which are used to successfully build a program that will learn how to play and win a game of checkers.

There are many different algorithms and learning techniques one can utilize to successfully build a machine to learn. Amongst machine learning, there are four main types of learning, supervised, unsupervised, semi-supervised, and reinforcement learning.

As the paper progresses, we learn about these different machine learning techniques, how they are implemented, and which learning experience we must utilize in order to accomplish our goals.

Supervised Learning

Of the three pillars of machine learning, supervised learning is the most popular in terms of use because the required training is straightforward and relatively easy to implement. The goal of supervised learning is to develop a model by observing mappings between inputs and outputs [2]. In this way, models are synonymous with functions. Training sets for supervised learning problems are labeled: for each example in a set, not only is an input value specified, but also an output value. It follows that the training process relies on the output value being known so the network that produces the model can check itself for the right answer. As more examples from a training set are given to a network to learn, the stronger and more accurate the model becomes.

Some supervised learning algorithms include support vector machines (SVMs), artificial neural networks, and logistic regression. SVMs have been shown to train more accurate models than those trained by artificial neural networks and logistic regression [3]. It is possible to use these algorithms in tandem, though. Initially, we planned on using a neural network and SVM. The network would analyze an image and pass its analysis onto a support vector machine for classification as a high or low-quality image.

We plan on using supervised learning to establish an initial model that will be able to rank images in terms of quality on a general basis. Once the user starts giving feedback, the model needs to be updated to reflect the user’s personal tastes. We may use reinforcement learning for this process, but there are other options as well. See the Design section for a more in-depth discussion.

Model Flexibility

There are two factors that determine the flexibility of a model: bias and variance. Bias represents assumptions made about the model [4]. If the bias for a model is low, it will take longer for the model to become accurate. An advantage of using a low-bias model is its superior ability to handle complex problems. High-bias models learn faster but are considered less flexible.

Different training sets used on a model will produce different functions for each set. Given an input, variance represents how different the outputs are when applying the input to the different functions. A model with low variance is desirable because this means the function it has learned is universally effective.

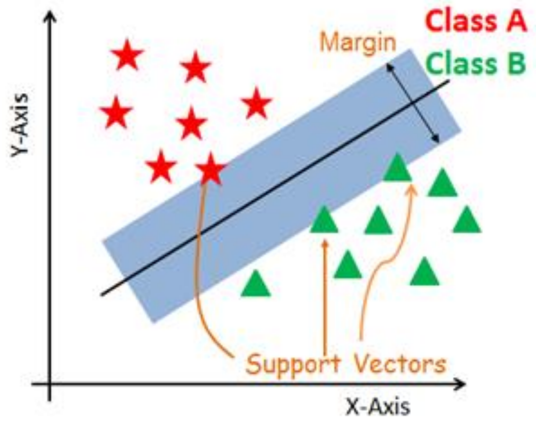
An accurate model can be learned for a simple problem using a small training set. This will result in low variance and high bias. Though high biases are generally not desirable, it is necessary to develop the model if the problem is simple. An issue that could arise from using a small training set is overfitting; the learned function requires that future inputs match the trained inputs to too high a degree. This will make the function unusable for unseen data.

Complex problems, such as assessing the aesthetic quality of an image, require larger training sets and possibly several different sets. There are more instances that could satisfy or dissatisfy the requirements and therefore the model needs to be exposed to them. A high variance, low bias model will likely be developed. Overfitting is less likely to occur since the training sets are large.

Support Vector Machines (SVMs)

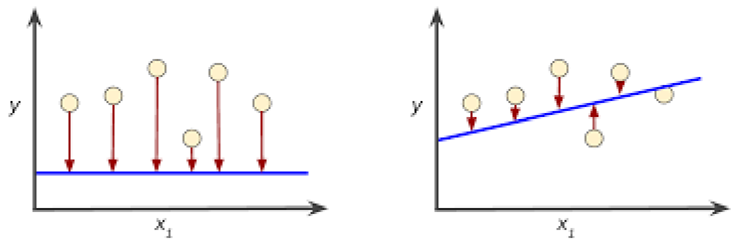
An SVM is a binary classifier; it maps inputs to one of two classes or types. There are no variable mappings. To understand how SVMs function, it is important to note the end goal: "the objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N is the number of features) that distinctly classifies the data points." [5] For an N-dimensional space, a hyperplane is an (N – 1)-dimensional subspace. It is used to separate the data space for easier classification [6]. For example, a 2-dimensional space will be split by a 1-dimensional hyperplane (in this case, a line). As with any other plane, hyperplanes have positive and negative sides. The two classes that an input can be classified as correspond to these positive and negative sides. Class A, in the figure below, would belong to the hyperplane's positive side, while Class B belongs to its negative side.

Optimal placement of the hyperplane would be at the maximum distance from the closest data points on either side of the hyperplane. This is known as a maximum margin separator [7]. These data points have great influence over the orientation of the hyperplane and are aptly name support vectors.



The data points closest to the hyperplane determine its orientation.

One characteristic that makes support vector machines special is their loss function. To preface, a loss function is used to determine the accuracy of a model [8].



The loss is the sum of the distances from the points to the blue line.

Furthermore, the effectiveness of a loss function directly influences the convergence rate (the rate that describes how quickly a model is moving towards an optimal orientation.) There are several loss functions used to train classification models. These include square, logistic, cross-entropy, exponential, and hinge loss. According to Rosasco et al. [9], hinge loss is the most effective because it leads to a better convergence rate.

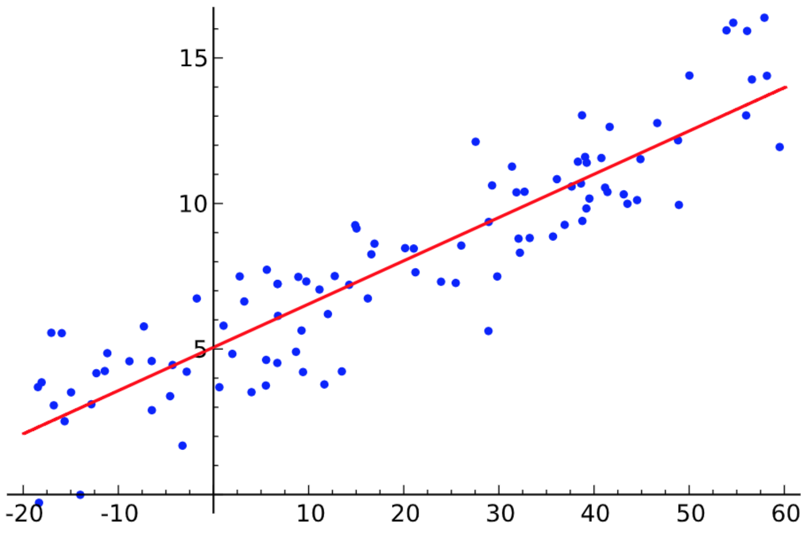
SVMs have been used in applications for face detection, text and hypertext categorization, bioinformatics, protein fold and remote homology detection, handwriting recognition, and classification of images [10]. For our purposes, we wanted to use an SVM to differentiate between images of high and low quality. This was before we learned about convolutional neural networks and their superior performance on automatic feature extraction. If we were to use an SVM, we would have to hand-pick features. More on features and convolutional neural networks in Sections 3 and 4, respectively.

Linear Regression

In this section the terms *regression* and *linear regression* will both refer to *linear regression*, where the regression equation is

$$: y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \ldots, n.$$

This is noted due to their being other forms of regression, for example, logistic regression which is mentioned in a later section. Linear regression is a mathematical technique that seeks to model the relationship of a dependent variable and one or more independent variables. Linear regression is also used as a predictive analysis tool, that allows for the prediction of a dependent variable from new independent variables. This is possible as long as the data being used is mostly linear or has a best fit that is linear. Another requisite to having a linear regression model is that the incoming independent variables cannot be outside of the range of the dataset. For example, a model that was fit for a dataset that has a range from 100 to 10000 can only predict values within that range. Anything outside of that range is extrapolation and will in most cases influence an inaccurate prediction. For this reason, it is important to have a regression model that is fit to the dataset and to make only valid predictions.



Regression is widely used in machine learning. Applications, where some aspect of regression is used, include the prediction of stock prices, the prediction of home values, and many forecasting applications. With so many applications, regression can sometimes be misused or applied to a system that does not need regression.

At the start of our project, we believed our system would involve some type of regression. We expected linear regression over logistic regression. However, after we dove deeper into understanding our problem we began to realize that our system would not benefit from regression as the main solution. We have not discarded regression from the picture completely but have decided that for our problem we will involve a classification system instead of a regression system. Initially, we believed that our system could use linear regression to predict if the incoming photo was a good or bad photo based on different features we would introduce. However, just by dissecting that above statement it is evident that our system’s true goal is to classify an image and not make value predictions, which is mainly what linear regression does. For that reason, we changed the scope of our problem to that of a classification problem instead of a regression problem.

Although classification algorithms do have similarities to regression algorithms. The similarities were not enough to continue pursuing a solution where regression was the main algorithm for our problem. Some of the main differences between a classification algorithm and a regression algorithm are that most classification algorithms seek to answer a yes or no question. For example, “Is the tumor cancerous?”, or in the case of our project, our question would be, “Is this picture a quality picture?”. Although quality is subjective, our project scope still attempts to provide the user with a binary answer of yes or no. Whereas a regression algorithm usually seeks to provide a value that is on the continuous spectrum and seeks to answer the question of “How much?” or “How many?”. For the scope of our project, we are not concerned with finding out how much quality a picture is expected to have but instead if the picture is of quality. For that reason, we choose a classification algorithm over a regression algorithm. It is not to say that later in our application process we will not be using regression to enhance our application in ways that classification cannot, but for our main algorithm classification will lead the project.

Logistic Regression

In this section, the terms *regression* and *logistic regression* will both refer to *logistic regression*, where the logistic sigmoid function is

$$p = \frac{1}{1 + e^{-y}}$$

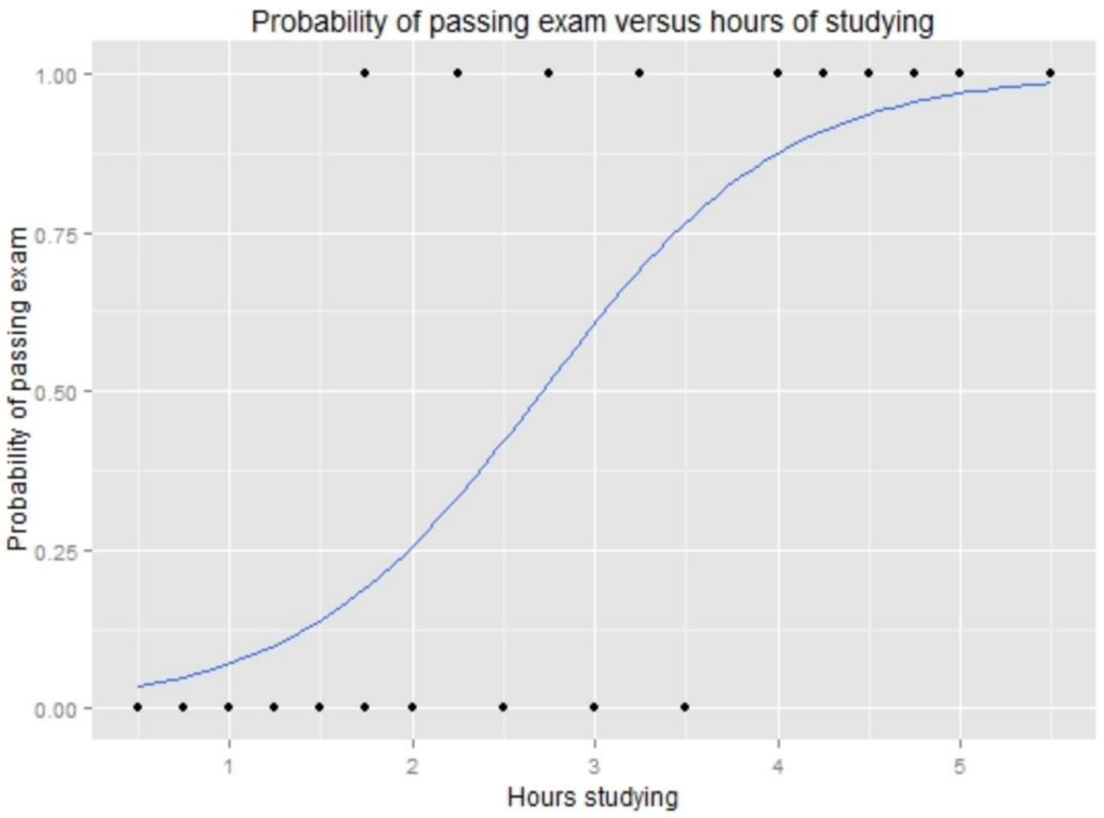
Logistic Sigmoid Function

This is noted due to their being other forms of regression, for example, linear regression which is mentioned in the above section. Logistic regression is similar to linear regression in that it is a predictive analysis tool. However, logistic regression differs from linear regression in that the dependent variable is not a number, but instead a qualitative property. For example, a yes or no prediction or answer. Logistic regression is called regression but behaves more like a classification, where it classifies the dependent variable into classes. Logistic regression uses linear regression to find its best fit line and to indicate whether the dependent variable passes or fails (i.e. if the binary output is 1 or 0).

$$y = \begin{cases} 1 & \beta_0 + \beta_1 x + \varepsilon > 0 \\ 0 & \text{else} \end{cases}$$

The equation above indicates the threshold of the logistic function. As seen, linear regression is used to calculate the upper threshold or class (which is represented as 1) and the other threshold or class is anything that is not calculated by the linear regression equation (which is represented by 0). This way when a dependent variable is classified by logistic regression it is either placed in the class represented by 1 or in the class represented by 0, making logistic regression useful for problems that require a binary output.

The graph below shows the probability of passing an exam versus hours of study. Logistic regression is used in this situation because the question that is trying to be answered is, how does the number of hours studying affect the probability that the student will pass or fail the exam. In other words, depending on the number of hours the student studies will they be more likely to pass or fail their exam. This question then becomes a binary output, either the student passes or fails. Using another form of regression would not be ideal in this situation because the output would not be the binary answer of yes or no that the question seeks to answer.



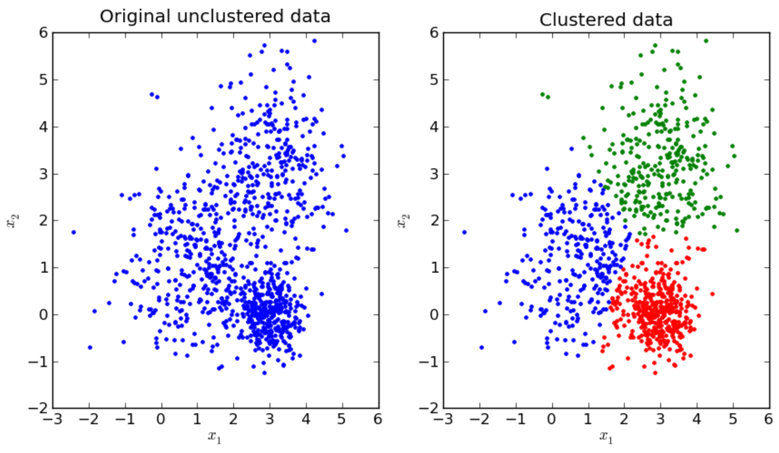
Logistic regression allows for the use of probability to determine a binary output, and for that reason, we would like to test logistic regression within our project's algorithm. At the moment, we have not implemented any logistic regression into our project. However, we believe that this type of regression will be a better fit for classification than other types of classification we have come across thus far.

The next steps to implement logistic regression into our project would be to do more research as to how our data would best fit into a logistic regression model and test whether the output makes sense for the scope of our project. After testing if logistic regression is the best fit for our data, we will need to create a test data set with clean data that has both quality photos (or a label of 1 in our logistic model) and photos we believe do not have quality (or a label of 0 in out logistic model) and test whether our model can correctly predict the quality of a photo. At this point, we will know if we should continue with logistic regression or move on to another form of classification.

Unsupervised Learning

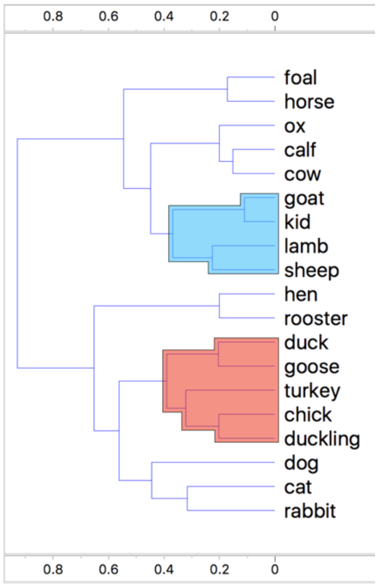
Unsupervised learning is a set of machine learning that deals with data that does not have any prior classification or label. The idea of unsupervised learning is to have a system that is capable of learning on its own. Since the data is not labeled, unsupervised learning algorithms look for and produce patterns within the dataset. These patterns are usually separated based on different features or inputs within the data that is being analyzed. A good way to think of unsupervised learning is to think of the dataset as only (X) inputs where there is no (Y) output to compare too. The goal of the unsupervised learning algorithm is to separate the data into what could be potential (Y) outputs (i.e. what the algorithm believes to be the outputs (Y) to the inputs (X) of the dataset that is being analyzed).

A popular unsupervised learning algorithm is clustering. Clustering is a technique that will run through a dataset and find natural clusters if any exist. Types of clustering are K-Means clustering, Hierarchical clustering, and Probabilistic clustering. K-Means clustering is the technique in where different data points aggregate together because of certain similarities and form a fixed number of k cluster where every cluster has a centroid. Every new data point that is analyzed is referenced with the curtain cluster's centroid and matched to the closest centroid. The number k is a fixed number of these centroids



.

Hierarchical clustering is the clustering of data points based on a Euclidian distance. The most popular technique for Hierarchical Clustering Analysis (HCA) is the Agglomerative approach or bottom-up approach. This data clustering technique works by treating every data point as its own cluster and comparing the similarities with other clusters based on the Euclidian distance. If and once similarities are found with other data points, then those similar data points become a cluster. This cluster is then compared to other data points and a bigger cluster is created. These steps are repeated for all the data points present. Most times a dendrogram is used to show the hierarchy of each cluster and when each cluster was established.



Our senior design project will implement unsupervised learning techniques to find patterns within large datasets and potentially help our application classify each photo correctly. We have decided to try out a clustering algorithm that can run on our dataset and find naturally occurring clusters. The unsupervised learning portion of our project will only be a piece within the entire architecture of our application. The clustering algorithm will work with classification algorithms to output a final result. We will be experimenting with unsupervised learning, Specifically, with K-Means clustering. The reason for choosing K-means clustering is first due to the simplicity of the algorithm compared to other clustering algorithms. Second, is because we believe that our application can be used for classifying many types of photographs. So the idea is to begin with a data set of many different kinds of photographs, with many different subjects, and use the K-Means algorithm to look through our data set. Ideally, our clustering algorithm will find the patterns and cluster each image by similarities. We can then use this clustered data to identify which type of photograph we are dealing with based on the cluster the data most resembles. A set of attributes would be set to each type of photograph we believe our later algorithm will encounter. We will then pass the newly identified photos to our algorithm that will detect if the photos in the data set are good or bad. Where good and bad photos will be based on the attributes specified for each kind of photo we expect to encounter.

Reinforcement Learning

To begin with, the definition of reinforcement learning (RL) does not apply to any single concept. In fact, it describes 3: RL is “a problem, a class of solution methods that work well on the problem, and the field that studies these problems.” [7] Most RL problems share the same defining components: a set of states, a set of actions, a reward function, a policy, and an environment. The environment is composed of states that, in turn, have corresponding rewards. It is in the learning agent’s best interest to maximize the amount of rewards it receives because this will reflect the desired behavior. An agent can gain the most rewards by developing and following a policy that selects actions which lead to states with high rewards.

Policy

Policies are constructs that represent the agent’s decision-making process; they influence which action will be selected in a given state. Suppose we have an agent and a set of states such that the agent is aware of the values for some but not all the states. Choosing the action that leads to the state with the highest value is logical since it has the greatest known probability of maximizing the total amount of rewards earned. However, one of the states with the unknown values could have a higher value than the current highest-value state. How does the agent know which state it needs to go to and consequently which action it needs to take?

This line of questioning leads to a dilemma: should the agent take an exploratory move or an exploitive one? An exploratory move is one that is randomly chosen. It may lead the agent to a new state with a high value. To exploit a move is to take the move while already having knowledge about the value that move produces [7]. It is also known as a greedy move. The policy influences how greedy the agent is and this, in turn, influences how often exploratory moves will be taken.

Choosing to follow policies that are more greedy than exploratory depends on whether the state values can change, and if so, how drastically. For example, if state values do not change, following an exploratory policy to examine all the possible state values allows the agent to find the highest-valued state as soon as possible. Now the agent can switch to a greedy policy and exploit the highest-valued state. On the other hand, consider if the state values are subject to change. The highest-valued state might be the lowest-valued state on the next iteration or the 5th iteration from now. In this case, it is better to have a balance between greedy and exploratory policies.

Markov Decision Processes

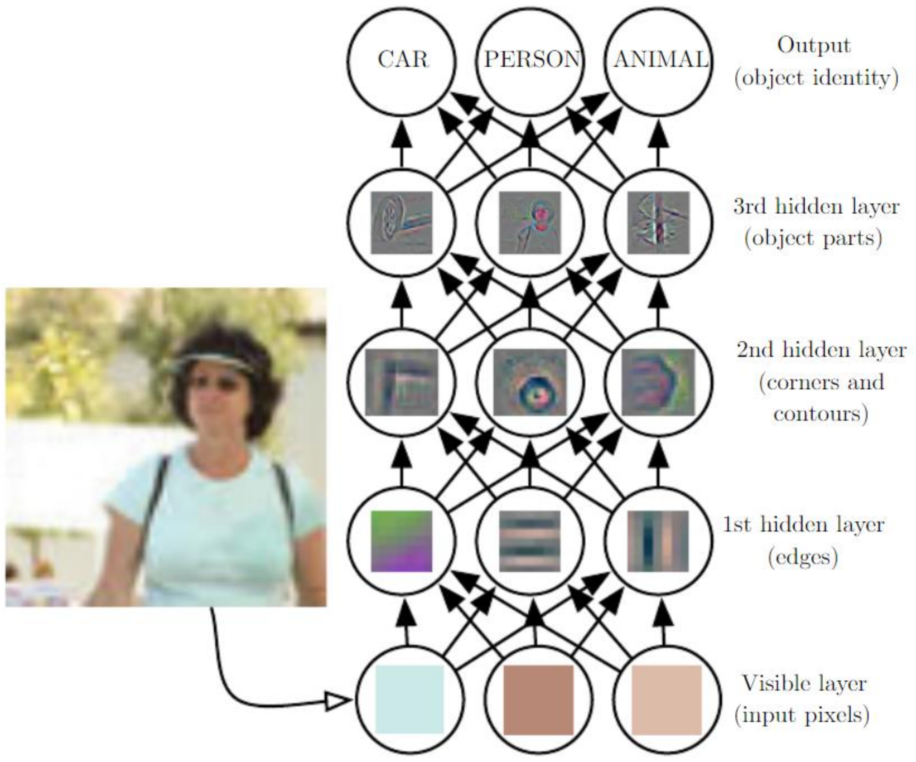
A formalization of the generalizations made above is known as a Markov Decision Process (MDP). The elements of an MDP are a set of states, a set of actions, a set of state transition probabilities, a discount factor, and a reward function [7]. Should a set of states be finite, the corresponding MDP is considered finite. The set of state transition probabilities describes the probabilities of entering a state given the current state and an action. Finally, the discount factor is a fraction that the value of a state is multiplied by. This factor grows in weight as an agent enters states that lead to a positive or negative reward. The reasoning behind this is that the initial states an agent enters are usually more important than the states leading up to the positive or negative reward. Consider an agent playing chess: it played all the optimal moves until the last two where it was subsequently checkmated. We do not want to dissuade the agent from making those initial, optimal moves.

Model-based vs Model-free

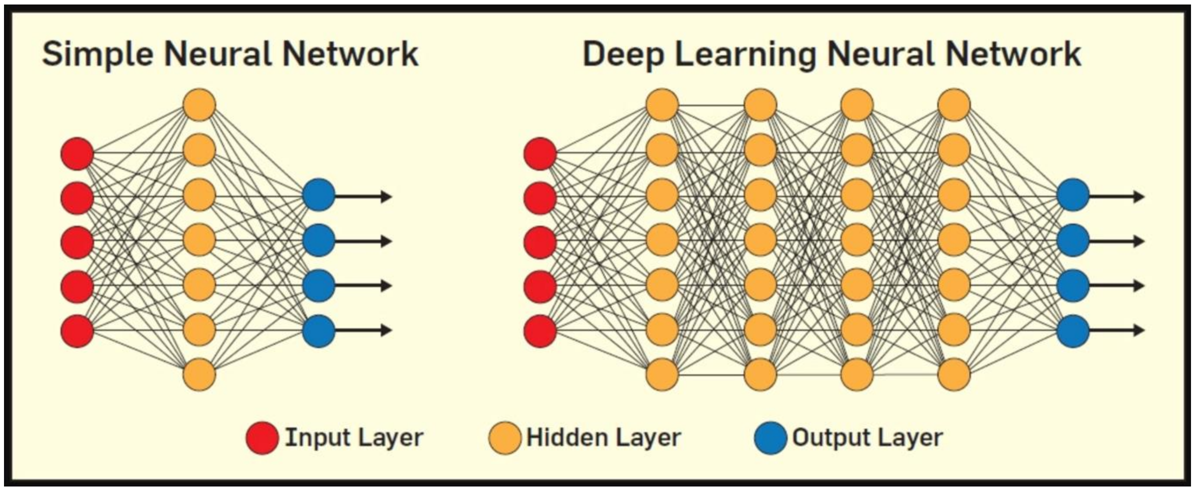
In reinforcement learning, a model is “something that allows inferences to be made about how the environment will behave.” [7] It is useful for a system that requires planning such as an agent learning to play chess. Both immediate and delayed consequences are factored into the decision of which move to make. This includes predicting the states to be seen and their values. It follows that model-based reinforcement learning algorithms that are more complex than model-free ones. The alternative to planning is a brute-force approach; model-free algorithms are simpler but incur higher costs as it is necessary to exhaust the state space to find the optimal action [7].

Deep Learning

Deep learning is a subset of machine learning which itself is a subset of artificial intelligence. Deep learning encompasses the area of machine learning that learns through an architecture of networks, known as neural networks. This architecture introduces an approach to solving problems through representing such problems as the representation of simpler problems. Deep learning enables computers to build complex systems out of simpler concepts. The figure below shows how deep learning can recognize a person within an image through the use of pixels. Where every output of each layer becomes the input of the next layer. Allowing for the architecture to perform complex analysis through the simple representations of the previous output.


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Compared to classic rule-based systems where a problem is solved based on some predetermined features, deep learning uses a system where simple features are extracted, then processed through layers of networks that extract more abstract features. These new features are mapped to other networks that are then mapped to an output. Comparable to the above example, the first features extracted are simple features, such as the input of overall pixel. Then the outputs of those simple features are passed to the hidden layers of the network where more abstract features are analyzed such as edges, contours, and object parts. The next layer does a more complex analysis that finds actual objects within the image. This layer is the last piece of the system where each previous layer helped build up to a final output.



Deep learning can be used to develop applications from complex problems such as image processing. For our project, we propose and believe that the use of deep learning along with other computer science techniques will be the core of our project. The idea will be that our architecture will take images as input and run them through a model. Our model would consist of deep learning neural networks along with other algorithms. The neural networks will work similarly to the deep learning model described above. Taking in each image we present and breaking it down by visible layer, 1st hidden layer, 2nd hidden layer and so on until our model can come up with a desirable output.

Neural Networks

Receptors, a neural network, and effectors are the three stages which are consisted within the neural system of the human body [11]. This is a process of how animals and humans can receive input, whether it be internally or from the external world. They then make decisions based on that input and finally respond to the outside environment. Figure 2 represents the different parts that come into play in a biological neural network. The receptors receive the stimuli, also known as the input. After it receives the input, information is then passed into the neurons in the form of electrical impulses. The neural network receives that as input and then processes them in order to make a proper decision in order to produce a reaction, the outputs. In the end, the effectors translate the electrical impulses which it received from the neural network into responses to the outside world.

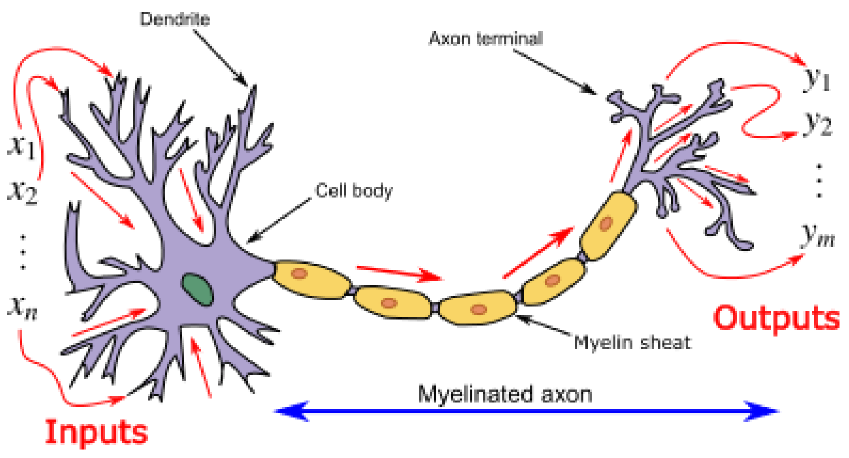


Figure 2: Illustration of a Biological Neural Network.

Artificial neural networks are inspired by the biological neural networks. This is a model which is used for many different kinds of machine learning algorithms. It helps systems “learn” to execute tasks by considering past examples. They are generally not programmed with any specific rules which dictate the tasks. An example of this is like image recognition. If we want a neural network to learn how to recognize a panda, then we would provide the neural network with images that have been manually labeled as “panda” or “no panda.” They will then be able to identify pandas in other images after the learning process. This is done without having any prior knowledge of how a panda looks like. Instead, it is accomplished by automatically generating identifying characteristics from the learning material that they process in the beginning stages. Initially, in an effort to solve problems in the same way as the human brain, artificial neural networks have been and continue to be extensively researched and developed. It is now performing a variety of tasks, such as computer vision, speech recognition, social network filtering, and much more.

The structure of an artificial neural network is similar to that of a biological neural network. It contains elements called artificial neurons. There is an input layer which is consisted of an input node(s) that receive data. There is an output layer which is consisted of the output node(s) which produces the appropriate output. Finally, there is a section called the hidden layer. There can be multiple hidden layers within a neural network. The job of this layer is to perform an action called forward propagation, this simply transforms the inputs into something that the output layer can use. It is also used to perform a backpropagation, which is commonly used to train the neural network. Figure 3 represents the different types of layers in an artificial neural network.

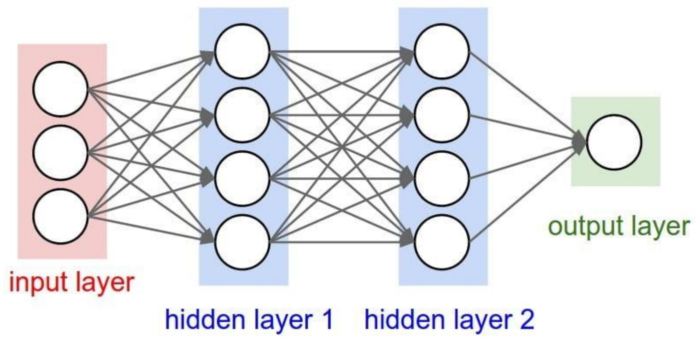


Figure 3: Illustration of an Artificial Neural Network

Forward Propagation

Forward propagation applies a series of functions to the data that was inputted. The type of function depends on which neural network you decide to use. The job of this layer is to transform the inputs into something that the output layer can use. An example, Figure 4 shows the basic computational unit for a single neuron may be consisted of two inputs (x1,x2) which is the data you want to model, weights (w1,w2) which are model parameters, and a bias (b) which allows you to shift the activation function by adding a constant. These values will be used to compute the weighted summation (z) in order to perform linear modelling: z = x1*w1 + x2*w2 + b. After you compute the weighted sum, you will then apply the weighted sum to an activation function (a). The value of the activation function will then become the output of that unit.

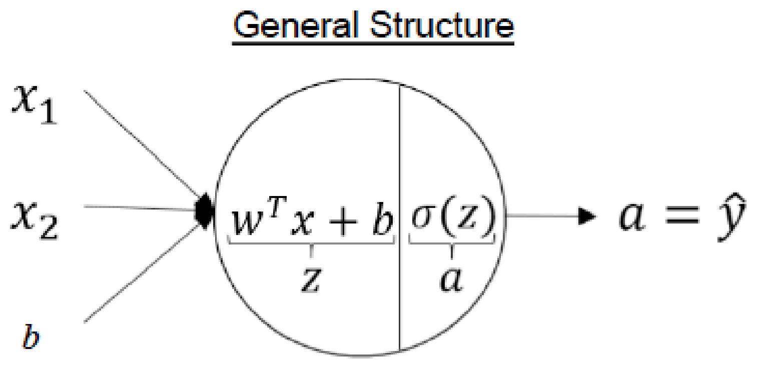


Figure 4: Illustration of the basic computational unit of a single neuron

Activation Functions

As said in the previous section, the value of the activation function for a particular node defines the output of that node, or “neuron.” The output of this function is then used as the input for the next node. Activation functions are inspired by the way our brains work. Different neurons fire, or are activated, by different stimuli. Activation functions are used to determine the output of neural networks. It is like producing a *yes* or a *no*. These values represent whether the neuron is firing or not. Figure 5 shows different types of activation functions.

Activation functions introduce non-linear properties to our network. This makes it easy for the model to generalize or adapt to a variety of data and to differentiate between the output. This is the reason why non-linear activation functions are the most used. Linear activation functions are straight lines thus, they do not adapt to a variety of data the way non-linear functions do.

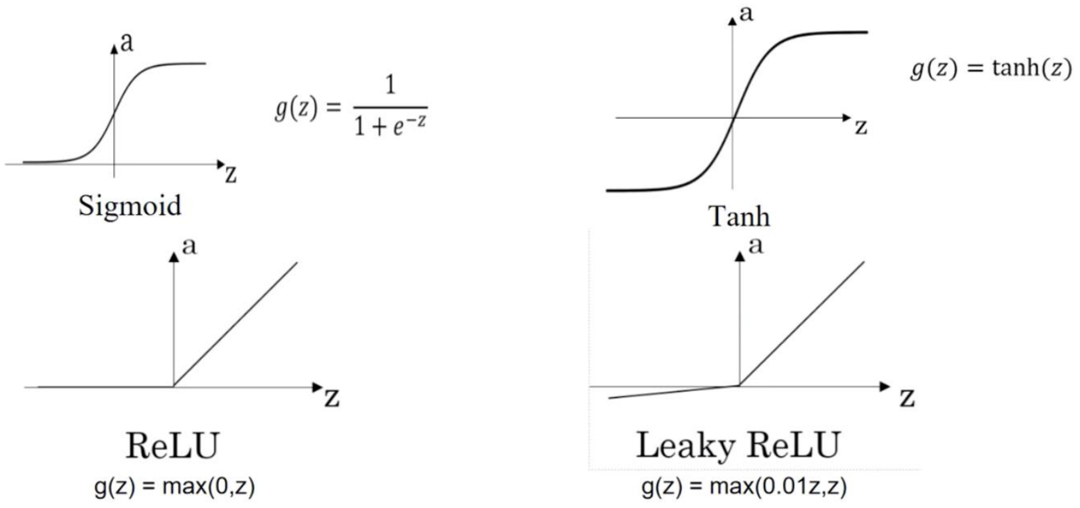


Figure 5: Illustration of some activation functions

Backpropagation

Backpropagation is used to train neural networks. To see the accuracy of the predictions made by the neural network, a lost function must be calculated. This is used to measure the inconsistency between the predicted and actual value. Backpropagation is used to update each of the weights in the network so that they cause the actual output to be closer to the target output, thereby minimizing the error for each output neuron and the network as a whole.

Gradient Descent

A gradient descent measures how much the output of a function changes if you change the inputs by just a little bit. This is a commonly used optimization algorithm while training a machine learning model. It tweaks model parameters iteratively to minimize a given function to its local minimum. At each step, the gradient descent tries to converge to a minimum.

Steps in Gradient Descent:

1) Perform a forward pass
2) Calculate the error
3) Backpropagate error as gradients
4) These gradients at each step update the weights (w) using the equation:

   $w^{k+1} = w^k$ – learning_rate * gradient
5) Perform above steps iteratively until error reaches a minimal value

The learning rate is a major component of the gradient descent. Learning rate decides how big the steps are that the gradient descent takes in the direction of the local minimum. For the gradient descent to reach the local minimum, we have to set the learning rate to an appropriate value, which is neither too low nor too high. In the figure below, it shows where the initial weight is and where the minimum is. The gradient descent will take the necessary steps in the direction of the local minimum. Figure 6 shows the process of gradient descent. It illustrates the small and steady changes to the weights as it approaches the optimal value.

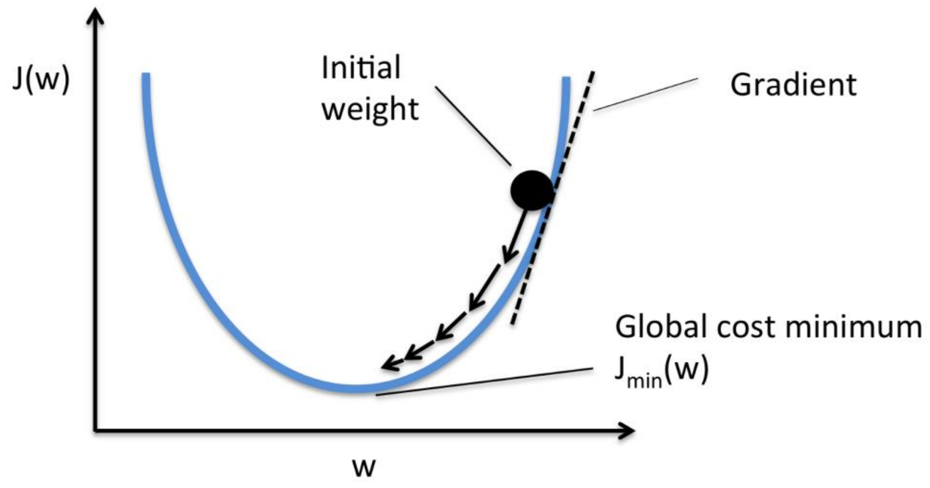
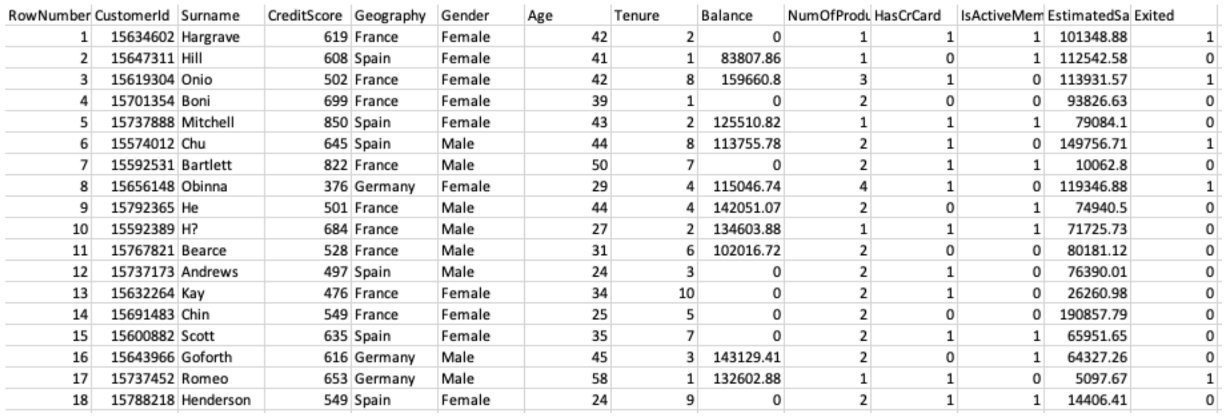


Figure 6: Illustration of the learning rate

Example Artificial Neural Network

In this example, we will build an artificial neural network that a bank can utilize in order to predict whether a customer will leave the bank or not. This model will be trained on a dataset of 10,000 customers. In the dataset, we have metadata about each customer. In Figure 7, you can see what type of information we have for each customer.

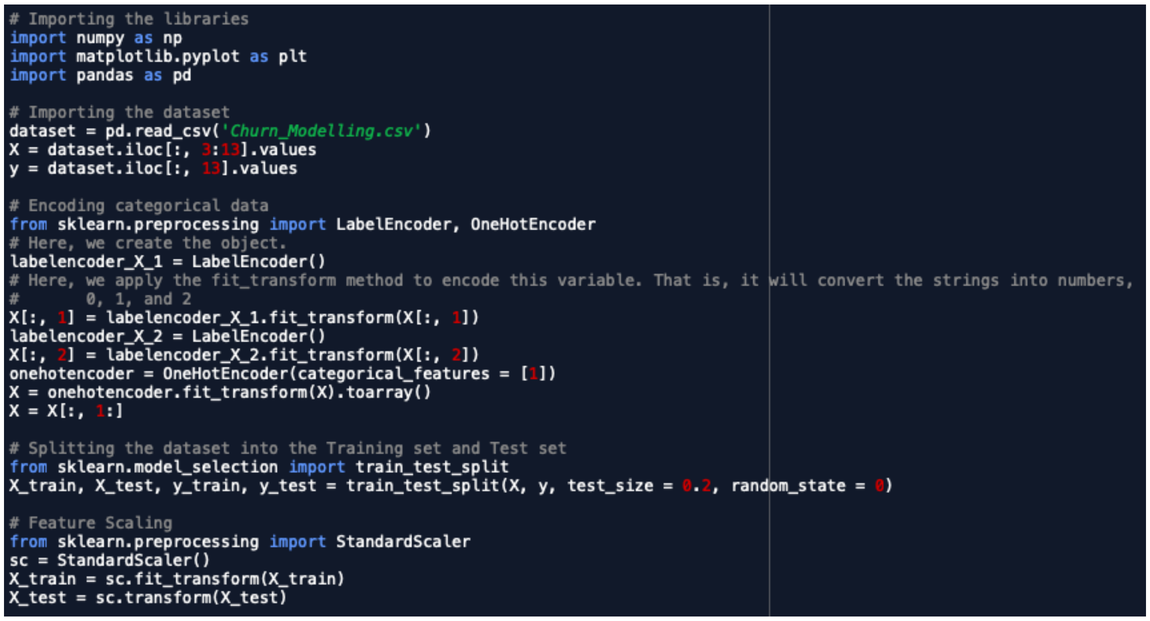


RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProdu	HasCrCard	IsActiveMem	EstimatedSa	Exited
1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0
6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0
10	15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0
11	15767821	Bearce	528	France	Male	31	6	102016.72	2	0	0	80181.12	0
12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390.01	0
13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26260.98	0
14	15691483	Chin	549	France	Female	25	5	0	2	0	0	190857.79	0
15	15600882	Scott	635	Spain	Female	35	7	0	2	1	1	65951.65	0
16	15643966	Goforth	616	Germany	Male	45	3	143129.41	2	0	1	64327.26	0
17	15737452	Romeo	653	Germany	Male	58	1	132602.88	1	1	0	5097.67	1
18	15788218	Henderson	549	Spain	Female	24	9	0	2	1	1	14406.41	0

Figure 7: Dataset from the bank

This artificial neural network creates a lot of added value to the bank because it targets the customers most likely to leave the bank. The bank can use the information which is given by the model to take some measures in order to prevent these customers from leaving.

As you can see in figure 8, we first import the necessary libraries in order to complete the task. We then start preparing the data we received from the bank. We split the data into training and testing sets. The training set is meant to help the network learn and become familiar with the data. The testing set is meant to help verify the accuracy of the network after the model has finished training.

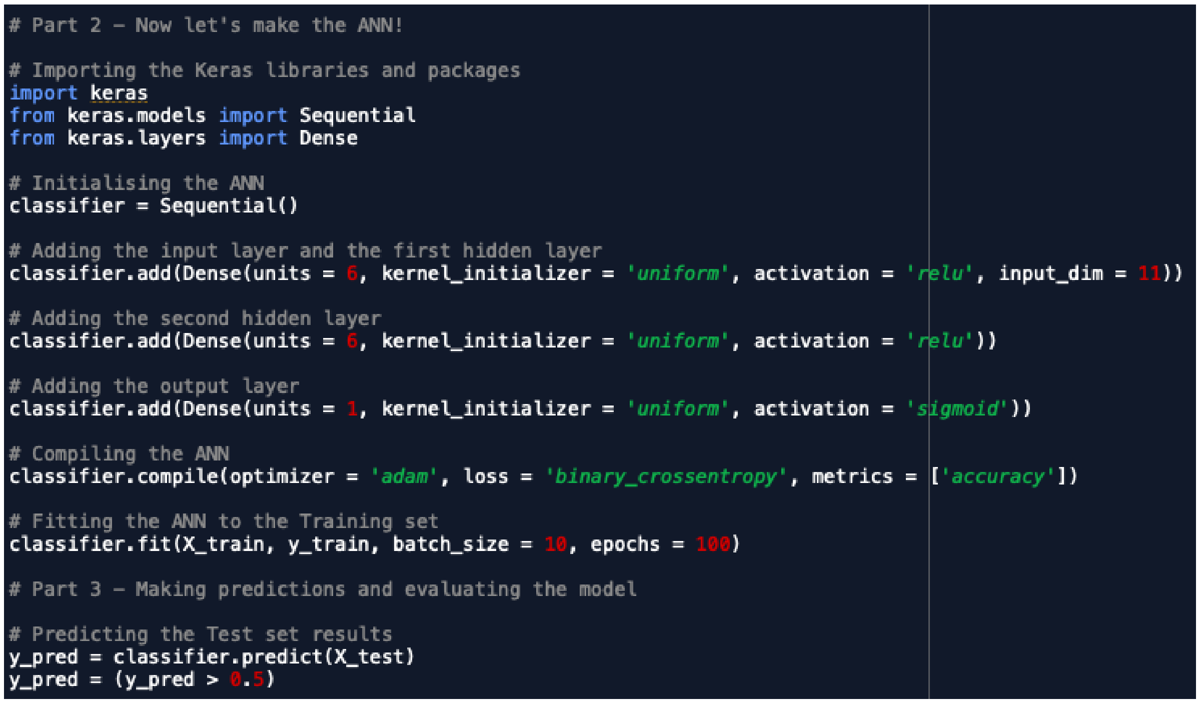


```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# Here, we create the object.
labelencoder_X_1 = LabelEncoder()
# Here, we apply the fit_transform method to encode this variable. That is, it will convert the strings into numbers,
#       0, 1, and 2
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Figure 8: Setting up the dataset

After we have finished prepping the data, it is now time to create the actual artificial neural network, Figure 9. Same as before, we start by importing some libraries and packages that will allow us to create the model. We then initialize the network and define it as a sequence of layers. We are now ready to start adding layers to this network. In this network, we have 2 hidden layers. Each of these layers has 6 neurons. We initiate each layer with weights close to zero, and with ReLu being the activation function. We then add the output layer. There will only be 1 neuron in the output layer. We are using sigmoid as the activation function because we are making a geodemographic segmentation model. Thus, we want to have probabilities for the outcome. We then compile the network and prepare to fit it to the training set.

After fitting it to the training set, we then make predictions and evaluate the model. The test set is used for the predictions and we see how accurate the model was. After training the model and making the predictions, we found that our model reached an accuracy of 85%.



```
# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

# Part 3 - Making predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

Figure 9: Creating the Artificial Neural Network

Convolutional Neural Networks

Convolutional Neural Networks are Deep Learning algorithms. They are similar to artificial neural networks as they also consist of neurons that self-optimize through learning. The main difference between them is that convolutional neural networks are used primarily for pattern recognition within images. This allows us to assign some kind of significant value to different aspects and details in an image. Convolutional Neural Networks are used primarily to classify images, group them by similarity, and perform object recognition. The algorithms can be used to identify faces, individuals, street signs, numbers, etc.. The first successful architecture developed for image recognition was LeNet. It classifies digits and was applied by several banks to recognize handwritten numbers on checks digitized in 32x32 pixel images.

Convolution Layer

The core building block of Convolutional Neural Networks is called Convolutional Layer. The convolutional layer performs convolution operations. These operations are performed using a set of filters. The filters represent learnable parameters for the network. Every filter is a small matrix that extends through the full depth of the input volume. These filter parameters extract complex features from the data. For example, a curve detector filter will result in a high number when it is convolved on a curve in the input image.

Figure 10 represents a simplified version of the convolution layer. In the figure, we have a grayscale image with a size of 32x32 pixels and we use 6 filters of size 5x5x1 to perform convolution. Each filter produces a feature map independently. Thus, we have 6 feature maps corresponding to each filter.

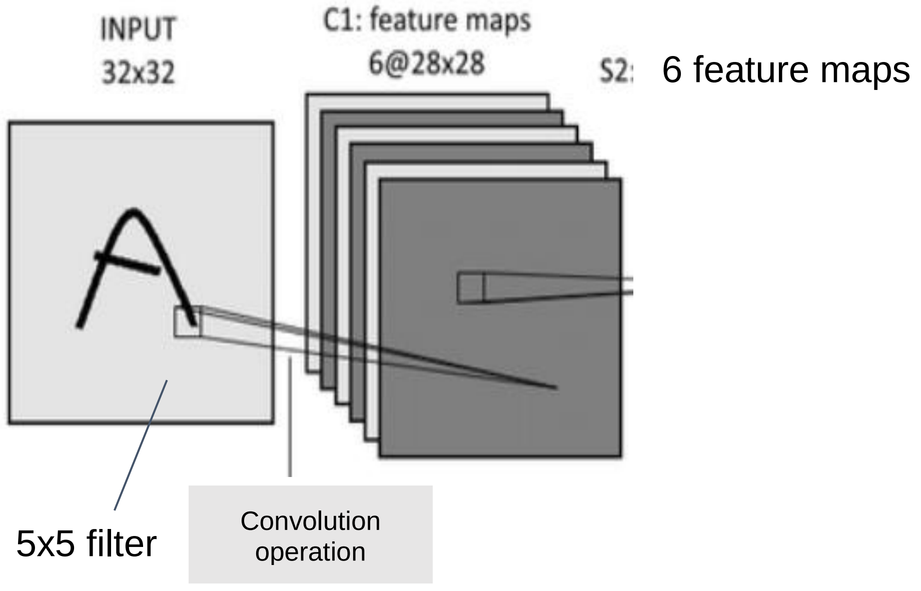


Figure 10: Representation of Convolution Layer

Pooling

In Convolutional Neural Networks, it is common to periodically insert a pooling layer in-between convolution layers. The reason for this is to reduce the number of parameters and computation in the network. It is also to control overfitting. Pooling is done by performing a pooling method to a group of pixels and mapping them to a single pixel. The common types of pooling are Max Pooling and Average Pooling. Max pooling is achieved by selecting the pixel with the highest intensity, whereas average pooling is completed by averaging a group of pixels and passing that value to a single pixel in the next layer.

Convolutional Neural Network Layers

There are three main types of layers when building a convolutional neural network. The convolutional layer, pooling layer, and fully-connected layer.

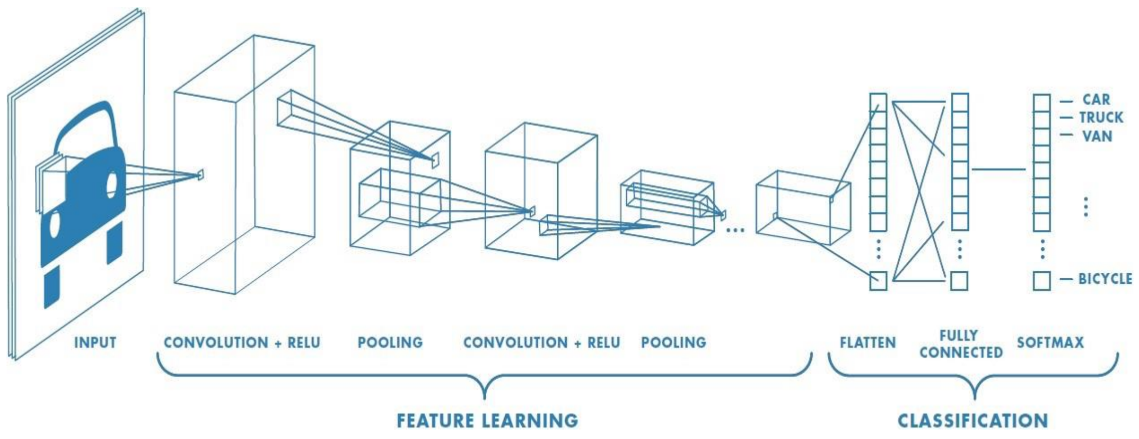


Figure 11: Individual layers in a Convolutional Neural Network

Input Layer

- This layer will contain the raw pixel values of the image. It will contain a width, height, and three color channels R, G, and B. This layer is represented as a three-dimensional array of pixel values. The width is one dimension, the height being the second and the RGB values representing the third dimension.

Convolution Layer

- This is the core building block of a Convolutional Neural Network. There consists of a set of learnable filters, an array of numbers that represents the weights, which will be convolved during the forward propagation. The dot product will be computed between the input and the entries of the filter. An activation map will then be produced. The network will notice certain filters that activate certain types of features at some position within the input. As mentioned before, the convolution of a curve detector filter on a curve in the input will result in a high number.

ReLu Layer

- The ReLu layer, Rectified Linear Unit, applies an activation function to the output from the convolution layer. It is responsible for removing negative values from an activation map. The function which is applied to all of the values is f(x) = max(0,x). The nonlinear properties of the model and the overall network are increased. This is done without affecting the receptive fields of the convolution layer.

Pooling Layer

- This layer is sometimes applied in order to reduce the spatial size of the representation and thus reduce the number of parameters and computation in the network. Maxpooling is the most popular layer option.

Flattening

- In this step, we will flatten the feature maps to a single vector. This is needed in order to make use of fully connected layers.

Fully Connected Layer

- At the end of the network, we attach a fully connected layer. The fully connected layer is responsible for taking in input volume and outputs an N-dimensional vector, N representing the number of classes that the program will choose from. Each value in the N-dimensional vector represents the probability of a certain class. The way this layer works is by taking the output from the previous layer and determining which features match to which class.

Softmax Function

- This function normalizes an input, a vector of size N, into a probability distribution consisting of N probabilities. The softmax function takes the output of each unit and puts it in the range of 0 and 1.

Feature Extraction

In convolutional neural networks, filters are not defined. Convolutional neural networks learn feature extractors. During the training process, the actual values of each filter are learned. This allows convolutional neural networks to find deeper meanings within images when compared to filters that have been designed to perform a certain task. For example, we may use the filter in Figure 12 to blur a picture.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 12: Box Blur filter

Filters in convolutional layers learn to detect things such as the beak of a bird, the face of a person, etc.. This is accomplished by stacking convolutional layers on one another. By doing this, convolutional neural networks are capable of detecting abstract concepts. The image below shows how different types of filters, in each layer, attempt to interpret the number 0.

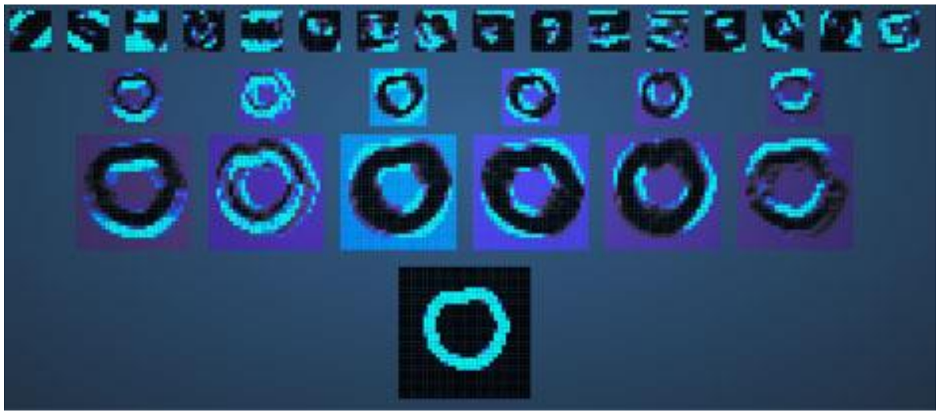
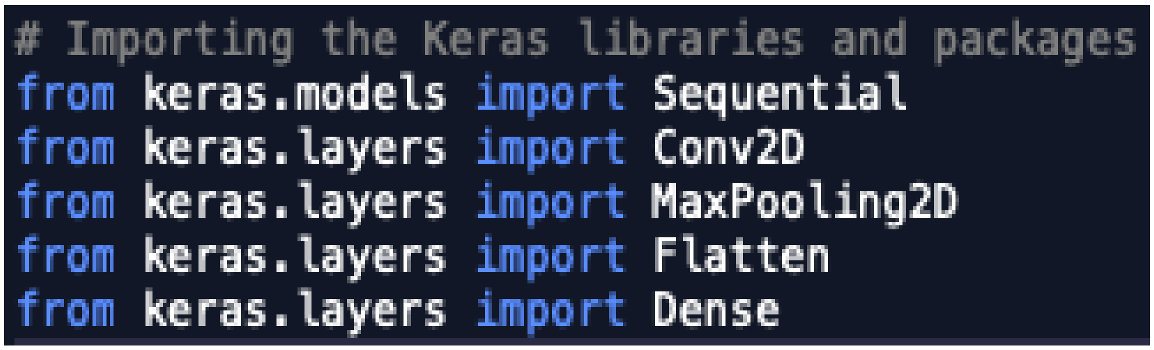


Figure 13: Different filters being applied in order to interpret the number 0

Example Convolutional Neural Network

This is an example of a convolutional neural network which was designed utilizing the Keras libraries. This model can successfully classify images of Dogs and Cats. The model was trained on 8000 images, 4000 cats and 4000 dogs.

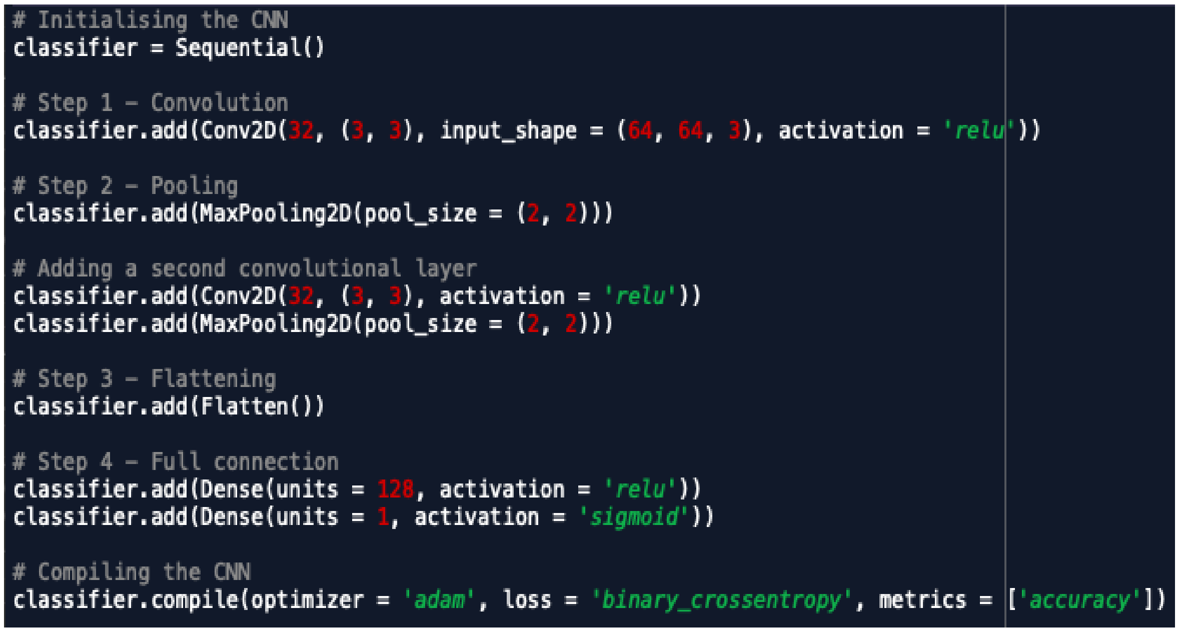
Figure 14 is a list of libraries and packages used in order to build the Convolutional Neural Network. Sequential is used in order to initialize the convolutional neural network. Conv2D is used to execute the convolution step. MaxPooling2D is used in order to implement pooling. Flatten is used in order to flatten the feature maps to a single vector. Finally, Dense is used in order to add the fully connected layer in a classic artificial neural network.



```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Figure 14: List of libraries and packages

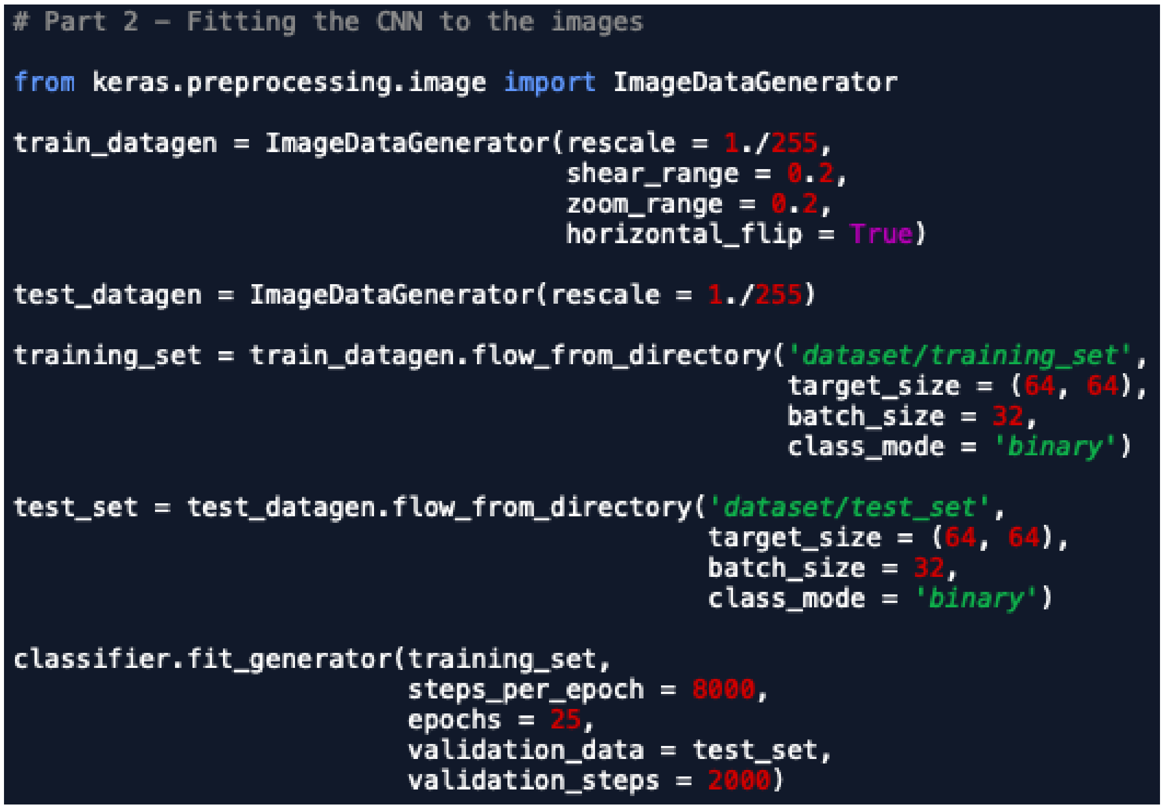
Figure 15 illustrates the steps of building the convolutional neural network. First, we initiate the convolutional neural network. We then perform convolution. Within the convolution, we set the number of filters we want to use, the size of the filter, the dimensions of the image, and the activation function. After we are done with the convolution, we perform max pooling and we set the size of the pooling matrix. We add another convolution layer and pooling operation. After we have completed the convolution layer, we start to flatten the feature maps to a single vector in order to attach a fully connected layer. We then compile the convolutional neural network in order to continue to the next step and fit the model to the images.



```
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Figure 15: Steps to building the CNN

Figure 16 illustrates the steps necessary to take in order to fit the convolutional neural network to the images. We first perform image augmentation. We do this because it will create many batches of our images, and in each batch it will apply some random transformations on a random selection of our images, like rotating them, flipping them, shifting them, or even shearing them, and eventually what we'll get during the training is many more diverse images inside these batches, and therefore a lot more material to train. So, Image Augmentation is a technique that allows us to enrich our data set, our training set, without adding more images and therefore that allows us to get good performance results with little or no overfitting, even with a small number of images. So, we apply several transformations to the images. We then create the training set. We first set the directory of the images and we apply some parameters such as the target size of the images, the batch size, and the class mode. We then do the same thing again but for the test set. Finally, we fit the CNN to the training set, while also testing its performance on the test set.



```
# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = 8000,
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = 2000)
```

Figure 16: Fitting the CNN to the images.

Notable Convolutional Neural Networks

There are many network architectures that have utilized convolutional neural networks. These architectures have been successful in doing things such as object detection and image classification. We will see how some of these architectures work and how they are different from the standard convolution neural network.

For the purpose of computer vision research, ImageNet manually labelled and categorized about 22,000 separate object categories. Now, ImageNet has a challenge called ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The goal of this challenge is to allow collaborators to present their algorithms and compete in object detection and classification precision. As you will see in this section, some notable CNN's has done incredibly well in this challenge.

AlexNet

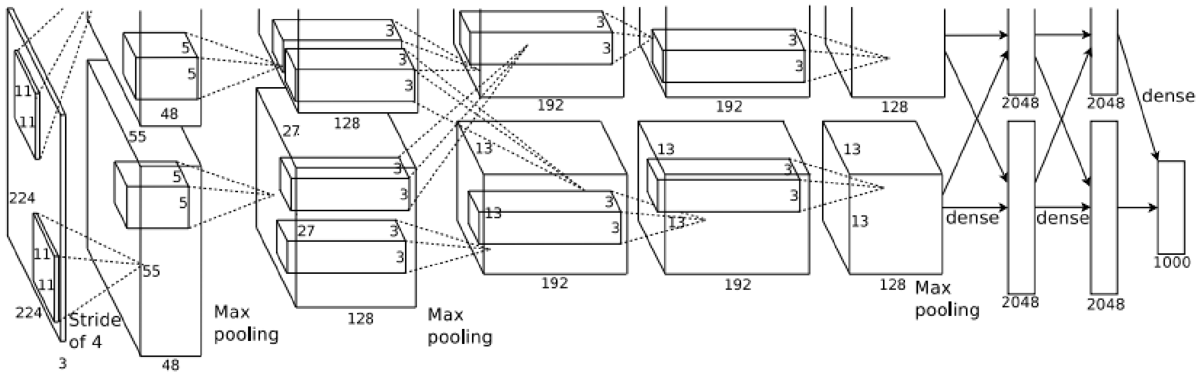


Figure 17: AlexNet representation

AlexNet won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This is a competition where the algorithms are evaluated for object detection and image classification. AlexNet consists of 8 learned layers - 5 convolutional layers and 3 fully connected layers. Key features about the AlexNet when compared to traditional convolutional neural networks are:

- AlexNet utilized the ReLu activation function instead of the Tanh function. This was done in order to add non-linearity. It was found that it was 6 times faster with the same accuracy.
- AlexNet used dropout to deal with overfitting, rather than regularization. The downside to this, however, is that the training time increased.
- Overlap pooling was utilized and it reduced the size of the network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3% respectively.

GoogleNet

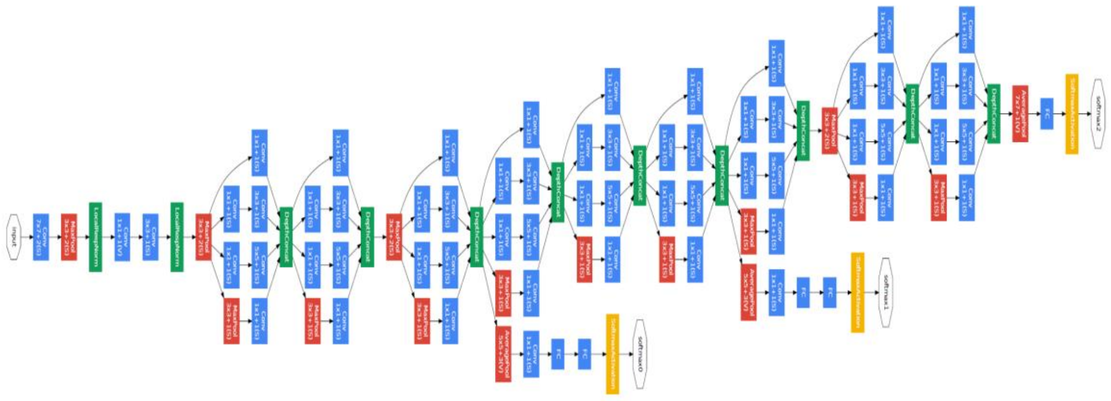
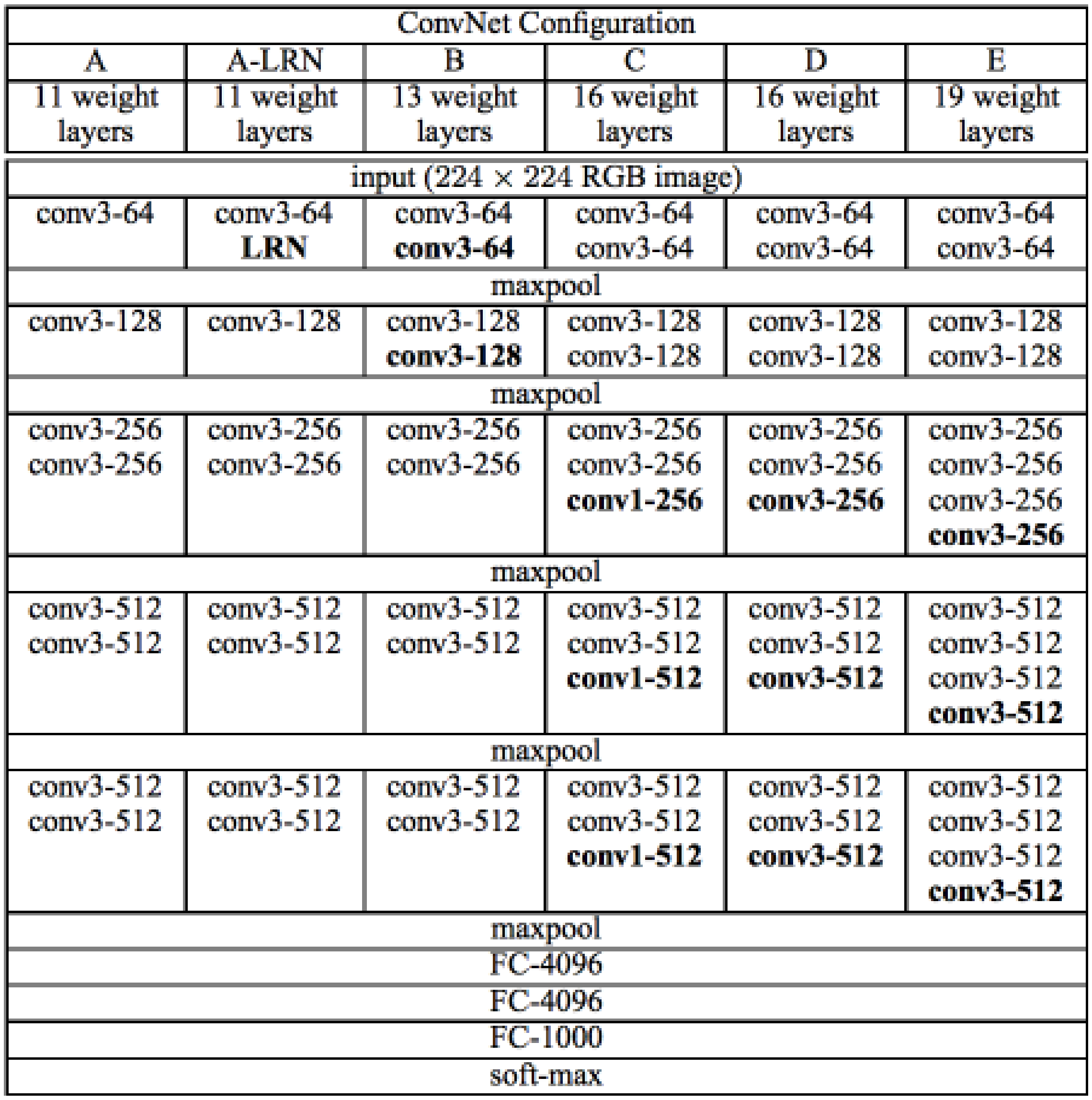


Figure 18: GoogleNet architecture

GoogleNet is the winner of the ILSVRC 2014 competition. A top-5 error rate of 6.67% was achieved. This is very close to the human level performance. The improved utilization of the computing resources inside the network is the main achievement of GoogleNet. The depth and width of the network was increased, and the computational budget was kept constant. The network was able to take advantage of multi-level feature extraction at each step. Different sized filters can be used for different features which are present in the input.

VGGNet



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
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 19: Very deep network architecture.

VGGNet was the runner-up at the ILSVRC 2014 challenge. It is consisted of 16 convolutional layers. Just like AlexNet, it has a lot of filters and only 3x3 convolutional layers. The convolutional layers are stacked on top of each other in increasing depth. VGGNet showed that the depth of the network is a critical component for a network to perform well.

DenseNet

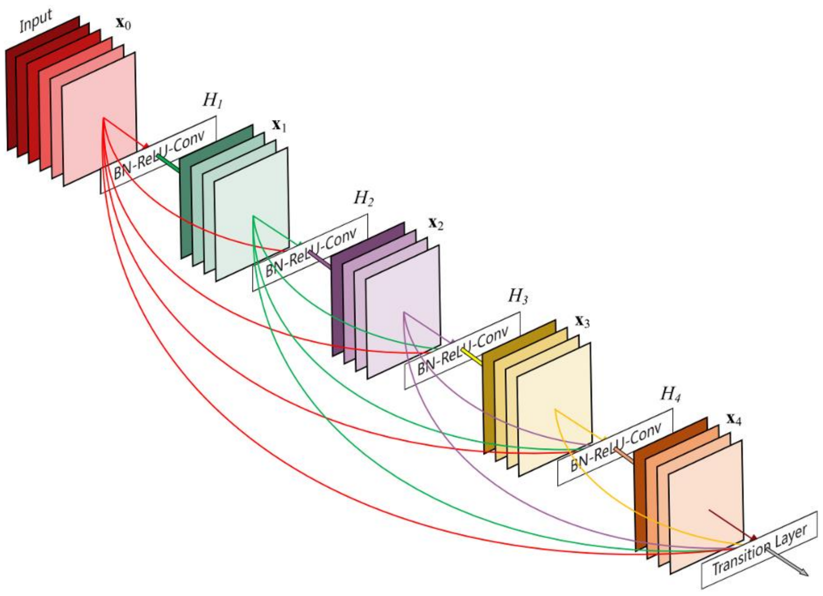


Figure 20: DenseNet architecture

Densely Connected Convolutional Network, DenseNet, is a network architecture. In a feed-forward fashion, each layer within the network is connected to every other layer. The feature maps of all preceding layers are treated as separate inputs for each layer, while its own feature maps are treated as inputs to all following layers. The advantages of DenseNet are, it decreases the vanishing-gradient problem, it enhances feature propagation, it encourages feature reuse, and it reduces the number of parameters [12].

ResNet

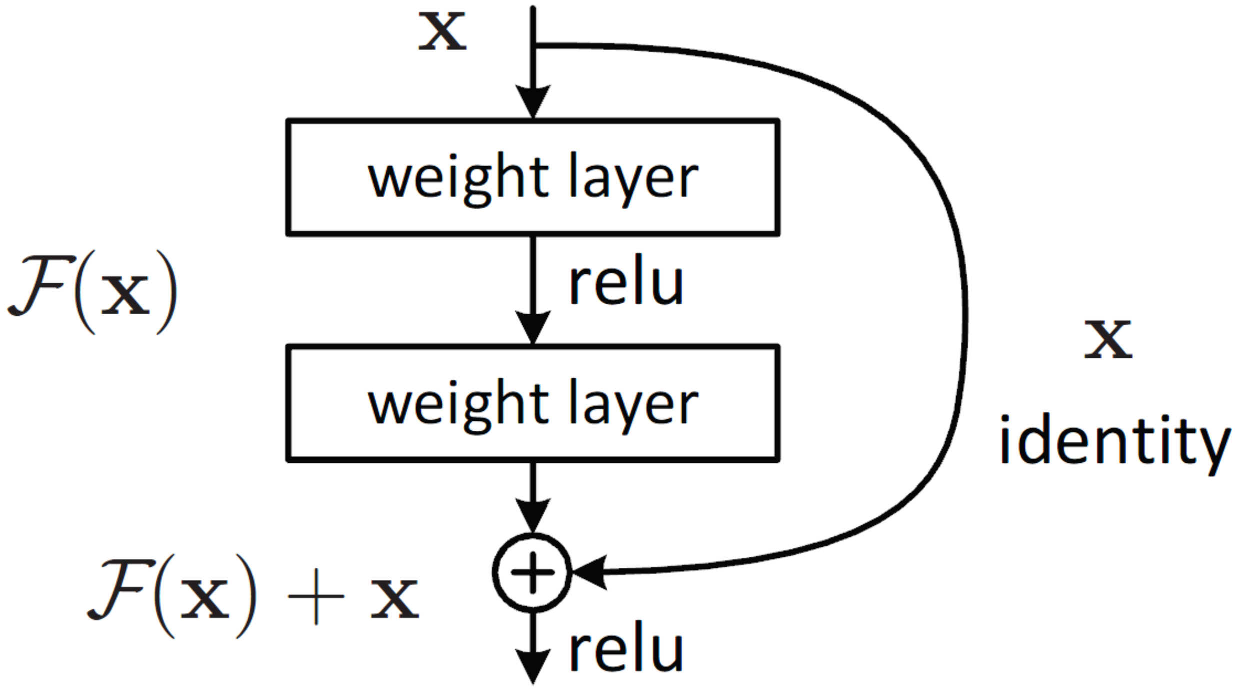


Figure 21: ResNet architecture

Residual Neural Network, ResNet, is a network that has similar structures to the cerebral cortex in the brain. ResNet's execute this by performing skip connections to skip over some layers. One reason why it skips over layers is to avoid the vanishing gradient problem. When skipping is done effectively, it simplifies the network. It will use fewer layers in the initial training stages. The learning becomes faster by reducing the impact of vanishing gradients. This is simply because it has fewer layers to propagate through.

Image Aesthetics

Features

A feature can be described as a characteristic that is measurable to a precise degree. When discussing images, there are low-level, mid-level, and high-level features that can be observed. Some low-level features include the exposure, colorfulness, and texture of the photographs. Mid-level features also called visual attributes, are general qualities that multiple objects can reflect such as "striped" or "round" [13]. Finally, high-level features are related to an image's layout [14]. This includes the spatial composition of the image i.e. the placement of subjects within the scene.

A well-known technique photographers use is known as the Rule of Thirds. See Figure 22 for an image that follows this rule. The idea is to divide the shot into three sections both vertically and horizontally. For a good photo, it is recommended to place subjects along the dividing lines or at line intersections.

In addition to different types of image features, there are different methods for *selecting* the features. One of the goals of feature selection is choosing the least number of features. The more features there are, the higher the cost of analyzing each image. This is known as the curse of dimensionality. The more dimensions (features) there are, the sparser the data is, the harder it is to determine an accurate function. The only way to break the curse is to introduce exponentially larger amounts of data.

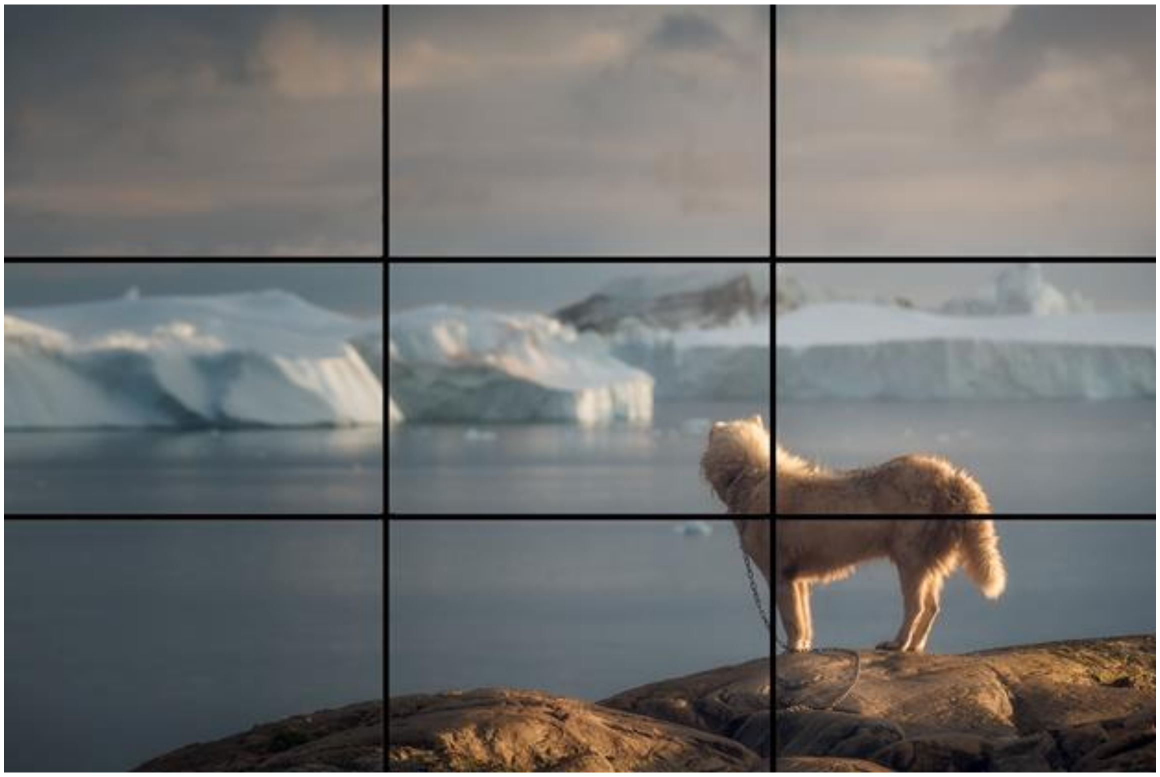


Figure 22. This image adheres to the Rule of Thirds. The dog is positioned at the lower right intersection point.

It is important to note that the task of feature selection is directed at problems where there are not many samples and there are many features. Regarding our project, we have many more samples than there are features to analyze (more than 250,000 photo samples versus an estimated 50 features per photo). We still wanted to choose the best features, however, therefore research into this technique continued.

Types of feature selection algorithms include:

1. Wrappers: these algorithms search through all possible subsets of features while measuring the resulting error rates. This makes them the most computationally demanding but also the most accurate.
2. Filters: the least computationally complex, these algorithms also search all subsets but are faster than wrappers because they use proxies to score

the subsets instead of measuring their error rates. The drawback is that they are not as precise as wrappers.

The challenge of assessing the quality of an image using computer vision has been around for a couple of decades. It follows that multiple image quality assessment (IQA) solutions were devised. Datta et al. used a “hand-crafted” selection technique where they explicitly specified which low or high-level features their models should use [15]. Marchesotti et al. used generic image descriptors that implicitly found features which “hand-crafted” techniques explicitly encoded [16]. Similarly, convolutional neural networks automatically extract features with no direction needed.

Quality Photos vs. Non-Quality Photos

The main question our project tries to answer is whether a photo the user presents is of quality or not of quality. However, this is not such a simple question to answer. The quality of a photo can be extremely subjective. What some people might consider to be a quality photo, others might think that the photo does not meet their standards of quality. Another thought on quality photos can be that some photos are meant to look a certain way, where users that do not take similar photos might deem such photos as bad quality. As none of our team members has much experience in photography, we decided to dive deeper and gain more insight into what makes a quality picture and why some quality pictures are perceived as bad pictures to other users.

After researching varies articles and talking with a few photographers, we found a common theme within the photography community. Most people within the photography community would agree that a good photo is a photo that evokes emotion to its viewers. A photo which contains an interesting subject matter, and a photo that tells a story. It was mentioned in an online article that a good photo sometimes does not require the technical qualities such as lighting, composition, or focus but rather a way to evoke the viewer and an interesting subject matter. With the newly gathered information we decided that out project needs to be more flexible towards all users, but focused enough that our photos would render quality results.

Although the information mentioned above was helpful in providing us with a different perspective on what photographers find as a quality photo, we wanted to focus more on the technicalities of photography and what makes a good photo. We decided to do this by capturing different features of a quality photo and those feature of non-quality photos. Some of the features we were interested in capturing was blur, lighting, frame of subject, and focus. Although some of the feed back we received on what makes a quality photo stated that a quality photo can fail to at all of the features we were looking for in an image, and still be a quality photo. We felt that for our program the non-negotiable for a quality photo needed to be the features that were previously mentioned. It is a difficult task, if not near impossible to have a computer understand an image based on pure emotion and the idea of a story. Therefore, our system needs to have clearly defined features that can better allows our algorithm to detect a quality image from a non-quality image. The following images are similar but their difference in quality is clear.

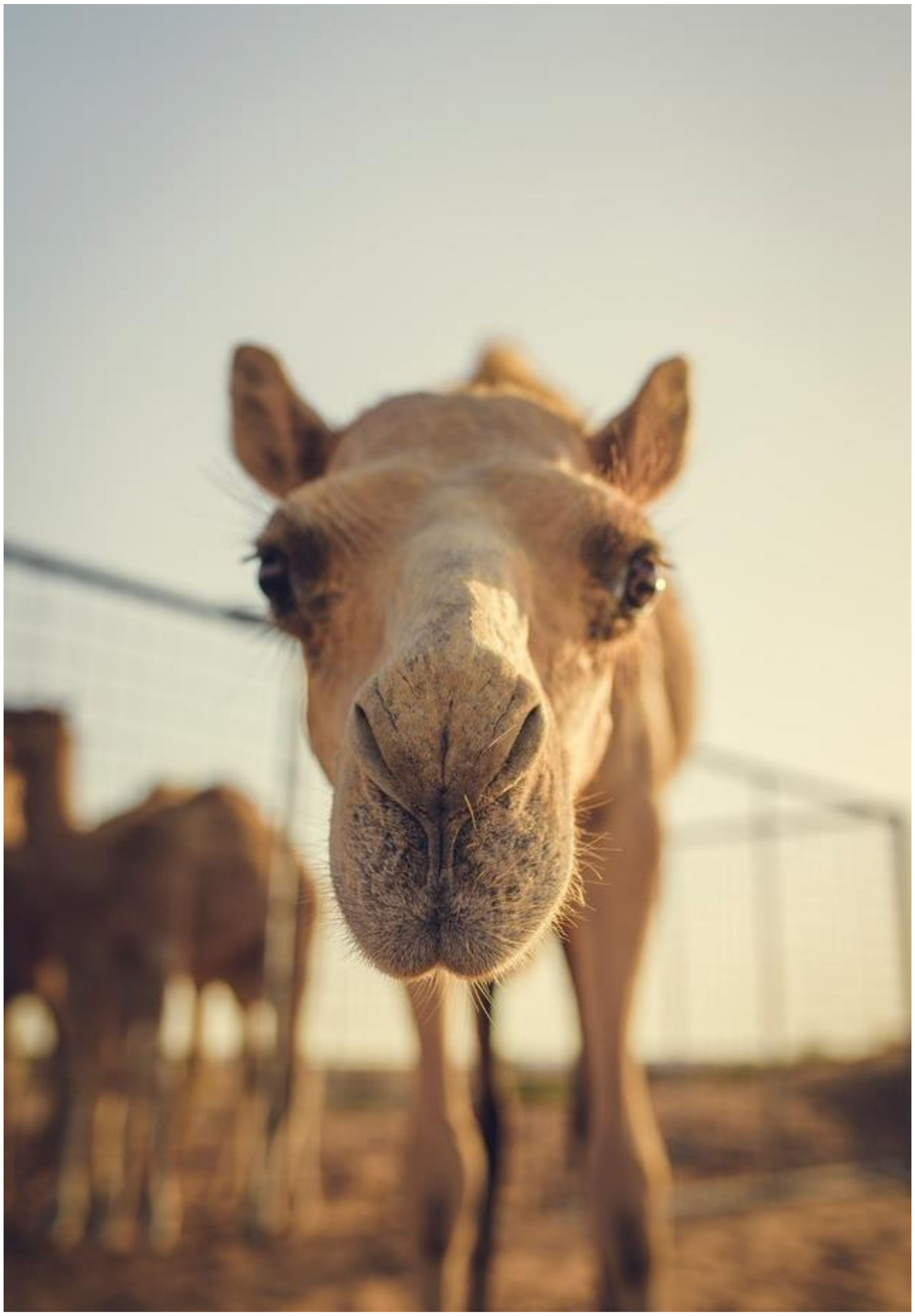


Figure 23: A quality image of a curious camel.

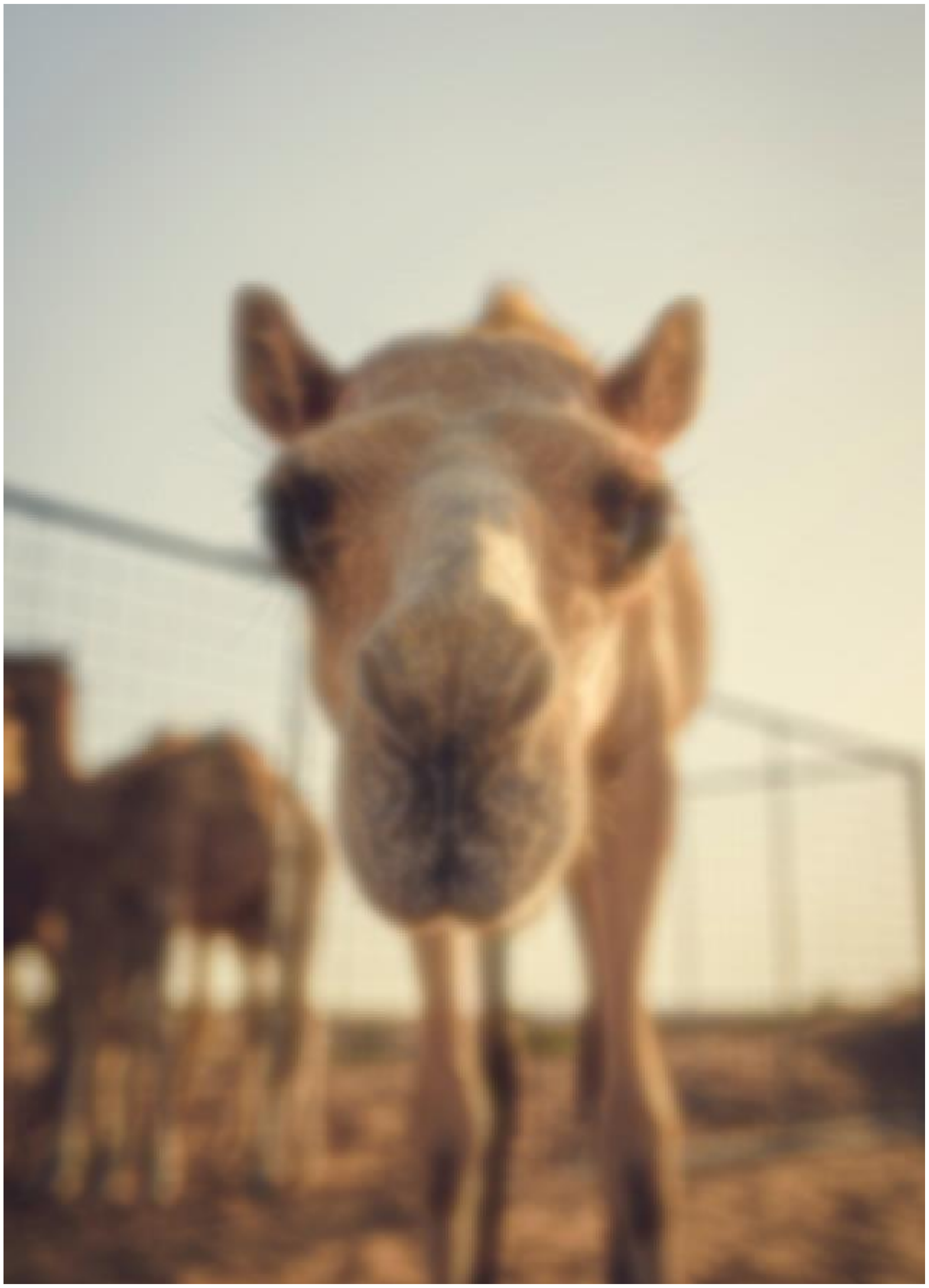


Figure 24: A low quality image of the same curious camel.

Although, both images contain the same subject in the exact same pose it is evident that one image is better than the other. They are actually the same picture with a change in the core features we are interested in defining as quality and non-quality. Figure 23 is clear and focused with a slight blur that adds depth to the photo. Thus Figure 23 is labeled as a quality photo. However, Figure 24 is shows to much blur and makes the photo lose it’s subject. It is still evident that the subject in the image is a camel, but it becomes hard to really tell what features the camel has. Figure 23 shows the the camel’s eye color, the hair that is on it’s mouth, and many other important details that are completely lost in Figure 24. Some photographers or user might in fact want an image like Figure 24 but for the purpose and scope of this project we will attempt to classify all images with t similar features and the features found in Figure 24 as a non-quality image.

PAC-NET

Researchers from the School of Electrical Engineering at Korea University developed a pairwise aesthetic comparison network for image aesthetic assessment. Half of our approach to a solution includes the need to discern image quality on a general level. PAC-NET is just one of the many possible solutions for this side of the problem. What makes this approach more attractive to us is the potential for a more straightforward method to update the network based on the user's feedback. Moreover, "experimental results demonstrate that PAC-NET achieves the state-of-the-art performance in both the ranking and classification applications." [17] While we will not be taking advantage of image classification (high or low-quality image), using the state-of-the-art algorithm ensures optimal performance in our application.

The architecture of PAC-NET is as follows: a pair of images I_1 and I_2 enter separate but identical convolutional neural networks (CNNs); see Figure 25. Each CNN will output a feature vector which serves as input to a comparator. These feature vectors represent the quality of an image. Using a softmax function, the comparator normalizes the feature vectors and produces a rank vector $r = (r_1, r_2)$. If $r_1 > r_2$, image I_1 is of higher quality than image I_2.

A drawback with PAC-NET is their "aesthetic-adaptive" cross entropy loss function. It is built in such a way as to maximize the loss in images that have a great difference in quality and minimize the loss with images that are close in their quality levels. They made this decision so that their network wouldn't get confused while training. A result of this decision is a decrease in accuracy when ranking images with similar quality levels.

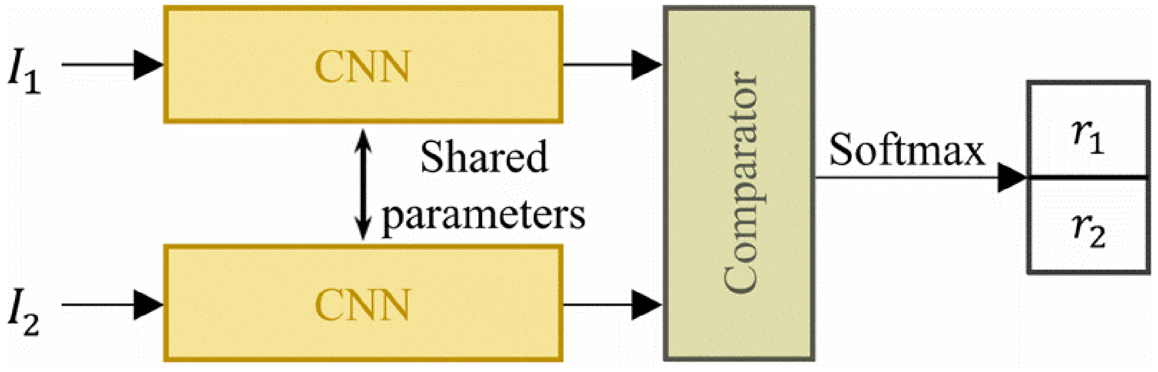
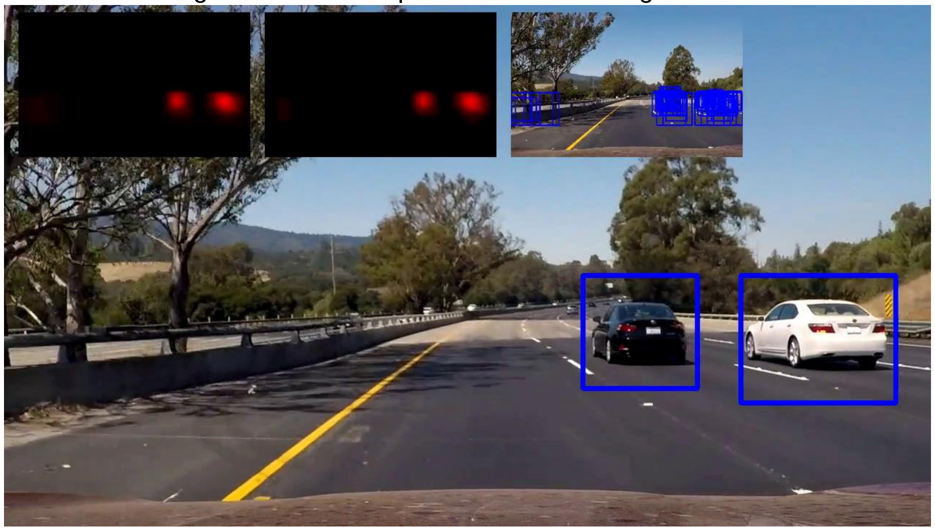


Figure 25: An overview of PAC-NET.

Computer Vision

Computer Vision deals with understanding how computers can be created to acquire, analyze, and process images. One of the main goals is to automate tasks that the human eye can and can't do. This may include extracting high-dimensional data from various images to output numerical or symbolic information. Other than engineering tools to improve our understanding of how we interpret images, there is a discipline which solely focuses on the theory behind artificial systems that use information from images. As a technological discipline, some sub-domains of computer vision include object recognition, scene reconstruction, video tracking, 3D pose estimation, learning, indexing, image restoration, and motion estimation.

Figure above: Computer Vision detecting different cars



Artificial intelligence can be applied to computer vision in order to assist with autonomous planning or deliberation in navigating through an environment. Computer vision collects highly detailed information of each image in order to be processed using artificial intelligence. Techniques to do so include using laser-based range finders (device that measures distance from the observer to a target) or CCD arrays to extract the visual features from the surrounding environment.

Image Filtering

Image filtering can be seen as a process to modify an image in order to improve the image or analyze for information. Images can provide us with plenty of information just by providing different operations onto the image. Processing operations such as smoothing, sharpening, and edge enhancement are all different techniques which can be applied to any given image. An image can be processed using a grid of intensity values. Each space in a grid represents a different pixel in an image. The intensity value of the grid space is broken down into three different values (R, G, B). Collectively, these values create the color being shown in the image. When an image is processed, the kernel, or convolution matrix can be sharpened, blurred, used to detect edges, etc. This is done by performing a convolution between a kernel and image.

Convolution

In image processing, a kernel, convolution matrix, or mask is a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between a kernel and an image. The convolution operation is the process of adding each element of an image to its local neighboring pixel, weighted by the kernel.

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Gaussian Blur

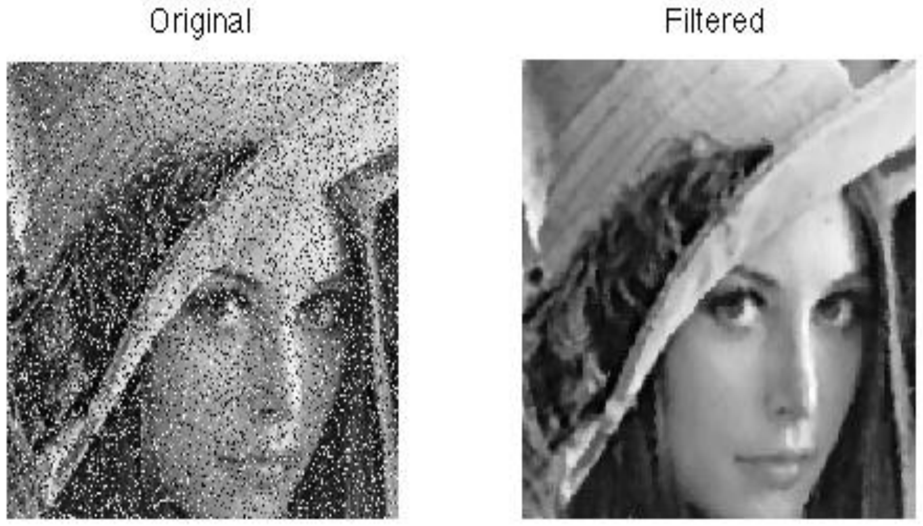
One type of effect created using image processing would be Gaussian Smoothing, also known as Gaussian Blur. This style is widely used to reduce image noise and detail. Gaussian Blur can be used as a pre-processing stage in computer vision algorithms so that image structures can be enhanced at different scales. One major use case would be applying Gaussian Smoothing in edge detection algorithms. Since most edge detection algorithms are sensitive to noise, Gaussian Blur assists in filtering out the noise in an image. The blurring effect is created by convolving an image with a kernel of Gaussian values. The process consists of two different operations. In the first operation, a kernel is used to blur the selected image in either the vertical or horizontal direction. After the image has been blurred, a second operation is performed using the same kernel to blur the image in the other direction.

Box Filter

The most basic filter is the Box Filter. They are frequently used to approximate a Gaussian Blur. The value of each pixel in an image is the average value of the neighboring pixels surrounding it. The algorithm is simple to create but is not as effective to use as a Gaussian Blur. Like the Gaussian Blur, the output image is blurrier than the original. Repeated Box Filtering operations will result in an approximation for the Gaussian Blur.

Median Filtering

Much like the Gaussian Blur, the Median Filter is used to remove noise from an image. This technique is popular in edge detection due to the fact that the edges are more likely to be preserved while still removing noise. Salt-and-pepper noise,



a form of noise that takes the appearance of white and black pixels, can be treated effectively using this method. The algorithm is mainly going through the pixels and replacing each value with the median from the surrounding pixels. One main issue with using this algorithm is the computational effort spent on each calculation. Depending on how the algorithm is implemented, the filtering process could take an extraneous amount of time for each image.

Gradient Operations

In order to change an image by using an image gradient, the intensity or color of an image needs to be changed in a certain direction, as shown above. In a gradient image, the value of each pixel is the result of the change in intensity from that point, in a given direction. Like the previous filters, gradient images can be applied in edge detection. The pixels in a gradient image which contain high gradient values are used as possible edge pixels. The largest gradient values from the pixels in the direction of the gradient are treated as edge pixels. These values can be traced using the direction perpendicular to the gradient direction.

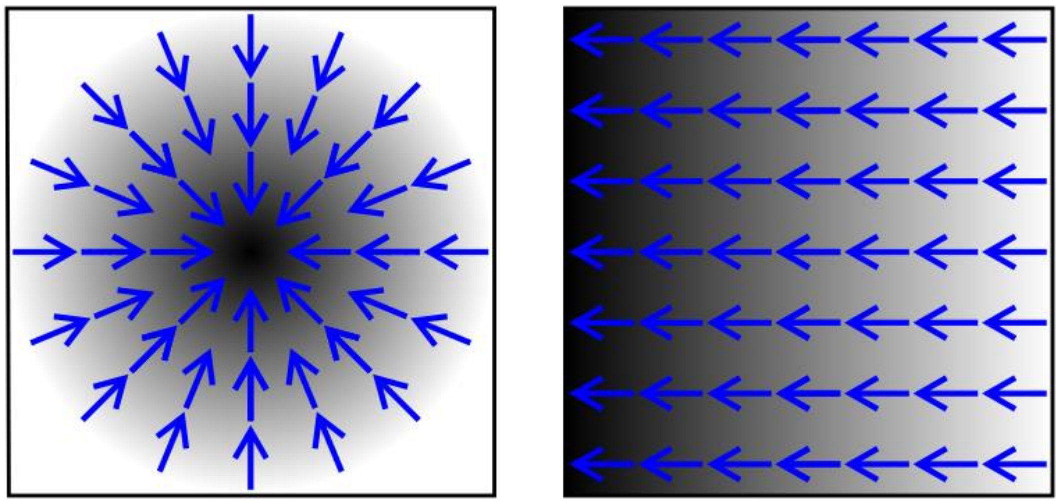
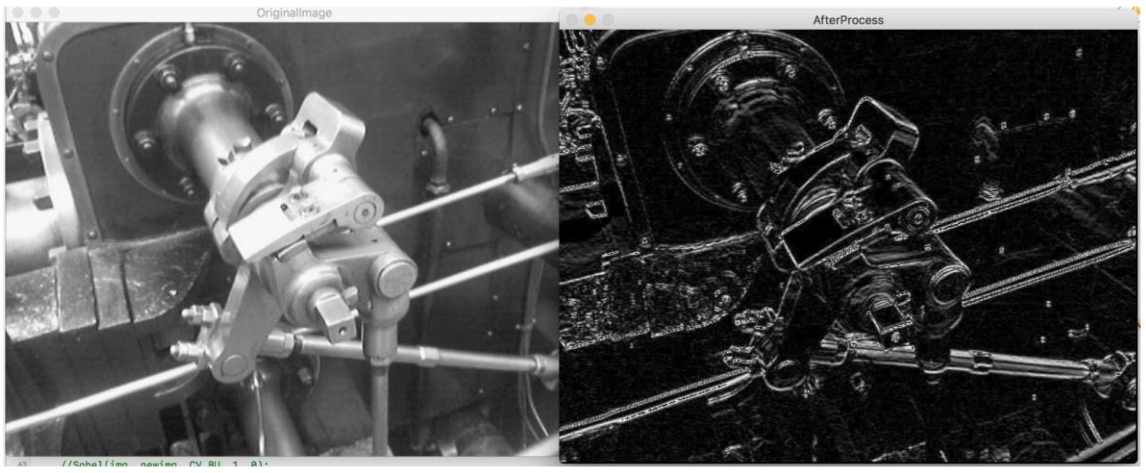


Figure above: Intensity is changed from the center (left), while intensity is changed from the right to left (right).

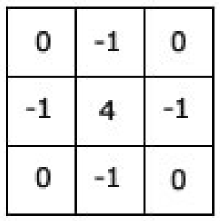
Sobel Filtering

The Sobel Filter creates an image that emphasizes edges. The algorithm computes an approximation of the gradient intensity function. Basically, the Sobel Operator tries to compute the amount of the difference by placing the gradient matrix on every pixel of the image. The result is two images, one for the x-direction and one for the y-direction.



```
//Sobel(img, newimg, CV_8U, 1, 0);
```

Laplacian Operator



0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

One of the most basic techniques for resolving an image’s quality is determining the blurriness of the image. Any out-of-focus photograph concentrated on any object will ruin any chances of being a good quality photo. By using an algorithm incorporating the Laplacian operator to calculate certain thresholds of blurriness, non-blurry photos can be extracted from blurry ones.

The difference between the Laplacian Operator compared to Sobel is that Sobel is a first order derivative mask. The Laplacian Operator is a second order derivative mask which can be used to find edges in an image. One of OpenCV's libraries contains a method which calculates the second derivative of an image. For better performance, users should make sure the image is converted to grayscale first. This is to allow the method to be able to convolve with the kernel, typically of size 3 x 3.

The Laplacian Operator can easily distinguish areas of an image that show drastic differences in intensity. By using the NumPy library, users are able to calculate the statistical variance of the intensity levels. With this information, the user can examine the way in which the edges and responses are distributed. In a typical blurry image, the contents of the image are blurred, making it difficult to accurately perceive what the image was trying to display. Because of this, blurry images will not have very many edges, resulting in a low variance. This is unlike a non-blurry image will typically show a high variance and large number of edges.

Once the user has a set of Laplacian variances, they have to be able to interpret the values in such a way to ensure that the image looks as least blurry as possible. There is a certain threshold that is acceptable for clear, non-blurry images that users have to be able to define. The user has to define the threshold of variance in which what constitutes as a blurry image and what is not. The user has to be careful when selecting the boundaries, however. If they select an unrealistically high threshold, images that most would consider as non-blurry would be classified as blurry. The same goes if the user sets the threshold too low. Once the user has selected a variance that they feel classifies the right selection of blurry and non-blurry photos, the photos can be broken down into deeper categories of what classifies as a "good" image.

Canny Edge Detection

Canny Edge Detection is a widely used method in Robot Vision and Computer Vision to detect a wide range of edges in images. John Canny developed the multi-stage algorithm in 1986 along with the *computational theory of edge detection* which elaborates on the method and how it works. The technique extracts important pieces from vision objects in order to reduce the processing required from the data gathered. There are certain requirements that have to be met in order to have a successful edge detection algorithm implementation. This includes making sure the detection captures as many edges possible from the image shown, the edge point detected from the operator is marked on the center of the edge, and finally, all edges in the image are marked once. Compared to other modern edge detection algorithms, Canny Edge detection remains as one of the best and most reliable techniques still used today.

User Interface

One of the requirements for our project was to build the program as a Windows client-side application. After thinking about the potential for this project, though, we decided it would be beneficial to use a cross-platform framework in case we'd like to broaden our support. The first step of constructing the system architecture was to choose the UI framework. Microsoft has three official frameworks: Universal Windows Platform (UWP), Windows Presentation Foundation (WPF), and Windows Forms. While this is not fact-based, the general opinion appears to be that they all have steep learning curves. Furthermore, they are not cross-platform and, while that is not an essential requirement, we would prefer a framework that is. Therefore, we moved on to assess Qt, Electron, JavaFX, and Swing. Half of us have experience with C++ and so more biased towards Qt. JavaFX and Swing also require the user to have the Java Runtime Environment (JRE) installed, which only further distanced our desire to work with them.

Electron

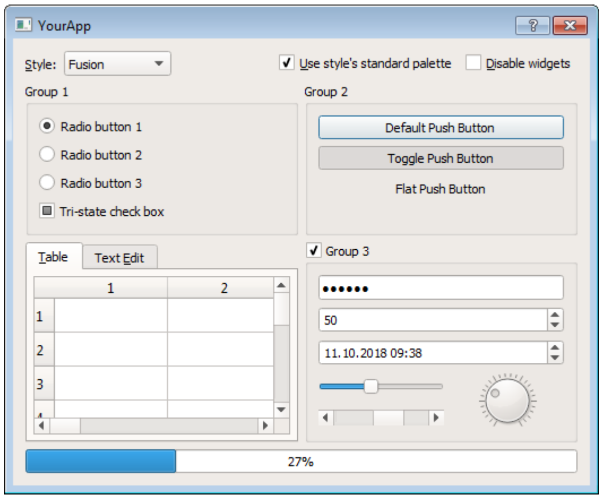
Electron is an open-source framework that allows for the development of desktop GUI software applications. The framework allows for both front and back end development capabilities, traditionally seen in web applications. Electron, originally known as Atom Shell, was developed and still maintained by GitHub. Several known applications such as Discord, Visual Studio Code, and Atom were made using the Electron GUI framework. Since the applications created are used using tools based off of web applications running locally, the components communicating between each other use up more resources than a regular application would. The application may also be vulnerable to any web-related security attacks which makes using Electron not the safest choice for GUI application development.

The framework has an attractive user interface that supports automatic updates, debugging, native menus and notifications. Electron also creates native application and context menus for easy user experience. They also feature crash reporting in case the application runs into unforeseen issues as well as detecting slow performance through Chromium’s content module. This enables smooth installation through Windows using their packages.

PyQt

PyQt is a Python binding of Qt, a GUI widget toolkit for creating graphical user interfaces that can run on various

software platforms such as Windows, MacOS, Linux, and Android. Due to the fact that a majority of image processing applications are created using Python libraries, we looked into using PyQt for tying the application to a graphical user interface for the user to operate. PyQt is free and implements over 400 classes as well as over 6,000 methods and functions. Some of the Python modules include GUI classes, classes for writing User Datagram Protocol and Transmission Control Protocol clients and servers, classes that incorporate 2D and 3D vector graphics, etc.



Style: Fusion
✓ Use style's standard palette
Disable widgets
50
11.10.2018 09:38

Figure above: PyQt UI that incorporates various basic widgets, including progress bar, radio buttons, check buttons, tables, etc.

All the different features you see in a PyQt interface are known as widgets. These comprise of dialogs, progress bars, etc. Widgets are typically nested. An example would be a window that contains a button, which in turn contains a label. Each application is started by instantiating a QApplication. This can be done by importing through the PyQt5.QtWidgets library as shown below.

```
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
app = QApplication([])
window = QWidget()
layout = QVBoxLayout()
layout.addWidget(QPushButton('Top'))
layout.addWidget(QPushButton('Bottom'))
window.setLayout(layout)
window.show()
app.exec_()
```

After including all the widgets into the application, finishing the execution with a app.exec_() will run the program until the user manually closes the application. Qt also has a slot / signal feature. Signals react to certain events that occur, such as the user clicking a button. A slot is a function that gets called whenever the signal occurs. In the case of the user clicking a button, a slot could show a message box. Slots are more important in C++ since they must be handled differently. However, in PyQt, any function can be treated as a slot since we are using Python.

PyInstaller

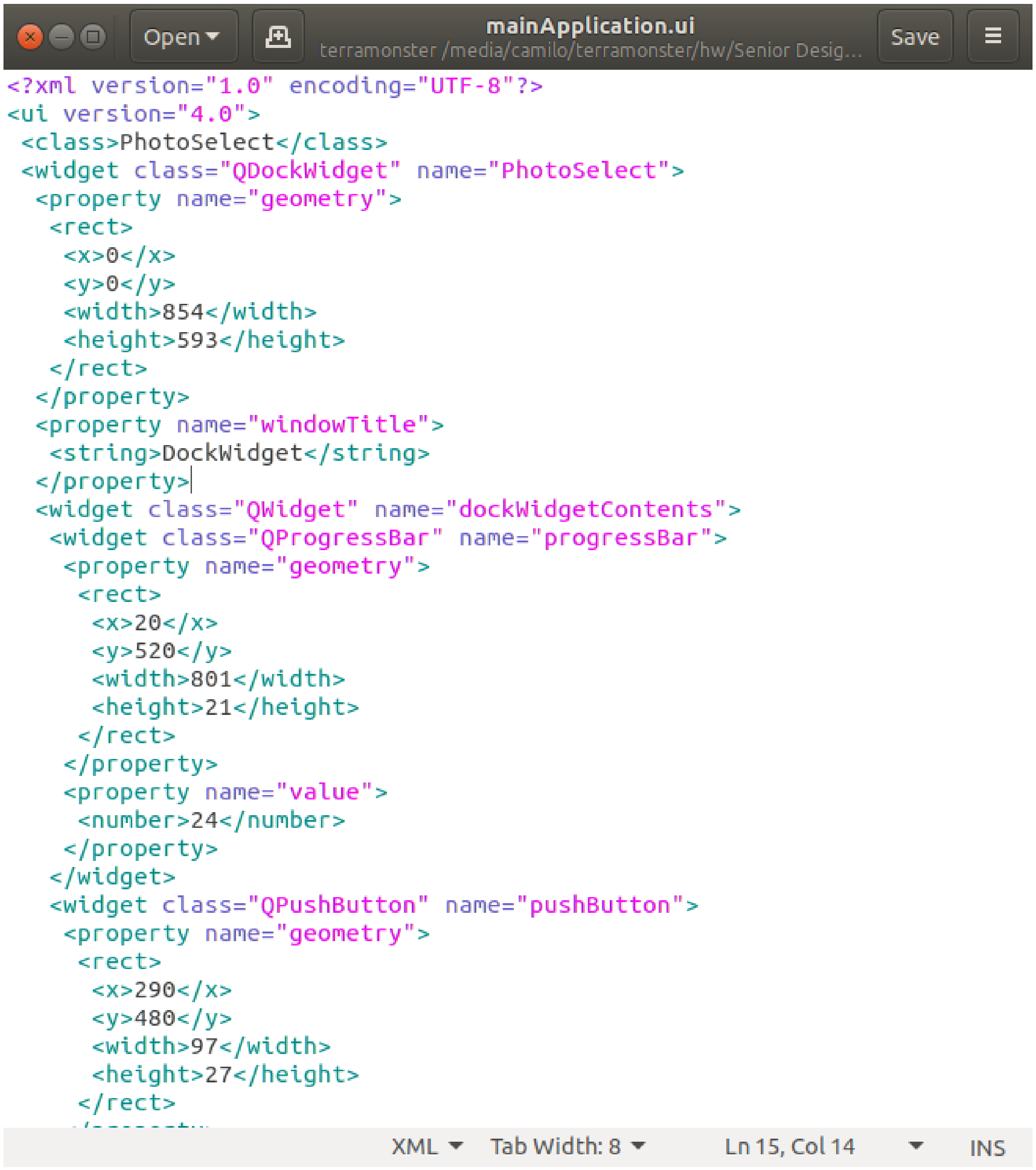
PyInstaller is a tool used to bundle a Python application with all its dependencies into a single package. The user would not need to install any Python interpreter or module to execute the program. This creates a user-friendly method of installing the application for customers who lack any in-depth programming experience. The software operates by parsing through a created Python script to see which modules and libraries the script will use in order to execute the program. PyInstaller then creates copies of the files and places them in a single executable file.

Due to PyInstaller's cross-platform capabilities, we came to the conclusion that we would be able to create a complex multi-functional application while still being able to reach as many users as we can. PyInstaller is also compatible with PyQt4 and PyQt5 which allows for easy integration. The software works for Python versions 2.7 / 3.4-3.7 as well. For the purpose of this project we chose to use PyQt4 since there is more documentation on how to integrate PyInstaller with PyQt4. There is also very little difference between PyQt4 and PyQt5 in terms of functionalities.

Qt 4 Designer

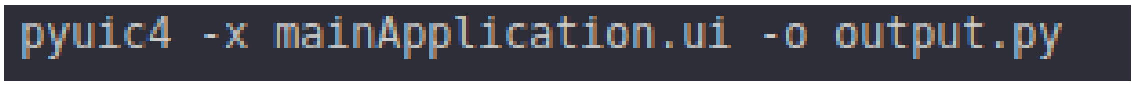
In order to create the interface, Qt provides an application for building and designing graphical user interfaces without having to write any code. Using a simple interface, the user can drag-and-drop the necessary Widgets, Labels, etc. onto main windows. As the user completes their design, they can preview their work to make sure everything works as designed. However, the application does not have the functionality to build the interface into a fully-functional application.

When the user starts up the application, he is prompted to select which type of Widget they will use for their application. From there, the user has a range of options to begin working on their Widget. This can include adding or removing Buttons, Item Views, Item Widgets, Containers, Spacers, Display Widgets, etc. There’s even support for Q3 objects.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>PhotoSelect</class>
 <widget class="QDockWidget" name="PhotoSelect">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>854</width>
    <height>593</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>DockWidget</string>
  </property>
  <widget class="QWidget" name="dockWidgetContents">
   <widget class="QProgressBar" name="progressBar">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>520</y>
      <width>801</width>
      <height>21</height>
     </rect>
    </property>
    <property name="value">
     <number>24</number>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>290</x>
      <y>480</y>
      <width>97</width>
      <height>27</height>
     </rect>
```

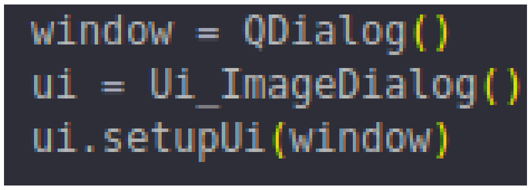
Each creation in Qt 4 Designer is saved with a ui extension. If you open the file, you will see a widgets with their properties and configurations displayed in xml formatting. In order to translate the ui into working Python code, there is a pyyuic4 command line utility which is a wrapper for the uic module.



```
pyuic4 -x mainApplication.ui -o output.py
```

Running this command with the “-x” will generate a self-executable standalone python application. For the purpose of this project, however, we need to include more behavior to the user interface, so the “-x” was removed from the command. As a result, just an “output.py” file was generated which can display all the bare bone user interface created in Qt 4 Designer.

The python file that is generated has a very similar design to Qt’s “uic”. The name of the created class is the name of the toplevel object set in Qt 4 Designer with Ui_ as the beginning (e.g Ui_Train). This is typically referred to as the form class, which can be seen as soon as you begin creating a project. Each form class contains a method called setupUi(), which takes in a Widget as its only argument. The Widget is treated as the Qt base class since its the Widget in which the user interface is created.



```
window = QDialog()
ui = Ui_ImageDialog()
ui.setupUi(window)
```

Here you can see how a newly generated python file uses the QDialog object to setup the application. The QDialog object in this case is ImageDialog. In order to actually the class you would use show(), which in this case would be window.show().

User Interface Draft

This section contains a mock-up of Slekit's user interface along with some explanation of the core functionality the user interface, will provide to users. The user interface for Slekit is a simple user interface that attempts to make the process of evaluating a user's photos simple and with as little effort as possible. Since Slekit will be a native application, the start up process will happen from the user's main operating system's screen or wherever the user decides to store their application. Once the application is opened Slekit will run and default to the main user interface screen.

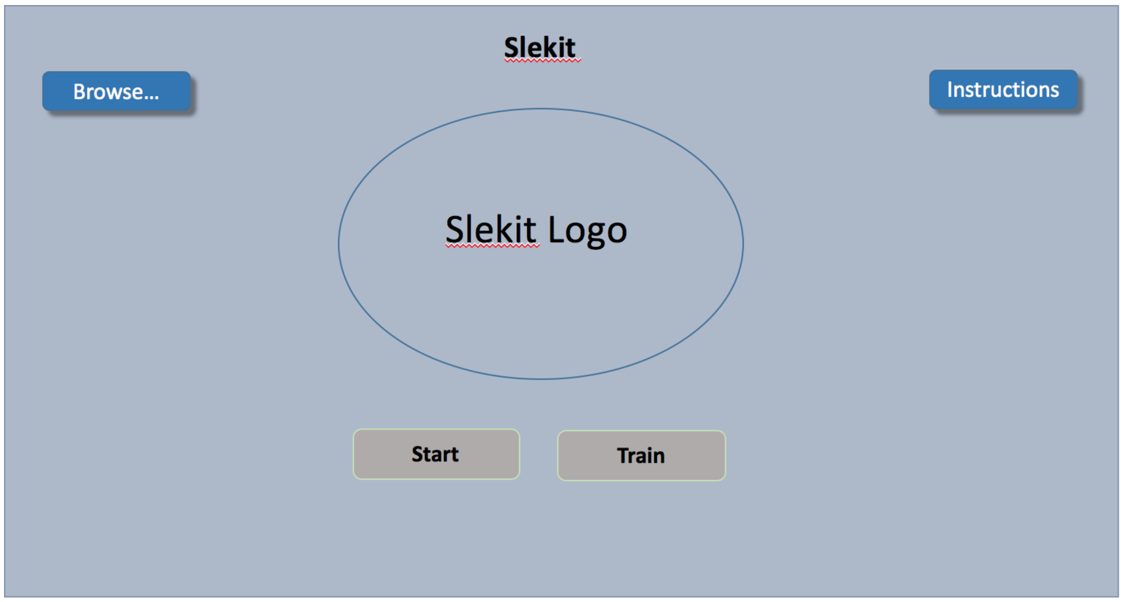


Figure 28: Slekit Main Screen

Figure 28 shows Slekit main screen upon the start of the application. The main screen is a simple start page where the user will have some areas of functionality. Upon the start of the application the user will only have access to the “Browse” and “Instructions” button. The “Start” and “Train” button will be showing but will not be clickable until the user as selected a directory that contains images. Once the user has selected their directory that contains images will the “Start” and “Train” buttons become available.

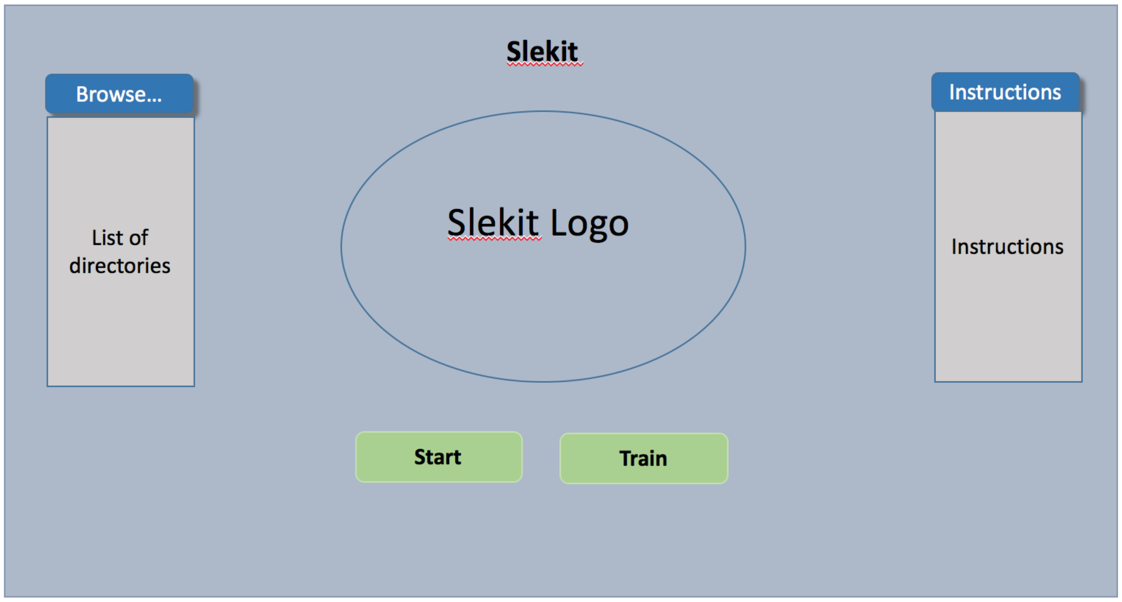


Figure 29: Slekit Browse and Instructions

Figure 29 shows the options for browsing a directory and displaying the user interface instructions. After the user has selected the directory where the images are stored, the “Start” and “Train” buttons will become clickable.

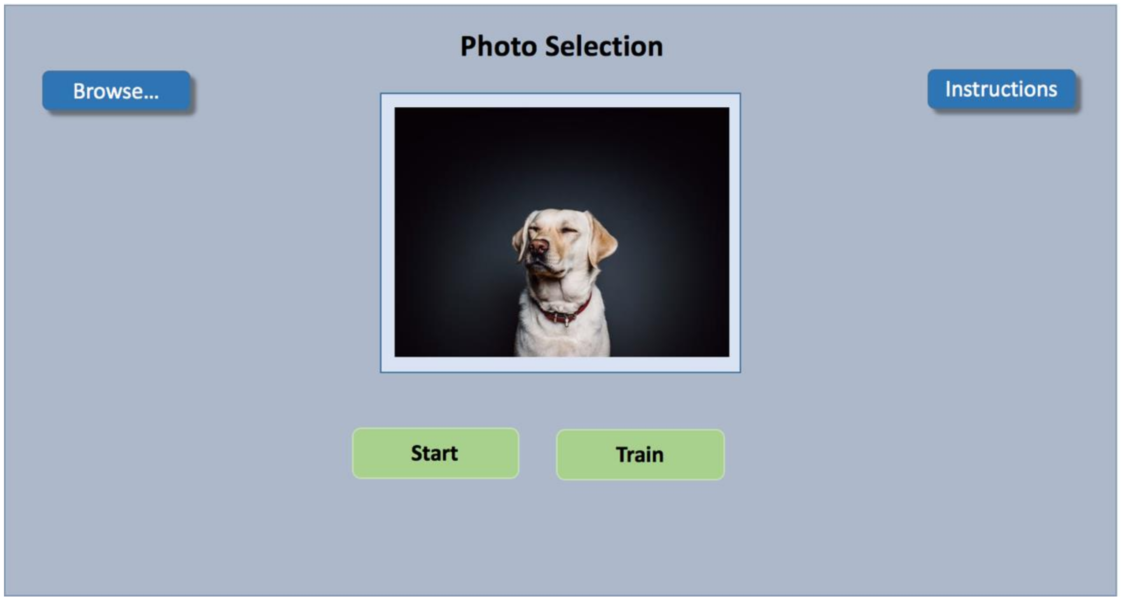
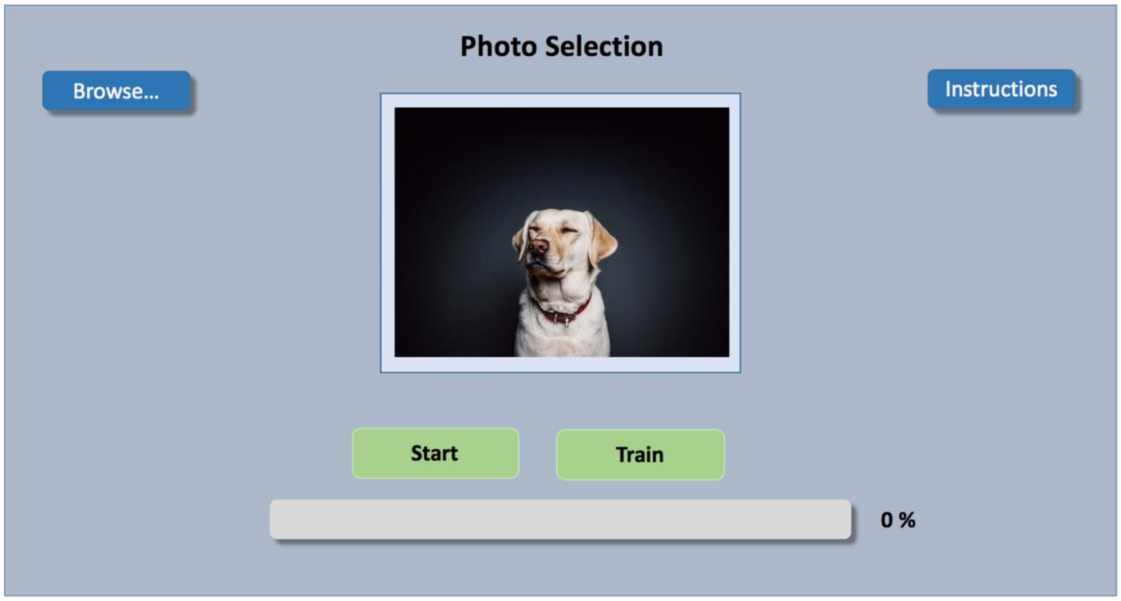


Figure 30: Selected Directory

Now that the directory has been chosen and the “Start” and “Train” button are enabled, the first image of the user’s directory will be displayed. We decided to go with this approach, in order to ensure the user has picked the correct directory. The image being displayed will be the first image to be processed by our system. We would like to implement a system that could display an image to the interface every certain amount of time as a form of progress the user can experience.



Figures 31: Progress bar

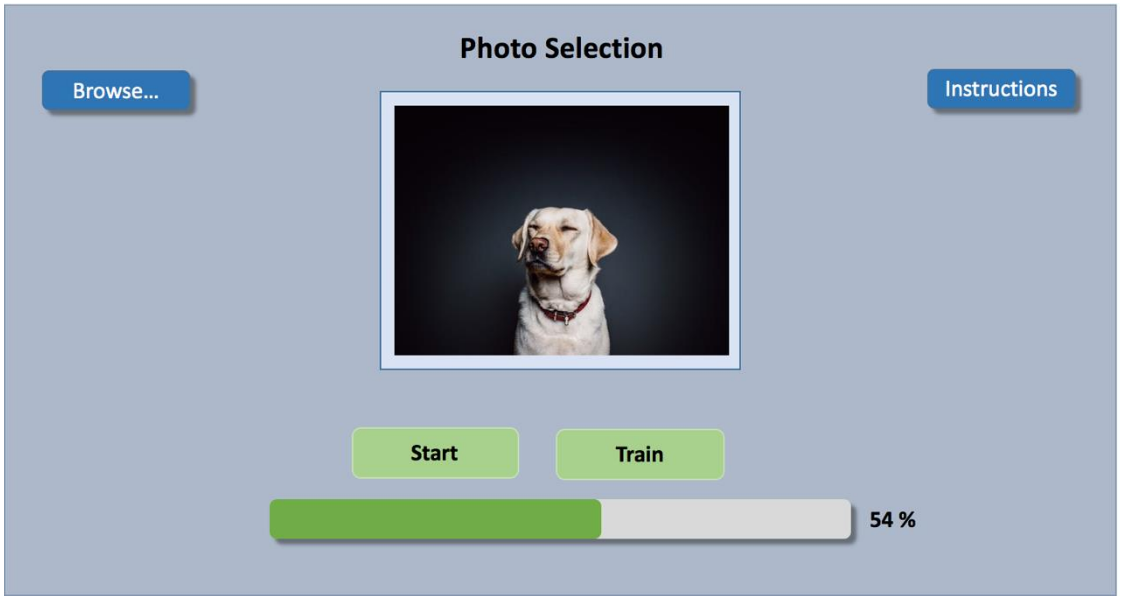


Figure 32: Progress bar

In addition to displaying a different picture every certain amount of time we will have a progress bar that will also give the user progress feedback. Figures 31 and 32 show what the progress bar will look like at the start of the image processing action and how it will look half way through the processing action. The user will not be able to click on the “Train”, “Start”, or “Browse” button during the duration of the image processing time. They may however, click on the “Instructions” button for more clarification on the functionality of the application.

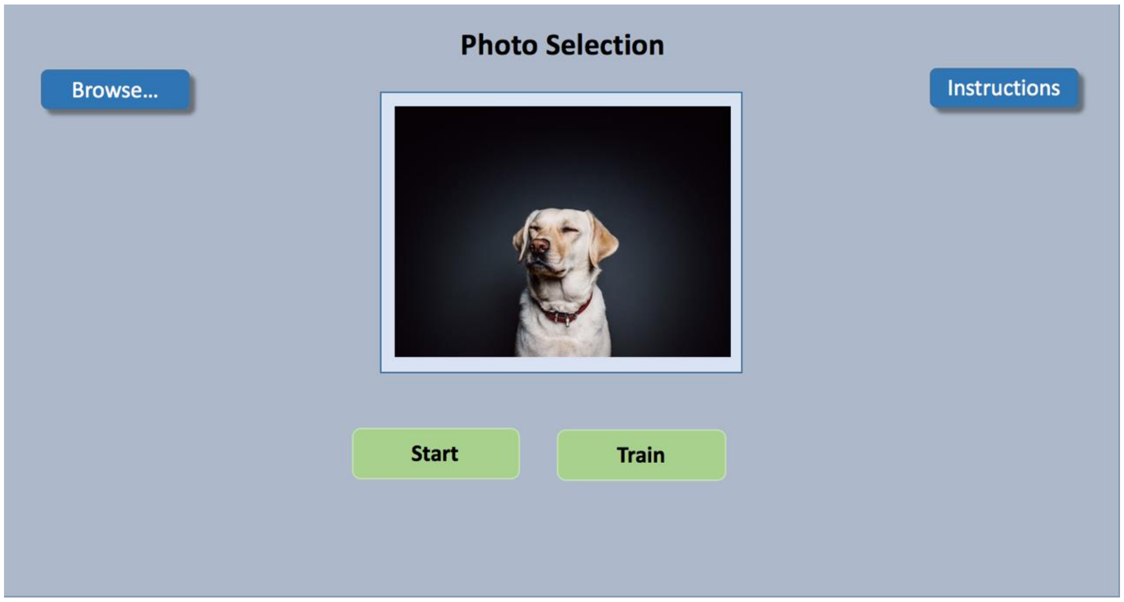


Figure 33: End of Image processing

Once the image processing has finalized the application will return to the main screen and display an image from the selected top twenty percent of the quality pictures. Figure 33 shows the what the final screen will look like. The progress bar will not be displayed at that moment as there will be not other pictures to process for that current directory. At this point the user has a few options as to what they want to do. The user may choose to export the top photos into another directory, the user may choose to close the application, or the user may choose to dive deeper into their image set and train the model based on categories the user deems to bee a quality image. The training section of the application will look at sets of two pictures and have the user choose which one the two pictures is better than the other. The user will be able to do this for as many pictures as they would like. The more pictures are categorized correctly the better out data model will be. Figure 34 shows how the images will be displayed within the application.

During the training portion of the application the user only has the option to choose which picture they found to have more quality, to finish the training process, or to choose the “Instruction” button that will display the instructions for the training piece of the application.

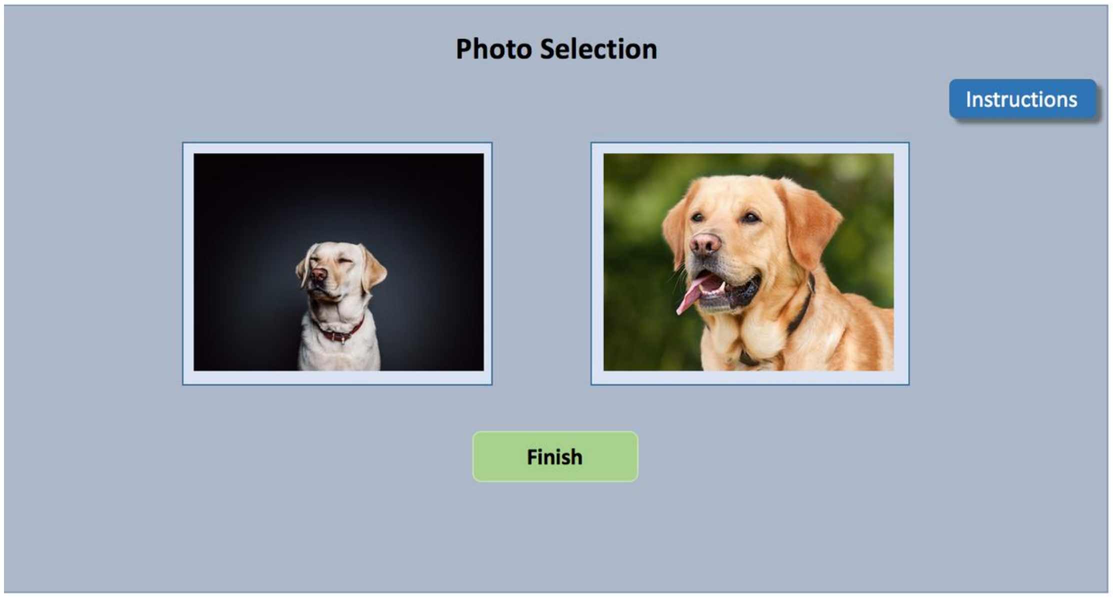


Figure 34: Manual training of data

Testing

In order to ensure that all of the components of the application are working accordingly, test cases have to be created that covers all use cases. In the case of testing the user interface, all of the basic functionalities will be tested extensively, which may include making sure all push buttons display the proper tooltips when the user hovers their mouse over them, the application loads and closes properly, etc. Creating scenarios which cover all circumstances establishes confidence in a working product and provides the user with a high quality product.

In practice, test cases should be written before the implementation of the user interface. The test cases describe the functionalities that the application will perform. These guidelines prevent any deviation to occur during the implementation of the design. If there are any modifications that need to be made to the implementation, new test cases will be created.

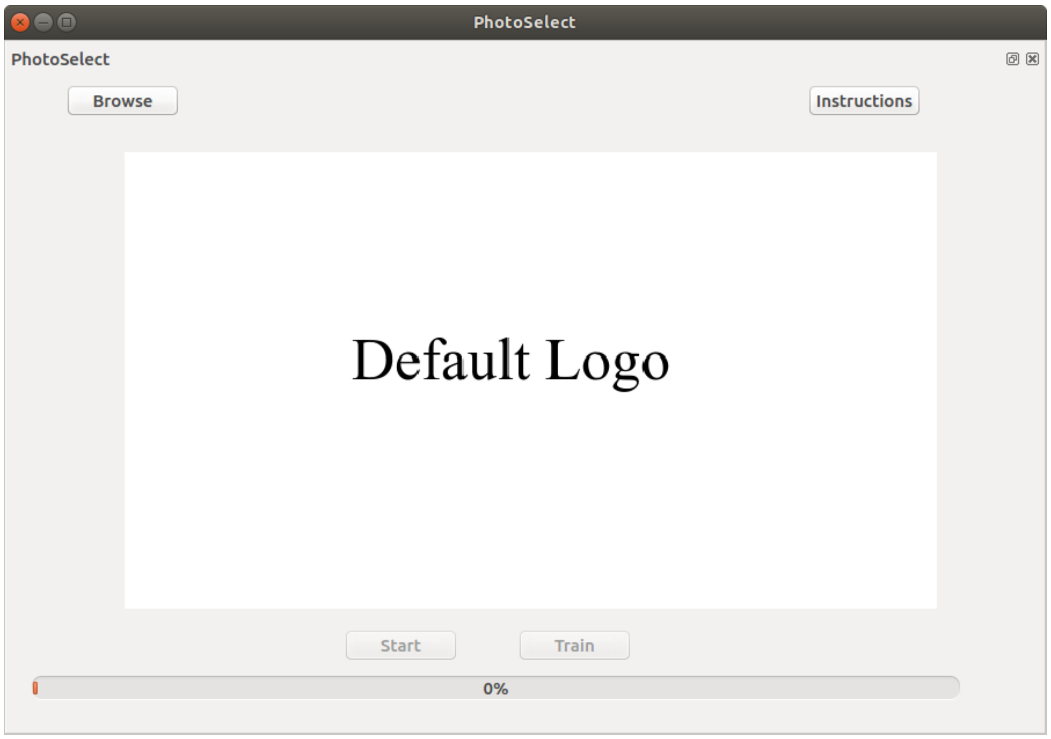
Starting Up / Shutting Down

Starting Up

When the application loads initially, a default logo will be presented to the user in the center of the screen. This is shown when the user has not selected a directory or has chosen an invalid directory.

- PhotoSelect Screen

- “Browse” pushButton will be enabled.
- Center Image will be the Default Logo
- “Instructions” pushButton will be enabled.
- “Start” pushButton will be disabled.
- “Train” pushButton will be disabled
- ProgressBar will be at 0%



Train Screen

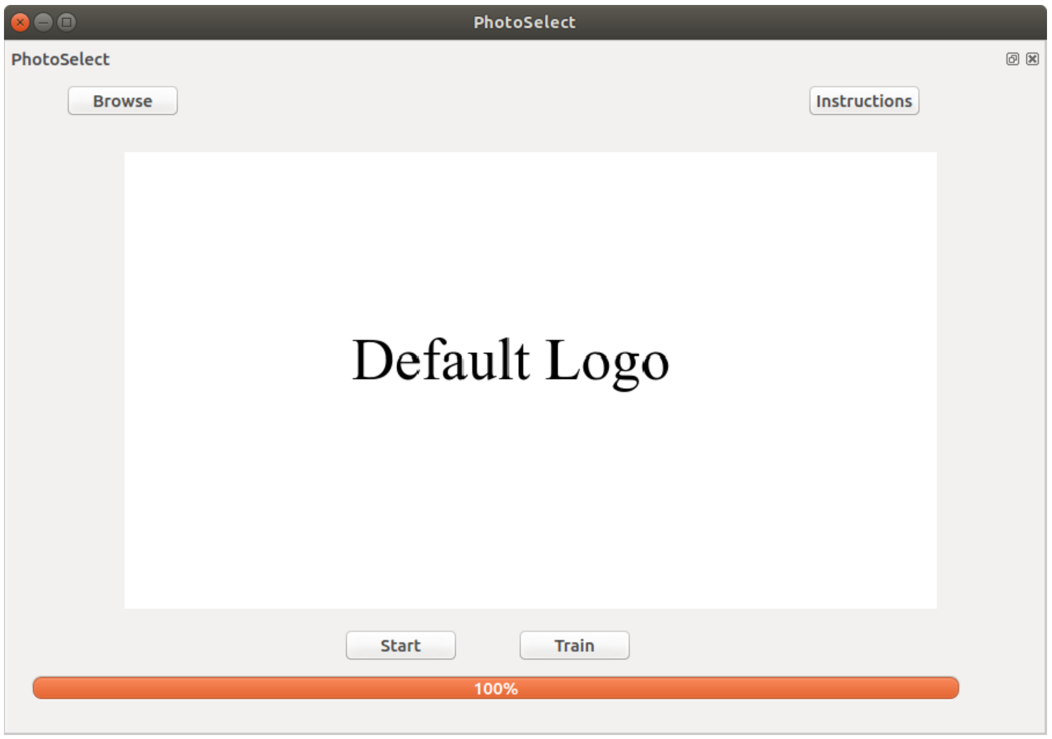
- NA

Shutting Down

Once the user is finished using the application, the software should be in a state ready to be closed safely without corrupting any data. This may include:

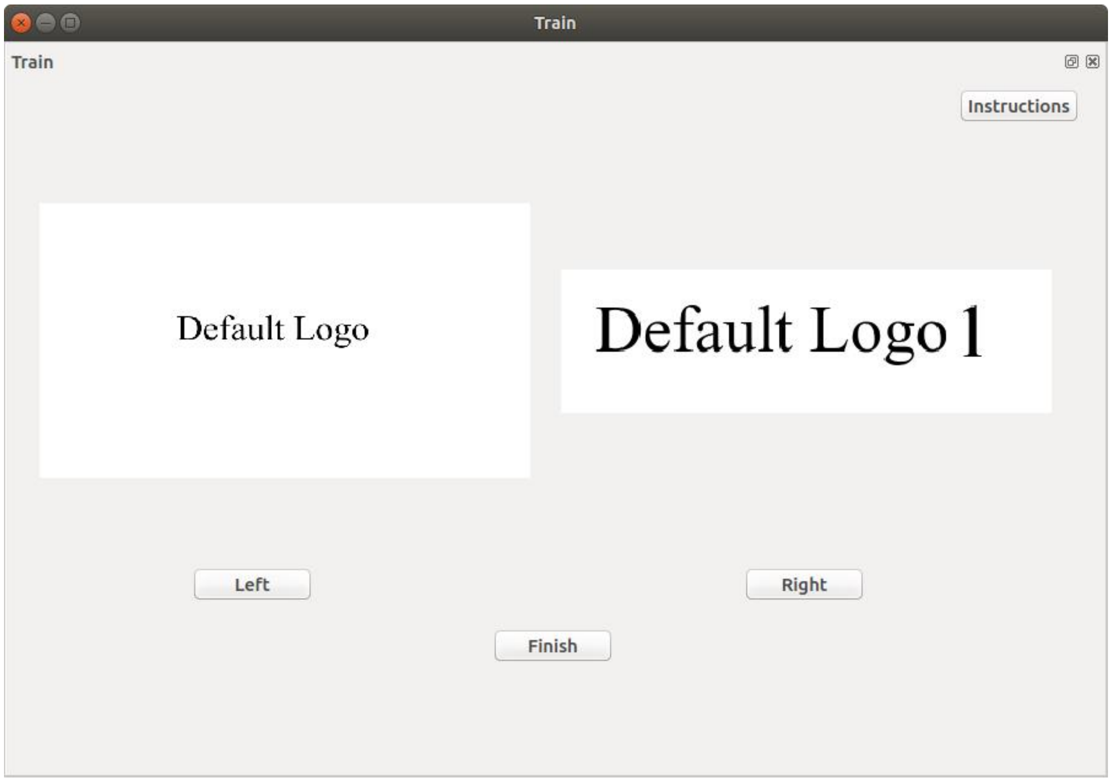
- PhotoSelect Screen

- “Browse” pushButton will be enabled.
- Center Image will consist of one of three options:
  - Default Logo
  - First Image from the directory user chose
  - Last Image from the directory user chose
- “Instructions” pushButton will be enabled.
- “Start” pushButton will be enabled/disabled.
- “Train” pushButton will be enabled/disabled.ProgressBar will be at either 0% or 100%.



Train Screen

- “Left” pushButton will be enabled
- “Right” pushButton will be enabled
- Left Image will be any image from the directory user chose as long as it is not the same as the Right Image.
- Right Image will be any image from the directory user chose as long as it is not the same as the Left Image.
- “Finish” pushButton will be enabled
- “Instructions” pushButton will be enabled



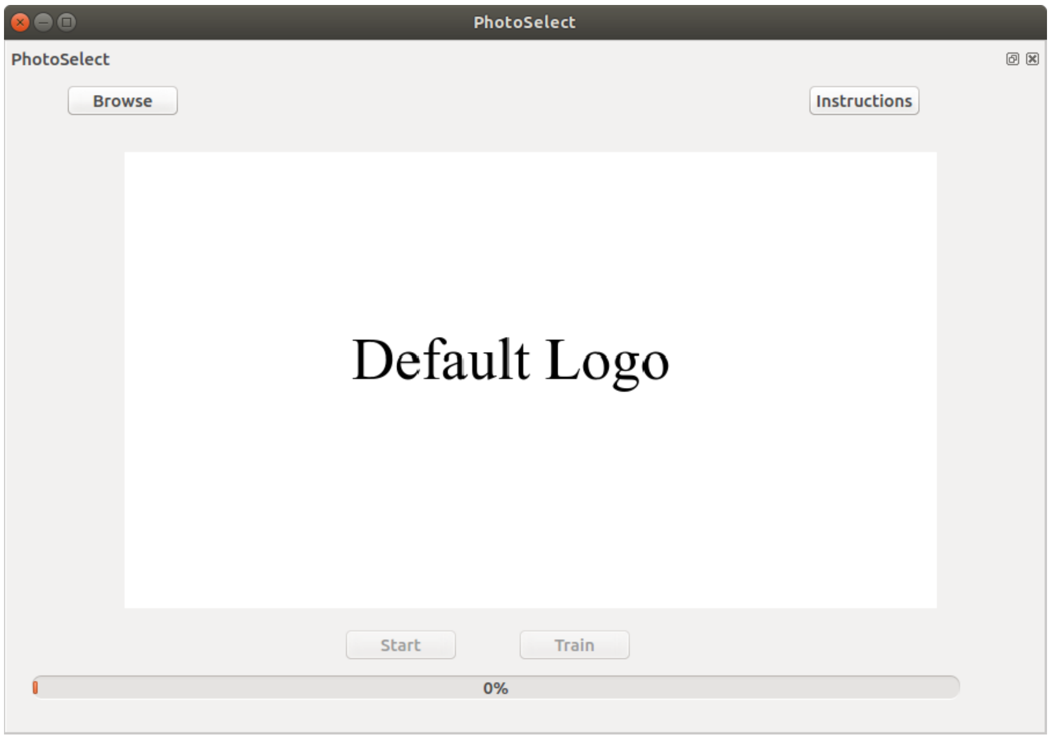
Loading an invalid directory

Once the application has loaded, the user will begin by choosing his directory using the “Browse” pushButton. This will let them navigate through their computer until they find the directory of photos they will be using. If the user selects a valid directory, valid meaning the directory consists of only images, the application will be updated accordingly as shown below. If the user selects an invalid directory, none of the states will be updated and the user will have to select a different directory. The following states should be updated:

- PhotoSelect Screen

- “Browse” pushButton will be enabled.
- Center Image will be the First Image from the directory user chose.
- “Instructions” pushButton will be enabled.
- “Start” pushButton will be disabled.
- “Train” pushButton will be disabled.

- ProgressBar will be at 0%.



- Train Screen

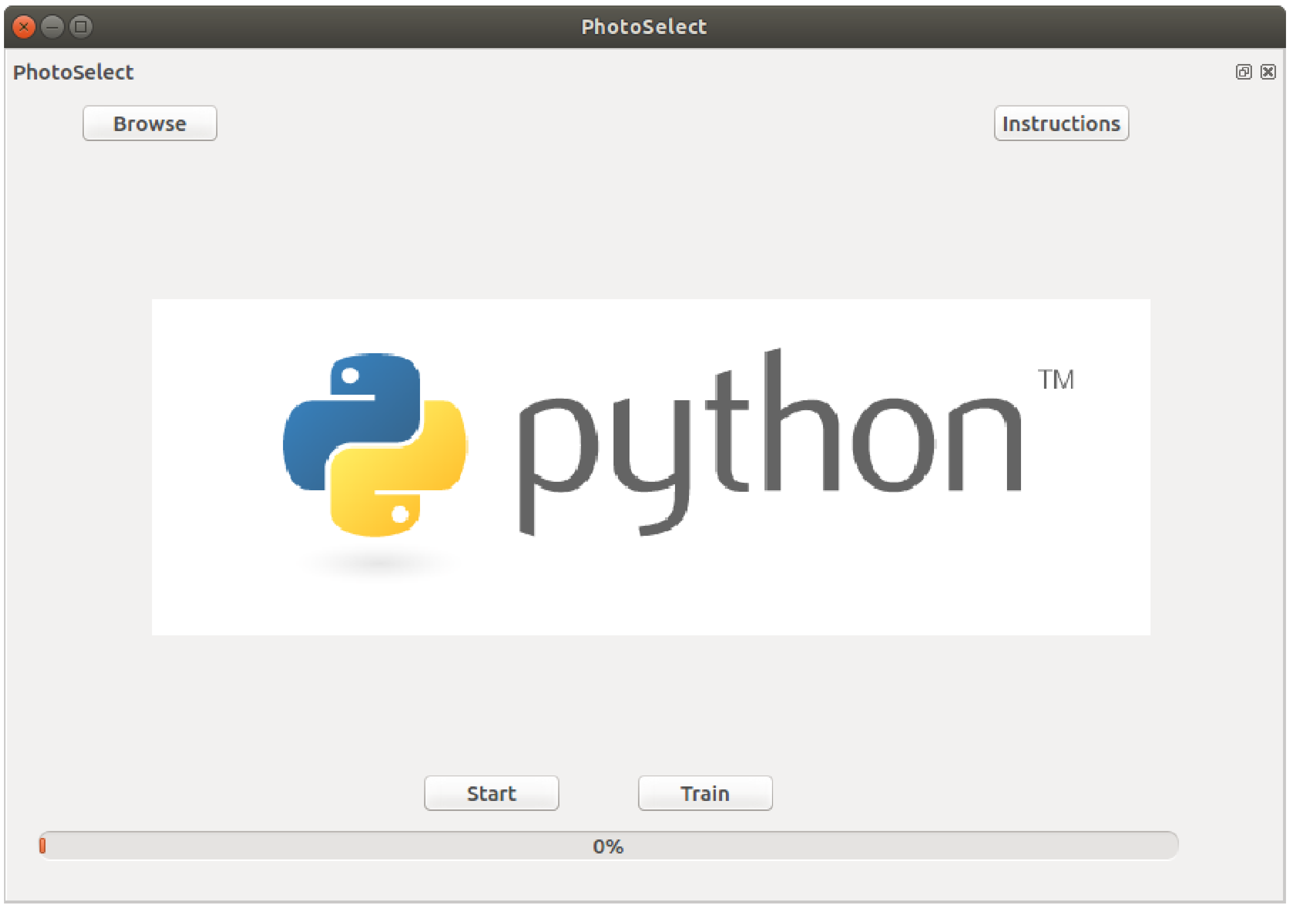
- NA

Training a directory

Once the directory has been loaded, the user will have the option to either begin running the application using a pre-trained algorithm or begin training their own by using the training function implemented in the application.

- PhotoSelect Screen

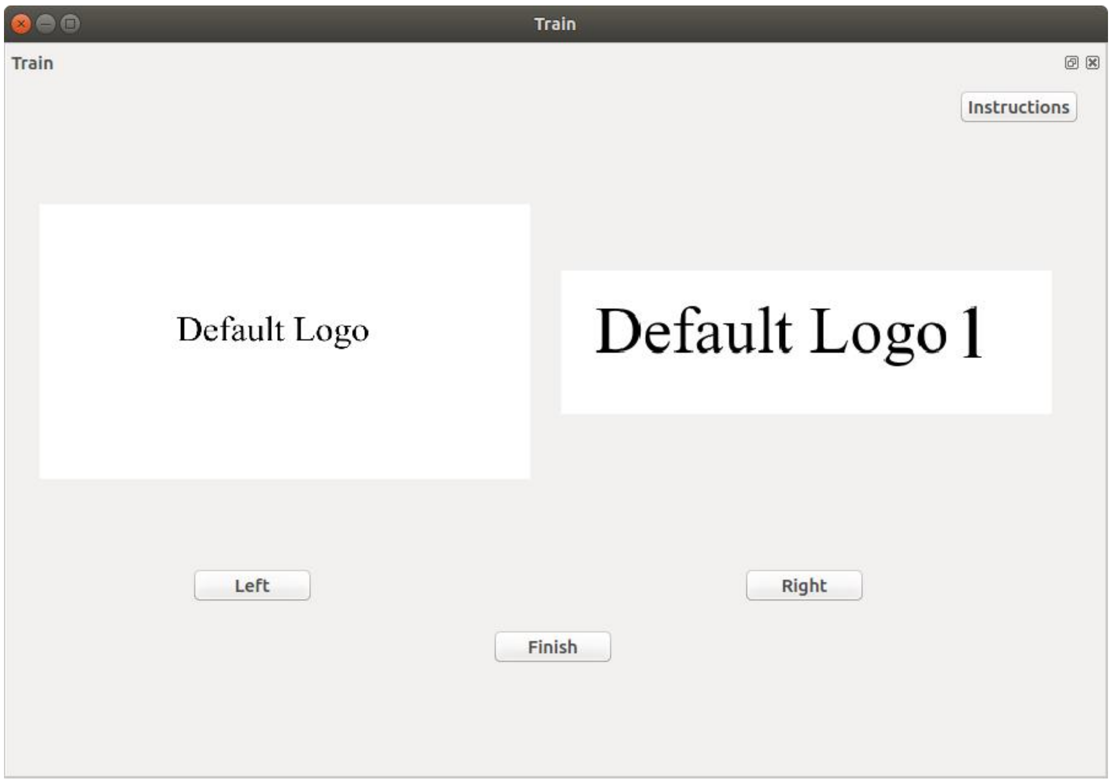
- “Browse” pushButton will be enabled.
- Center Image will be the First Image from the directory user chose.
- “Instructions” pushButton will be enabled.
- “Start” pushButton will be enabled/disabled.
- “Train” pushButton will be enabled/disabled.
- ProgressBar will be at 0%.



The Train Screen will enable the user to personalize their algorithm by training the algorithm to the user's preference. The user will be given two photos and will have to compare to see which is of higher quality. After comparing several photos, the user can choose to return to the PhotoSelect Screen and start running the updated algorithm.

- Train Screen

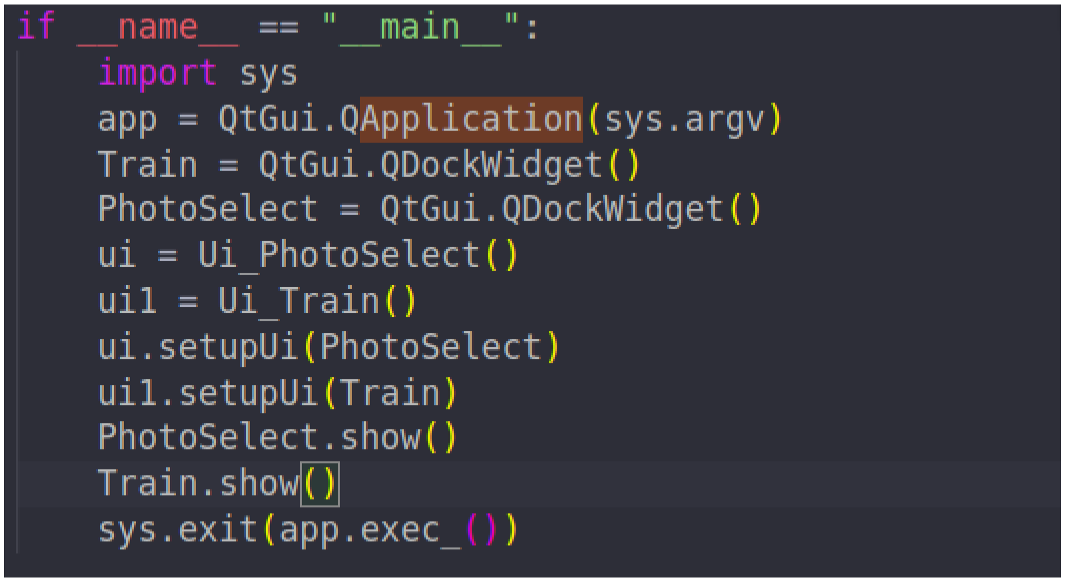
- "Left" pushButton will be enabled
- "Right" pushButton will be enabled
- Left Image will be any image from the directory user chose as long as it is not the same as the Right Image.
- Right Image will be any image from the directory user chose as long as it is not the same as the Left Image.
- "Finish" pushButton will be enabled
- "Instructions" pushButton will be enabled


Train
Train
Instructions
Default Logo
Default Logo 1
Left
Right
Finish

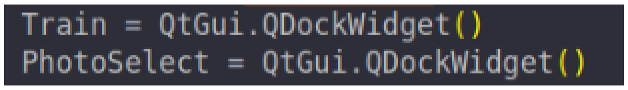
Functionality

When deciding on the framework for the Slekti application, one of our main focuses was on the functionality of the application. Since we are designing an application that has the potential to process multitudinous images at any given time, we wanted to avoid any unnecessary features that would bog down the application. However, we do understand that user experience also includes a visually-pleasing interface as well efficiency. As the application runs, we will have images that will update the interface as well as a progress bar that tracks how many photos are left to be processed. Typically, users enjoy seeing progress in an application. Including small changes such as these makes a huge difference in the user’s experience.

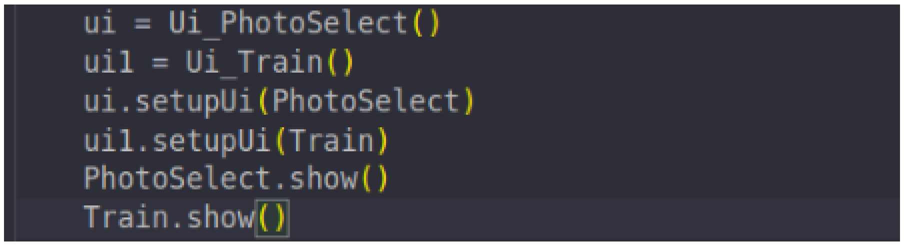
We decided to go with PyQt4 as the framework due to its efficiency and reliability. PyQt4 has been around for more than 10 years and has been supported by many other applications. It is no surprise that we have found plenty of documentation to help us accomplish the tasks necessary. We are not using the newest version, PyQt5, because it contains issues which would be time-consuming to resolve, such as integrating with PyInstaller. We decided to use PyInstaller in order to bundle the application into a stand-alone executable. Users would not have to go through and download all the dependencies needed to make and run the Python file. Instead, they can just run the executable like any other application.


```
if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    Train = QtGui.QDockWidget()
    PhotoSelect = QtGui.QDockWidget()
    ui = Ui_PhotoSelect()
    ui1 = Ui_Train()
    ui.setupUi(PhotoSelect)
    ui1.setupUi(Train)
    PhotoSelect.show()
    Train.show()
    sys.exit(app.exec_())
```

When the application begins, an application object has to be created. This goes for every single PyQt4 application. The application object is located in the QtGui module. The parameter “sys.argv” is a list of arguments from the terminal.


```
Train = QtGui.QDockWidget()
PhotoSelect = QtGui.QDockWidget()
```

The QtGui.QDockWidget class is a type of widget. Widgets are the base class of user interface objects. They receive events from users such as mouse or keyboard, or from different sources such as the window system, and provides output onto the screen. In this case, two new QtGui.QDockWidget classes are created for Train and PhotoSelect. These will be the two main windows of the application.


```
    ui = Ui_PhotoSelect()
    ui1 = Ui_Train()
    ui.setupUi(PhotoSelect)
    ui1.setupUi(Train)
    PhotoSelect.show()
    Train.show()
```

The following two lines are the names of the toplevel objects from Qt 4 Designer. Since there were two windows needed for this application, two classes had to be created. As previously mentioned, Qt 4 Designer prepends the name each of the classes with "Ui_". The classes are then both stored in temp variables and displayed at the same time. Both classes have a pre-made method from Qt 4 Designer which passes in the Widget in which the user interface is created. Since we were only concerned with having a bare bone design for this portion of the project, we decided to show both windows upon starting the application. Later on, complete functionality will be add and the Train pushButton will be able to seamlessly open the Train window while closing the PhotoSelect window.

Methods for PhotoSelect class

- **setupUi(self, PhotoSelect):**
  - The method passes in the PhotoSelect Window and creates all the Widgets that will be seen in the Window. This will include all the QPushbuttons such as the Train and Start as well as the QprogressBar and QLabel which displays the image.
- **retranslateUi(self, PhotoSelect):**
  - The method passes in the PhotoSelect Window and sets the text and title of the Widgets.
- **train_Button_clicked(self):**
  - The method will be called whenever the Train QPushbutton is clicked. The current PhotoSelect Window will close and the Train Window will open.
- **start_Button_clicked(self):**
  - The method will be called whenever the Start QPushbutton is clicked. The learning algorithm will begin processing through each photo and create a new directory for the chosen images. As each photo is being processed, the QProgressbar is keeping a percentage of how many photos are left to be processed. As an optional functionality, the image being displayed in the center will be updated to show which photos are being selected.
- **instructions_Button_clicked(self):**

    - The method will be called whenever the Instructions QPushbutton is clicked. A new Widget will appear which will show instructions on how to begin using the application. The Widget will display the instructions in list formatting.
- **browse_Button_clicked(self):**
    - The method will be called whenever the Browse QPushbutton is clicked. The user will select a directory which contains only images that will be processed. The method will be updated to support the following image types: TIFF, JPEG, GIF, PNG.

Methods for Train class

- **setupUi(self, Train):**
    - The method passes in the Train Window and creates all the Widgets that will be seen in the Window. This will include all the QPushbuttons such as the Left and Right as well as the QLabels which will display the images.
- **retranslateUi(self, Train):**
    - The method passes in the Train Window and sets the text and title of the Widgets.
- **left_or_right_Button_clicked(self, left_Button, right_Button):**
    - The method will be called whenever either the Left or Right QPushbutton are pressed. Depending on the button pressed, the algorithm will be updated to the user's preference and two new images will be shown in the Train Window for the user to choose. For example, if the user selected the left button, left_UserImages will be passed to the algorithm as the preferred image.
- **finish_Button_clicked(self):**
    - The method will be called whenever the Finish QPushbutton is clicked. Similar to the train_Button_clicked(self) method, the

finish_Button_clicked will close the current Train Window and open the PhotoSelect Window.

- **instructions_Button_clicked(self):**
    - The method will be called whenever the Instructions QPushbutton is clicked. Similar to the instruction_Button_clicked(self) in the PhotoSelect class, this method will open a new Widget which with instructions on how to operate the applications. The instructions will focus primarily on how to use the Train portion of the software. The Widget will display the instructions in list formatting.

Design

Summary

A challenge (before stumbling upon modern methods) was determining which features to specify and similarly, how many we could specify. The more features we expect to be evaluated, the more computation must be done and, therefore, the speed at which we can rank images decreases. However, if too few features are analyzed, an accurate model would most likely not come to fruition. Thus, selecting features with the most influence in determining a high-quality image was the next step.

Before we moved on to the next step, we found PAC-NET and realized we didn't need to manually select any features. Not only that, the architecture of PAC-NET lends itself to an easy way for us to update the model based on the user's feedback. More on PAC-NET is discussed above.

Having an easy-to-use graphical user interface (GUI) was one of the more important essentials our sponsor had, which is why simplicity was one of the core themes we kept in mind while we designed the interface. Similarly, we wanted to follow the best practices for user interface design to maximize every user's experience. One way we achieved this was to make sure the user wasn't staring at a static page while the algorithm was ranking the images. To reassure the user that progress was being made, we designed the interface to display not just a loading bar, but also which image was being ranked.

One of the risks we anticipate is that our algorithm will take too long to rank all the user’s pictures. We plan on experimenting with different deep learning frameworks as well as different orientations of the network layers to see if we can optimize the runtime. Another risk stems from the heavy computational load required to train an accurate model. We expect training times to last anywhere from hours to days thus limiting the number of experiments we will be able to hold.

Data Sets

The first half of our algorithm deals with image quality assessment. Here we plan to use a supervised learning approach to develop a model that can predict the subjective quality of an arbitrary photo. As with all supervised learning implementations, a labeled training set is required. For our purposes, we need photos that range from low to high quality. We could manually acquire photos by either taking them ourselves, asking our friends and family for their photos or by scouring the internet. If we were to take any of these options, it would also be necessary to manually score each photo’s quality.

Fortunately, image quality assessment solutions have been sought after by researchers for more than 20 years. This has resulted in the creation of several data sets comprised of hundreds to hundreds of thousands of images with corresponding quality scores. The largest data set is known as the Aesthetic Visual Analysis (AVA) data set. It has more than 255,000 images and each image has several aesthetic ratings on a scale of 1-10 [20]. For example, an image that was rated by 20 people may have 5 counts of the rating 10, 3 counts of the rating 9, and 12 counts of the rating 8. We consider this data set an invaluable resource due to the number of images it provides as well as the number of ratings per image.

Figure 26: An image from the AVA dataset. It was rated by 162 individuals for a mean score of 7.3/10.0.

While there are other image quality datasets, they do not compare with the AVA dataset in size or number of ratings per image. The Hidden Beauty (HB) of Flickr Photos dataset sports 15,000 photos with 5 ratings per photo, on a scale of 1-5 [21]. The Kodak Aesthetics Database has more than 1500 photos and each photo was rated by 4 people on a scale of 1-100 [22]. Another data set came from researchers at the Chinese University of Hong Kong. It is called the CUHK-PQ data set and has nearly 18,000 photos [23].

We expect (and hope) our model reaches a quality-assessment accuracy of at least 80% on the AVA dataset. The researchers who created the model we are using achieved an accuracy of 82% [17]. If we are not satisfied with the performance of our model by training it on the AVA dataset, we may train our model on the other datasets.

Selecting Images

There are two criteria that determine how the assistant will select photos out of a user's album:

1. The estimated aesthetic quality of the image.
2. The probability the user will find the image desirable.

Before the user has the chance to train the assistant, they will have the option to run a general quality assessment algorithm on their photos. Currently, we are unsure how much feedback the user needs to give until the assistant can accurately predict the probability of an image pleasing the user. This brings us to another risk; we may construct a model that will eventually learn the user's tastes, but the user would have to provide vast amounts of feedback, all the while wondering when or if the program will even work.

Depending on the contents or theme of an album, the user may have different preferences when it comes to selecting photos. For example, a user may be more likely to select photos where faces are visible when looking through an album of a family outing. On the other hand, for an album depicting a day spent at a car show, in-focus, well-placed car-photos might be valued more than photos with faces in them.

We are considering giving the user the option to select one or more filters to help the selection process. These filters could include features of images such as the Rule of Thirds, symmetry, or the absence of a face. It is also possible to classify the objects in an image and allow the user to specify which images they are looking for by selecting a tag. This feature will increase the time needed for processing but it would greatly assist the assistant's ability to choose images the user likes.

Learning from the User

To clarify, "user feedback" does not pertain to the user's opinion on our application; it describes the information collected from the user that will be used to train the assistant. We have different approaches to how this information will be collected.

When we implement the PAC-NET solution we present the user with two random images from their photo album and have them decide on which one they like more. From here, the feedback could either be dependent or independent:

A. Dependent: user feedback on image pairs is transferable to other images, i.e. if the user likes image X more than image Y, and likes image Y more than image Z, then we will not present the image pair (X, Z) and assume the user likes image X more than image Z.
B. Independent: No assumptions will be made with regard to which images the user prefers - only the user will be able to pass judgment on image pairs.

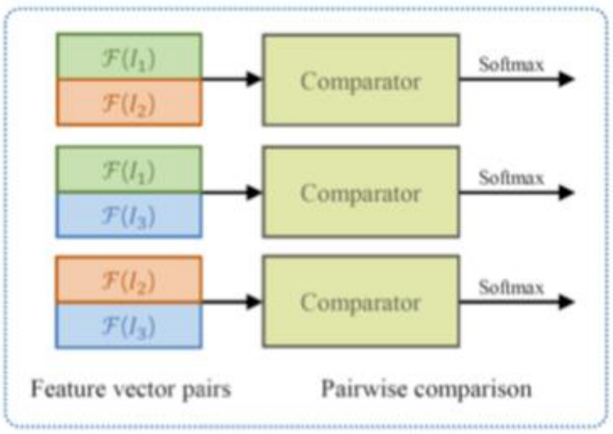
The end goal is to eliminate the need for the user to give feedback on an album, but when the user first installs the application, the assistant will have no inkling as to what kind of photos the user likes. Therefore, requiring the user to spend time training the assistant is desirable. The more images the user gives feedback on, the more accurate the assistant will become; however, we do not expect users to go through *all* their images for every album. We are considering a feature that allows users to specify how many images out of their album they are willing to give feedback on. We can suggest an amount, such as 30% of their album, or they can give feedback on the whole album.

The schema where we assume the user's feedback is dependent on their previous feedback is advantageous because, for the same number of image pairs we present to the user, we gain more information about which images the user prefers than if we follow the schema where we assume the user's feedback is independent of previous feedback. The drawback lies with the fact that users' opinions on images vary greatly. It might not be the case that a user's disposition adheres to the transitive property (if X > Y, and Y > Z, then X > Z). For example, suppose a user is presented with images of a cat, dog, and lion. The user likes the image of a cat more than the dog, and the dog more than the lion. However, the image of the lion is favored more than the cat. This may happen when users compare different aspects of each photo against each other.

Feature Extraction and Comparison

As discussed in the section about convolutional neural networks, the ability of convolutional neural networks to detect abstract and complex features makes them very useful for image recognition problems. In our case, we will be utilizing a pretrained network, during the training process, in order to extract features of the image. We will be utilizing a pretrained GoogLeNet in order to do this. After we input an image into the network, it will extract the aesthetic features and output a feature vector. These feature vectors represent the quality of the image. After we extract the aesthetic features, we will then perform feature comparison by inputting the feature vector into a comparator.

We will now compare the features in pairs. Similar to the PacNet, we will train identical comparators that will contain two fully-connected layers. The last fully-connected layer will output a 2-dimensional rank vector. This vector will then be normalized in the softmax layer. The figure below represents how we will execute this process. $F(I_1)$, $F(I_2)$, and $F(I_3)$ represent the feature vectors. Those feature vectors are inputted as pairs into the comparator and then passed through the softmax layer.



Representation of PacNet's pairwise feature comparison

Loss Function

Machine learning algorithms learn by means of a loss function. This is a way of assessing how well an algorithm represents the data. The equation below represents the loss function we will be utilizing in our design.

$$Rank_{loss}(I_1, I_2) = -L \cdot log(P) - (1 - L) \cdot log(1 - P),$$

This loss function asymptotes to a linear function. This makes it robust to noise when compared to a quadratic function. It also handles input pairs with similar ground-truth attribute strengths in a principled manner as it becomes symmetric with minimum value at 0 when L = 0.5 [24].

Possible Reinforcement Learning Implementation

To implement a reinforcement learning solution, we must first formalize our problem as Markov Decision Process (MDP). To recap, elements of an MDP include a set of states, a set of actions, a set of state transition probabilities, a discount factor, and a reward function. There may be many ways we can choose what each of these elements represents.

We want to reward the agent when it selects photos the user would select and punish (or abstain from rewarding) the agent when it selects a photo the user dislikes. Here, the agent represents the “trusted assistant.” The environment is responsible for presenting the agent with images to assess. See Figure X for an overview of the relationships in a reinforcement learning problem.

The agent will begin in an initial state that does not have a value assigned to it. From here, it wants to move to states that have high rewards, but what do these states look like? The state with a positive reward in this problem is the state where the agent chose a photo the user approves of. Likewise, the state with a negative reward (or a neutral reward depending on implementation) is the state where the agent chose incorrectly. Now we have defined the set of states.

The set of actions is comprised of only two actions; of the two presented photos, select one or the other. This will take the agent to one of the two states and the process will repeat from there. As this approach stands, there is not a high level of interactivity between the agent and the environment. Actions taken by the agent do not influence the environment, i.e. which photos will be presented next. The only way for any *learning* to happen is for the agent to reflect on which actions lead to rewarding states and for which images those actions were taken for. Learning what the user likes will help the agent accurately assign the state transition probabilities. It boils down to answering the question of “How likely is it for the user to select this photo over the other?”

While it is customary for the discount factor to increase as the agent progresses through states, we think a constant discount factor is necessary because each iteration of the agent being presented with photos and the subsequent transition to a positive/negative state is equal in importance. The sum of rewards collected in these iterations will reflect the effectiveness of the agent. More recent states have equal value as the initial states.

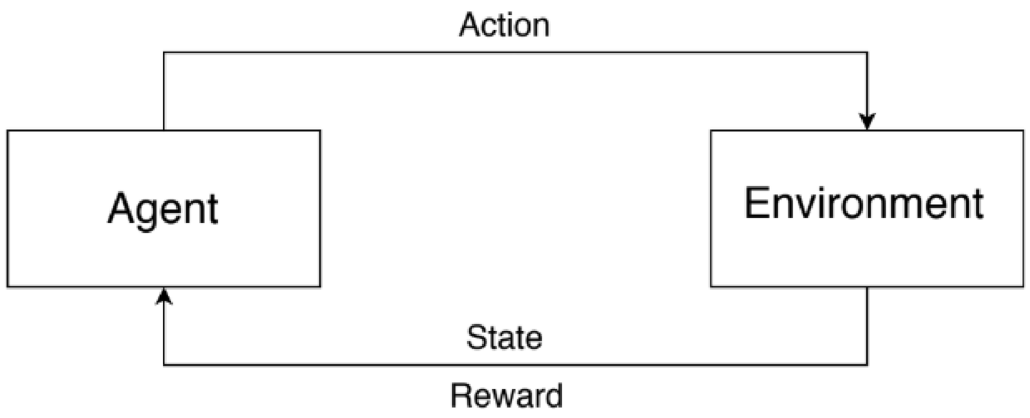
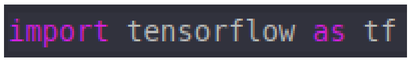


Figure 27: Every action an agent takes in an environment results in a reward and new state

Frameworks

TensorFlow

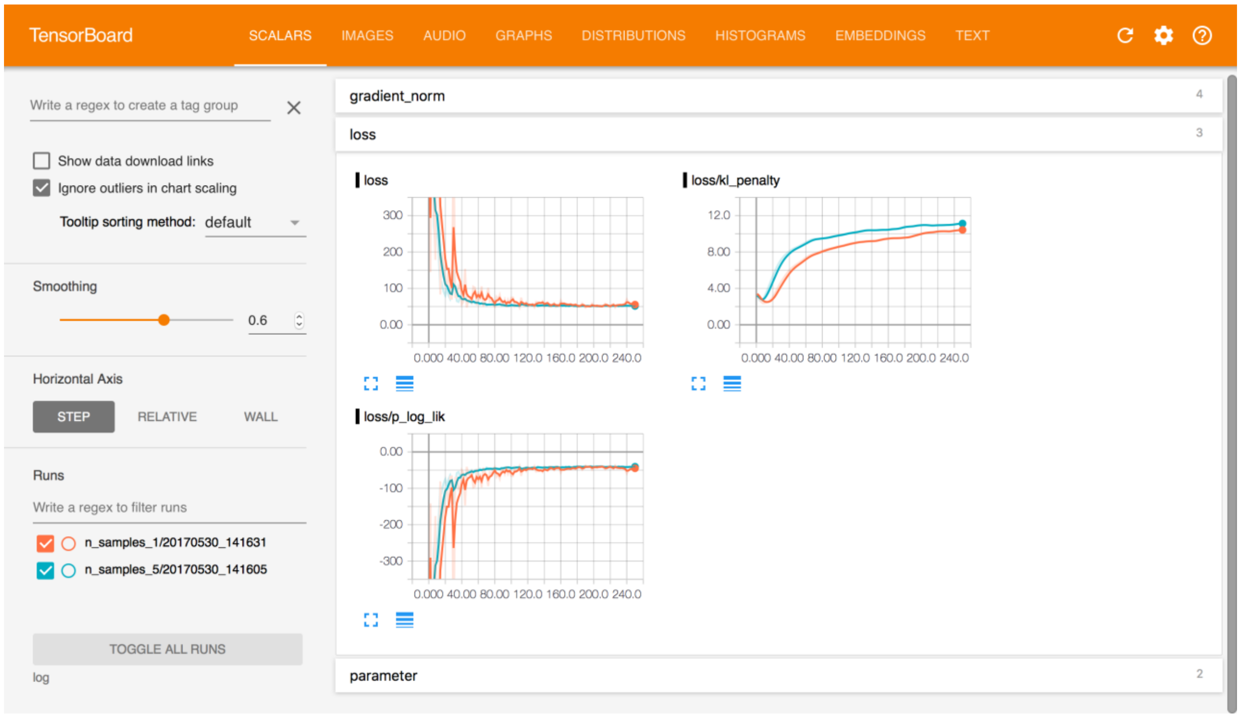
Created by Google Brain Team in order to further deep learning research, TensorFlow is an open-source library that utilizes machine learning in order to accomplish complex numerical computation. Most TensorFlow development is done through Python. Consider how quickly and easily you look at a picture and immediately know what you are looking at. While this process may seem simple enough for anyone to do, this task is incredibly arduous for a machine to perform. TensorFlow delves deep into the field to develop image processing technology for modern-day applications such as facial recognition. The tool provides various APIs that can be supported on platforms such as Windows, MacOS, Linux, and Android. As long as the computation can be represented through a data flow graph, TensorFlow will be able to utilize the information. Most of the information TensorFlow's library works with are called tensors. These are arrays of n-dimensions, where n represents the data flow graph edge, or rank. Even though tensors can be seen as arrays, there are not treated the same as matrices in mathematics. Tensors can be seen as an entity defined by both its magnitude and number of directions. For example, there could exist a tensor represented with an array of 2 numbers in a 3 dimensional space. The 2 in this situation represents the rank of the tensor.



```
import tensorflow as tf
```

In order to begin using TensorFlow, the user would include the import statement somewhere in the beginning of their Python file. The import statement gives the user access to all of the library's symbols, methods, and classes. TensorFlow is available on many popular platforms such as Windows, Linux, and MacOS.

TensorBoard



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
☑ n_samples_1/20170530_141631
☑ n_samples_5/20170530_141605

Figure Above: TensorBoard application displaying loss

TensorFlow provides the capability to work with computationally expensive neural networks that would take even longer to analyze. Though it is possible to go through and take time to analyze the output shown on the terminal without any sort of tool to assist in the process, this process may take hours or even days. Luckily, TensorFlow has developed a tool to assist in interpreting those large data sets and displaying the information in the form of graphs.

TensorBoard makes all the computations that TensorFlow can do, such as training a deep neural network, into visually-pleasing graphs to better understand the output of the user’s program. TensorBoard can plot quantitative metrics about the execution of a graph, and provide information about the data processed such as the images that go through it. The user can determine if the layers they setup were connected to their respective regions correctly, or whether the right tensors are being displayed, and if so, what form they are taking.

Keras

Keras is a high-level neural networks API. It is written in Python and can have TensorFlow, CNTK, or Theano run in the background. Keras makes implementation fast and easy. Keras supports both convolutional and recurrent neural networks. Keras was designed in order to enable fast experimentation, be user-friendly, modular, and extensible. The Keras libraries were utilized in the example programs in sections 2 and 3.

PyTorch

TensorFlow is not the only library used for machine learning - PyTorch stands as a worthy opponent to Google’s brainchild. It is described as “an optimized tensor library for deep learning using GPUs and CPUs.” [18] Some features of PyTorch include a C++ front-end for those who seek an even finer level of granularity, a hybrid front-end that provides a clear interface to both eager and graph modes, a Python-first construction of the library that ensures compatibility with popular existing Python libraries, and an extensive toolset created by researchers that can be applied to images, text, and even reinforcement learning.

Compared to TensorFlow, PyTorch is easier for beginners to get started with because the relationship between classes is more intuitive than TensorFlow. Obviously, both libraries require knowledge of machine learning concepts, but since PyTorch was built specifically for Python, it lends itself to quicker understandings. Furthermore, PyTorch provides dynamic graphs that can update as parameters to the model change. On the other hand, TensorFlow only supports static graphs. These require a predefined model and do not update in real time [19].

A disadvantage of PyTorch is its small community. Though released only a year after TensorFlow, there are far less PyTorch users and tutorials. A discount to this disadvantage, as mentioned above, is PyTorch's gentle learning curve. The most important drawback of PyTorch is its inferior ability to scale. The library is useful for experimentation and small projects, but it cannot keep up with TensorFlow's support for production-level machinery.

OpenCV

The Open Source Computer Vision library, also known as OpenCV, is a library of functions which focus primarily on computer vision. Like many other tools being used for the application, this library is cross-platform and free to use. This means OpenCV can be applied to any operating system such as MacOs, Linux, or Windows. OpenCV also supports deep learning frameworks such as TensorFlow. OpenCV contains more than 2400 algorithms which focus on efficiency and simple implementation. The algorithms include tracking moving objects, automated cropping, classifying and recognizing all or parts of an object or scenery, etc. Companies such as Yahoo, Microsoft, Google, and many others use the library extensively in their large scale projects due to how functional it is.

Intel Research launched OpenCV in 1999 to further CPU-intensive applications. This included attempting to 3D display walls and real-time ray tracing. Intel's located in Russia heavily contributed to the project. They focused on advancing currently existing vision research, creating a common foundation for developers to create from, and making optimized code available for free. The library is now supported by Itseez and Willow Garage. The library is fairly easy to interpret and implement. This is due to how well the source-code is documented along with the hand-tuned assembly language binaries.

Recently, most of the updates OpenCV has released have been focused on C++. These changes allow for more flexibility in changing interface, creating type-safe patterns, and more efficient functions. These releases come out around every six months and are done mostly by an independent Russian team. Although most of OpenCV is written in C++, there are Python bindings that can be used instead to accomplish the same tasks. There are also bindings for Java, JavaScript, and MATLAB/OCTAVE. The documentation for all these bindings are easily accessible online, and are still currently being maintained and improved upon to support a bigger crowd of developers.

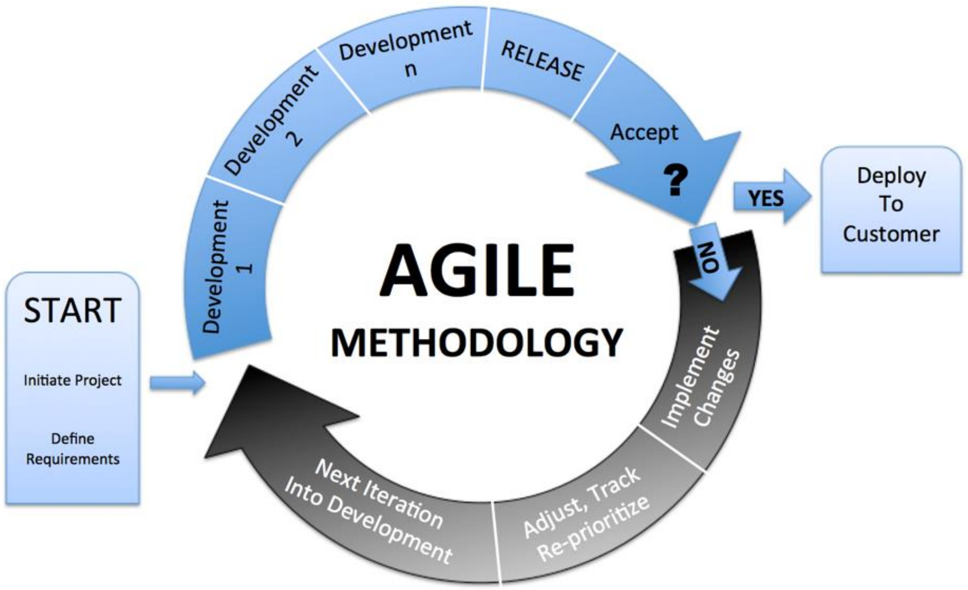
Managerial Topics

Agile

Since our project will require a large amount of development time and has the possibility of changing over time, we have decided to implement the Agile methodology. The Agile methodology is an approach to software development that allows for projects and teams to self organize and adapt to changing requirements. We believe that our project is large enough and complex enough for different members of the team to take on different portions of the project and meet the final requirement. Another reason we would like to take on the Agile method is because it would allow us to use a project management tool that is an industry standard. Throughout the semester of senior design one we have been doing most of the research and gathering the majority, if not all of the product requirements. This phase would be the analysis and product requirement phase of our project. The next step will be the first development phase, followed be a second and possibly a third development phase.

Sprints

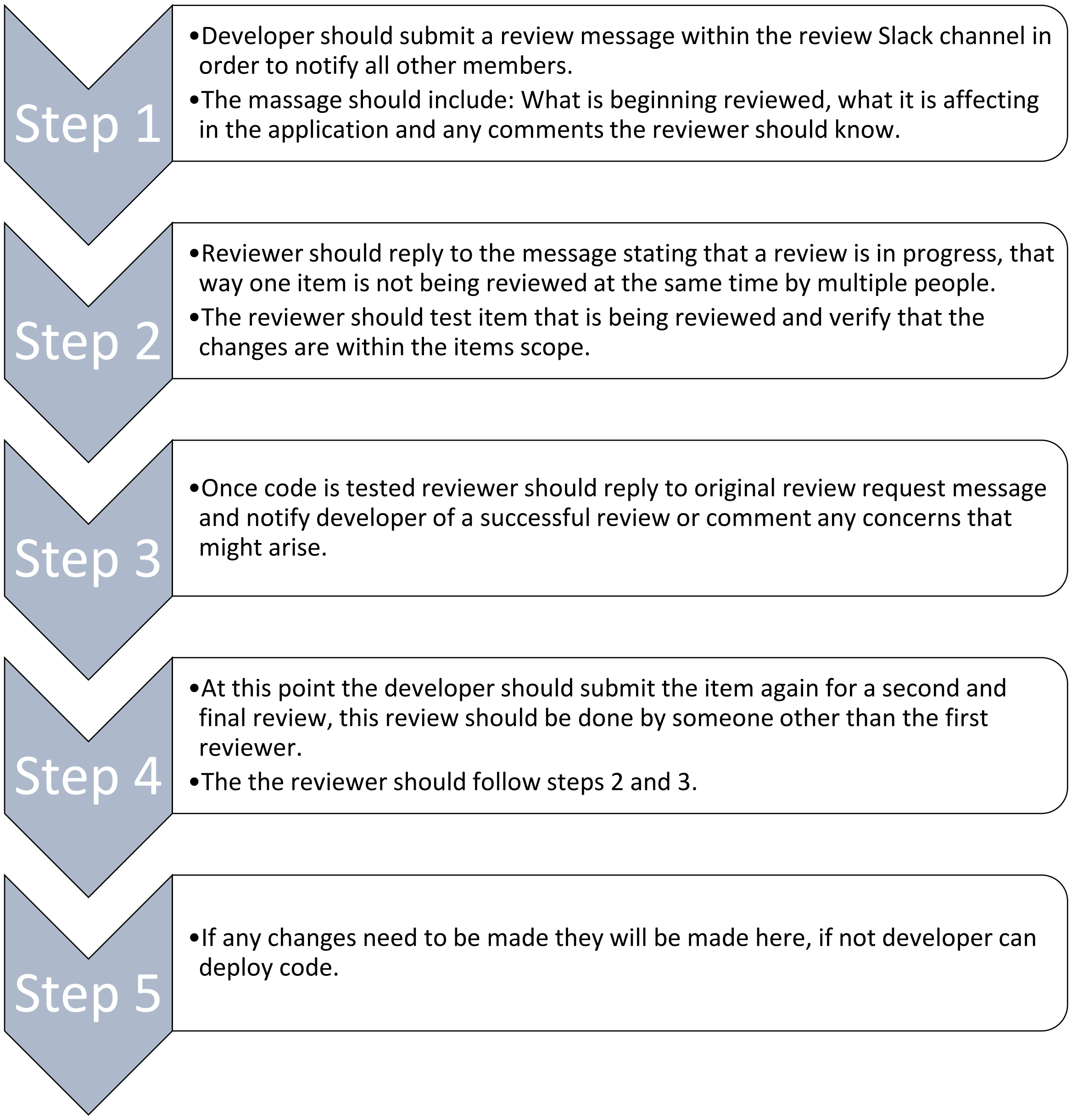
We plan to use sprints within the development phases of our project to better keep track of our work and give better visibility of what work is being accomplished at what point of our project time line. As of the time of this document we are deciding to create four-week development sprints with enough work in each item that will keep all members of the team occupied but also allow for collaboration among team members if one item is finished before the sprint ends. In addition to having sprints we are potentially going to switch the role of scrum master between every team member. The reason for this is that it will provide a more diverse working environment and will allow for each member to experience time as a project lead. At the end of each sprint we plan on having a retrospective with all the team members and gathering all input and feedback we had for the sprint. This retrospective will allow us to better understand what we did good as a team and where we could improve for the next sprint.


START
Initiate Project
Define Requirements
Development 1
Development 2
Development n
RELEASE
Accept
?
YES
Deploy To Customer
NO
AGILE
METHODOLOGY
Implement Changes
Adjust, Track Re-prioritize
Next Iteration Into Development

Peer Reviews

As part of the development phase we plan on having all of our code reviewed by a team member in order to help prevent bugs from entering into our final deliverable application. The way we will approach this is by creating a peer review process for all code we plan to deploy into our final application. The process should ultimately prevent any bugs from entering into the production application and also allow other team members to explore and understand the code that is being pushed into production. This will later help the entire team better understand what each team member has contributed towards the application. The review process should be used as a checks and balance system for all code that is being created, updated, or removed from the system. The following diagram explains more in depth what the peer review process will look like. We will be utilizing Slack to communicate when a new review has been submitted and the code can be reviewed by any team member.

Peer Review Process



Step 1

- Developer should submit a review message within the review Slack channel in order to notify all other members.
- The massage should include: What is beginning reviewed, what it is affecting in the application and any comments the reviewer should know.

Step 2

- Reviewer should reply to the message stating that a review is in progress, that way one item is not being reviewed at the same time by multiple people.
- The reviewer should test item that is being reviewed and verify that the changes are within the items scope.

Step 3

- Once code is tested reviewer should reply to original review request message and notify developer of a successful review or comment any concerns that might arise.

Step 4

- At this point the developer should submit the item again for a second and final review, this review should be done by someone other than the first reviewer.
- The the reviewer should follow steps 2 and 3.

Step 5

- If any changes need to be made they will be made here, if not developer can deploy code.

Project Planning

For planning our project, we decided to track all of the larger task and provided visibility to all involved parties. The way we approached this was allowing all parties to have a tracking chart also known as a RACI chart. The RACI chart allows for every person within the project to understand the tasks and responsibility of every other person. The chart is broken down by project task and each member or stakeholder on our team. Under project task there is a brief description of each task that should be completed. Under each team members name there are the letters, A, I, R, C. Each of these letters lets each member know what the are the required action for that specific task. The following is a description of what each letter in the chart represents.

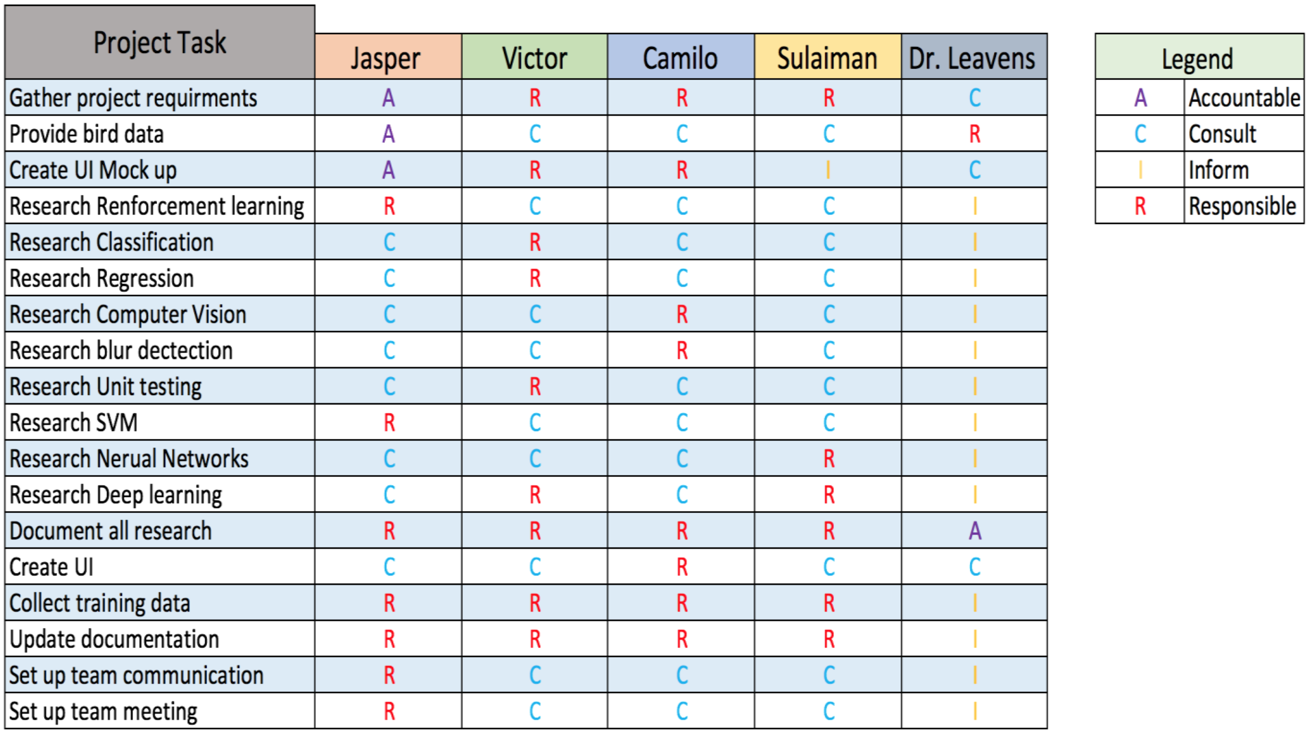
- **Accountable (A) –** This team member delegates work and is the last one to review the task or deliverable before it's deemed complete. On some tasks, the Responsible party may also serve as the Accountable one.

- **Responsible (R) –** This team member does the work to complete the task.

- **Consulted (C**) **–** Every deliverable is strengthened by review and consultation from more than one team member. Consulted parties are typically the people who provide input based on either how it will impact their future project work or their domain of expertise on the deliverable itself.

- **Informed (I) –** These team members simply need to be kept in the loop on project progress, rather than roped into the details of every deliverable.

The following image contains a mock up of our RACI chart. We still have a living document that will continue to be updated as the project progresses and new task are created. The RACI chart is kept within an Excel file that all team members have access to updating at any time.



Project Task	Jasper	Victor	Camilo	Sulaiman	Dr. Leavens
Gather project requirments	A	R	R	R	C
Provide bird data	A	C	C	C	R
Create UI Mock up	A	R	R	I	C
Research Renforcement learning	R	C	C	C	I
Research Classification	C	R	C	C	I
Research Regression	C	R	C	C	I
Research Computer Vision	C	C	R	C	I
Research blur dectection	C	C	R	C	I
Research Unit testing	C	R	C	C	I
Research SVM	R	C	C	C	I
Research Nerual Networks	C	C	C	R	I
Research Deep learning	C	R	C	R	I
Document all research	R	R	R	R	A
Create UI	C	C	R	C	C
Collect training data	R	R	R	R	I
Update documentation	R	R	R	R	I
Set up team communication	R	C	C	C	I
Set up team meeting	R	C	C	C	I

Legend	
A	Accountable
C	Consult
I	Inform
R	Responsible

Unit Testing

The purpose of unit testing is to ensure that the algorithms and models are working as expected and that all code works properly before and after each final commit to our master repository. We will be using Unittest, Python's unit testing frame work to have an easier transition to tools that will assist with testing all of our code. Unittest provides built in functions such as test fixture, test case, test suite, and test runner that make up the the majority of unit test cases. Each function is described by Python's documentation page as follows:

- test fixture
    - A *test fixture* represents the preparation needed to perform one or more tests, and any associate cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.
- test case
    - A *test case* is the individual unit of testing. It checks for a specific response to a particular set of inputs. unittest provides a base class, Testcase, which may be used to create new test cases.
- test suite
    - A *test suite* is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.
- test runner
    - A *test runner* is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests

We will mostly be using this frame to test and validate the accuracy of our models along with some of our user interface code. It is important that we are constantly collecting data on the progress of our model, as this will later become insightful on how we should tweak or improve our models. Unit testing is the first step in having continuous integration and continuous improvement.

Project Summary

Slekit will be an application that utilizes the power of machine learning to automate the process of parsing a user’s image directory for satisfactory images. A satisfactory image can be defined as one of two things in this situation: 1) the image meets or exceeds a general level of aesthetic quality, or 2) the image is desirable to the user. Each user will have the opportunity to train his or her own “assistant” which will learn to mimic their selection preferences. Both professional and amateur photographers will derive utility from this time-saving application.

Budget

Our project did not include any finical assistance from outside sponsors or professors. However, as a team we worked on trying to reduce as much cost as possible and to manage all cost among ourselves. Some of the ways we accomplished this is by using free software.
For communication we used Slack's free version, which allows for 10k searchable messages, 10 apps and integrations, 1-to-1 video calls, and two-factor authentication, the free version gives our team access to Slack's basic features. For research, we used Google Scholar and tried to search for free peer reviewed journals that would provide more insight into the machine learning topics we were interested in. As far as our image data, we were provided with a large amount of pictures by Dr. Leavens. We also were able to find some data set of images online and through other sources such as family and friends.
We did inquire some cost on learning martial. Some of that cost came from online learning material such as Udemy, Coursera, and other online courses. Those cost were covered amongst each other. We do not have an official budget but are managing all cost within our team, amongst each other. With that being said we do not have a final budget to include in our project.

Consultants

We were very fortunate to receive advice and guidance from Dr. Gita Reese Sukthankar, director of the Intelligent Agents Lab at UCF, and Dr. Mahdi Kalayeh, an expert in deep learning systems.

Milestones

DATE (2019)	DESCRIPTION	COMPLETED	MISSED
02/02	Senior Design Bootcamp	✓	
02/06	Discuss problem & requirements w/ sponsor	✓	
02/13	Proposal/Requirements Document	✓	
02/20	Discuss solutions & system layout w/ sponsor	✓	
02/27	Review previous SD team’s Final Design Document, 1 page each on research ML, RL, DL, and CV	✓	
02/28	TA Check-in	✓	
03/01	5 pages each (*Realistic*)	✓	
03/04	Project Status & Doc Review	✓	
03/04	15 pages each		X

03/08	10 pages each (*Realistic*)		X
03/08	Decide on UI		X
03/08	Sample SVM program		X
03/08	Select features to analyze		X
03/18	Project Status & Doc Review	✓	
03/22	15 pages each (*Realistic*)		X
03/25	Research AlexNet	✓	
03/29	Set up SVM		X
03/29	Design RL Solution		X
03/29	20 pages each (*Realistic*)		X
04/05	25 pages each (*Realistic*)		X
04/05	Integrate NN and SVM		X
04/08	TA Check-in	✓	

04/08	20-25 pages each		X
04/12	30 pages each (*Realistic*)		X
04/13	Barebones GUI	✓	
04/24	Final Design Doc Due; 30 pages each		
08/30	Build PAC-NET using Keras		
09/13	Evaluate PAC-NET performance		
09/27	App will support general image quality assessment		
10/01	Develop testing suite		
10/11	App will support user feedback		
11/01	Test run with real users		
11/15	Finished product		
12/01	Final Design Document		

Despite not having (senior design) class for this coming summer, we are all eager to work on the project. Below is an unofficial set of milestones for Summer 2019.

DATE (2019)	DESCRIPTION	COMPLETED	MISSED
05/24	Build an SVM using PyTorch and Keras		
05/24	Build GUI in Electron		
06/07	Build PAC-NET using PyTorch and Keras		
06/21	Evaluate SVM, PAC-NET performance on IQA using different data sets		
07/05	Evaluate PAC-NET's ability to adapt to user taste		
07/19	Research other methods of adaptive learning		
08/02	Create installation guide		
08/02	Experiment with object classification		
08/16	Lay foundation for future documentation		

References

[1] Mitchell, T. M. (2017). *Machine learning*. New York: McGraw Hill.

[2] S. J. Russell and P. Norvig, *Artificial intelligence: A Modern Approach*. Upper Saddle River: Pearson, 2016.

[3] Caruana, Rich, and Alexandru Niculescu-Mizil. 2006. “An Empirical Comparison of Supervised Learning Algorithms.” In *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA.

[4] https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

[5] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[6] https://deepai.org/machine-learning-glossary-and-terms/hyperplane

[7] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. Cambridge (Mass.): The MIT Press.

[8] https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[9] Rosasco, L.; De Vito, E. D.; Caponnetto, A.; Piana, M.; Verri, A. (2004).”Are Loss Functions All the Same?”. *Neural Computation*.

[10] https://data-flair.training/blogs/applications-of-svm/

[11] http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node10.html

[12] https://arxiv.org/pdf/1608.06993.pdf

[13] http://www.robots.ox.ac.uk:5000/~vgg/publications/papers/ferrari07.pdf

[14] L. Marchesotti, F. Perronnin, D. Larlus, G. Csurka, "Assessing the aesthetic quality of photographs using generic image descriptors", 2011.

[15] R. Datta, J. Z. Wang, "Studying Aesthetics in Photographic Images", 2006.

[16] L. Marchesotti, F. Perronnin, D. Larlus, G. Csurka, "Assessing the aesthetic quality of photographs using generic image descriptors", 2011.

[17] K. Ko, J.-T. Lee, and C.-S. Kim, “PAC-Net: pairwise aesthetic comparison network for image aesthetic assessment,” in IEEE Int. Conf. Image Process. (2018).

[18] https://pytorch.org/docs/stable/index.html

[19] https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4

[20] N. Murray, L. Marchesotti, and F. Perronnin, ‘‘AVA: A large-scale database for aesthetic visual analysis,’’ in Proc. IEEE Conf. Comput. Vis. Pattern

[21] R. Schifanella, M. Redi, and L. Aiello, ‘‘An image is worth more than a thousand favorites: Surfacing the hidden beauty of flickr pictures,’’ in Proc. Int. Conf. Web Social Media, 2015, pp. 296–317.

[22] W. Jiang, A. C. Loui, and C. D. Cerosaletti, ‘‘Automatic aesthetic value assessment in photographic images,’’ in Proc. IEEE Int. Conf. Multimedia Expo, vol. 41, no. 3, Jul. 2010, pp. 920–925.

[23] X. Tang, W. Luo, and X. Wang, ‘‘Content-based photo quality assessment,’’ IEEE Trans. Multimedia, vol. 15, no. 8, pp. 1930–1943, Dec. 2013

[24] https://arxiv.org/pdf/1608.02676.pdf

Figures

[1] https://machinelearningmastery.com/basic-concepts-in-machine-learning/

[2] Created by Professor Loc Vu-Quoc

[3] https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

[4] https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[5] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[6] https://www.bogotobogo.com/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron.php

[7] Custom image.

[8] Custom image.

[9] Custom image.

[10] https://www.jeremyjordan.me/convnet-architectures/

[11] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[12] Simonyan, K., Zisserman, A. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv e-prints arXiv:1409.1556.

[13] Custom image.

[14] Custom image.

[15] Custom image.

[16] Custom image.

[20] G. Huang, Z. Liu and L. van der Maaten, “Densely Connected Convolutional Networks,” 2018.

[22] https://www.capturelandscapes.com/the-rule-of-thirds-explained/

[25] K. Ko, J.-T. Lee, and C.-S. Kim, “PAC-Net: pairwise aesthetic comparison network for image aesthetic assessment,” in IEEE Int. Conf. Image Process. (2018).

[26] N. Murray, L. Marchesotti, and F. Perronnin, ‘‘AVA: A large-scale database for aesthetic visual analysis,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., vol. 157, no. 10, Jun. 2012, pp. 2408–2415.

[27] Custom image.

[28] Custom image.

[29] Custom image.

[30] Custom image.