DSO 593: Independent Research Final Report

**Kofi Buahin**

**2nd Year MSBA Student – Student #: 9607811646**

# Project Description/Overview:

During the last lecture of DSO530 with Professor Xin Tong, we were introduced to Neural Networks as an optional portion of the course. I learned that Neural networks are a subset of machine learning that set out to reflect the behaviors of the human brain and give computer programs the ability to recognize patterns and solve common problems in different settings. This introduction got me curious to learn more. For my Independent Research Project, I would like to complete a ***deep dive into Neural Networks, how they work and their real-life applications***. Using external course material and resources briefly detailed below, I hope to build my knowledge in this area and submit a set of deliverables that displays my learning.

Over the course of the summer I have spent time refining my knowledge in this space of machine learning. I have referenced industry reading material and coursework to help build my knowledge of the theory behind Neural Networks and build my technical skillset with the . Specifically I read the [Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition](https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/) textbook by **Aurélien Géron** and completed relevant sections of well respected Data Scientist Andrew Ng’s coursera course, [Introduction to Machine Learning with Python](https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/).

The purpose of this report is to reflect on the following topics:

* Review Key Concepts I learned surrounding Neural Networks and their applications.
* Discuss the two Neural Network Models I built to apply the knowledge I developed over the summer.
* Highlight how I can build on my learning moving forward.

# Learning Timeline:

Here is a quick timeline documenting how I completed my research over the summer. I started going through the reading material I found surrounding Neural Networks at the end of May. I went through the segments of I completed Aurélien Géron’stext book in the middle of July and went through

Once this was done, I spent the 3rd week of July working on the first of the two models I would build to apply all that I had learned over the summer – a Neural Network Regressor to enter the [Housing Prices – Advanced Regression Techniques](https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview) Kaggle Competition. With this first model, I trained both a traditional Multi-Linear Regression Model on the dataset, then built a Neural Network and submitted the predictions for both models to the competition for scoring to compare performance.

In the Last week of July, working up into the first of August, I started work on my second model, a series of image classification models based on different Convolutional Neural Network structures that I learned about over the summer. I found the [Natural Images dataset](https://www.kaggle.com/datasets/prasunroy/natural-images) on Kaggle and used parts of this dataset (over 2,000 images of cats, dogs and human faces) to train my models and test the predictive power of these models on some of my friend’s and their pets.

# Learning Summary:i

## Neural Networks – A Brief History:

Let’s start with a simple definition for Artificial Neural Networks(ANN) – They are a Machine learning model based on the relationships and networks of the neurons that make up our brains**(citation)**. The first model of an artificial neuron was proposed back in 1943 by Warren McColluch and Walter Pitts. They established a computational representation of the way Biological Neurons work – each neuron possesses one or many binary input(s) and a single binary output **(citation maybe).** Their work illustrated that even in the most basic models, a collection of artificial neurons can be built to carry out sophisticated calculations.

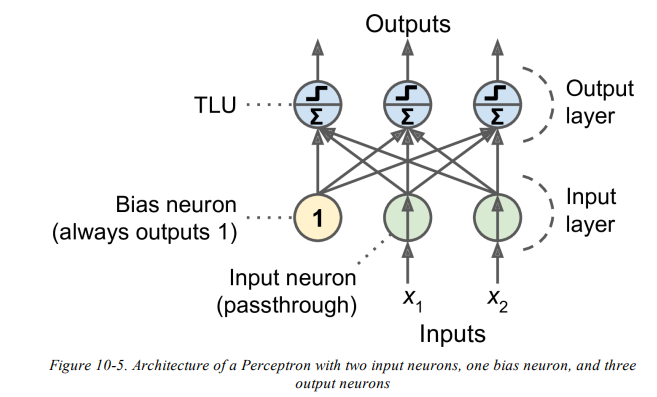
After the initial work of McColluch & Pitts, there was a time in the 1960’s where the optimism surrounding Neural Networks died down and resources were applied to other area. Nonetheless, there was more progress with the development of new neural network architectures with enhanced model training capabilities in the 1980’s. Then came a period in the 1990’s where alternative Machine Learning Algorithms with higher performance, like supper vector machines, were developed and this again slowed down the progress of research into Neural Networks.

With all this in mind though, we are currently in a cycle of renewed interest in Neural Networks. With the proliferation of big data, there is now vast amounts of resources to train these ANNs. Furthermore, there has been a distinct increase in computing capabilities since the last major wave of interest in Neural Networks in the 90’s. These two factors combined with the fact that there is immense funding towards researching Neural Networks, there is strong reason that this new wave of commitment to the applications of Neural networks is here to stay.

## Perceptrons – Base Level Artificial Neuron

To get started with Neural Networks, we must briefly define Perceptrons, what they are and how they work. Invented in 1957 by Frank Rosenblatt, Perceptrons represent one of the most basic ANN architectures and they are built off a unique type of Artificial Neuron known as the **Threshold Logic Unit (TLU)** – a neuron where inputs and outputs are numbers rather than binary on vs. off values as originally proposed by McColluch and Pitts. With a TLU, every input connection has a weight and the TLU calculates the weighted sum of all inputs fed into the network and then uses a step function on this weighted sum that returns a result.

Essentially, a Perceptron is made up of one layer of TLUs, where all the TLUs are connected to every single input that is fed to the network. A sample use case of a single TLU is a simple linear binary classification. You can also see a sample diagram of a Perceptron with two inputs and three outputs shared below:



As Perceptrons have weights assigned to each of the inputs, the network must be trained to determine the optimal weights assigned to each input. Perceptrons are trained on an algorithm based off of Hebb’s Rule – this rule proposes that the strength of a connection between two biological neurons increases when one of these neurons triggers the other frequently. The rule that Perceptrons are trained on factors in the error of a network when predicting an output based on the weighted inputs – the rule strengthens connections that work towards reducing the error of the overall network.

Perceptrons are a key concept with Neural Networks as they represent the most basic form of Artificial Neural Networks. They do possess some limitations, most prominently they can’t provide solutions for a number of simple problems including the Exclusive OR (XOR) classification problem. Some of these limitations can be rectified in with a **Multi-layered Perceptron(MLP)**, which is the next ANN architecture I explored.

## Multilayer Perceptrons – Development on Basic Perceptrons:

A MLP is an ANN architecture that is built of an input layer, one (or multiple) layers of TLUs known as **hidden layers** and a final layer of TLUs referred to as the **output layer**. It is essentially a neural network of dense-layered perceptrons stacked on top of one another. In the case where an ANN is composed of a deep stack of hidden layers, it is referred to as a **Deep Neural Network.** You can see the sample structure of an MLP ANN architecture presented in the diagram below:

Diagram

Description automatically generated

MLPs are trained using the **backpropagation** training algorithm proposed by David Rumelhart et al in their renowned [research paper](https://scholar.google.com/scholar?q=Learning+Internal+Representations+by+Error+Propagation+author%3Arumelhart) in 1986. The backpropagation algorithm implements two passes through the Neural Network (one forward, and one backward). Through these two passes it calculates the gradient of the network’s error for each parameter that is a part of the model(**citation**). This means it determines the way in which every connection weight and bias term in the network should be altered to reduce the network’s error. With these gradient’s the algorithm then conducts a gradient descent step & repeats this cycle until a solution is reached. In summary, the Backpropagation algorithm follows 3 steps:

1. Makes a prediction through the neural network with current weights and inputs (**forward pass**) and calculates the error of this prediction.
2. Goes back through every layer in the network backwards (**backward pass**) and calculates each connections contribution to the overall error of the network.
3. Alters the connection weights to reduce the error of the overall network (**Gradient Descent Step**).

For this to work, Rumelhart et al. made a key alteration to the MLP architecture, where the implemented a logistic activation function rather than the traditional step function with Perceptrons. There are a number of other activation functions that can be used in place of the sigmoid function (depending on the use case) including the **hyperbolic tangent function ( tanh(z) )** and the **Rectified Linear Unit function** (**ReLU (z) )**.

The MLP ANN architecture can be applied to both Regression and Classification problems. This implementation is commonly done through **Keras**, a Deep Learning API that allows you to develop a plethora of neural networks. This API was at the center of two of the deliverables that I will walk through later in this report.

### Training an MLP Neural Network

ANNs possess a large amount of flexibility – the individual tuning the network has a number of parameters they can modify in order to come to the Network architecture that provides the strongest predictive power or classification accuracy. Some of the key hyperparameters that can be tuned include the following:

* The number of layers included in the network
* The number of neurons in each layer included in the network
* The activation function that is applied to each layer that is included in the network
* The weight initialization logic applied to each network (this can also heavily depend on the activation function selected).
* The optimizer used when compiling the neural network.
* The learning rate that is passed to the optimizer.

One of the most common ways to set the hyperparameters for the model is to use multiple different combinations of the hyperparameters and evaluate the error or accuracy provided by each variation of the model. This can be done with either GridSearchCV or RandomizedSearchCV from the sci-kit learn library on python.

## CNNs - Convolutional Neural Networks

Once I developed an understanding of the basic ANN architectures, I moved onto a more complex class of ANN architectures, known as **Convolutional Neural Networks (CNN)**. The CNN class of ANN is popularly known for it’s applications to image recognition and came into existence from studies surrounding the brain’s visual cortex.

Through their research in the late 50’s David Hubel and Torsten Weisel, showed that numerous neurons located in the visual cortex possess only a small local receptive field – This means that these neurons only have the capability to react to visual stimuli found in a limited region of the visual field **(citation)**. The key idea derived from their research was that higher-level neurons are built on outputs generated by nearby low-level neurons. This structure creates a situation where sophisticated patterns can be identified from different areas of the visual field.

This research from Hubel and Weisel was the inspiration for the Neocognitron, a self-organizing ANN model applied to pattern recognition **(citation)** which evolved over time to become what is commonly known as the **Convolutional Neural Network**. On top of the fully connected layers and activation functions I learned about with the Multi-layered Perceptron ANN architecture I learned about before, CNN’s added two new types of layers for a Neural Network architecture.

### Convolutional Layer

The first of these two new layers is the **convolutional layer** which represents the central building block of a CNN. Neurons in the first convolutional layer are only connected to the pixels that fall in their receptive field, unlike layer structures I learned about and detailed earlier, that are connected to every pixel in the given input passed to the Neural Network. Every neuron in the second convolutional layer of the CNN, is connected to neurons found within a small rectangular subsection of the first layer. The basic architecture of a CNN, as displayed in the image presented below, gives the network the freedom to focus on small low-level features in the first layer and then, in the following hidden layer, combines these features into more refined, higher-level features. This continues for the number of layers that comprise the CNN.

Shape

Description automatically generated

The weights of a given neuron can be portrayed as a small image that is the size of a receptive field. These filters impact how a neuron processes a receptive field. A feature map that highlights the regions of an image that the filter activates most strongly is produced by a layer of neurons that all use the same filter. A convolutional layer is most properly represented in 3D vs. 2D since it includes numerous filters (determined by the individual training the network), each of which produces a feature map. Moreover every neuron that is part of a certain feature maps share the same features (weights and bias terms). Essentially, a convolutional layer can recognize many features wherever in its given inputs by simultaneously applying multiple trainable features to it.

When implementing a convolutional layer there are several hyperparameters that must be tuned or decided on:

* The number of filters to include
* The height and width of the convolutional layer
* The strides (these are the shifts from one receptive field to another)
* Type of Padding (whether or not to add zeroes around inputs to ensure a layer maintains the same dimensions (height & width) as the previous layer)/

Now that we have detailed the key elements of the convolutional layer, we will look at the 2nd key layer time with CNNs.

### Pooling Layer

The Second of the two new layers introduced with the CNN class of Neural Networks is the **Pooling Layer** . The main function of the pooling layer is to subsample the input. This is done in order to lessen the computation effort, memory utilization and number of parameters(this portion is specifically key to minimize the risk of overfitting).

Similarly, to the convolutional layer, every neuron in the pooling layer is linked to the outputs of a small subset of neurons found in the previous layer situated inside a constrained rectangular receptive field. Also like the convolutional layer, the pooling layer requires you to define the size, stride and padding. The key difference between the two types of layers is that a pooling layer is that the neurons in the pooling layers have no weights – all they do are aggregate the inputs passed to the network using a specified aggregation function (for example the max or mean). In most cases it is more effective to use a max aggregation function for the pooling layer rather than an average aggregation function.

### CNN Architectures:

Once I grasped the key new elements of a CNN, I started to explore the different architectures that have been built in the past. A typical CNN architecture consists of a stack of some convolutional layers, each of which is typically followed by a ReLU layer. After this sequence of convolutional layers there is a pooling layer added and then that pooling layer is followed by another stack of convolutional layers (again followed by a ReLU layer) and then another pooling layer. This repeats for as long (or as deep) as the neural net architecture has been set. Naturally, the image size reduces as it moves through the CNN and also becomes deeper (meaning there are more feature maps), both of which are a result of the presence of convolutional layers. Generally, at the top of the stack of layers, a normal feedforward ANN is included in the CNN and the last layer provides the final prediction. The typical structure of a CNN can be seen depicted in the diagram below:

Diagram

Description automatically generated

There are a number of more defined architectures that have been created for multiple classification problems in the past. Here is a brief summary of 6 of the architectures that I was able to learn more about in my research:

* **LeNet-5 Architecture** – This is probably one of the most famous CNN architectures in the Deep Learning Community. It was created by Yann LeCun in 1998 and has been commonly applied handwritten digit recognition tasks. The structure consists of 3 convolutional layers, 2 pooling layers and 2 fully connected layers. Each of the layers use the Tanh activation function except the output layer.

Table

Description automatically generated

* **AlexNet Architecture** – This architecture was developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton. It has proven successful as it won the 2012 ImageNet ILSVRC[[1]](#footnote-1) challenge commandingly. It’s architecture is very similar to that of LeNet-5 however it is significantly larger & deeper. It also stacks convolutional layers on top of one another as opposed to stacking a pooling layer on top of each convolutional layer. It consists of 5 convolutional layers, 2 max pooling layers and 3 fully connected. The different layers all use the ReLU activation layer.

Table

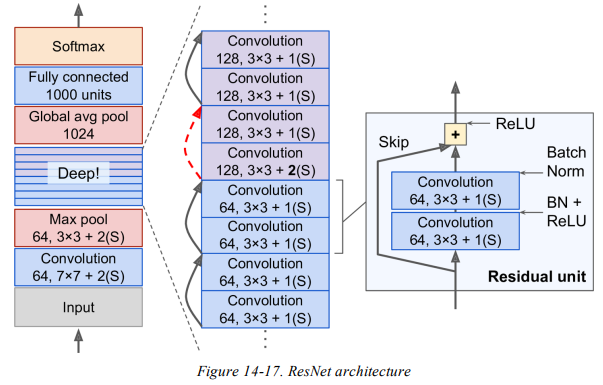
Description automatically generated

* **GoogLeNet Architecture** – This architecture was built by Christian Szegedy et al from Google Research and has also seen classification success, winning the 2014 ILSVRC. The strong performance of this network is spurred by the fact that it is much deeper than previous strong CNN networks that have been developed. This is made presence of *inception modules* in the network, which give the GoogLeNet architecture the capability to utilize parameters in a more efficient manner than other architectures. When it comes to actual architecture, the GoogLeNet CNN architecture is essentially one tall stack of layers that includes 9 inception modules.

Diagram

Description automatically generated

* **VGGNet** – This specific architecture was developed by Karen Simonyan and Andrew Zisserman. It has a straightforward architecture that consists of 2-3 convolutional layers and a pooling layer followed by another 2-3 convolutional layers and so on. This pattern repeats until you have a total of 16-19 convolutional layers in the architecture (this is dependent on the variant of the VGG). The architecture is concluded with a dense network that consists of 2 hidden layers and the final output layer.
* **ResNet** – This stands for Residual Network. This architecture was conceived by Kaiming He et al and also showed its classification prowess by winning the ILSVRC 2015. The winning variant of this network’s architecture implemented a very deep CNN made up of 152 layers (much larger than previous variants). This highlighted the trend of CNN architectures moving towards a deeper stack of layers. The central element of developing a deep network like the ResNet architecture is to implement *skip connections*. These connections serve to input signals to the output layer situated higher up in the stack of layers. By adding a given input x to the output of the network (adding a skip connection) you force the network to implement residual learning (modelling ***f(x) = h(x) – x*** as opposed to ***h(x)***). Adding these skip connections allows the network to begin its learning progress before multiple layers even begin learning.



* **Xception** – This CNN architecture combines the concepts of the GoogLeNet and ResNet, with a key difference in that it replaces inception modules found in the traditional GoogLeNet with a different type of layer called a *separable convolution layer*. This unique type of layer works under the assumption that spatial patterns and cross-channel patterns can be modeled separately from one another. The first portion of this layer applies one spatial filter for every input feature map present, while the second searches only for cross-channel patterns. The actual architecture of an Xception ANN begins with 2 normal convolutional layers that are followed by predominantly separable convolutions (34). The architecture also incorporates a few max pooling layers and the traditional finishing layers (average pooling layer & output layer).
* **SENet** – Standing for Squeeze-and-Excitation Network, the SENet was the last popular CNN architecture I researched. This architecture represents an extension of inception networks and ResNets and enhances their performance (the SENet won the ILSVRC 2017 challenge). This boost in performance comes from the added feature of a SE Block (a small neural network) to the original ResNet and Inception model. This means each inception module or residual unit has an SE Block attached to it. The SE block’s function is to analyze the output of the unit that is linked to with a primary goal to evaluate depth dimension (ignores spatial patterns) and learn which combination of features are most active together. The SE block then utilizes this data to recalibrate the feature maps. For example, in training an SENet for a classification task, a SE Block exposed to pictures that include humans could learn that mouths, noses, and eyes typically show up together in images, ergo if the model sees a mouth and nose, it should anticipate seeing eyes as well. How the SE block operates is that if it observes strong activation in the mouth & nose feature but only light activation in the eye feature map, it will heighten the eye feature map to incorporate this new learning it makes (reduce the irrelevant feature maps). An SE block is made of 3 layers: a global average pooling layer a hidden dense layer (ReLU activation) and a dense output layer (Sigmoid activation). You can a sample SE block structure as well as the process by which it recalibrates feature maps below.

Diagram

Description automatically generatedDiagram

Description automatically generated

## Real Life Applications:

Once I developed my understanding of how different forms of Neural Networks work and their respective architectures, I was curious to learn more about how they are used in different settings. Artificial Neural Networks are the center of forefront of Deep Learning. Due to their versatility and scalability, they are a very useful tool in dealing with complicated Machine Learning challenges and tasks. Here are three of my favorite applications that I’ve read about as a part of my research:

1. **Image Classification**: One of the most popular applications of Neural Networks is Image Classification. One of the key strengths of Deep Neural Networks is that they have the ability to identify images at a pixel level. CNN based models have been used from image classifiers on social media (think when Facebook used to have the tag recommendation feature) to applications in classifying images on X-rays and CT scan images.
2. **Speech & Voice Recognition**: Speech Recognition plays a role in a number of key everyday tasks in our modern world today such as video gaming, virtual assistance home automation and more. Neural networks play a huge role in these tasks specifically CNNs. In 2014, A mixed team of researchers from Microsoft Research, York University and the University of Toronto described in their paper how CNNs can create an additional reduction in error on speech recognition technology here(**citation**).It is said that the CNN is allows for some inconsistency in speech recognition such as varying speaking rates.
3. **Video Recommendation**: I spend an exorbitant amount of time on YouTube, be it watching soccer highlights or searching for product reviews. It was cool to learn that YouTube implements Deep Neural Networks to provide video recommendations. In their paper for Google Research, Paul Covington, Jay Adams and Emre Sargin detailed the key role deep learning plays in Googles deep candidate generation and deep ranking models(citation).

With this summary of what I learned, we can now walk through a quick summary of how I applied these new tools, with two specific deliverables that are available through my GitHub profile.

# Final Deliverables:

## Regression Neural Network Model:

My first deliverable was a regression problem – specifically I wanted to test the basic learnings I had around Multi-layer Perceptrons and try to compare the performance of a traditional Linear regression model to the performance of a MLP Neural Network regressor. As a litmus test, I entered Kaggle's Housing Prices Advanced Regression Learning Competition, which is an advanced regression challenge where entering contestants are given access to the Ames Housing Dataset (a dataset with 80 variables) and can build models to predict housing prices. Contestants are then asked to submit predictions on a test can then make predictions on a test dataset provided by the organizers of the competition and are scored on the RMSE of the logarithm of sale price and logarithm of the predicted price.

I started out by building a Multiple Linear Regression Model. My approach to building this model is summarized below:

1. **Pre-process Data:** The data set wasn’t particularly messy however there were some preliminary
2. **Exploratory Data Analysis:**
3. **Model Building Part:**
4. **Addressing Multicollinearity:**
5. **Final Predictions:**

Once I built this out, I proceeded to develop and train a Multi-Layer Perceptron to attempt predictions on the same test set. My approach to building this basic Neural Network can be seen summarized below:

The predictions I submitted generated from the Neural Network Regressor were slightly more accurate than the predictions submitted from the Multiple Linear Regression model I trained.

The final Jupyter Notebook for submission can be found on my github project – [Ames Housing Prices Prediction Neural Network Regressor vs MLR](https://github.com/kofibuahin/Ames-Housing-Prices-Prediction-Neural-Network-Regressor-vs-MLR). The Jupyter Notebook provides annotations of my thought process at each step of the approach and a conclusion into my findings.

## Convolutional Neural Network & Image Classification:

The second deliverable was based on the more complex Neural Network Architecture I learned about this summer – the CNN. I decided to use the Natural Images Dataset that I sourced from Kaggle (<https://www.kaggle.com/datasets/prasunroy/natural-images>) to train 3 image classification models. In this deliverable, I built and trained 3 different types of Networks (2 Convolutional Neural Networks and 1 Multi-Layer Perceptron) on the Natural Images dataset and test their efficacy.

I then ended the project off At the end of the project, we conclude with a short fun activity where I apply the trained models to a 25-image dataset made up of images of my friends, some of their pets, famous cats & dogs and famous individuals in society.

The final Jupyter Notebook for submission can be found on my github project --

The Jupyter Notebook provides annotations of my thought process at each step of the approach and a conclusion into my findings.

## Conclusion & Moving Forward:

This summer I was able to learn a lot about Deep Learning and Neural Networks. I was able to take a journey through the history of this Machine Learning Concept, starting out with the most basic architecture, the perceptron, and worked my way up to one of the most common architectures – the Convolutional Neural Network. I was also able to sharpen my technical skills in Python, working through two separate applications of Neural Networks that will be an asset for me professionally as I move forward.

With all this in mind, there is still much to learn about Neural Networks and their applications. While I focused on the basics and one additional complex architecture, there are some more key architectures to explore. The most prominent one is the **Recurrent neural network (RNN)** but there are other key architectures such as the Long short-term memory (LSTM) and autoencoders. In the future, I hope to learn more about these architectures and apply them to personal projects and in the industry where I end up working, wherever that may be.

## Citations:

* <https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf>
* Cite Aurelian Gueron’s Book
* <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf>
* <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf>

1. [ImageNet Large Scale Visual Recognition Challenge](https://www.image-net.org/challenges/LSVRC/#:~:text=The%20ImageNet%20Large%20Scale%20Visual,image%20classification%20at%20large%20scale.): evaluates algorithms for object detection & image classification at large scale. [↑](#footnote-ref-1)