Deep Convolutional Neural Networks for Remote Sensing Investigation of Looting of the Archeological Site of Al-Lisht, Egypt

by

Timberlynn Woolf

A Thesis Presented to the
Faculty of the USC Graduate School
University of Southern California
In Partial Fulfillment of the
Requirements for the Degree
Master of Science
(GEOGRAPHIC INFORMATION SCIENCE AND TECHNOLOGY)

August 2018

For Alice

Table of Contents

List of Figures

List of Tables

Acknowledgements

Thank you to the DigitalGlobe Foundation for the satellite imagery grant and images courtesy of the DigitalGlobe Foundation. Thank you to my brother, John, for all of your love and support and to my faculty, family, and friends who have supported and encouraged me through this process.

List of Abbreviations

AlexNet	CNN architecture developed by Alex Krizhevsky
BOVW	Bag of visual words
Caffe	Convolutional Architecture for Fast Feature Embedding
CaffeNet	CNN architecture provided by Caffe
CNN	Convolutional neural network
DGF	DigitalGlobe Foundation
DNN	Deep convolutional neural network
FC layer	Fully-connected layer
GEP	Google Earth Pro
GIS	Geographic information system
GISci	Geographic information science
HRRS	High-Resolution Remote Sensing
IFK	Improved Fisher kernel
ReLU	Rectified linear units
SVM	Support vector machines
SSI	Spatial Sciences Institute
UFL	Unsupervised feature learning
USC	University of Southern California
VGG-F	Fast CNN architecture developed by Chatfield
VGG16	16 layer Deep CNN

Abstract

Looting of archaeological sites is a global problem. To quantify looting on a nationwide scale and to assess the validity and scope of the looting reports and modern encroachment, satellite archaeologist have turned to mapping looting from space. High-resolution satellite imagery has become a powerful tool and resource for monitoring looting and site destruction remotely and proves to be an independent way to cross check and analyze against varied and unreliable reports from media and government agencies. It is estimated that over a quarter of Egypt's 1100 known archaeological areas have sustained major damage and site destruction directly linked to looting. The organized looting and illicit trafficking of art and antiques, known as cultural racketeering, is a multi-billion dollar worldwide criminal industry that thrives in Egypt during times of political and economic turmoil and potentially funds drug cartels, armed insurgents, and even terrorist networks. This study analyzes methods used to monitor site looting at the archaeological site of al-Lisht which is located in the Egyptian governorate of Giza south of Cairo. Monitoring damage and looting over time has been largely dependent upon direct human interpretation of images. The manual image comparison method is laborious, time consuming, and prone to human-induced error. Recently, partially-supervised methods using deep convolutional neural networks (CNNs) have shown astounding performance in object recognition and detection. This study seeks to demonstrate the viability of using deep convolutional neural networks (CNNs) within the field of archaeology and cultural heritage preservation for the purpose of augmenting or replacing the manual detection of looting. It brings recent advancements from the field of Artificial Intelligence to an applied GIS challenge at the intersection of remote sensing and archaeology. The objective is to show that CNNs are a more accurate and expedient method for the detecting of looting with wide-ranging application beyond this specific research.

Chapter 1 Introduction

Cultural heritage theft and small-scale looting has been a part of the history of Egypt for thousands of years. In recent years the looting has increased dramatically which has been shown to be primarily a result of economic factors. Looting in Egypt has escalated dramatically from economic and political instability seeing a dramatic spike with the onset of the economic crisis of 2009 and the Egyptian Revolution of 2011. In the recession of 2009, the consumer price index rose and tourism fell with record numbers of unemployment reaching 36% (Parcak 2016). The economic crisis was then shortly followed by the 2011 Egyptian revolution known as the Arab Spring. During this timeframe Egypt experienced the highest overall values of both looting pits and encroachment (Parcak 2016). High unemployment and financial incentives have increased looting dramatically. An increase in looted goods flooded the underground antiquities market during these timeframes and were thought to be potentially used to fuel international crime from drug trafficking, illegal arms trafficking, and even terrorism. With archaeological site destruction comes the loss of the material culture from the site and the loss of the piece of history that was taken. Satellite Archaeologist are taking dramatic steps using satellite remote sensed imagery and GIS for manual and partially-supervised processing and assessment in order to gain a better understanding of the damage and loss and to aid the future prevention.

This project focuses on one previously studied archaeological site of al-Lisht, Egypt. Al-Lisht has seen a dramatic spike in looting following the 2009 economic downturn and 2011 Arab Spring Revolution. This study analyzes the looted site using the manual method of detection of direct human interpretation of remotely sensed images monitoring/ detecting change over time. In an effort to improve upon the manual comparison method which is time consuming and prone to human-induced error, this study investigates the use of a partially-supervised method of deep convolutional neural networks (CNN) for looting pit classification using high-resolution satellite imagery.

1.1 Study Area

The study area is located south of Cairo archaeological site of al-Lisht in the Egyptian governorate of Giza south of Cairo. Al-Lisht is the site of Twelfth and Thirteenth Dynasties Middle Kingdom royals and elite burials, including two pyramids built by Amenemhat I and Senuseret II.

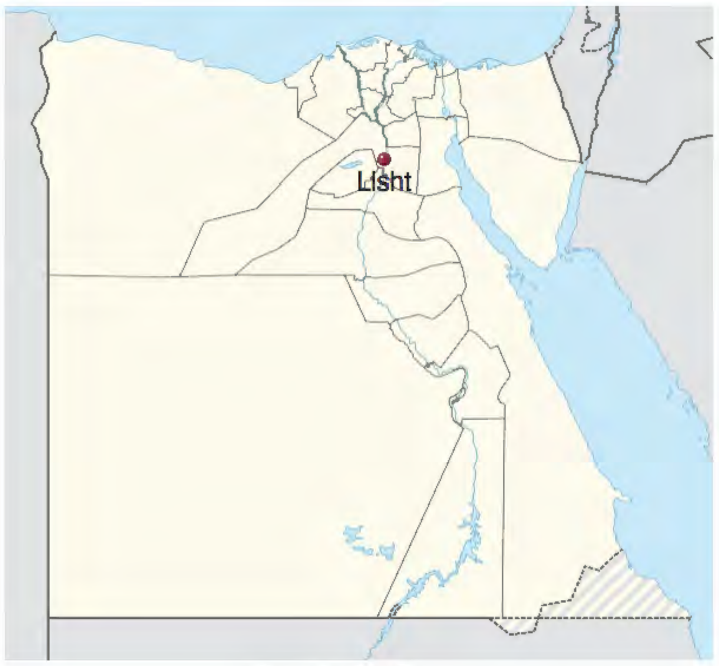


Figure 1: Study Area al-Lisht

1.2 Motivation

Time is running out for many cultural heritage and archaeological sites. Once a site has been looted there is no way to determine what exactly has been lost. It is a part of history that may never be known to the world. Monitoring damage and looting over time has been largely dependent upon direct human interpretation of images. The manual image comparison method is laborious, time consuming, and prone to human-induced error (Lauricella et al., 2017) Current advances in deep convolutional neural networks have achieved breakthrough performance in object recognition and detection achieving up to 96% accuracy. This study seeks to show the potential to expedite the looting detection process using Deep Convolutional Neural Networks (CNNs).

Monitoring of looting is complicated in that it is an illicit activity, subject to legal sanction (Contreras 2010). Poor or unreliable information about damage from looting has an impact on policy making. It enables claims that the extent of looting damage is being exaggerated. This allows artifacts that reach the market to be claimed as chance finds (objects discovered not related to the activity of illicit digging), or pre-existing collections which in turn do not call for strong policy making or response (Contreras et al., 2010). Additionally, it is difficult to monitor the effectiveness of any ameliorating policies whether direct at demand (the illicit trafficking of antiquities) or supply (illicit digging itself) (Contreras 2010). In an effort for site preservation, and for stopping the illicit trafficking of antiquities, this study hopes to identify the best method for monitoring, detecting, and for the prevention of looting of archaeological sites over time. Additionally, this study hopes to bring awareness and understanding to local government agencies who can use this information for counteractive measures and preventative policies as well as proactive measures such as community outreach programs.

1.3 Project Purpose and Scope

The purpose of this project is to expand and build upon CNNs methods that have been used for other forms of scene and object recognition and to train and apply it to the specific task of monitoring looting. The intent of this effort is for it to provide a more accurate and expedient method for looting pit detection. The scope of this project covers the previous manual methods of looting detection, the use of partially-supervised methods, the use of CNNs for other forms of scene classification and object recognition, and lastly how CNNs were used in this study for looting pit detection.

The benefits of monitoring looting are that it provides a basis for quantification estimates of damage from looting. It also can allow for temporal estimates to be made of the time periods in which the looting occurred. Furthermore, quantifying looting in an expedient manor using the CNN method can potentially provide a means of assessing damage still being done to archaeological sites with the chance of linking the damage to the trade in illicit activities. Possible uses of this study and its results can aid in predictive policing. Knowing what dates and times the sites were looted and the economic factors that motivate them can aid in predicting of looting and the predicting of possible attempts in the future. Actions can be taken to protect these sites and provide possible monitoring of artifacts that could be illicitly trafficked making their way to the antiquities black market from these sites. It can also be used to hypothesize what to look for in the illegal international antiquities trade market and aids in the creation of an international watch list. Monitoring looting helps us know what types of sites have been looted and from what time period. This knowledge can aid in notifying international agencies such as US Immigration and Customs Enforcement and INTERPOL (Parcak 2016). Additional motivations are archaeological site discovery, looting detection, and ultimately, preservation of cultural heritage sites and artifacts.

Chapter 2 Related Work

In satellite archaeology and remote sensing there are many methods employed for the investigation of looting. However, there is still a lack of investigation using CNNs with high-resolution remotely sensed imagery for object recognition and detection that could be potentially used for looting detection. This section first discusses the manual direct human interpretation of satellite images for looting detection. Second, it discusses other partially supervised methods and other CNN methods for scene classification. Third it discusses CNNs and the potential for the supervised deep convolutional neural networks to be trained for looting detection.

2.1 Manual direct human interpretation of satellite images for looting detection

In the article, *Satellite Evidence of Archaeological Site Looting in Egypt*: 2002-2013, Sarah Parcak and her team turned to the use of satellite imagery and GIS to map the looting in Egypt from 2002-2013. They used Google Earth Pro (GEP) to obtain satellite data from 2002-2013. The imagery quality that was available varied over time with the changes that came in the initial development phases in high-resolution satellite imagery. Initially image availability was in the range of 20% in 2003 but changed dramatically over time increasing to around 50% in in 2005 and 70% for 2009 to 2011 (Parcak 2016). In order to reach 100% coverage they extrapolated the data for incomplete coverage years (Parcak 2016).

Looting pit encroachment on sites was then assessed between the years of 2002-2013 to determine the extent of the looting and the damage. They assessed the 1100 sites they surveyed for damage and determined that 24.3% displayed evidence of damage and looting (Parcak 2016). 267 sites were georeferenced within ArcGIS and individual polygons were drawn over each looting pit and a series of larger polygons were placed over areas that had been affected by encroachment. Natural boundaries such as roads and rivers were used to determine the extent of the perimeter for undefined sites. In the past satellite imagery did not have the resolution to do this, but now it is possible with GEP. Polygons were drawn around the location of the sites perimeter. By determining the perimeter, they were able to calculate the area of the site and the percentage of the site that had been looted. From the data in the research gathered by Parcak and her team quantifying site looting in Egypt, they were able to determine that looting in Egypt since 2002 did not increase steadily over time, but fluctuated dramatically with recent political and economic instability. By discerning and detecting the political and economic indicators that seem to come before the dramatic increase in looting, efforts can be made to protect archaeological sites that are known to be at risk.

A previous study of the Viru Valley, Peru by Contreras in 2010, used Google Earth imagery as a useful tool for addressing the scale of looting damage to archaeological sites (Contreras 2010). As with present day concerns when using remote-sensing, they encountered problems with coverage, appropriate resolution, and surface visibility (Beck 2006; Ur 2006; Scollar et al. 2008; Parcak 2009). The Viru Valley Study utilized remotely sensed imagery in conjunction with historical site surveys and then analyzed their findings using GIS analysis.

2.1.2 Looting Pit Quantification Methods

The method Parcak used to calculate this was inverse distance weighting (IDW). Inverse distance weighting is a special interpolation algorithm. Weights are proportional to the inverse of the distance between points and the predicted location. The IDW weighs the values from specific known points and uses the inverse of the distance of those points to assign and predict approximate weights for the unknown area within the extent of the known points (Parcak 2016). One of the drawbacks of this study and the IDW method is that it only indicates looting attempts and does not predict if it was a successful attempt or not. There is no specific way to determine what has been taken and to what extent, only that a site has been looted.

The Viru Valley, Peru 2010 study inspected the valley sites utilizing Google Earth for signs of obvious and extensive looting and visible pitting on aerial and satellite images. Correlation was made between looting pits identified in images and looting that was established from areas known to be badly damaged by looting utilizing survey information. Areas noted for looting were referenced with a polygon for later evaluation. 263 areas were identified using a combination of historical images, ground survey data, known looted sites, and remotely sensed imagery for further analysis. Once looted areas were identified, control points were selected and a .jpg image was downloaded from Google Earth at highest resolution possible (4800x 3229 pixels) adjusted (contrast, brightness and color balance) for improvement of visibility of features, and georeferenced in ArcGIS 9.2. Areas identified as damaged by looting were used to create boundary polygons in ArcGIS. Resolution quality did not allow for accurate or adequate counting of individual pits and thus did not allow for the direct calculation of the total number of pits and density of pits in each site. Instead, the total looted area was approximated by bounding the visibly disturbed areas. This lent to multiple polygons being defined for one select site location. As a result of this effort, polygon shapefiles were used to calculate the looted area. Published literature was then consulted to identify previously mentioned looting damage and when the site was most likely dated to be looted. By doing so, they were able to assess possible types of artifacts a particular site might yield or have yielded to the illicit antiquities market (Contreras 2010). The research and methodology allowed for quantifying the scale of looting in the Viru Valley. It also allowed for the identification of recent looting.

2.1.3 Remote sensing for looting detection

Previous use of satellite remote sensing in Syria dates back to 2012. Using satellite imagery in Syria for site looting detection and destruction. Unfortunately, most reports of war-related destruction in Syria come from journalists, and photos and videos posted on social media by potentially biased actors in the conflict. The Syrian government's Director General of Antiquities and Museums releases periodic reports regarding looting and site destruction but has been harshly criticized for being selective in reporting with political motivations. (Casana 2014). Previous work to document and analyze looting and damage to archaeological sites in Syria as a direct consequence of the ongoing civil war relied on-high resolution satellite imagery. The Syrian site assessment effort was based in the years 2012 and 2013 using GeoEye and WorldView imagery of 30 key sites. Additional free imagery was obtained from Google Earth and Bing Maps. Anaysis was further expanded through an imagery grant from the DigitalGlobe Foundation. In an attempt to safely and accurately quantify the true scale of the damage and looting, archaeologist and analyst turned to high-quality satellite imagery. In april of 2012, Quickbird imagery of the Roman city of Apamea was posted on Google Earth showing the full extent of the looting damage to the site from the previous 8 months. Imagery revealed that between July 2011 and April 2012, Apamea was intensely looted showing a pockmarked landscape (Casana 2014). The image was picked up and distributed around the world by media. Methodology for the analysis by Casana and Panhipour used freely available Google Earth and Bing Maps imagery, alongside GeoEye-1, and Worldview-1 and 2 imagery provided by the DigitalGlobe Foundation. Comparison was made between pre-war images and the most recent images available high resolution satellite imagery has become a powerful tool and resource for monitoring looting and site destruction remotely and proves to be an independent way to cross check and analyze against varied and unreliable reports from media and government agencies (Casana 2014).

2.1.4 Earlier Years of Remote Sensing

Earlier years of remote sensing using satellite imagery for looting detection in Iraq date back to before and after 2003 war. 1900 sites were investigated for signs of looting. Limited data was developed from the comparison of imagery from before and after the 2003 invasion of Iraq. Areas of focus included archaeological sites of the Nippur area near Eridu and around Uruk. These sites were chosen based on the reports of significant looting in the area and that the area had been imaged at 60cm resolution by the DigitalGlobe Corporation (Stone 2008). The study of DigitalGlobe Imagery spans from 2002 to 2006. 9728km^2 of imagery was examined, 0.87 percent was occupied by archaeological sites. The total number of sites studied in the end totaled 1949 sites. Intense looting was found to be most common close to the boundaries between settled areas and the desert. Conclusions determined that site selection for looting is close enough to draw a workforce yet far enough away to go undisturbed or noticed (Stone2008). Of the 1949 sites examined only 743 can be seen in more than one image, of those 213 were looted representing 26 percent of all looting sites (Stone 2008). However, there are 348 pairs of images taken at different times at sites allowing for progress in looting to be determined from previous older activity. Of the 114 images that were imaged multiple times and imaged in 2003, 85 percent showed evidence of fresh looting (Stone 2008). The total area of intensive looting for this time period adds up to 15.75km^2 which is larger than all archaeological investigations ever conducted it Iraq at the time (Stone 2008). Spatial analysis of looting distribution suggested that many interesting sites at the time, were intact which were in areas that had not been as badly hit.

2.2 Machine Learning

Lack of manual performance motivates research into computer assisted methods, mostly from machine learning and homegrown algorithms. Common methods consist of Support Vector Machines (SVMs), a supervised non-parametric statistical learning technique and Principal Component Analysis (PCA)- PCA is a statistical method for analysis of multivariate datasets. Benefits of SVMs are their ability to successfully handle small training datasets often producing greater accuracy than traditional methods, it allows for more rapid identification of pits. However, there are drawbacks with the kernel function and choices resulting in overfitting and oversmoothing. SVMs can show poor performance with noisy data. Benefits of PCA with multispectral satellite imagery is its ability to recombine data collected on the reflectance of visible and non-visible spectra of light, highlighting patterns in the landscape allowing for the display of looting pits. PCA with Interactive supervised classification tool in ArcGIS can define training polygons on pits and isolates pixels corresponding to pits. PCA allows for a more expedient quantification of looting patterns than manual visual inspection. A drawback of PCA method is the cost and availability of multispectral imagery.

Looting pit identification history parallels general object recognition and detection history in manual and machine learning. Methods like SVM and custom expert crafted methods (SIFT/SURF) achieved reasonable but unsatisfactory performance, (see ImageNet challenge). Recent findings favor usage of CNNs to SIFT/SURF, SVMs and PCAs. CNN fine-tuning yields competitive accuracy on various retrieval tasks and has proven to have advantages in efficiency over SIFT/SURF, SVMs and PCAs.

Advent of deep convolutional neural networks and ongoing refinements have achieved an average of 96% and are now the dominant algorithmic approach for object recognition and detection tasks (Krizhevsky et al., 2012) (Russakovsky, 2015) (Hu et al., 2015), see ImageNet large scale visual recognition challenge for performance data.

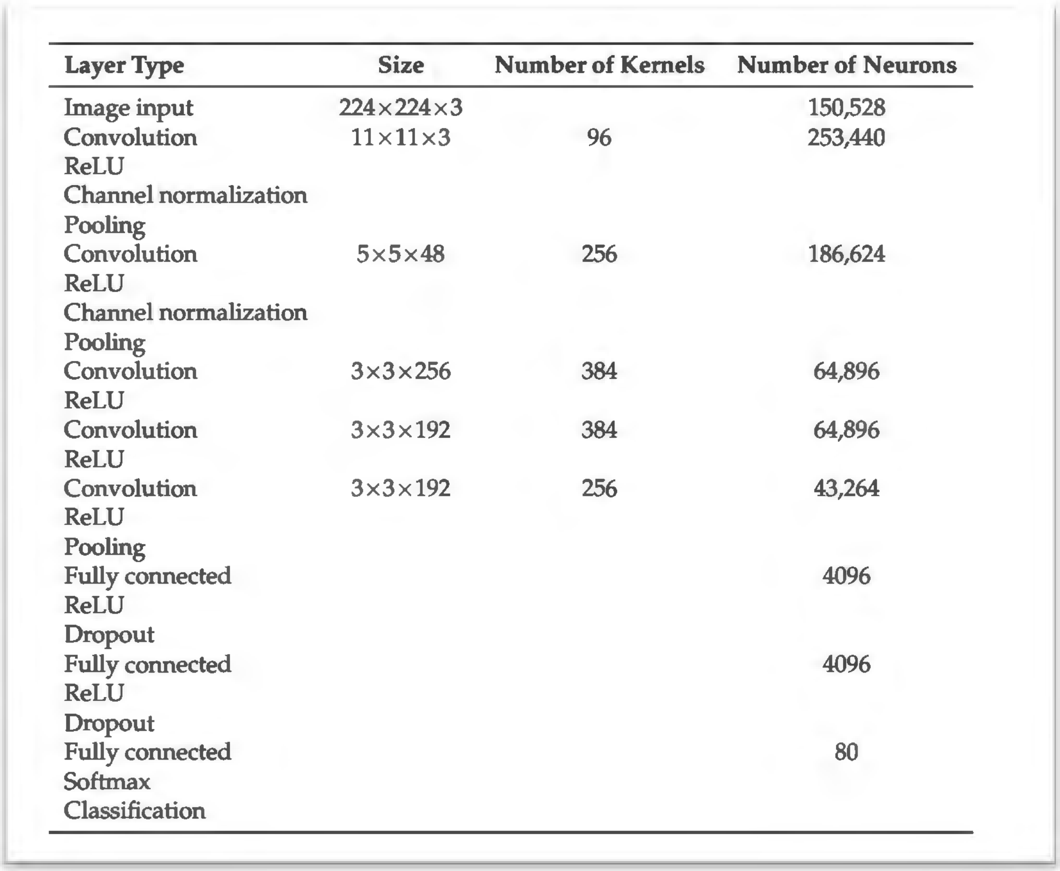
2.2.1 Convolutional neural networks

Deep learning is a new take on specific sub-field of machine learning. It consists of learning representations from data which puts an emphasis on learning successive layers of increasingly meaningful representations (Chollet 2017). The deep in deep learning refers to the layers of the model standing for the idea of successive layers of representations and how many layers there are that contribute to the model referring to the models depth. Deep neural networks map inputs (images) to targets (labels) via a deep sequence of data transformations (layers) and the transformation is learned by exposure to the examples (Chollet 2017). Deep convolutional neural networks (CNNs) architectures are typically comprised of several layers of various types that can be summarized into categories. (1) Convolutional layers compute the convolution of the input image with the weights of the network. Neurons in the first hidden layer only view a small image window and learn low-level features. Deeper layers view larger portions of the image and learn more expressive features by combining low-level features. Hyper-parameters characterize each layer with the number of filters to learn, spatial support, stride between different windows and zero padding which controls output layer size (Castelluccio et al., 2015). (2) Pooling layers are inserted in-between successive convolutional layers and progressively reduce the size of the input layer through local non-linear operations. They reduce the amount of parameters and computation in the network and also controlling overfitting (Castelluccio et al., 2015). (3) Normalization layers have the intentions of implementing inhibition schemes observed in the biological brain (Russakovsky et al., 2015). (4) Fully connected layers are used in the last few layers of the network. Neurons in a fully connected layer have full connections to all activations in the previous layer. Their activations can hence be computed with a matrix multiplication followed by bias offset ((Russakovsky et al., 2015). By removing the constraints, they can better summarize the information that is conveyed by the lower-level layers in view of final decisions (Castelluccio et al., 2015).

2.2.2 Deep CNN Architecture

AlexNet, developed by Alex Krizhevsky is a groundbreaking deep CNN architecture that consists of five convolutional layers with the first, second, and fifth of which are followed with pooling layers, and three fully-connected layers as displayed in Figure 2. AlexNet success is attributed to its use of Rectified Linear Units (ReLU) non-linearity, data augmentation, and dropouts (Hu et al., 2015) (Krizhevsky et al. 2014) table 1 below. ReLU is the half-wave rectifier function $f(x) = max(x,0)$ which can significantly accelerate the training phase. Data augmentation effectively reduces overfitting (the error in the training set is driven to a very small value, but when new data is presented to the network error is large). The network has memorized the training examples but it has not learned to generalize to new situations (Hu et al., 2015)). Augmentation reduces overfitting when training large CNNs, which generates more training image samples. Training image samples are created by cropping small-size patches and horizontally flipping these patches from original images (Hu et al., 2015).

Table 1: AlexNet architecture layers (Pedraza 2017)



Layer Type	Size	Number of Kernels	Number of Neurons
Image input	224×224×3		150,528
Convolution	11×11×3	96	253,440
ReLU			
Channel normalization			
Pooling			
Convolution	5×5×48	256	186,624
ReLU			
Channel normalization			
Pooling			
Convolution	3×3×256	384	64,896
ReLU			
Convolution	3×3×192	384	64,896
ReLU			
Convolution	3×3×192	256	43,264
ReLU			
Pooling			
Fully connected			4096
ReLU			
Dropout			
Fully connected			4096
ReLU			
Dropout			
Fully connected			80
Softmax			
Classification			

The dropout technique reduces the co-adaptation of neurons by randomly setting zeros to the output of each hidden neuron and is used in fully-connected layers to reduce substantial overfitting. AlexNet has become the baseline architecture for modern CNNs by popularizing the application of large CNNs in visual recognition tasks (Hu et al., 2015).

2.2.3 Caffe

CaffeNet is aPre-trained CNN Convolutional Architecture for Fast Feature Embedding *(CaffeNet)* also called Caffe. Caffe is a fully open-source deep learning framework that allows clear and easy implementations of deep architectures (Penatti 2015). For convolutional neural networks in particular, it is one of the most popular libraries for deep learning. It is developed by the Berkeley Vision and learning Center (BVLC) and community contributors. It provides possibly the fastest available implementations for effectively training and deploying general-purpose CNNs and other deep models (Hu et al., 2015). Caffe is implemented using C++ with support to CUDA a NVidia parallel programming based on graphics processing units (GPU). Caffe uses Protocol Buffer language, which makes it easier to create new architectures. Other functionalities of Caffe include; fine-tuning strategizing, layer visualization, and feature extraction (Penatti 2015). Caffe is very similar the CNN architecture AlexNet with the exception of a few small modifications. CaffeNet allows for training without data augmentation and it allows for the exchanging of the order of pooling and normalization layers (Hu et al., 2015).

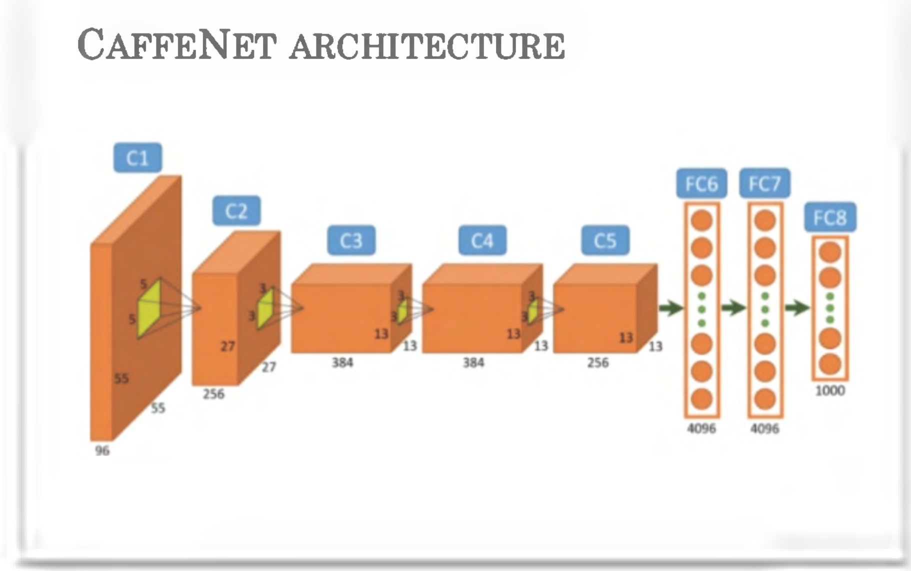


Figure 2: Image Credit Hu et al., 2015. Transferring Deep Convolutional Neural Networks for the Scene Classification of High-Resolution Remote Sensing Imagery. The Boxes show the size of each feature layer and for the fully connected layers, the size of the output.

2.2.4 VGG16

VGG16 was developed and trained by Oxford's Visual Geometry Group (VGG). It is a deep convolutional neural network for object recognition and has achieved very good performance on the ImageNet dataset securing first and second place in the ImageNet Challenge in 2014. Its convolutional layers use 3x3 size its Max pooling layers use 2x2 size and it has fully connected end layers with a total of 16 layers (VGG16). It has increased depth using architecture with very small 3x3 convolutional filters and has shown that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16 weight layers (Simonyan et. al 2015).

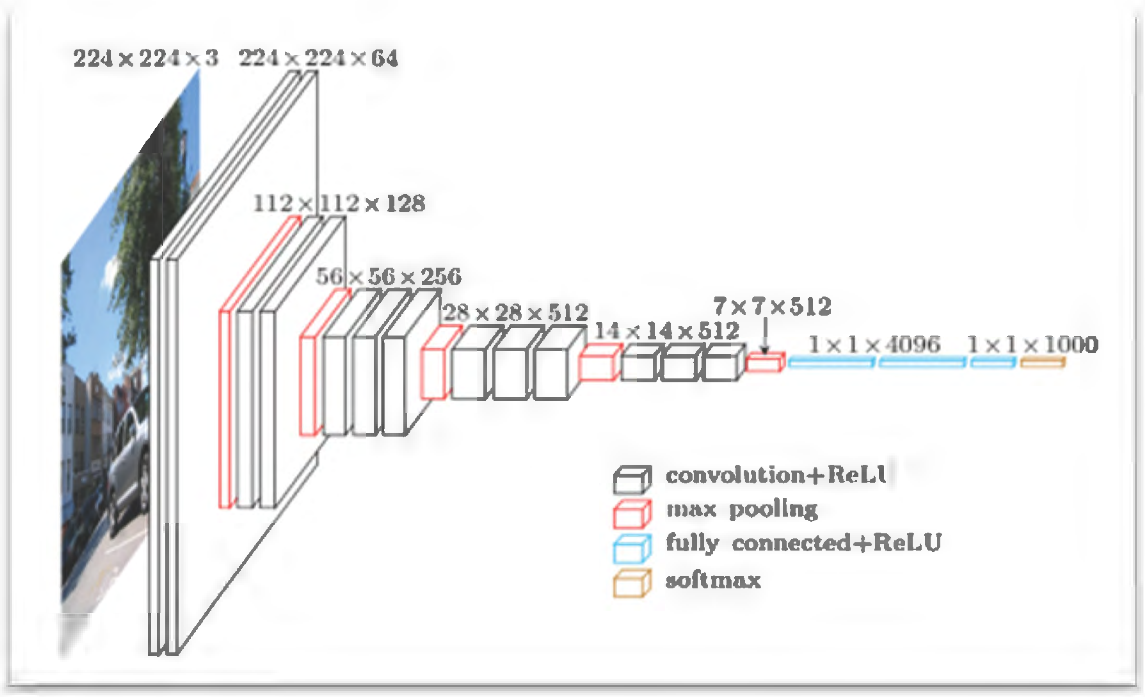


Figure 3: VGG116 Architecture

2.2.5 Feature Extraction in Deep Learning

The motivation behind the use of feature extraction for this study was based in how highly effective leveraging a pre-trained network has proven to be in deep learning while working with a small image dataset. Training deep learning networks from scratch takes large data and computational resources in order to obtain useful generalizable models (Rosebrock 2017). Deep convolutional neural networks applied to computer vision problems generate low-level object recognition features (e.g. edge detection, line and curve detection) that are common across image classification domains. By using a CNN previously trained on a large scale image recognition dataset, it’s possible to bootstrap the training of a new CNN classifier using layers of the previously trained network architecture and weights and with a new classifier and smaller dataset. This process is called transfer learning. In transfer learning the general features for image recognition (edges, etc) are transferred to the new network which is then optimized against the specific image data that this study attempts to classify (looting pits).

For looting pit image classification task feature extraction was used. The VGG16 architecture convolutional base of the network which is trained on ImageNet was used to extract interesting features from this study’s own pits versus non-pit images, and then trained a pits verses non-pit densely-connected classifier (with only two classes) on top of these features. VGG16 CNN was chosen as the starting point for transfer learning. VGG16 won ImageNet Large Scale Image Recognition Challenge (2012). Publically available weights from training against ImageNet dataset of 1.4 million images were used. The VGG16 architecture was modified by removing the final fully-connected classification layers and applying the ImageNet trained weights to the remaining lower layers. Note that the weights in these layers are not updated as part of the training for pit classification as they already generalize for low-level object recognition features. This study’s own fully-connected classifier was added fit to the recognition task (number and size of inputs, two classes in the classification problem). The weights in this network are the ones optimized by the gradient descent training process as described previously. During training, images pass through the VGG16 ImageNet weighted base network which finds general image features. The outputs of the VGG16 network feed into the fully-connected layers (created specifically for classifying the looting pits) which become optimized for looting pit image specific features over the course of training.

Feature extraction in deep learning differ from other methods. Features extracted by deep neural networks are the network weights learned over training that minimize the loss function. In other words, a feature in deep learning is a parameterization of the weights of the network. These weights correspond to the patterns in the image data that lead to correct classification of the image. While there are techniques for visualizing what the network is "seeing" in these features, the outputs are contrived in the sense that the feature is the parameterization of the network (the set of weights) while the visualization is a given data sample filtered through those weights which is perhaps representative, but not the same thing.

In contrast to deep learning methods, feature extraction in GIS, and across many domains in which statistical or machine learning methods are applied differ. Feature extraction is traditionally performed by experts that 'hand craft' what they (or the experts) believe to be most important to making a correct determination. These features tend to be human recognizable (e.g. nose and two eyes, or a circular sand berm). Part of what makes deep learning exciting and yet sometimes difficult for fields to adopt, is that the features that experts have long considered to be discriminatory in classification tasks are less so compared with the 'alien' features found in the data by neural networks as evidenced by the massive successes deep neural networks have achieved over traditional methods in recent years.

Chapter 3 Methodology

The method used for this study uses a supervised Deep Convolutional Neural Network (CNN) for object recognition and detection. CNNs are hierarchical architectures trained on large-scale data sets, which have recently shown astounding performance in object recognition and detection (Hu et al., 2015). The purpose of this analysis was to find a more expedient and accurate method with lower false-positive rates for looting detection using CNNs.

3.1 Deep convolutional neural network analysis

This section focuses on the processes that were used to create a dataset of looting pits that are representatives of the problem. These datasets were then applied to CNNs after for object recognition and classification. In order to perform the first task, looting pit characteristics needed to be defined. Looting pit selection and image acquisition choice is based on detailed descriptions by previous archaeological studies, descriptions, and findings. The main stages necessary to create a dataset of this kind are; image acquisition, looting pit selection, data labeling, image processing, and dataset building.

3.1.1 Data

Two datasets were used fir this study. The first dataset was gathered from Google Earth Pro (GEP). The image datasets consisted of the images of two classes; looting pits, and not pits, that were used to teach the machine learning classifier what the different categories looked like. The data set utilized satellite imagery from Google Earth Pro with a spatial resolution or ground sample distance (GSD) of 2.5 meters. Sets of publically-available satellite imagery from Google Earth Pro spanning from 2008 to 2018 were obtained of the sites al-Lisht, Dahshur, and Saqqara pyramid field regions of Egypt. Building upon a previous study from 2017, Algorithmic Identification of Looted Archaeological Sites from Space (Bowen et al, 2017), similar site locations and satellite images were selected. Sites were previously selected from regions with multiple instances of looted burial sites, substantial expanse of open desert, and multiple distractors such as graveyards, modern buildings, and farmlands (Bowen et al, 2017).

The second dataset consists of a set of high-resolution satellite imagery of the pyramid fields region in Egypt granted by the DigitalGlobe Foundation. Images were captured by the WorldView-3 satellite at a panchromatic mean ground sample distance (GSD) of 0.30 m per pixel. Collection Start = 2017-01-03T09:07:51.758306Z; Collection Stop = 2017-01-03T09:08:17.814179Z; Country Code = ""; Number Of Looks = "1"; Cloud Cover = 0.000; NW Lat = 30.01341060; NW Long = 31.11287091; SE Lat = 28.45609182; SE Long = 31.26501260. Once again, site selection was based on similar site locations. Sites were previously selected from regions with multiple instances of looted burial sites, substantial expanse of open desert, and multiple distractors.

3.1.2 Data Labeling

Once the GEP looting pits were selected and the images were resized to 500 x 500 pixels, over 300 images of looting pits were obtained. To use these images for training, they must be labeled. In this stage the pits are labeled with their grid number, their site location, and their latitude and longitude information for location. Additionally, this task consisted of indicating which class they belonged to; positive or negative and pits or non-pits.

Once the DigitalGlobe Foundation looting pits were selected and the images were resized to 112 x 112 pixels, over 300 images of looting pits were obtained. To use these images for training, they must be labeled. In this stage the pits are labeled with their grid location, site location, and their latitude and longitude information for location. Additionally, this task consisted of indicating which class they belonged to; positive or negative and pits or non-pits.

3.1.3 Dataset Building

After gathering and labeling, two primary datasets were available with 500 samples each. The image datasets consisted of the images of two classes; looting pits, and not pits, that were used to teach the machine learning classifier what the different categories looked like. The first dataset utilized satellite imagery from Google Earth Pro with a spatial resolution or ground sample distance (GSD) of 2.5 meters. Sets of publically-available satellite imagery from Google Earth Pro spanning from 2008 to 2018 were obtained of the site al-Lisht, Dahshur, and Saqqara pyramid field regions of Egypt. The second dataset consisted of DigitalGlobe Foundation WorldView-3 imagery from 2017 with a (GSD) of 0.30m (Figure 6).

3.2 Site Selection

Sites selected were al-Lisht, Dahshur, and Saqqara pyramid field regions of Egypt. Site selection of these regions were based on and compared with previously obtained and analyzed images of the selected archaeological site of these regions for algorithmic identification of looting in these same areas. These regions were selected for this study and by previous studies based on their multiple instances of looted burial sites, substantial expanses of open desert, and for multiple distractors such as farmland, non-archaeological structures, modern graveyards, and military bases with bomb craters (Bowen et al., 2017). For GEP imagery, a grid system was used, control points were selected and a jpeg image was downloaded at the highest resolution possible (4800x 3229 pixels), adjusted (contrast, brightness and color balance) for improvement of visibility of features, with the ability to be georeferenced in ArcGIS Pro (Figure 4). Provider date, catalog ID # of each image and images incorporated into GEP was identified and noted for metadata. Images of these sites were obtained and classified to create a dataset of a collection of images with each individual image labeled as a data point representing a looting pit, or not a looting pit. A uniform number of images of each category were selected. 300 Individual looting pit images were all labeled with site grid location (letter), site location name, location within that grid (upper left etc.), data point count number and data point longitude and latitude. 300 non-looting pit images were selected from the same regions (Figure 5). Both pit and non-pit images were then loaded and cropped to 500 x 500 pixels in the online photo editor Pixlr and filed in an image data tracker.

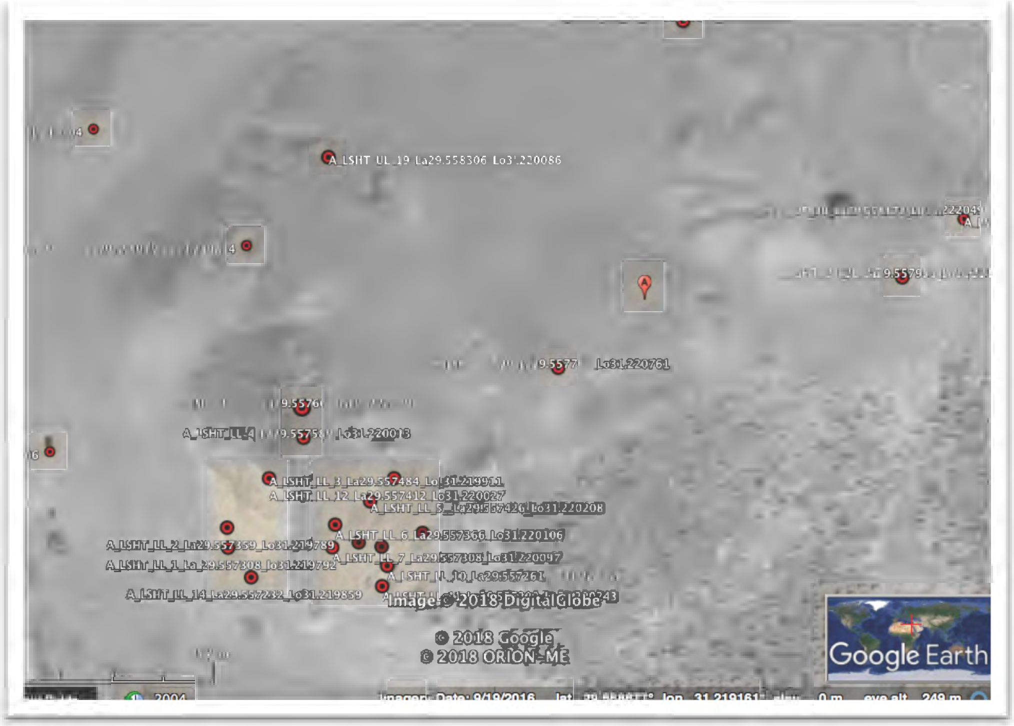

A_LSHT_UL_19_La29.558306_Lo31.220086
A_LSHT_LL_3_La29.557484_Lo31.219911
A_LSHT_LL_12_La29.557412_Lo31.220027
A_LSHT_LL_5_La29.557426_Lo31.220208
A_LSHT_LL_6_La29.557366_Lo31.220106
A_LSHT_LL_2_La29.557359_Lo31.219789
A_LSHT_LL_1_La29.557308_Lo31.219792
A_LSHT_LL_7_La29.557308_Lo31.220097
A_LSHT_LL_10_La29.557261
A_LSHT_LL_14_La29.557232_Lo31.219859
Image © 2018 DigitalGlobe
© 2018 Google
© 2018 ORION-ME
Google Earth

Figure 4: Labeling of pits Al-Lisht Google Earth Pro

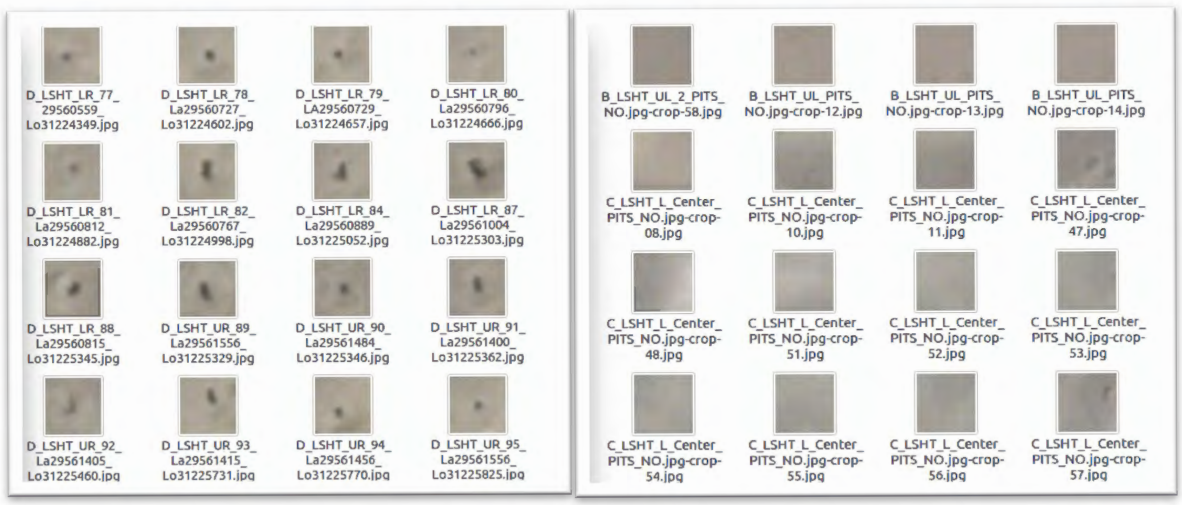

D_LSHT_LR_77_ 29560559_ Lo31224349.jpg
D_LSHT_LR_78_ La29560727_ Lo31224602.jpg
D_LSHT_LR_79_ LA29560729_ Lo31224657.jpg
D_LSHT_LR_80_ La29560796_ Lo31224666.jpg
D_LSHT_LR_81_ La29560812_ Lo31224882.jpg
D_LSHT_LR_82_ La29560767_ Lo31224998.jpg
D_LSHT_LR_84_ La29560889_ Lo31225052.jpg
D_LSHT_LR_87_ La29561004_ Lo31225303.jpg
D_LSHT_LR_88_ La29560815_ Lo31225345.jpg
D_LSHT_UR_89_ La29561556_ Lo31225329.jpg
D_LSHT_UR_90_ La29561484_ Lo31225346.jpg
D_LSHT_UR_91_ La29561400_ Lo31225362.jpg
D_LSHT_UR_92_ La29561405_ Lo31225460.jpg
D_LSHT_UR_93_ La29561415_ Lo31225731.jpg
D_LSHT_UR_94_ La29561456_ Lo31225770.jpg
D_LSHT_UR_95_ La29561556_ Lo31225825.jpg
B_LSHT_UL_2_PITS_ NO.jpg-crop-58.jpg
B_LSHT_UL_PITS_ NO.jpg-crop-12.jpg
B_LSHT_UL_PITS_ NO.jpg-crop-13.jpg
B_LSHT_UL_PITS_ NO.jpg-crop-14.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 08.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 10.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 11.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 47.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 48.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 51.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 52.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 53.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 54.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 55.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 56.jpg
C_LSHT_L_Center_ PITS_NO.jpg-crop- 57.jpg

Figure 5: Sample of images from dataset pits and non-pits (sand) Al-Lisht

Factors of variation that were considered through this process included viewpoint variation, scale, variation, deformation, occlusion, illumination, background clutter and intra-class variation. Looting pit data sets were curated with these in mind and based on previously selected looting locations. Any looting pit location of question was excluded from this study based on previously selected looting pit location. Looting pit selection, characteristics, and definition as well as image acquisition choices were based on detailed descriptions by previous archaeological studies and findings. Dataset samples were taken from each of the three pyramid field sites. Primary visual references for looting pit site selection include; Algorithmic Identification of Looted Archaeological Sites from Space (Bowen, et al. 2017) and Satellite Evidence of Archaeological Site Looting in Egypt 2002-2013 (Parcak et al. 2016).

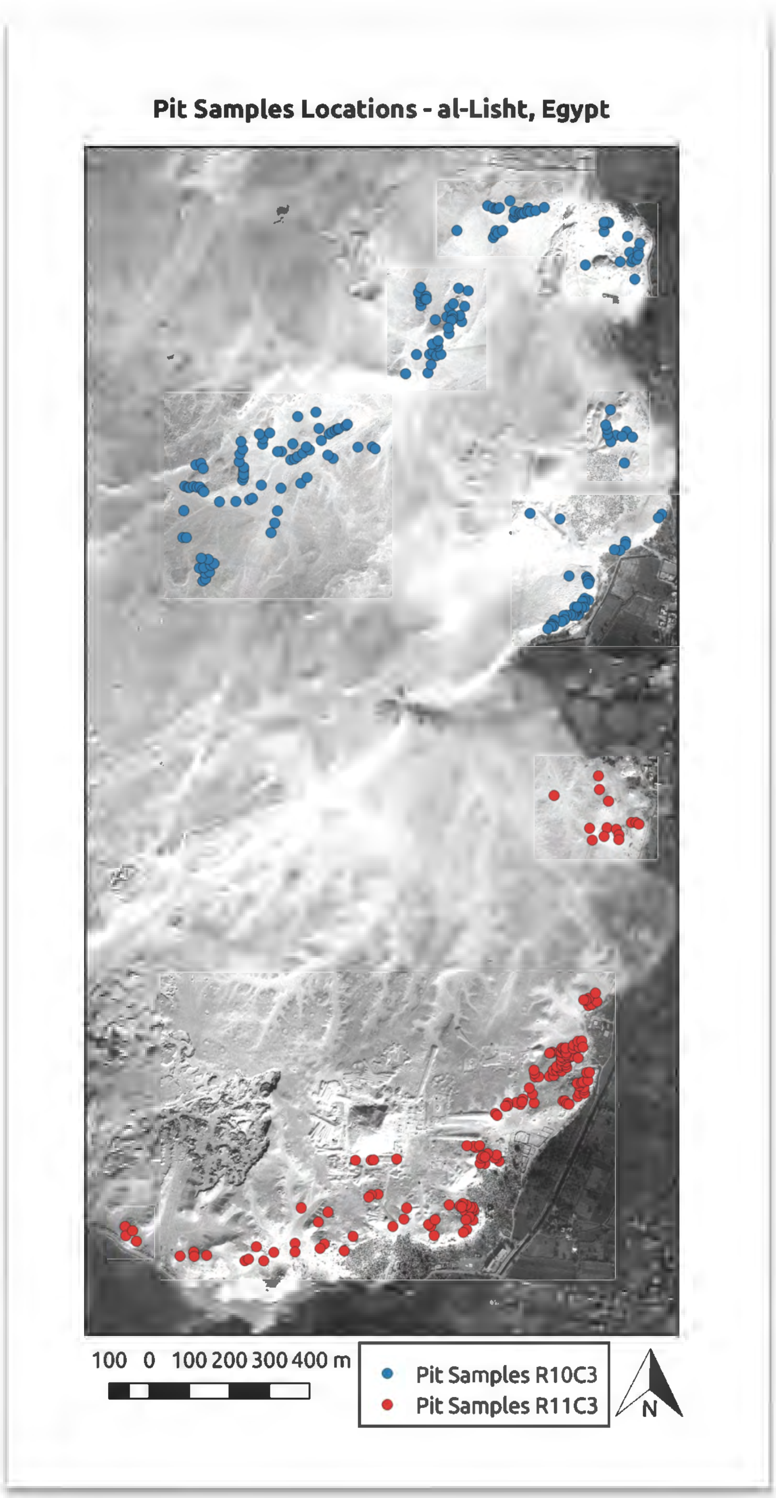


Figure 6: Image courtesy of DigitalGlobe Foundation WV3 2017 R10C3 and R11C3 al- Lisht

3.3 Machine Training

A common goal in machine learning is to achieve generalization. Models that generalize perform well on data that has never been seen before. The goal when training a machine learning model is to reduce the training loss as much as possible while ensuring that the gap between the training and testing loss is reasonably small (Challot 2017). In order for the neural network to learn, we have to find the combination of model parameters that minimize the loss function for the given set of training data samples (pits vs non-pits) and their corresponding targets (Rosebrock 2017). The loss function is used to guide the training process of a neural network. It informs us what is happening internally inside the network. Ideally, training loss and validation loss should be close and should get smaller and smaller. However, a loss of zero leads to overfitting and is not desired (Ng 2016). Overfitting is the error in the training set is driven to a very small value, but when new data is presented to the network error is large. The network has memorized the training examples but it has not learned to generalize to new situations (Hu et al., 2015).

After building the dataset and considering the image classification problem that needs to be solved, selection of the loss function, the network architecture, and optimization method used to train the model needs to be made. For this study the use of the neural network is for image classification. Keeping the loss function selection in line with the nature of the task at hand, and following nearly all deep learning image classification problems, cross-entropy loss is selected. As this project is a two class project binary cross-entropy is used. Architecture selection for this study was done on deep convolutional neural network VGG16 architecture developed by Karen Simonyan and Andrew Zisserman for the ImageNet large-scale object recognition challenge which achieved new benchmarks in 2014 outperforming other methods and neural networks

(Simonyan et al., 2015). For feature extraction, VGG16 architecture was used and used publicly available weights from training against the Imagenet dataset. Optimization method selection is RMSprop. Additionally, the dropout technique was used and activation consisted of using Rectified Linear Unit (ReLU). The dropout technique reduces the co-adaptation of neurons by randomly setting zeros to the output of each hidden neuron and is used in fully-connected layers to reduce substantial overfitting. Rectified Linear Unit (ReLU) is the half-wave rectifier function $f(x) = max\,(x,0)$ which can significantly accelerate the training phase. Activation is thresholded at zero.

3.3.1 Binary Cross-Entropy and RMSprop Optomizer for Gradient Descent

For this neural network, binary cross-entropy was used for a two-class classification problem; pits or non-pits. The loss function quantified the quality of any particular set of weight values for the weights in the network. It is differentiable which is what allowed for taking its gradient. The gradient showed how steep the function was at that time or how steeply the loss function was moving. The desire at that point, is to update the weights in the opposite direction in order to flatten the gradient.

The optimization strategy gradient decent is used for taking the gradient from steep to flat. It is the rule used to determine how much the weights (usually on an individual basis) are updated relative to the current gradient. RMSprop was used as the gradient decent optimizing algorithm. RMSprop uses the magnitude of the recent gradient to normalize the gradient calculations. It attempts to iteratively find weights that are slightly better at reducing the loss (given by the binary-cross-entropy loss function) than in the previous iteration. It is a process of updating parameters of the function such that the gradient gets shallower and shallower until it is basically flat. In turn, the loss function became smaller and smaller.

3.3.2 Backpropagation

Backpropagation starts after the forward pass where inputs are passed through the network and output predictions are obtained. Once final loss value is obtained, it works backwards from the top layers to the bottom layers, applying the chain rule to compute the contribution that each parameter had in the loss value (Chollet 2017). This is the backward pass where we compute the gradient of the loss function (which includes the loss function and optimizer) at the final layer of the network and use the gradient to recursively apply the calculus 'chain rule' to update the weights layer by layer in our network (Chollet 2017). This is also referred to as the weight update phase (Rosebrock 2017). Weights are adjusted layer by layer because changing weight values in the reverse direction has a cascading effect on weights in layers closer to the beginning of the network.

Feature extraction consisted of taking the convolutional base of the previously-trained network, running the new data through it, and training a new data classifier on top of the output. The convolutional base was run over the dataset, outputting a network feature array. It then used that data as an input to a standalone densely-connected classifier (Challot 2017). Once features were extracted from these images they were then flattened to be fed to the densely-connected classifier. The densely-connected classifier was defined and used the dropout method for regularization and was trained on the data and labels that were just recorded. Training happens through a forward pass, calculating loss. Once loss is calculated the weights are updated through backpropagation.

Chapter 4 Results

Looting of ancient archaeological sites continues to rise globally. Artifacts are stolen, lost to the black-market, or sometimes destroyed. These pieces of history are often lost to mankind forever. The goal of this study was to demonstrate the viability of using deep convolutional neural networks (CNNs) within the field of archaeology and cultural heritage preservation with the purpose of augmenting or replacing the manual detection of looting. It brings recent advancements from the field of Artificial Intelligence to an applied GIS challenge at the intersection of remote sensing and archaeology. Motivation is based in the need for advancements in the field of cultural heritage preservation with the purpose of expediting and aiding in the protection for ancient archaeological sites. Training is a transfer learning technique that employs fine-tuning of previously trained deep convolutional neural networks that have learned general features such as CaffeNet, AlexNet, or VGG16. The fine-tuned CNN is applied to our dataset and adapts the network weights for the problem at hand. Using VGG16 it was possible to achieve good results with 90-96% accuracy with only a few iterations.

4.1 GEP Findings

Ran on two GPU: NVIDIA GeForce GTX 1080 Ti, feature extraction was done on VGG16 architecture using publicly available weights for training against the ImageNet dataset. Feature extraction consisted of taking the convolutional base of the previously-trained network, running the new data through it, and training a new data classifier on top of the output with a batch size of 20 for 30 epochs.

Table 2: GEP Training, Testing, and Validation Loss and Accuracy Results

Training	Validation	Test
Training loss: 0.3871	Validation loss: 0.4158	Test loss is 0.4187
Training accuracy: 0.9150	Validation accuracy: 0.9600	Test accuracy is 0.9600

4.1.1 Loss

The loss function quantifies how well, or how bad the given predictor is at classifying the input data points in the dataset. As stated earlier, smaller the loss the better the classifier is doing at modeling the relationship between the input data and output class labels (Rosebrock 2017). Loss is cumulative per epoch. At the beginning of each epoch, loss is zero. For each calculation of the loss (given by the loss function) the loss is added to the loss metric on the graph. What is seen over time in the plotted results in Figure: 7 below, is the total loss of training and validation decreasing (generally) which means the weights of the network (in this case, the fully connected classifier head on the convnet) are getting more accurate. However, when reading such a low loss of near zero, it is evident that because the weights are near perfectly tuned to the training and validation data, it has crossed the point where it has overfit the model which leads our model to have problems generalizing.

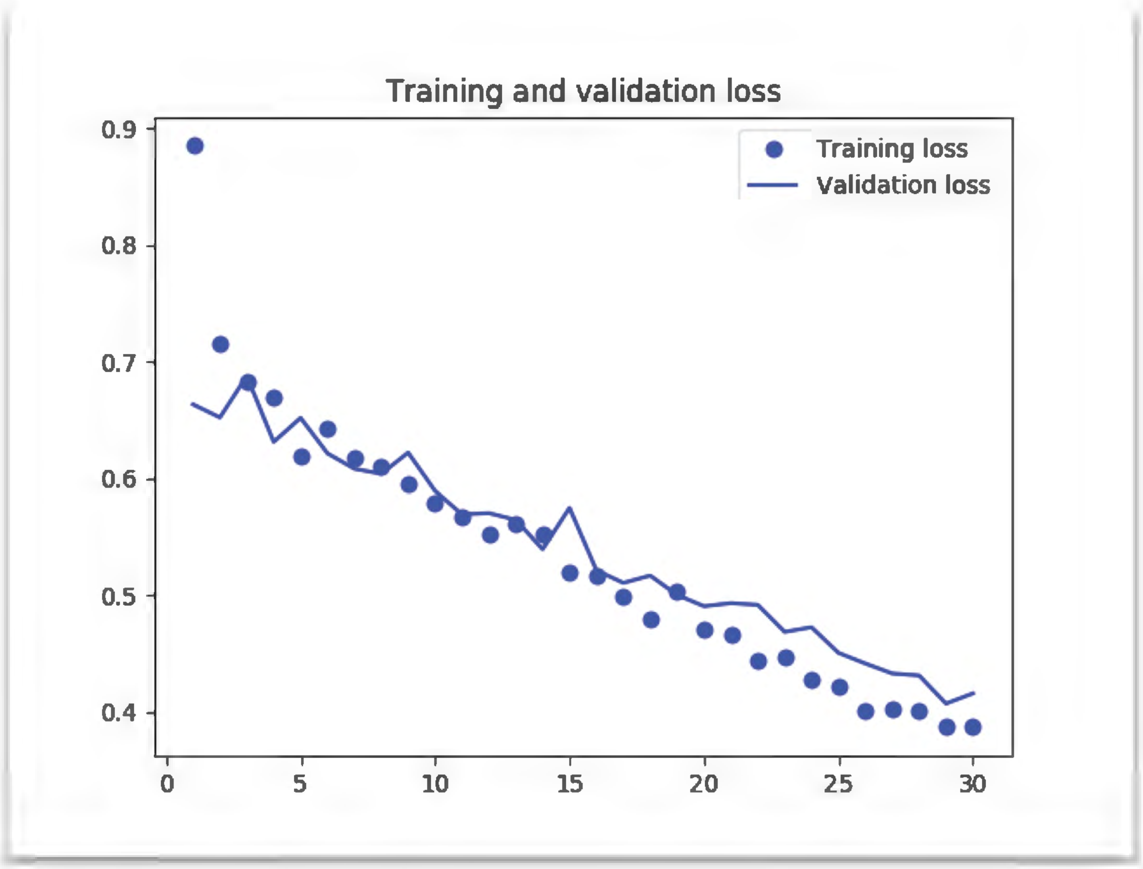


Figure 7: Plotted GEP Training Loss and Validation Loss

4.1.2 Accuracy

In this gradient decent optimization, training accuracy increases with every epoch (Figure 8), however a model that performs well on training data may not necessarily do well on data it has never seen before (generalization). Training accuracy reaches 92%, validation accuracy reaches 96%, and test accuracy reaches 96%.

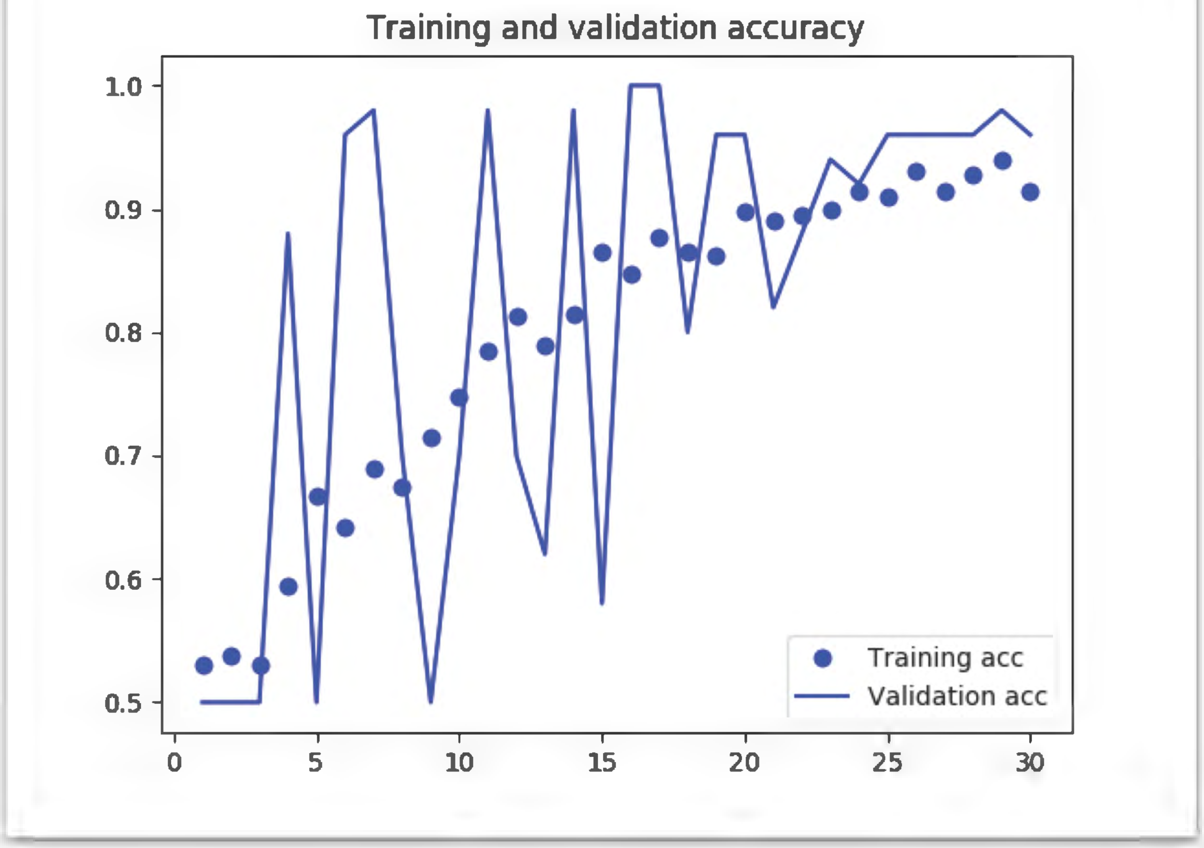


Figure 8: Plotted GEP Training and Validation Accuracy

4.1.3 Evaluating Error, Bias, and Variance

Bias relates to the ability of your model function to approximate the data, so *high bias* is related to under-fitting (Ng 2011). Low bias in turn relates to your models ability to differentiate well. A high bias percentile relates to misclassification. Variance is about the stability of your models response to new training examples. *High variance* relates to overfitting (Ng 2011). Measuring the error rate human level performance on looting pit classification has yet to be quantified. For the sake of analysis, a conservative line was taken, assuming a baseline of 1% error rate by a team of experts.

Table 3: GEP Measure of Training Error and Val Error for Calculating Bias and Variance

Human Error	1%	Human Error – Train Error = 7%	High Bias
Training Error	8%	Training Error – Validation Error = 4%	High Variance
Validation Error	4%	Validation Error – Test Error = 0	Overfitting of Validation
Test Error	4%	Test Error – Validation Error = 0	Overfitting of Validation

By looking at the algorithm error on the test set and the algorithm error on the validation set, *high bias,* or *high variance* can be diagnosed. In this case, there is both high bias and high variance. Some of the high bias relates directly to the ImageNet weights because they were not optimized for the classes of images in this projects dataset. There is high bias because it is a mostly linear classifier, but it also has high variance as well because of too much flexibility.

4.2 DGF Findings

Ran on two GPU: NVIDIA GeForce GTX 1080 Ti, feature extraction was done on VGG16 architecture using publicly available weights for training against the ImageNet dataset. Feature extraction consisted of taking the convolutional base of the previously-trained network, running the new data through it, and training a new data classifier on top of the output with a batch size of 20 for 30 epochs. One epoch is equal to one forward pass and one backward pass of all the training examples. Batch size is equal to the number of training examples in one forward /backward pass. Results:

Table 4: DGF Training, Testing, and Validation Loss and Accuracy Results

Training	Validation	Test
Training loss: 0.3228	Validation loss: 0.2895	Test loss is 0.3200
Training accuracy: 0.9275	Validation accuracy: 0.9200	Test accuracy is 0.9000

Table 5: DGF Measure of Training Error and Val Error for Calculating Bias and Variance

Human Error	1%	Human Error – Train Error = 6%	High Bias
Training Error	7%	Training Error – Validation Error = 1%	Low Variance
Validation Error	8%	Validation Error – Test Error = 2%	Some Overfitting of Validation
Test Error	10%	Test Error – Validation Error = 2%	Some Overfitting of Validation

4.2.1 Evaluating Error, Bias, and Variance

By looking at the algorithm error on the test set and the algorithm error on the validation set, *high bias,* or *high variance* can be diagnosed. In this case, there is high bias and low variance. Some of the high bias relates directly to the ImageNet weights because they were not optimized for the classes of images in this projects dataset. As bias is a measure of how well your model fits your data and how well it differentiates your data set, the lower the bias the better job it is doing at differentiating. As seen with DGF data, with high bias there is misclassification.

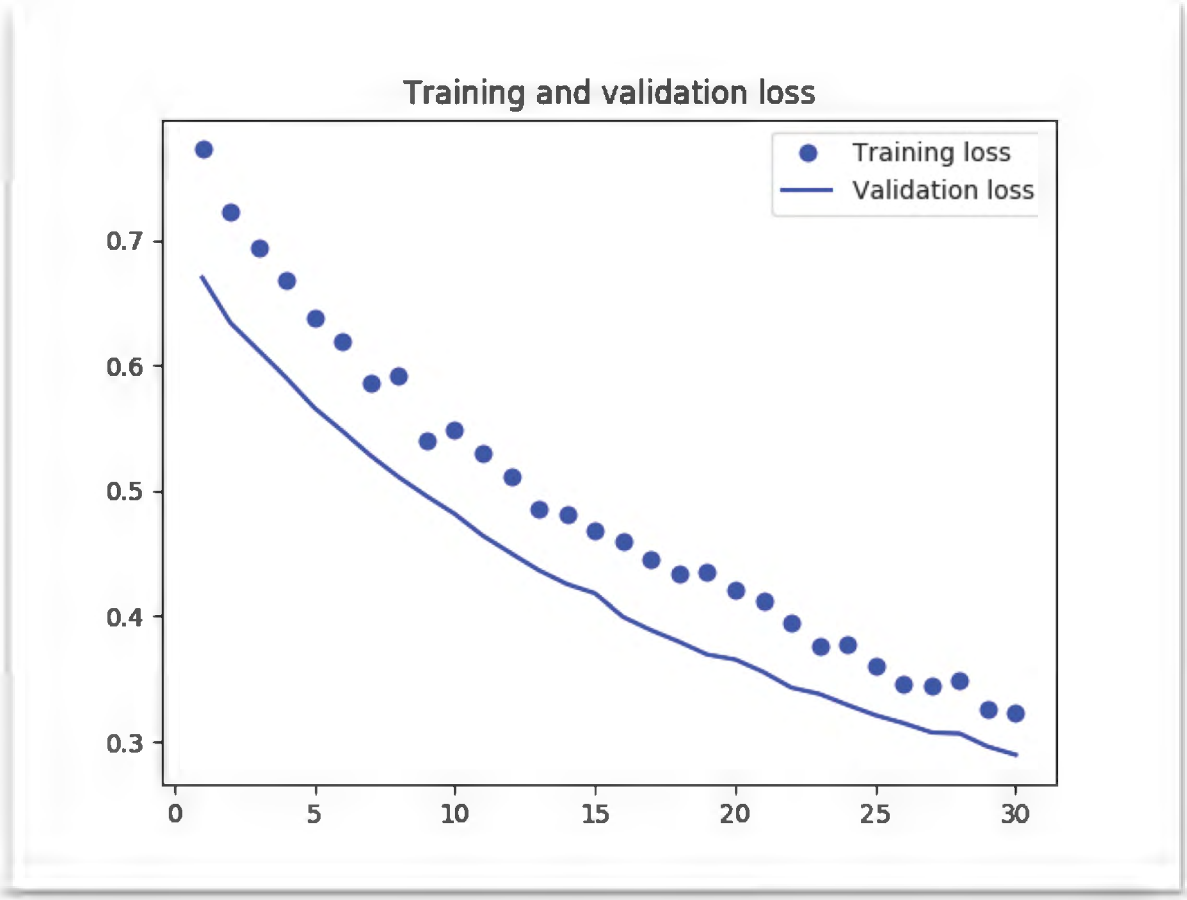


Figure 9: Plotted DGF Training Loss and Validation Loss

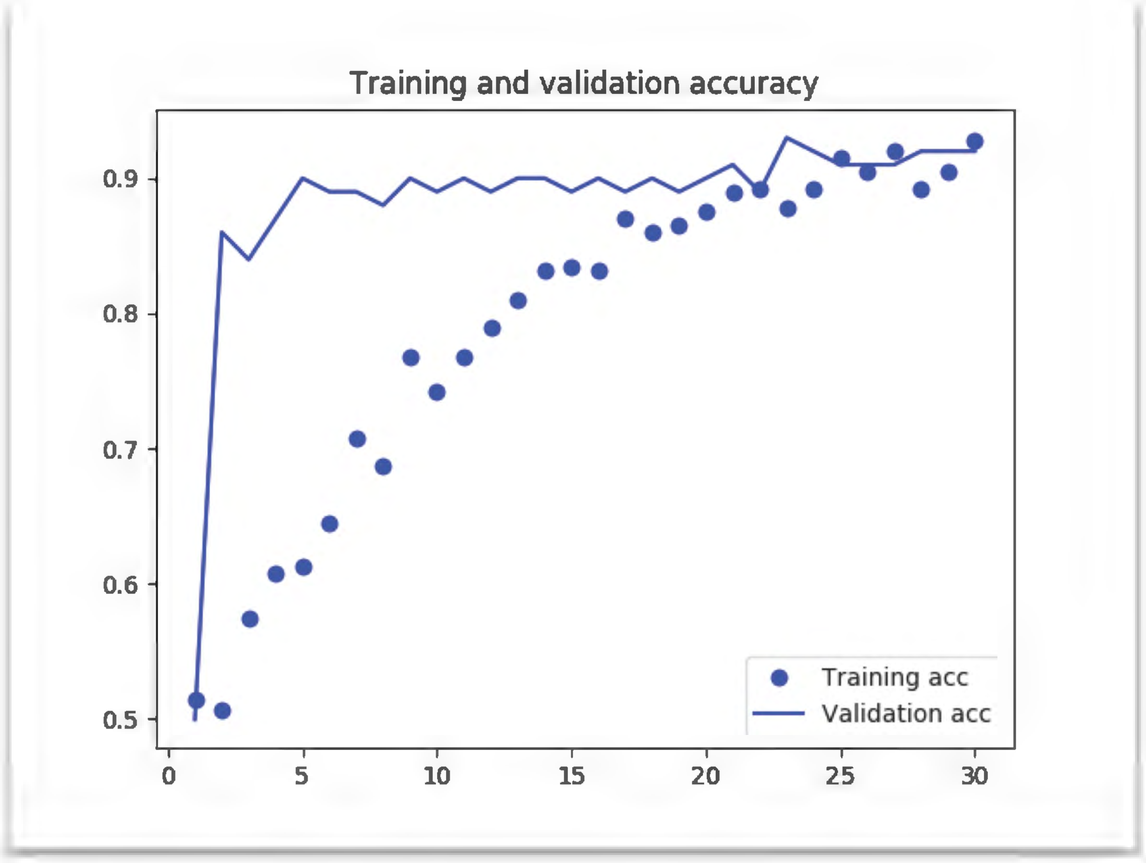


Figure 10: Plotted DGF Training and Validation Accuracy

4.2.2 DGF Accuracy

In this gradient decent optimization (Figure 10), training accuracy increases with every epoch, however a model that performs well on training data may not necessarily do well on data it has never seen before (generalization). Training accuracy reaches 93%, validation accuracy reaches 92%, and test accuracy reaches 90%.

4.2.3 DGF Loss

The loss function quantifies how well, or how bad the given predictor is at classifying the input data points in the dataset. The smaller the loss the better the classifier is doing at modeling the relationship between the input data and output class labels (Rosebrock 2017). What is seen is the total loss of training and validation decreasing epoch by epoch (figure 9) which means the weights of the network (in this case, the fully connected classifier head on the convnet) are getting more accurate. However, when reading such a low loss of near zero, it is evident that because the weights are tuned to the training and validation data, it has crossed the point where it has overfit the model which leads our model to have problems generalizing.

This study’s solution is to assist in looting prevention and detection by showing that CNNs are a highly accurate and expedient method for machine learning and detecting of looting with wide-ranging application beyond this specific research. This study applied machine learning and deep learning algorithms to discover underlying patterns to its looting pit dataset which enabled the correct classification of data points (pits) that the algorithm had yet to encounter. The image datasets consisted of the images of two classes; looting pits, and not-pits, that were used to teach the machine learning classifier what the different categories looked like with near human accuracy but with a fraction of the time expenditure.

Chapter 5: Conclusion

This study utilized a pre-trained deep convolutional neural network VGG16 architecture for feature extraction using publicly available weights for training against the ImageNet dataset. Feature extraction consisted of taking the convolutional base of the previously-trained network, running the new data through it, and training a new data classifier on top of the output. The convolutional base was run over the dataset, recording its output to a numpy array. It then used that data as an input to a standalone densely-connected classifier. The workflow of object recognition and image classification process with a deep convolutional neural network for looting pit recognition was covered and applied and yielded positive results. The applied methodology has shown that CNNs can be highly efficient in expediting image recognition of looting pits obtaining an overall accuracy of 96% with the GEP dataset and 90% accuracy with the DGF dataset. Thoughts behind accuracy variation of the two datasets accuracy rating are possibly due to image quality. The non-pit images in the DGF imagery are of a much higher spatial resolution (0.30m) than that of the GEP images (2.5m) leading to higher detail quality and a more varied non-pit dataset leading to a more complicated differentiation. GEP datasets training and validation errors show high bias and high variance, while DGF dataset shows high bias with low variance.

Convolutional neural networks are proving to be the best type of machine learning models when faced with an image recognition task yielding higher and more accurate results than other previously used methods. They are particularly favorable when working with small datasets and are able to still yield decent to positive results. However, one of the drawbacks with small datasets consist of overfitting, but this can be mitigated with data augmentation.

Future work to improve the shortcomings in performance of this CNN on this dataset should start with following Ng's recommendation for deep learning workflow as outlined in his Nuts and Bolts of Applying Deep Learning talk from 2016, fine-tuning, and lastly visualizing what the convnet learned.

Ng suggests starting by addressing high training error with high bias with the solution of possibly training on a bigger model, training longer, ensuring optimization is working, or moving to new model architecture. Once bias and training error is low, the next step is to address high validation error with high variance. Ng suggests possible solutions for this include obtaining more data, working on regularization, data augmentation or more data similar to test data or new model architecture. Once high validation error with high variance is low, if test set data error is high get more data, or possible new model architecture (Ng 2016).

After addressing variance and bias, the next recommendation would be fine-tuning. Fine-tuning unfreezes a few of the top frozen layers of a frozen model base used for feature extraction and together trains the newly added fully-connected classifier part of the model and those top layers allowing for slight adjustments to the more abstract representations of the model that are being reused which makes them more relevant to the task at hand. Fine-tuning adapts to a new problem some of the representations previously learned by an existing model which helps the model to perform better (Chollet 2017).

For a final recommendation, visualizing the intermediate activation outputs that the convolutional neural network learned is useful in understanding how successive layers transform their input in order to have a better understanding of the meaning of individual filters. Deep learning models are often referred to as difficult to extract and present in a human readable form but it is possible to do so with convolutional neural networks. Visualizing the networks filters helps in understanding exactly what visual pattern concept each filter in the network is receptive to. Heatmaps of class activation in an image can also be visualized and aid in the understanding of which part of an image were identified as belonging to an individual class and in doing so allows to localize objects in images.

Revolutionary recent advancements in the machine learning subfield of deep learning have shown remarkable results in near-human level image classifications as well as in all perceptual problems. Deep learning and AI’s long-term picture is looking bright as it is starting to be applied to many problems in which it has the possible capability to be transformative to many fields. Archaeology is one of them. The use of deep convolutional neural networks is applicable and will prove to be a valuable tool for archaeological research and site preservation. Within the field of archaeology efforts are currently being made to use GIS, remote sensing, and algorithmic identification techniques to mitigate against looted archaeological sites. Building upon these efforts, this study acquired high-resolution satellite imagery, built its datasets, and was able to achieve its goal of using a deep convolutional neural network to obtain near-human level accuracy (90% and 96%) of looting pit detection expediting the time and the process of detection with the potential of broader future applications. The work done for this study using deep convolutional neural networks for looting pit detection is an early example of what will one day be dominant research modality.

References

Antiquities Coalition , 2016. Maps. https://theantiquitiescoalition.org/resources/maps/ (Accessed November 23, 2016).

Atwood, R. 2004. *Stealing History: Tomb Raiders, Smugglers, and the Looting of the ancient world*. New York: St. Martin's.

Brodie, N. & D. Contreras 2012. Lazrus & A.W. Barker (ed.), "*All the Kings Horses: Looting, antiquities trafficking and the integrity of the archaeological record*": 9-24. Washington, D.C.: Society for American Archaeology.

Brodie, N. & C. Renfrew, 2005. "*Looting the Worlds Archaeological Heritage: the inadequate response*". Annual review of Anthropology 34:343-361 http://dx.doi.org.libproxy1.usc.edu/10.1146/annurev.anthro34.081.084.120551

Brodie, N. & K. Tubb, 2001. "*Illicit Antiquities: The Theft of Culture and the Extinction of Archaeology*". London: Routledge.

Bowen, E., et al., 2017. *"Algorithmic Identification of Looted Archaeological Sites from Space"* *https://www.frontiersin.org/articles/10.3389/fict.2017.00004/full* (accessed 15 January 2018).

CAPMAS/CAPMASTAT 1.0: DevInfo adaptation released by Population Statistics and Censuses Sector affiliated to the Central Agency for Public Mobilization & Statistics (CAPMAS). http://www.devinfo.org/capmas/libraries/aspx/Catalog.aspx (accessed 26 October 2016).

Casana, J. & N. Panahipour 2014. "*Notes on a Disappearing Past: Satellite- based monitoring of looting and damaged to archaeological sites in Syria*".

Castelluccio,M. G. Poggi, C. Sansone, L. Verdoliva, 2015. "*Land Use Classification in Remote Sensing Images by Convolutional Neural Networks*" *https://arxiv.org/pdf/1508.00092.pdf* (accessed 9 November 2017).

Central Bank of Egypt, (2014). Economic research. Time series. Available at: http://www.cbe.org.eg/english/economic+research/time+series/ (accessed 31 October, 2016).

Chollet, F. 2017. *"Deep Learning with Python" Manning Publications. www. Manning.com* (accessed 28 April 2018)

Coluzzi, R. et al., 2010. *"Satellite Imagery Time Series for the Detection of Looting Activities at Archaeological Sites"*. CNR-IMAA. EGU General Assembly 2010: 10569.

Contreras, D.& Brodie, N. 2010. *"The Utility of Publicly Available Satellite Imagery for Investigating Looting of Archaeological Sites in Jordan"*. Journal of Field Archaeology 35: 101-14. http://dx.doi.org.libproxy1.usc.edu/10.1179/009346910X12707320296838

Crossona, A. 2012. *"Egypt Looters Ransack Archaeological Sites"*. The World, TI March 2012. Available at: http://www.theworld.org/2012/03/egypt-looters-ransack-sites/(accessed 30 October 2016).

El Dorry, M.-A. 2011. *"Why Do People Loot? The Case of the Egyptian Revolution"*. al-Rawi Egypt's Heritage Review 2011: 20-28.

Reske Henry, J. 2012. *"Egypt's Poverty, Unemployment, Push Youths to Breaking Point"*. Retrieved 10 February 2012. http://www.newsmax.com/Newsfront/Egypt-poverty-unemployment-unrest/2011/01/31/id/384555/.

Hanna, M. 2013. *"What has Happened to Egyptian Heritage After the 2011 Unfinished Revolution?"* Journal of Eastern Mediterranean Archaeology & Heritage Studies 1: 371-75. http://dx.doi.org.libproxy1.usc.edu/10.5325/jeasmedarcherstu.1.4.0371

Hu, Fan, Xia, Gui-Song; Hu, Jingwen Zhang, Liangpei.2015. "*Transferring Deep Convolutional Neural Networks for the Scene Classification of High-Resolution Remote Sensing Imagery"*Remote Sensing; Basel Vol. 7, Iss. 11, (2015): 14680-14707. https://search-proquest-com.libproxy1.usc.edu/docview/1748566503?pq-origsite=summon&accountid=14749(accessed 9 November 2017)

Jemahs 2013. "*Cultural Heritage in Times of Crisis"*. Journal of Eastern Mediterranean Archaeology Heritage Studies 1: 366-71.

Krizhevsky A. et al., 2014. *"ImageNet Classification with Deep Convolutional Neural Networks*" https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (accessed 20 February 2018)

Ikram, S. et al., 2013. *"Looting and land grabbing: the current situation in Egypt"*. Bulletin of the American Research Center in Egypt 202: 34-39.

Lauricella, A. et al., 2017 *"Semi-automated detection of looting in Afghanistan using multispectral imagery and principal component analysis"* Antiquity Volume 91, Issue 359. https://www-cambridge-org.libproxy1.usc.edu/core/journals/antiquity/article/semiautomated-detection-of-looting-in-afghanistan-using-multispectral-imagery-and-principal-component-analysis/A649663703443DBAB767D434587BBB1C (Accesed 10 November 2017).

Muller,T. 2016. National Geographic: *"Plundering the past"*. (accessed 25 October 2016) http://www.nationalgeographic.com/magazine/2016/06/looting-ancient-blood-antiquities/

Ng, A., 2016, *"Nuts and Bolts of Applying Deep Learning"* *https://www.youtube.com/watch?v=F1ka6a13S9I* (accessed 20 April 2018).

Parcak, S. 2007. *"Going, Going, Gone: towards a satellite remote-sensing methodology for monitoring archaeological tell sites under threat in the Middle East"*. Journal of Field Archaeology 42: 61-83.

Parcak, S. 2009. *"Satellite Remote Sensing for Archaeology"*. New York: Routledge.

Parcak, S. 2016. *"Satellite evidence of Archaeological site looting in Egypt: 2002-2013"*. Antiquity Volume 90 issue 394, pp. 188-205. (accessed 30 October 2016). https://www.cambridge.org/core/journals/antiquity/article/satellite-evidence-of-archaeological-site-looting-in-egypt-20022013/D23EA939FC4767D8BF50CAC6DE96D005

Pedraza A., Gloria Bueno, Oscar Deniz, Gabriel Cristóbal , Saúl Blanco,and María Borrego-Ramos 3 2017. *"Automated Diatom Classification (Part B): A Deep Learning Approach"*VISILAB Research Group, University Castilla La Mancha, Av. Camilo José Cela s/n,Ciudad Real 13071, Spain. http://digital.csic.es/handle/10261/149382 (accessed 11 November 2017).

Penatti , A. 2015. *"Do Deep Features Generalize from Everyday Objects to Remote Sensing and Aerial Scenes Domains?"*Computer Vision Foundation *https://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W13/papers/Penatti_Do_Deep_Features_2015_CVPR_paper.pdf* (accessed 9 November 2017).

Rosebrock, A. 2017. *"Deep Learning for Computer Vision with Python"* Pyimagesearch *https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/* (last acceded 20 May 2018).

Russakovsky et al., 2015 *"ImageNet Large Scale Visual Recognition Challenge"* *https://link.springer.com/article/10.1007/s11263-015-0816-y* (accessed 12 February 2018)

Simonyan, K. & Zisserman 2015. *"Very Deep Convolutional networks for Large-Scale Image Recognition"* *https://arxiv.org/abs/1409.1556* (accessed 2 January 2018)

Stone, Elizabeth C. 2008. *"Patterns of Looting in Iraq"*, Antiquity 82, 125-138. (accessed 17 September 2017) http://www.academia.edu/4764494/Elizabeth_C._Stone_2008_Patterns_of_Looting_in_Iraq_Antiquity_82_125-138

United Nations Statistics Division: UN Data: Egypt. (accessed 28 October 2016). http://data.un.org/CountryProfile.aspx?crName=egypt

World Bank Database. 2014. http://www.worldbank.org/en/country/egypt (accessed 31 October 2016).

The World Bank, (2013) Egyptian Overview. .http://www.worldbank.org/en/country/egypt/overview

Appendix A: GEP Code

```
import os

import matplotlib.pyplot as plt

import numpy as np

from keras.preprocessing.image import ImageDataGenerator

from keras.applications import VGG16

from keras import models

from keras import layers

from keras import optimizers


print("Starting image classification process...")

conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(500, 500, 3))

print("VGG16 base summary: " + str(conv_base.summary()))

print("Starting training data augmentation")

print("Starting feature extration process...")


base_dir = '/home/jw/Timber_Thesis/pits_v_sand4'

print("Using base directory for training, validation and test images: " + base_dir)


train_dir = os.path.join(base_dir, 'train')

validation_dir = os.path.join(base_dir, 'validation')

test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)

batch_size = 20


def extract_features(directory, sample_count):

        features = np.zeros(shape=(sample_count, 15, 15, 512))

        labels = np.zeros(shape=(sample_count))
        generator = datagen.flow_from_directory(
                directory,
                target_size=(500, 500),
                batch_size=batch_size,
                class_mode='binary')
        i = 0
        for inputs_batch, labels_batch in generator:
                features_batch = conv_base.predict(inputs_batch)
                features[i * batch_size : (i + 1) * batch_size] = features_batch
                labels[i * batch_size : (i + 1) * batch_size] = labels_batch
                i += 1
                if i * batch_size >= sample_count:
                        break
        return features, labels
print("Starting feature extration process on training images...")
train_features, train_labels = extract_features(train_dir, 400)
print("Ending feature extration process on training images...")
print("Starting feature extration process on validation images...")
validation_features, validation_labels = extract_features(validation_dir, 50)
print("Ending feature extration process on validation images...")
print("Starting feature extration process on test images...")
test_features, test_labels = extract_features(test_dir, 50)
print("Ending feature extration process on test images...")
print("Ending feature extration process...")
train_features = np.reshape(train_features, (400, 15 * 15 * 512))
validation_features = np.reshape(validation_features, (50, 15 * 15 * 512))
test_features = np.reshape(test_features, (50, 15 * 15 * 512))
```

```
model = models.Sequential()

model.add(layers.Dense(256, activation='relu', input_dim=15 * 15 * 512))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),

                        loss='binary_crossentropy',

                        metrics=['acc'])


print("Starting training of NN on training, validation features...")

history = model.fit(train_features, train_labels,

                                     epochs=30,

                                     batch_size=20,

                                     validation_data=(validation_features,
validation_labels))


print("Ending training of NN on training, validation features...")


acc = history.history['acc']

val_acc = history.history['val_acc']

loss = history.history['loss']

val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)


plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()
```

```
  plt.plot(epochs, loss, 'bo', label='Training loss')

  plt.plot(epochs, val_loss, 'b', label='Validation loss')

  plt.title('Training and validation loss')

  plt.legend()

  plt.show()


  print("Starting models evaluation of NN on test dataset...")

  test_datagen = ImageDataGenerator(rescale=1./255)

  test_generator = test_datagen.flow_from_directory(test_dir,

          target_size=(500, 500),

          batch_size=20,

          class_mode='binary')


  test_loss_and_metrics = model.evaluate(test_features, test_labels)

  print("Ending evaluation of NN on test dataset...")

  print('Available test metrics: ', str(model.metrics_names))

  print('CNN model loss and accuracy on test dataset: ', str(test_loss_and_metrics))


  print("Starting prediction on test dataset...")

  predictions = model.predict(test_features)

  print(predictions)

  print("Ending prediction on test dataset...")

print("Ending image classification process...")

  print("Done")
```

Appendix B: GEP Console Output

```
(deeplearning) jw@Orion:~/Timber_Thesis/pits_v_sand4$ python listing_529.py

Using TensorFlow backend.

/usr/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compiletime version 3.5
of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version
3.6

  return f(*args, **kwds)

Starting image classification process...

2018-04-05 07:15:55.763735: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2
AVX AVX2 FMA

2018-04-05 07:15:55.930950: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:892]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero

2018-04-05 07:15:55.931228: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030]
Found device 0 with properties:

name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.721

pciBusID: 0000:01:00.0

totalMemory: 10.91GiB freeMemory: 10.08GiB

2018-04-05 07:15:56.048337: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:892]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero

2018-04-05 07:15:56.048597: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030]
Found device 1 with properties:

name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.721

pciBusID: 0000:02:00.0

totalMemory: 10.91GiB freeMemory: 10.75GiB

2018-04-05 07:15:56.049326: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045]
Device peer to peer matrix

2018-04-05 07:15:56.049347: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1051] DMA:
0 1

2018-04-05 07:15:56.049352: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1061] 0:
Y Y

2018-04-05 07:15:56.049354: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1061] 1:
Y Y
```

```
2018-04-05 07:15:56.049378: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120]
Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: GeForce GTX 1080 Ti, pci
bus id: 0000:01:00.0, compute capability: 6.1)

2018-04-05 07:15:56.049385: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120]
Creating TensorFlow device (/device:GPU:1) -> (device: 1, name: GeForce GTX 1080 Ti, pci
bus id: 0000:02:00.0, compute capability: 6.1)

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

input_1 (InputLayer)         (None, 500, 500, 3)       0

_________________________________________________________________

block1_conv1 (Conv2D)        (None, 500, 500, 64)      1792

_________________________________________________________________

block1_conv2 (Conv2D)        (None, 500, 500, 64)      36928

_________________________________________________________________

block1_pool (MaxPooling2D)   (None, 250, 250, 64)      0

_________________________________________________________________

block2_conv1 (Conv2D)        (None, 250, 250, 128)     73856

_________________________________________________________________

block2_conv2 (Conv2D)        (None, 250, 250, 128)     147584

_________________________________________________________________

block2_pool (MaxPooling2D)   (None, 125, 125, 128)     0

_________________________________________________________________

block3_conv1 (Conv2D)        (None, 125, 125, 256)     295168

_________________________________________________________________

block3_conv2 (Conv2D)        (None, 125, 125, 256)     590080

_________________________________________________________________

block3_conv3 (Conv2D)        (None, 125, 125, 256)     590080

_________________________________________________________________

block3_pool (MaxPooling2D)   (None, 62, 62, 256)       0
```

```
_________________________________________________________________

block4_conv1 (Conv2D)        (None, 62, 62, 512)       1180160

_________________________________________________________________

block4_conv2 (Conv2D)        (None, 62, 62, 512)       2359808

_________________________________________________________________

block4_conv3 (Conv2D)        (None, 62, 62, 512)       2359808

_________________________________________________________________

block4_pool (MaxPooling2D)   (None, 31, 31, 512)       0

_________________________________________________________________

block5_conv1 (Conv2D)        (None, 31, 31, 512)       2359808

_________________________________________________________________

block5_conv2 (Conv2D)        (None, 31, 31, 512)       2359808

_________________________________________________________________

block5_conv3 (Conv2D)        (None, 31, 31, 512)       2359808

_________________________________________________________________

block5_pool (MaxPooling2D)   (None, 15, 15, 512)       0
=================================================================

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

_________________________________________________________________

VGG16 base summary: None

Starting training data augmentation

Starting feature extration process...

Using base directory for training, validation and test images:
/home/jw/Timber_Thesis/pits_v_sand4

Starting feature extration process on training images...

Found 400 images belonging to 2 classes.
```

```
Ending feature extration process on training images...

Starting feature extration process on validation images...

Found 50 images belonging to 2 classes.

Ending feature extration process on validation images...

Starting feature extration process on test images...

Found 50 images belonging to 2 classes.

Ending feature extration process on test images...

Ending feature extration process...

Starting training of NN on training, validation features...

Train on 400 samples, validate on 50 samples

Epoch 1/30

400/400 [==============================] - 1s 2ms/step - loss: 0.8846 - acc: 0.5300 -
val_loss: 0.6629 - val_acc: 0.5000

Epoch 2/30

400/400 [==============================] - 0s 1ms/step - loss: 0.7143 - acc: 0.5375 -
val_loss: 0.6518 - val_acc: 0.5000

Epoch 3/30

400/400 [==============================] - 0s 1ms/step - loss: 0.6826 - acc: 0.5300 -
val_loss: 0.6872 - val_acc: 0.5000

Epoch 4/30

400/400 [==============================] - 0s 1ms/step - loss: 0.6697 - acc: 0.5950 -
val_loss: 0.6309 - val_acc: 0.8800

Epoch 5/30

400/400 [==============================] - 0s 935us/step - loss: 0.6180 - acc: 0.6675 -
val_loss: 0.6514 - val_acc: 0.5000

Epoch 6/30

400/400 [==============================] - 0s 1ms/step - loss: 0.6420 - acc: 0.6425 -
val_loss: 0.6210 - val_acc: 0.9600

Epoch 7/30

400/400 [==============================] - 0s 982us/step - loss: 0.6174 - acc: 0.6900 -
val_loss: 0.6079 - val_acc: 0.9800

Epoch 8/30
```

```
400/400 [==============================] - 0s 895us/step - loss: 0.6095 - acc: 0.6750 -
val_loss: 0.6036 - val_acc: 0.7000

Epoch 9/30

400/400 [==============================] - 0s 900us/step - loss: 0.5953 - acc: 0.7150 -
val_loss: 0.6217 - val_acc: 0.5000

Epoch 10/30

400/400 [==============================] - 0s 946us/step - loss: 0.5781 - acc: 0.7475 -
val_loss: 0.5890 - val_acc: 0.7000

Epoch 11/30

400/400 [==============================] - 0s 919us/step - loss: 0.5663 - acc: 0.7850 -
val_loss: 0.5690 - val_acc: 0.9800

Epoch 12/30

400/400 [==============================] - 0s 945us/step - loss: 0.5521 - acc: 0.8125 -
val_loss: 0.5701 - val_acc: 0.7000

Epoch 13/30

400/400 [==============================] - 0s 1ms/step - loss: 0.5605 - acc: 0.7900 -
val_loss: 0.5643 - val_acc: 0.6200

Epoch 14/30

400/400 [==============================] - 0s 1ms/step - loss: 0.5513 - acc: 0.8150 -
val_loss: 0.5392 - val_acc: 0.9800

Epoch 15/30

400/400 [==============================] - 0s 941us/step - loss: 0.5195 - acc: 0.8650 -
val_loss: 0.5746 - val_acc: 0.5800

Epoch 16/30

400/400 [==============================] - 0s 1ms/step - loss: 0.5164 - acc: 0.8475 -
val_loss: 0.5212 - val_acc: 1.0000

Epoch 17/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4981 - acc: 0.8775 -
val_loss: 0.5104 - val_acc: 1.0000

Epoch 18/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4797 - acc: 0.8650 -
val_loss: 0.5166 - val_acc: 0.8000

Epoch 19/30

400/400 [==============================] - 1s 1ms/step - loss: 0.5026 - acc: 0.8625 -
val_loss: 0.5001 - val_acc: 0.9600
```

```
Epoch 20/30

400/400 [==============================] - 0s 988us/step - loss: 0.4702 - acc: 0.8975 -
val_loss: 0.4904 - val_acc: 0.9600

Epoch 21/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4657 - acc: 0.8900 -
val_loss: 0.4930 - val_acc: 0.8200

Epoch 22/30

400/400 [==============================] - 1s 1ms/step - loss: 0.4444 - acc: 0.8950 -
val_loss: 0.4916 - val_acc: 0.8800

Epoch 23/30

400/400 [==============================] - 0s 918us/step - loss: 0.4465 - acc: 0.9000 -
val_loss: 0.4685 - val_acc: 0.9400

Epoch 24/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4276 - acc: 0.9150 -
val_loss: 0.4724 - val_acc: 0.9200

Epoch 25/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4222 - acc: 0.9100 -
val_loss: 0.4505 - val_acc: 0.9600

Epoch 26/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4007 - acc: 0.9300 -
val_loss: 0.4414 - val_acc: 0.9600

Epoch 27/30

400/400 [==============================] - 0s 935us/step - loss: 0.4027 - acc: 0.9150 -
val_loss: 0.4329 - val_acc: 0.9600

Epoch 28/30

400/400 [==============================] - 0s 1ms/step - loss: 0.4009 - acc: 0.9275 -
val_loss: 0.4312 - val_acc: 0.9600

Epoch 29/30

400/400 [==============================] - 0s 1ms/step - loss: 0.3884 - acc: 0.9400 -
val_loss: 0.4071 - val_acc: 0.9800

Epoch 30/30

400/400 [==============================] - 0s 1ms/step - loss: 0.3871 - acc: 0.9150 -
val_loss: 0.4158 - val_acc: 0.9600

Ending training of NN on training, validation features...

Starting evaluation of NN on test dataset...
```

```
Found 50 images belonging to 2 classes.

50/50 [==============================] - 0s 484us/step

Ending evaluation of NN on test dataset...

Available test metrics:  ['loss', 'acc']

CNN accuracy on test dataset:  [0.41870951652526855, 0.95999999046325679]

Ending image classification process...

Done
```

Appendix C: DGF Code

```
import os

import matplotlib.pyplot as plt

import numpy as np

from keras.preprocessing.image import ImageDataGenerator

from keras.applications import VGG16

from keras import models

from keras import layers

from keras import optimizers

from pathlib import Path


print("Starting image classification process...")

conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(112, 112, 3))

print("VGG16 base summary: " + str(conv_base.summary()))

print("Starting feature extration process...")

base_dir = str(Path.cwd())

print("Using base directory for training, validation and test images: " + base_dir)


train_dir = os.path.join(base_dir, 'train')

validation_dir = os.path.join(base_dir, 'validation')

test_dir = os.path.join(base_dir, 'test')


datagen = ImageDataGenerator(rescale=1./255)

batch_size = 20


def extract_features(directory, sample_count):

        features = np.zeros(shape=(sample_count, 3, 3, 512))
       labels = np.zeros(shape=(sample_count))
       generator = datagen.flow_from_directory(
               directory,
               target_size=(112, 112),
               batch_size=batch_size,
               class_mode='binary')
       i = 0
       for inputs_batch, labels_batch in generator:
               features_batch = conv_base.predict(inputs_batch)
               features[i * batch_size : (i + 1) * batch_size] = features_batch
               labels[i * batch_size : (i + 1) * batch_size] = labels_batch
               i += 1
               if i * batch_size >= sample_count:
                       break
       return features, labels


print("Starting feature extration process on training images...")
train_features, train_labels = extract_features(train_dir, 400)
print("Ending feature extration process on training images...")
print("Starting feature extration process on validation images...")
validation_features, validation_labels = extract_features(validation_dir, 100)
print("Ending feature extration process on validation images...")
print("Starting feature extration process on test images...")
test_features, test_labels = extract_features(test_dir, 100)
print("Ending feature extration process on test images...")
print("Ending feature extration process...")
```

```
train_features = np.reshape(train_features, (400, 3 * 3 * 512))

validation_features = np.reshape(validation_features, (100, 3 * 3 * 512))

test_features = np.reshape(test_features, (100, 3 * 3 * 512))


model = models.Sequential()

model.add(layers.Dense(256, activation='relu', input_dim=3 * 3 * 512))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(1, activation='sigmoid'))


model.compile(optimizer=optimizers.RMSprop(lr=2e-5),

                        loss='binary_crossentropy',

                        metrics=['acc'])


print("Starting training of NN on training, validation features...")

history = model.fit(train_features, train_labels,

                                     epochs=30,

                                     batch_size=20,

                                     validation_data=(validation_features,
validation_labels))


print("Ending training of NN on training, validation features...")


acc = history.history['acc']

val_acc = history.history['val_acc']

loss = history.history['loss']

val_loss = history.history['val_loss']


epochs = range(1, len(acc) + 1)
```

```
plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()


print("Starting evaluation of NN on test dataset...")

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(test_dir,

        target_size=(112, 112),

        batch_size=20,

        class_mode='binary')


test_loss_and_metrics = model.evaluate(test_features, test_labels)

print("Ending evaluation of NN on test dataset...")

print('Available test metrics: ', str(model.metrics_names))

print('CNN accuracy on test dataset: ', str(test_loss_and_metrics))

print("Ending image classification process...")

print("Done")
```

Appendix D: DGF Console Output

```
I jw@Orion:~/Dropbox/MASTERS_THESIS_W/pits_v_sand5$ python3 listing_530.py

Using TensorFlow backend.

/usr/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compiletime version 3.5
of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version
3.6

  return f(*args, **kwds)

Starting image classification process...

2018-04-27 18:15:33.020201: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2
AVX AVX2 FMA

2018-04-27 18:15:33.021330: E tensorflow/stream_executor/cuda/cuda_driver.cc:406] failed
call to cuInit: CUDA_ERROR_UNKNOWN

2018-04-27 18:15:33.021350: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:158]
retrieving CUDA diagnostic information for host: Orion

2018-04-27 18:15:33.021355: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:165]
hostname: Orion

2018-04-27 18:15:33.021377: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:189]
libcuda reported version is: 387.34.0

2018-04-27 18:15:33.021389: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:369]
driver version file contents: """NVRM version: NVIDIA UNIX x86_64 Kernel Module  387.34
Tue Nov 21 03:09:00 PST 2017

GCC version:  gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3.2)

"""

2018-04-27 18:15:33.021397: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:193]
kernel reported version is: 387.34.0

2018-04-27 18:15:33.021401: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:300]
kernel version seems to match DSO: 387.34.0

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

input_1 (InputLayer)         (None, 112, 112, 3)       0

_________________________________________________________________

block1_conv1 (Conv2D)        (None, 112, 112, 64)      1792

_________________________________________________________________
```

```
block1_conv2 (Conv2D)        (None, 112, 112, 64)      36928

_________________________________________________________________

block1_pool (MaxPooling2D)   (None, 56, 56, 64)        0

_________________________________________________________________

block2_conv1 (Conv2D)        (None, 56, 56, 128)       73856

_________________________________________________________________

block2_conv2 (Conv2D)        (None, 56, 56, 128)       147584

_________________________________________________________________

block2_pool (MaxPooling2D)   (None, 28, 28, 128)       0

_________________________________________________________________

block3_conv1 (Conv2D)        (None, 28, 28, 256)       295168

_________________________________________________________________

block3_conv2 (Conv2D)        (None, 28, 28, 256)       590080

_________________________________________________________________

block3_conv3 (Conv2D)        (None, 28, 28, 256)       590080

_________________________________________________________________

block3_pool (MaxPooling2D)   (None, 14, 14, 256)       0

_________________________________________________________________

block4_conv1 (Conv2D)        (None, 14, 14, 512)       1180160

_________________________________________________________________

block4_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

_________________________________________________________________

block4_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

_________________________________________________________________

block4_pool (MaxPooling2D)   (None, 7, 7, 512)         0

_________________________________________________________________

block5_conv1 (Conv2D)        (None, 7, 7, 512)         2359808

_________________________________________________________________
```

```
block5_conv2 (Conv2D)        (None, 7, 7, 512)        2359808

_________________________________________________________________

block5_conv3 (Conv2D)        (None, 7, 7, 512)        2359808

_________________________________________________________________

block5_pool (MaxPooling2D)   (None, 3, 3, 512)        0

=================================================================

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

_________________________________________________________________

VGG16 base summary: None

Starting feature extration process...

Using base directory for training, validation and test images:
/media/dropbox/Dropbox/MASTERS_THESIS_W/pits_v_sand5

Starting feature extration process on training images...

Found 400 images belonging to 2 classes.

Ending feature extration process on training images...

Starting feature extration process on validation images...

Found 100 images belonging to 2 classes.

Ending feature extration process on validation images...

Starting feature extration process on test images...

Found 100 images belonging to 2 classes.

Ending feature extration process on test images...

Ending feature extration process...

Starting training of NN on training, validation features...

Train on 400 samples, validate on 100 samples

Epoch 1/30

400/400 [==============================] - 0s 440us/step - loss: 0.7721 - acc: 0.5150 -
val_loss: 0.6698 - val_acc: 0.5000
```

```
Epoch 2/30

400/400 [==============================] - 0s 279us/step - loss: 0.7230 - acc: 0.5075 -
val_loss: 0.6338 - val_acc: 0.8600

Epoch 3/30

400/400 [==============================] - 0s 281us/step - loss: 0.6939 - acc: 0.5750 -
val_loss: 0.6118 - val_acc: 0.8400

Epoch 4/30

400/400 [==============================] - 0s 280us/step - loss: 0.6676 - acc: 0.6075 -
val_loss: 0.5898 - val_acc: 0.8700

Epoch 5/30

400/400 [==============================] - 0s 275us/step - loss: 0.6373 - acc: 0.6125 -
val_loss: 0.5660 - val_acc: 0.9000

Epoch 6/30

400/400 [==============================] - 0s 283us/step - loss: 0.6185 - acc: 0.6450 -
val_loss: 0.5474 - val_acc: 0.8900

Epoch 7/30

400/400 [==============================] - 0s 283us/step - loss: 0.5861 - acc: 0.7075 -
val_loss: 0.5280 - val_acc: 0.8900

Epoch 8/30

400/400 [==============================] - 0s 281us/step - loss: 0.5918 - acc: 0.6875 -
val_loss: 0.5108 - val_acc: 0.8800

Epoch 9/30

400/400 [==============================] - 0s 275us/step - loss: 0.5400 - acc: 0.7675 -
val_loss: 0.4956 - val_acc: 0.9000

Epoch 10/30

400/400 [==============================] - 0s 277us/step - loss: 0.5484 - acc: 0.7425 -
val_loss: 0.4813 - val_acc: 0.8900

Epoch 11/30

400/400 [==============================] - 0s 284us/step - loss: 0.5302 - acc: 0.7675 -
val_loss: 0.4640 - val_acc: 0.9000

Epoch 12/30

400/400 [==============================] - 0s 282us/step - loss: 0.5108 - acc: 0.7900 -
val_loss: 0.4501 - val_acc: 0.8900

Epoch 13/30
```

```
400/400 [==============================] - 0s 284us/step - loss: 0.4860 - acc: 0.8100 -
val_loss: 0.4364 - val_acc: 0.9000

Epoch 14/30

400/400 [==============================] - 0s 279us/step - loss: 0.4807 - acc: 0.8325 -
val_loss: 0.4254 - val_acc: 0.9000

Epoch 15/30

400/400 [==============================] - 0s 285us/step - loss: 0.4678 - acc: 0.8350 -
val_loss: 0.4181 - val_acc: 0.8900

Epoch 16/30

400/400 [==============================] - 0s 286us/step - loss: 0.4595 - acc: 0.8325 -
val_loss: 0.3995 - val_acc: 0.9000

Epoch 17/30

400/400 [==============================] - 0s 279us/step - loss: 0.4457 - acc: 0.8700 -
val_loss: 0.3889 - val_acc: 0.8900

Epoch 18/30

400/400 [==============================] - 0s 281us/step - loss: 0.4339 - acc: 0.8600 -
val_loss: 0.3797 - val_acc: 0.9000

Epoch 19/30

400/400 [==============================] - 0s 285us/step - loss: 0.4356 - acc: 0.8650 -
val_loss: 0.3695 - val_acc: 0.8900

Epoch 20/30

400/400 [==============================] - 0s 276us/step - loss: 0.4215 - acc: 0.8750 -
val_loss: 0.3655 - val_acc: 0.9000

Epoch 21/30

400/400 [==============================] - 0s 284us/step - loss: 0.4115 - acc: 0.8900 -
val_loss: 0.3556 - val_acc: 0.9100

Epoch 22/30

400/400 [==============================] - 0s 283us/step - loss: 0.3947 - acc: 0.8925 -
val_loss: 0.3431 - val_acc: 0.8900

Epoch 23/30

400/400 [==============================] - 0s 282us/step - loss: 0.3769 - acc: 0.8775 -
val_loss: 0.3381 - val_acc: 0.9300

Epoch 24/30

400/400 [==============================] - 0s 288us/step - loss: 0.3771 - acc: 0.8925 -
val_loss: 0.3293 - val_acc: 0.9200
```

```
Epoch 25/30

400/400 [==============================] - 0s 278us/step - loss: 0.3602 - acc: 0.9150 -
val_loss: 0.3211 - val_acc: 0.9100

Epoch 26/30

400/400 [==============================] - 0s 297us/step - loss: 0.3465 - acc: 0.9050 -
val_loss: 0.3147 - val_acc: 0.9100

Epoch 27/30

400/400 [==============================] - 0s 283us/step - loss: 0.3451 - acc: 0.9200 -
val_loss: 0.3074 - val_acc: 0.9100

Epoch 28/30

400/400 [==============================] - 0s 277us/step - loss: 0.3492 - acc: 0.8925 -
val_loss: 0.3065 - val_acc: 0.9200

Epoch 29/30

400/400 [==============================] - 0s 277us/step - loss: 0.3265 - acc: 0.9050 -
val_loss: 0.2961 - val_acc: 0.9200

Epoch 30/30

400/400 [==============================] - 0s 282us/step - loss: 0.3228 - acc: 0.9275 -
val_loss: 0.2895 - val_acc: 0.9200

Ending training of NN on training, validation features...

Starting evaluation of NN on test dataset...

Found 100 images belonging to 2 classes.

100/100 [==============================] - 0s 51us/step

Ending evaluation of NN on test dataset...

Available test metrics:  ['loss', 'acc']

CNN accuracy on test dataset:  [0.3282634776830673, 0.90000000000000002]

Ending image classification process...

Done
```