2022

Development and Optimization of Classification Neural Networks for Disaster-Assessment Using Unmanned Aerial Vehicle Systems

Maria Isabel Gonzalez Bocanegra

Development and Optimization of Classification Neural Networks for Disaster-Assessment Using Unmanned Aerial Vehicle Systems

An Honors Thesis submitted in partial fulfillment of the requirements for Honors in *Department Name.*

By
Maria Isabel Gonzalez Bocanegra

Under the mentorship of *Rami J. Haddad*

ABSTRACT

This research focuses on increasing the classification accuracy of convolutional neural networks in an autonomous network of unmanned aerial vehicles for transportation disaster management. The autonomous network of UAVs will allow first responders to optimize their rescue plans by providing relevant information on inaccessible roads. The research seeks to explore different methods to optimize the architecture of convolutional networks for the multiclass classification of disaster-damaged roads.

Thesis Mentor:________________________

Dr. *Rami J. Haddad*

Honors Director:_______________________

Dr. Steven Engel

April 2022
Department Electrical and Computer Engineering
Honors College
Georgia Southern University

Development and Optimization of Classification Neural Networks for Disaster-Assessment Using Unmanned Aerial Vehicle Systems

By
Maria Isabel Gonzalez Bocanegra

Table of Contents

List of Figures

Acknowledgements

I wanted to use this section to thank my mom, Marisol Bocanegra, who has been right next to me every step of the way (even though she lives 2805.2 miles away). Thank you for giving me the space, opportunity, and understanding to become the person I am now. I would also like to thank Dr. Rami J. Haddad for his mentorship and advise throughout my four years at Georgia Southern University. I am extremely thankful to him for all his help and advice, without him this project would not have been possible. I also want to thank the people who have never left my side during my last few years at Southern, especially Zach Hamilton, Irene Bueso, Analucia Yanar, Ana Abadie, and Ana Oviedo. You guys inspired me to become the best version of myself and enjoy the journey while doing so.

Chapter 1: Introduction

The occurrence of natural disasters has increased with the accelerated rate of climate change in the United States and the world. These natural disasters can have deep consequences in communities and can drastically affect human infrastructure, such as roads. There exists a need for effective means to assess natural disaster damages and aid first responders in their recovery efforts. The costs brought up after such events is influenced by damage assessment and cleanup efforts. These assessments are handled by state and federal ground teams, which require great manpower. With this in mind, the project seeks to develop an automated damage assessment process to streamline disaster preparedness, response, and recovery operations. The implementation of such systems would allow for disaster management teams to optimize their recovery efforts by providing to real-time transportation network status information. This information can be leveraged by federal agencies, such as FEMA, to provide aid to natural disaster affected communities and infrastructure.

The main characteristic of this system is that it utilizes image processing and deep learning techniques to assess damages to transportation systems (roads). Once the neural network positively identifies damages, it automatically retrieves the image's geo-tag metadata to an ArcGIS map. These online geo-tagged maps can then be accessed by response and recovery teams to facilitate their recovery efforts. The system is predominantly useful for the restoration of the state transportation system after natural disasters. The system provides the damage assessment team with a list summarizing all damages that were assessed and their geographical locations and live streaming of the UAV's video feed to an RTMP server, enabling the first responders to assess damages.

The project also developed a Graphic User Interface application using Python and the MATLAB software to automate and centralize the operation of this system. The application included managing, sampling, classifying, and ArcGIS map tagging of the UAV-generated video streams. This application also provided some flexibility to customize the operating settings of this system.

A library of bird's view disaster damaged road images was compiled through extensive research on natural disasters. The library was divided into six different classes of natural disaster damaged roads that are commonly observed. Additionally, a total of three different Convolutional Neural Networks were researched and tested after implementing accuracy optimization techniques. Information for network performance was obtained through different metric assessments. The best network accuracy for multiclass classification achieved 74.1% accuracy while the binary classification achieved 99% accuracy.

Chapter 2: Background

This chapter serves as an introduction to the theory behind Convolutional Neural Networks, Pretrained Convolutional Neural Networks, Transfer Learning, K-Fold Cross Validation, and Network Performance Assessments.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of Artificial Neural Networks that help analyze images through image processing. Such networks have a flexible network configuration that allows for image data mapping. These networks were proposed in the 1995 paper by Yann Lecun [6]. The overall architecture of the CNN consists of three parts that help classify raw data. The first part is called the local receptive fields, which have artificial neurons. These artificial neurons consist of mathematical functions that calculate the weighted sum of multiple inputs and outputs. The artificial neuron in a receptive field of a Convolutional Neural Network deals with sections of high-dimensional inputs (e.g., images). Connecting the artificial neurons will result in the creation of the receptive field. This process creates a feature map that can be used as the starting point to guide the output. Figure 1 showcases a visual example of the reception field computation. The receptive field in each convolutional layer can be calculated by using the following equation:

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1 \tag{1}$$

$$n_{in} = number\ of\ input\ features$$

$$n_{out} = number\ of\ output\ features$$

k = *convolution kernel size*

$$p = convolution\ padding\ size$$

$$s = convolution\ stride\ size$$

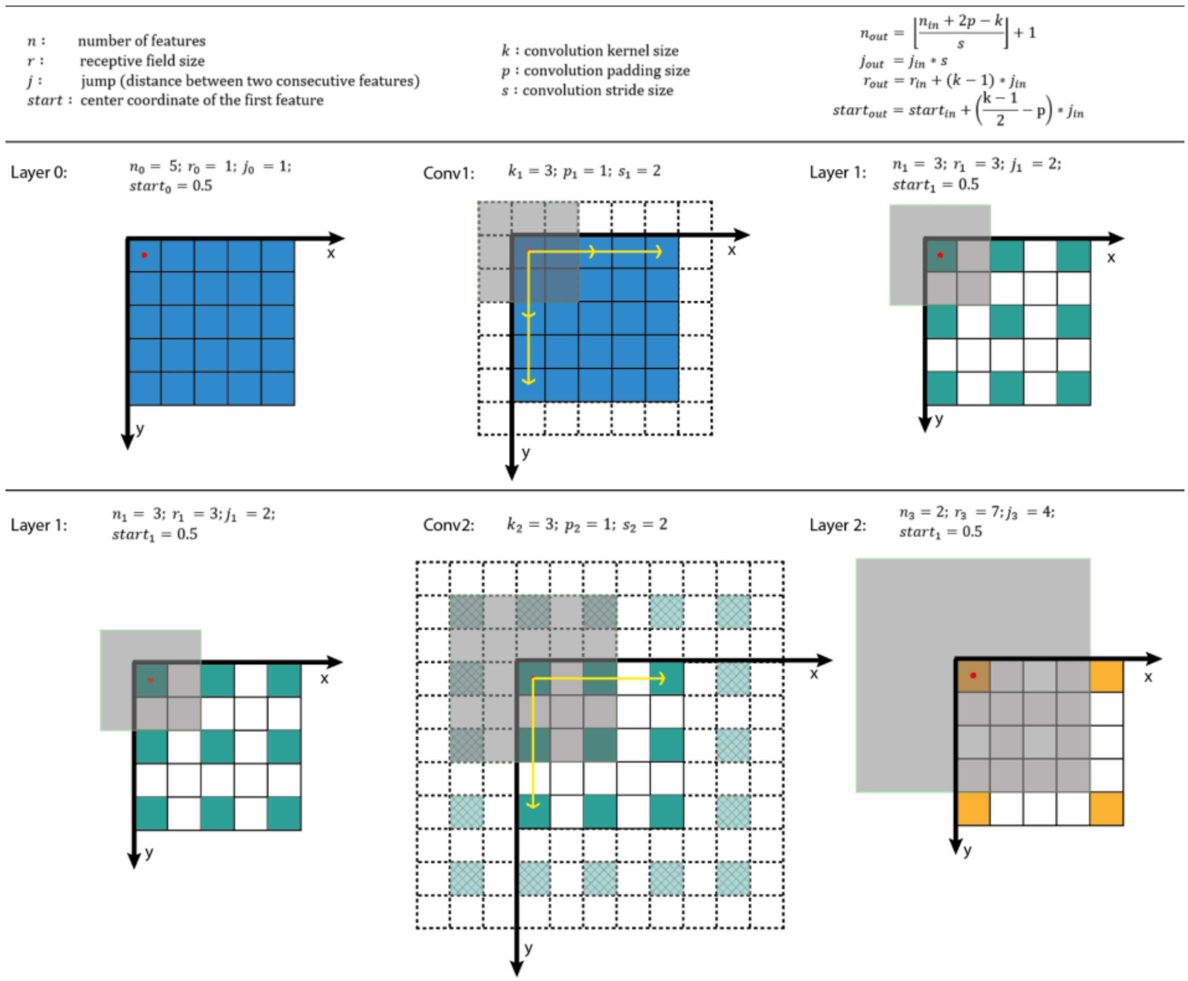

Figure 1. Details for Receptive Field Computation [36]

The next important concept of a Convolutional Neural Network is spatial subsampling. The spatial subsampling, or pooling layer, is applied to reduce the resolution of feature maps and their sensitivity to outputs produced by their current convolutional layer [6]. The convolutional layer in the network creates a general feature map with the essential features from the raw data. The spatial subsampling down samples the output of the convolutional layer in the height and width spatial dimension. This feature reduces the reliance of precise positioning of features and retains important feature map information. Figure 2 showcases the subsampling, or pooling, layer which groups the feature maps and extracts their most important feature.

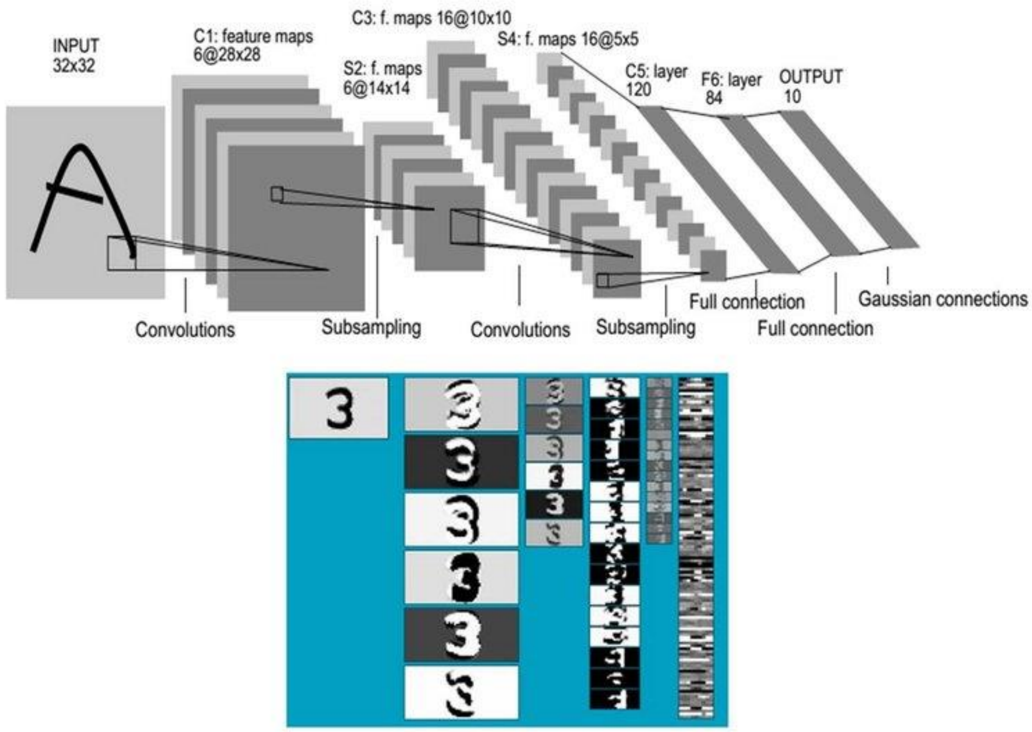

Figure 2. Spatial Subsampling Example [37]

The last important aspect of the Convolutional Neural Networks is weight sharing. Weight sharing is the use of a reception field across the visual field, in this case images. This allows for the creation of different feature maps that can extract a reception field specific feature from an image.

Figure 3 exemplifies the complete architecture of a Convolutional Neural Network where convolutional layers can be observed, creating feature maps that are then input to a spatial subsampling layer. The process continues until a matrix of feature layers is achieved and then fed to fully connecting layers that combine these feature maps into a model [7]. The output model is then classified by an activation function, in most cases the SoftMax function.

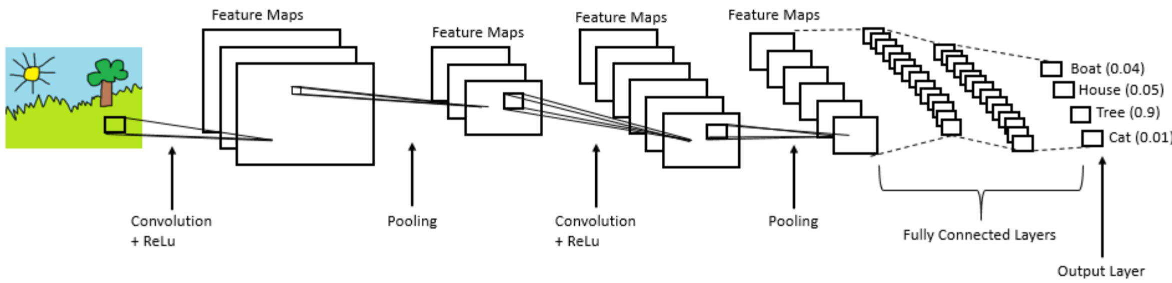

Figure 3. Illustration of Convolutional Neural Network Architecture [8]

2.2 Pre-Trained Convolutional Neural Networks

Pre-trained convolutional neural networks have revolutionized the industry due to their capability of using image processing to leverage object classification. The project uses this capability to its advantage for the recognition of disaster damaged roads. Each network has been already pretrained on the ImageNet dataset, a 15 million high-resolution image dataset. By implementing pretrained neural network, the computational complexity and time in the project's training process is significantly reduced by implementing transfer learning. The three networks tested are AlexNet, GoogLeNet, and ResNet50, which are some of the most successful networks today.

The AlexNet CNN is a 22 layers deep network that helped classify 1.2 million high-resolution images into 1000 different classes in the ImageNet Large-Scale Visual

Recognition Challenge (LSVRC) 2010 contest [8]. The architecture consists of five convolutional layers, three max-pooling layers, two normalization layers, two fully connected layers, and one SoftMax layer. GoogLeNet is 22 layers deep and was trained on the same dataset as AlexNet. Additionally, GoogLeNet is able to reduce the input image while maintaining important spatial information across its convolutional layers. This technique allows for the network to obtain more details from the reduced image . on the other hand, ResNet50 is a 50 layers deep CNN trained on over a million images that implement residual learning. ResNet50 skips connections instead of layering convolutional layers to address vanishing gradient descent.

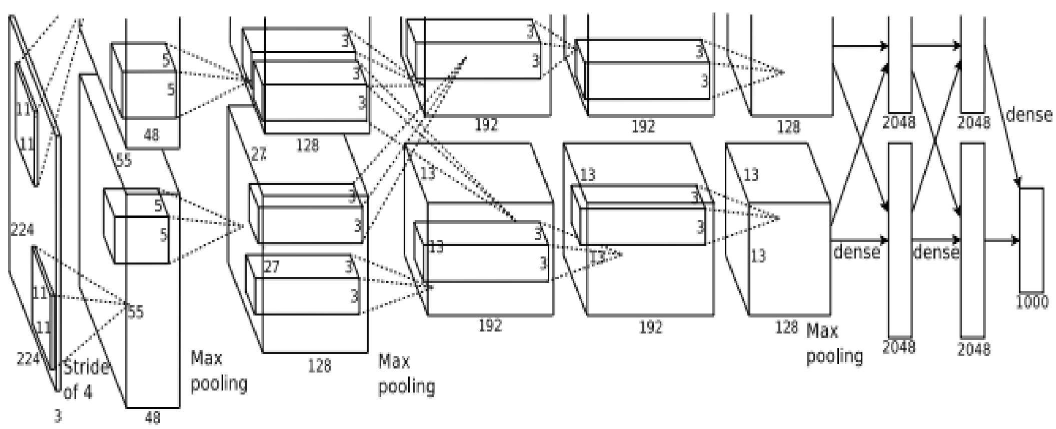

Figure 4. Architecture Representation of AlexNet CNN [10]

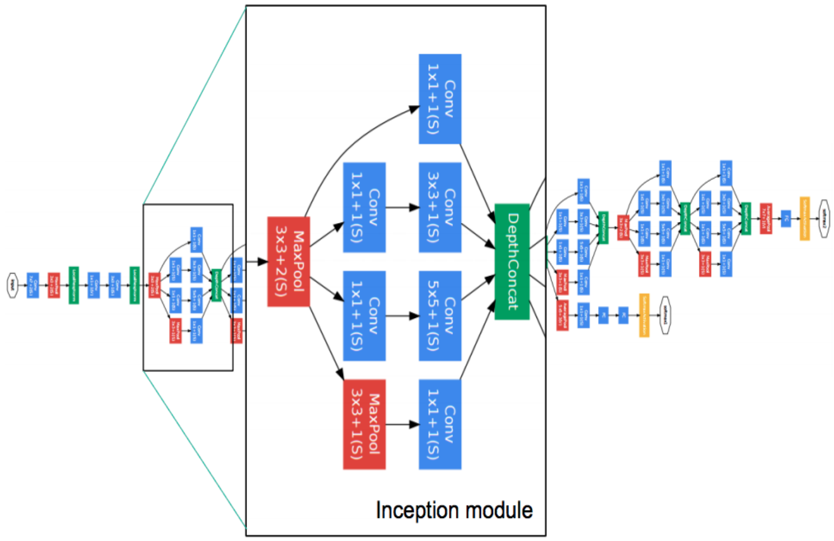

Figure 5. Architecture Representation of GoogLeNet CNN [8]

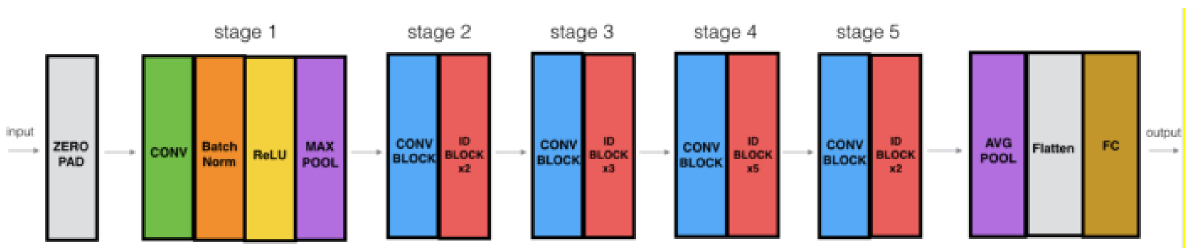

Figure 6. Architecture Representation of ResNet50 CNN [11]

2.3 Transfer Learning

Transfer learning is a technique that leverages pre-trained Convolutional Neural Networks. Through this process the user selectively changes the output categories of a pre-trained classification network. The original network model is built to classify specific tasks, but with transfer learning the network is repurposed the to classify the required task by the user. A network trained on a large dataset to classify 1000+ objects can be repurposed to classify a smaller dataset. The network's learned parameters obtained by being pre-trained with a large dataset are kept unchanged, except for the final few layers. The last few layers of the network are repurposed for specific dataset classification. The application of transfer learning addresses the time constraints of building large data sets and performing supervised learning. Transfer learning also avoids the use of costly hardware, such as a GPU, required to conduct mathematically intense computational analysis with large datasets. Figure 7 showcases the overall theory of transfer learning.

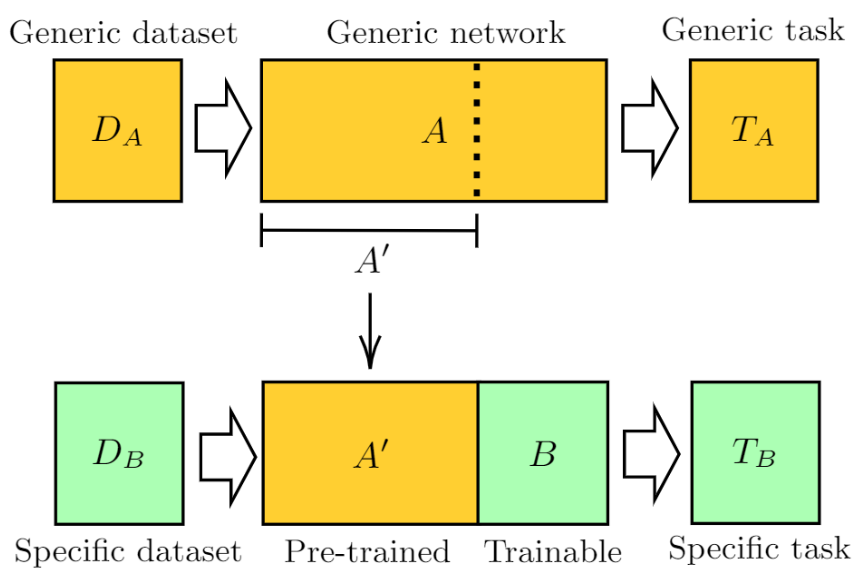

Figure 7. General Demonstration of Transfer Learning [38]

2.4 K-Fold Cross-Validation

K-Fold Cross-Validation is a technique used to partition a dataset into K number of sections. Each section will at one point of the experiment be used to test the Convolutional Neural Network after the training phase is completed. This technique is used to evaluate the performance of the network on data it has not be trained on to obtain true system accuracy. The implementation of K-Fold Cross-Validation is a powerful tool to avoiding data overfitting issues that rise from training and testing on a limited dataset of disaster-damaged roads. In the case of this work, the dataset was divided into four folds (K = 4). Three folds are used as a training dataset while the remaining fold is used as the testing dataset. Each fold will eventually be used as a testing dataset thus there are four testing and training trials. This methodology allows for a less biased model and faster generalization.

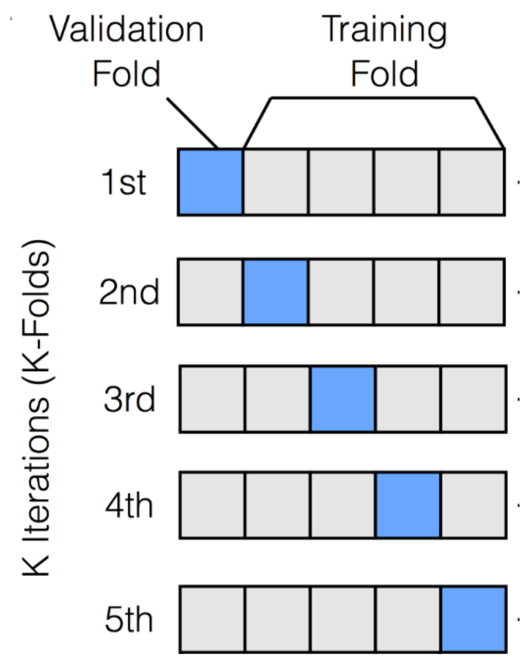

Figure 8. General Demonstration of K-Fold Cross-Validation

https://androidkt.com/pytorch-k-fold-cross-validation-using-dataloader-and-sklearn/

2.5 Network Performance Assessments

The network's classification performance can be measure in multiple ways, in this section we explore the metric used to assess network's performance. The classification output can be characterized by one of the following four categories: true positive (TP), false positive (FP), true negative (TN), and false-negative (FN). These categories will help in the calculations of the metrics to assess the network's performance.

2.5.1 Recall Metric

Recall, also known as sensitivity, refers to the ratio between the number of correct positive classified images (true positives) to that of the total number of possible positive classified images (true positive + false negative). This metric allows for the understanding of how well the network recognizes positive cases.

$$Recall = \frac{TP}{TP+FN} \quad (2)$$

2.5.2 Precision Metric

The precision metric refers to the ratio of correct positive classified images (true positive) to the number of all positive classified images (true positive + false positive) , as shown in Eq.3. This measure demonstrates the model's classification accuracy of positive cases.

$$Precision = \frac{TP}{TP+FP}$$

(3)

2.5.3 F1 Score Metric

The harmonic mean between statistical precision and recall is called the F1 score, shown in Eq. 4. This measurement is preferred when there exists some degree of class imbalance in the dataset. The F1 score is suited for measuring incorrectly classified cases by a network and is represented by a number between 0 and 1.

$$F1 = \frac{TP}{TP+0.5(FP+FN)}$$

(4)

2.6 ArcGIS Online Mapping Software

ArcGIS is a cloud based geographical information software used to map, visualize, and analyze geospatial information. The software delivers fast solutions to the development of apps, maps, and data. This project leverages the ArcGIS Online tool for complex map creation and development through its Python API. ArcGIS is implemented to create maps with exact pinpoints of the locations where the UAVs identified disaster-damaged roads, as illustrated in Figure 9. The software can expand into more complex workflows and is suitable for this project due to its API design flexibility.

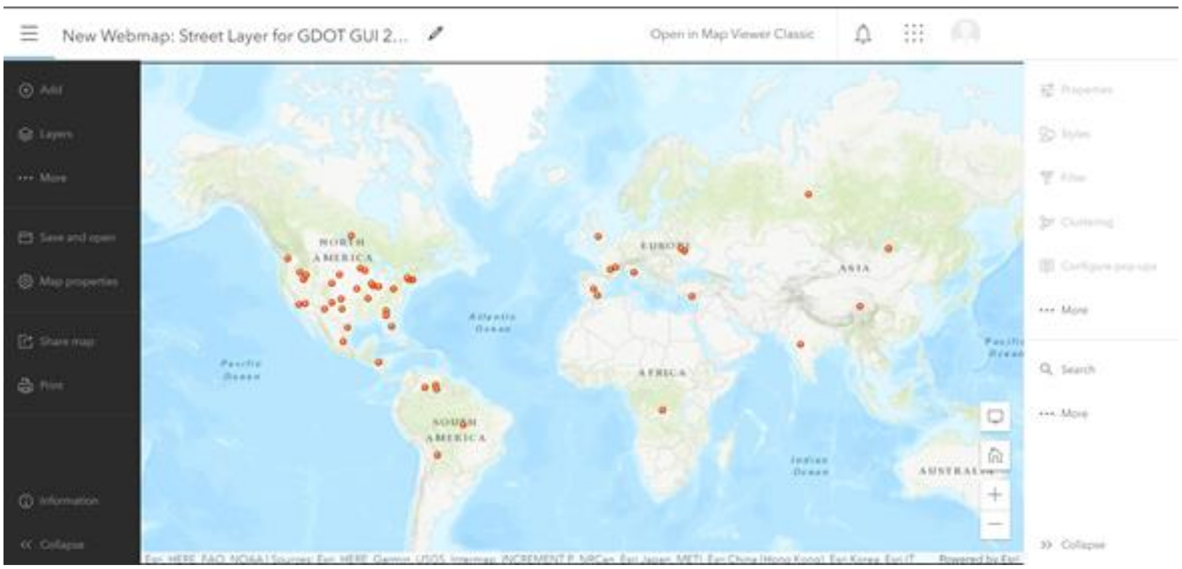

Figure 9. ArcGIS web map with tagged disaster-damaged roads

Each pinpoint in the web map represents a disaster damaged road. The pinpoint contains the image's metadata information of the disaster damaged road. This metadata includes latitude, longitude, and type of damage associated with a specific location as well as original file location in Google Drive. Figure 10 provides an example of the information each pin holds. The ArcGIS API also supports the automation of map development on their online platform. The API serves as the foundation for the creation of the graphic user interface (GUI) to automate workflows and speed up data retrieval.

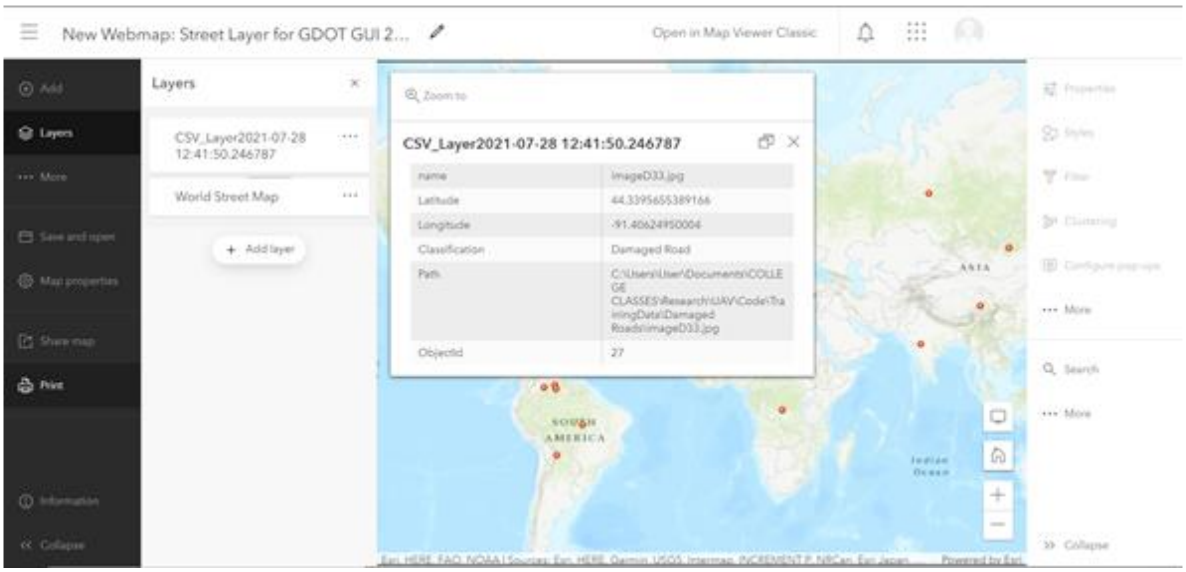

Figure 10. Information tag for each disaster-damaged road on the web map

Chapter 3: Methodology

3.1 Image Library

Convolutional Neural Network development requires a dataset from which the network will be trained and tested on. This research requires a large set of images representing the different categories on which the network will be tasked to classify images. This research developed a library of disaster-damaged road images due to the lack of natural disaster damaged road datasets. The created library contains six different classes of damages encountered by the Georgia Department of Transportation across the state during and after natural disasters. The library is partitioned into the following classes: Damaged Roads, Clear Roads, Blocked Roads, Boat in Roads, Fallen Power Lines, and Flooded Roads. A specific characteristic of this dataset is that the images are exclusively taken at a bird's eye view or high camera angle to resemble what a flying UAV would capture in real-time. This project also took into consideration the requirement of evenly distributed classes to avoid feature unbalance during training and testing of the network. Figures 11-17 showcase image samples that are part of the image library.

Figure 11. Disaster Damaged Road - Library Sample Image [29]

Figure 12. Clear Road - Library Sample Image [30]

Figure 13. Disaster Blocked Road - Library Sample Image [31]

Figure 14. Boat in a Road - Library Sample Image [32]

Figure 15. Fallen Power Lines - Library Sample Image [33]

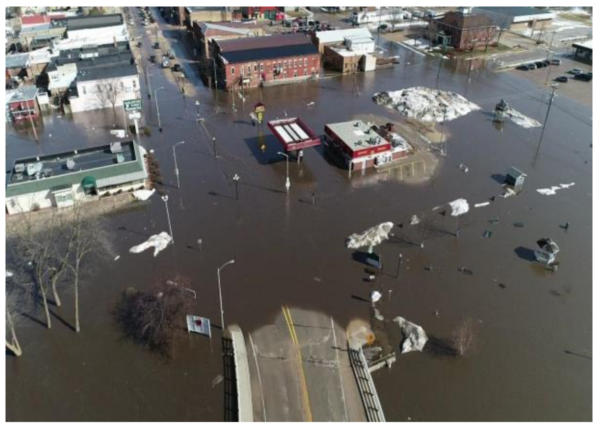

Figure 16. Flooded Road - Library Sample Image [34]

3.2 Convolutional Neural Network Training, Testing, Optimization

For the purposes of this research project, three different but highly successful Convolutional Neural Networks were tested. As previously mentioned, the main networks investigated are AlexNet, GoogLeNet, and ResNet50. These networks were chosen given their architectural prowess for learning and generalizing to properly conduct classification.

The MATLAB software was used to create the networks through the Deep Learning Toolbox. Moreover, parameters such as mini-batch size, max epochs, and learning rate, were set to their default values for each network. The networks were tested with a mini-batch size of 64, 15 epochs, and a learning rate of 10-4 with an image input of 224×224×3. A MATLAB script was written to accommodate for different image sizes in the dataset and convert them into 224×224×3.

The project implemented transfer learning at this stage to avoid network retraining and exploit the pretrained networks as well as to optimize the networks. This technique allows for domain adaptation by using the learned features from the trained network and moving them into the target network. Furthermore, the last few layers of the network are retrained to avoid data overfitting, which can happen due to the small size of the dataset. The last layers also obtain more specific features from the smaller dataset. Transfer learning helps avoid time constraints of gathering images to create large datasets and lowers hardware costs, e.g., GPU, required to conduct big mathematical computational analyses with big datasets.

The network is ready to train after completing the transfer learning. The original dataset is partitioned through K-Fold cross validation. To train the network, a 4-fold cross-validation is implemented, meaning the dataset is partitioned in four groups. Cross-validation is a validation technique implemented to assess the Convolutional Neural Network's performance under different training and testing mixed folds. Similarly, to transfer learning, K-Fold Cross-Validation is also a method that helps avoid overfitting issues caused by training and testing on a small dataset of disaster-damaged roads.

For this project, the dataset was divided into four-fold from which three folds are selected for training and the other fold is used as the testing dataset. Through this methodology each fold will be used as a testing dataset, achieving four different training phases and thus four different iterations of the network. After each training phase the model parameters are saved for testing. All the metrics are averaged to estimate the model's performance more accurately.

3.3 Graphical User Interface

A Graphical User Interface (GUI) is a subset of the User Interface (UI) group which allows for the users to interact with electronic devices. For this project, and to provide a system that can assess disaster-damaged roads, there exists a need to centralize all the components. The centralization system will take the form of a Graphical User Interface developed through Python 3. In addition to Python 3, the GUI also incorporated ArcGIS API to help with the automation of web map development of disaster-damaged roads. The GUI provides a wide variation of web map creation and modification as well as data handling and Neural Network reconfiguration. Figure 17 provides the complete GUI system diagram and its functionality. The user can access their online ArcGIS account from the GUI by using their credentials. Figures 18-20 show the main page where the login process starts.

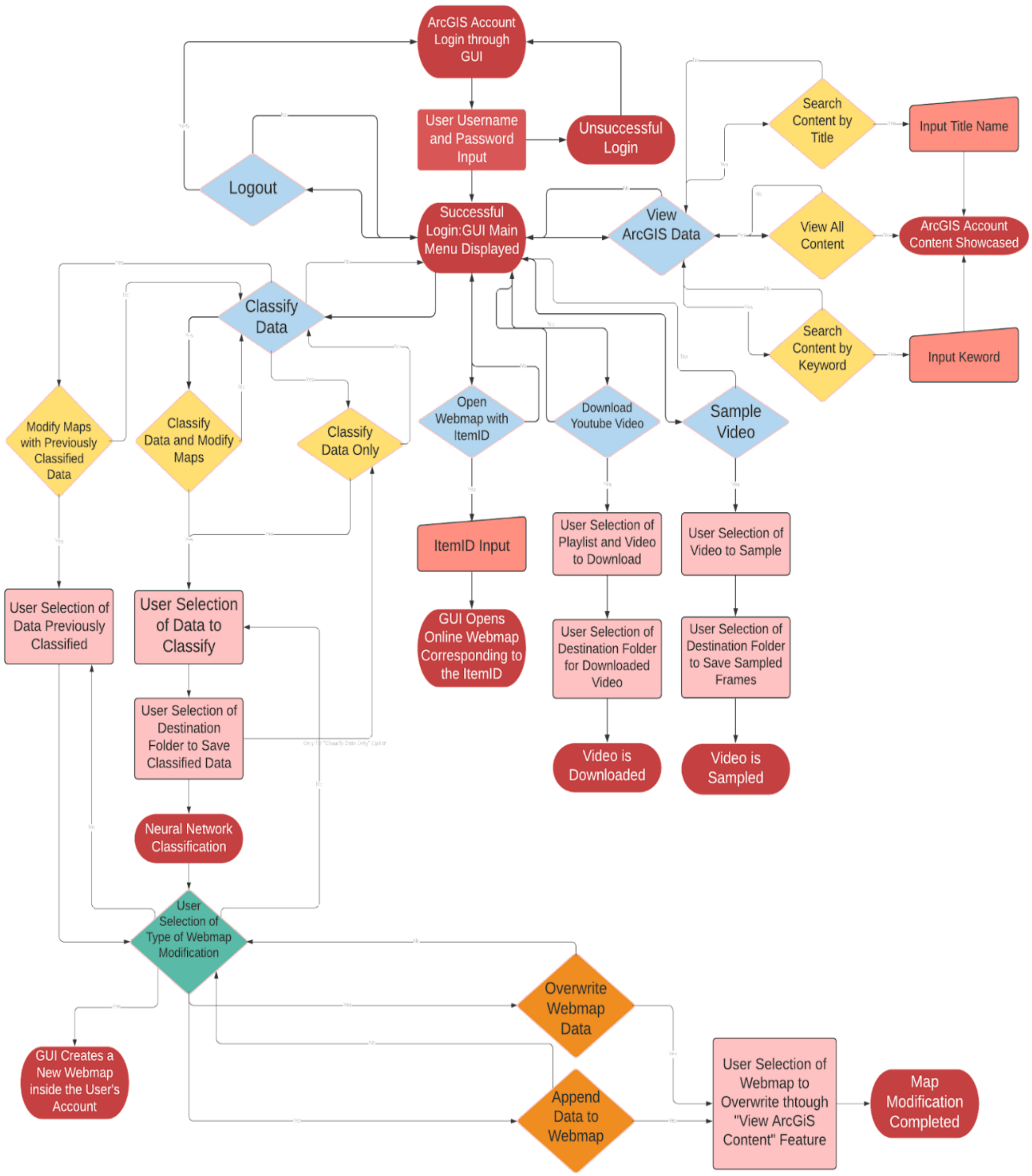

Figure 17. Complete GUI System Diagram

.

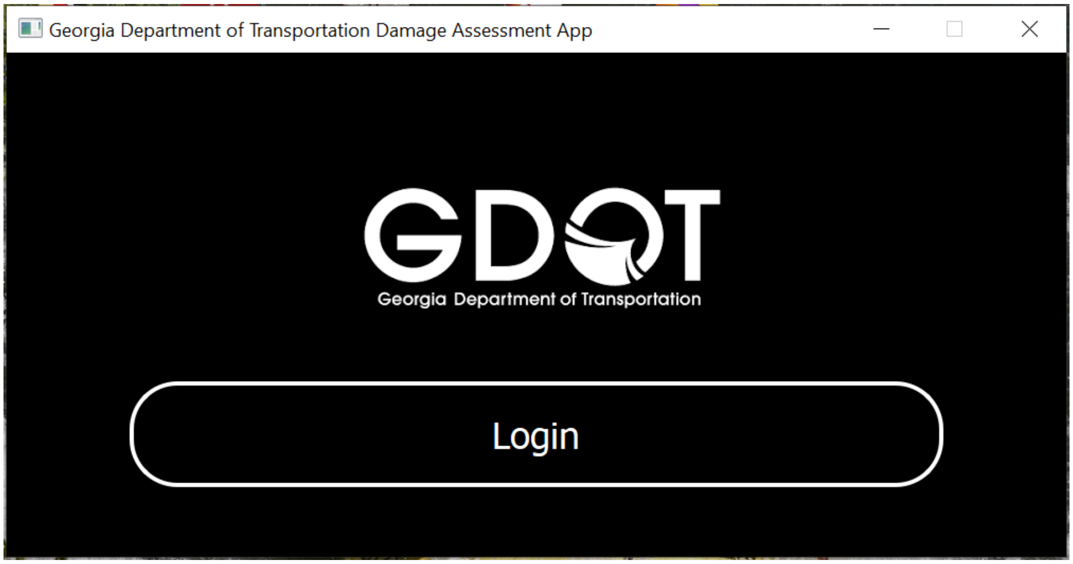

Figure 18. GUI Login Window

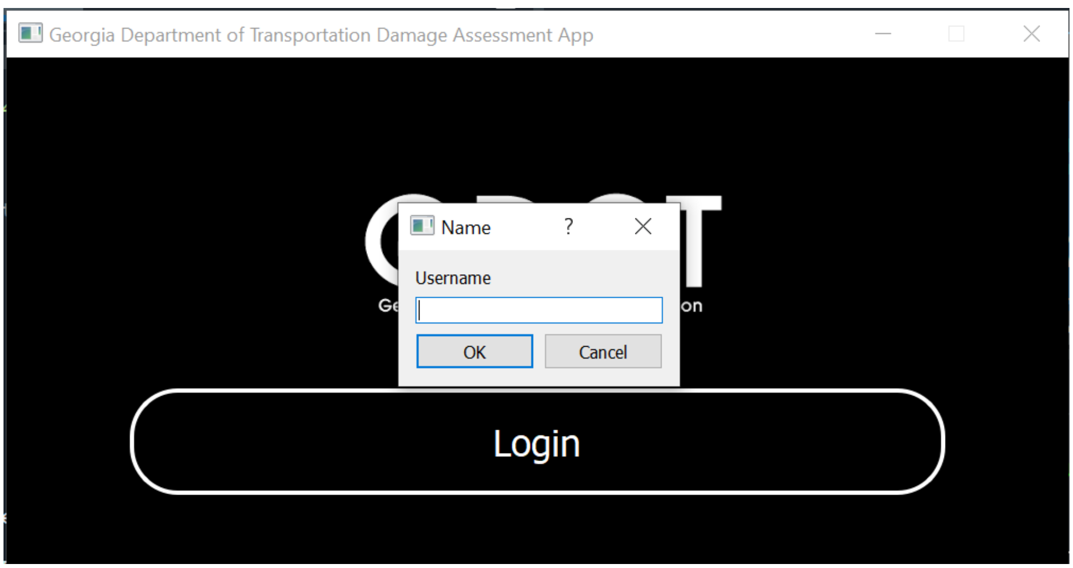

Figure 19. GUI Prompt Window for ArcGIS Account Username

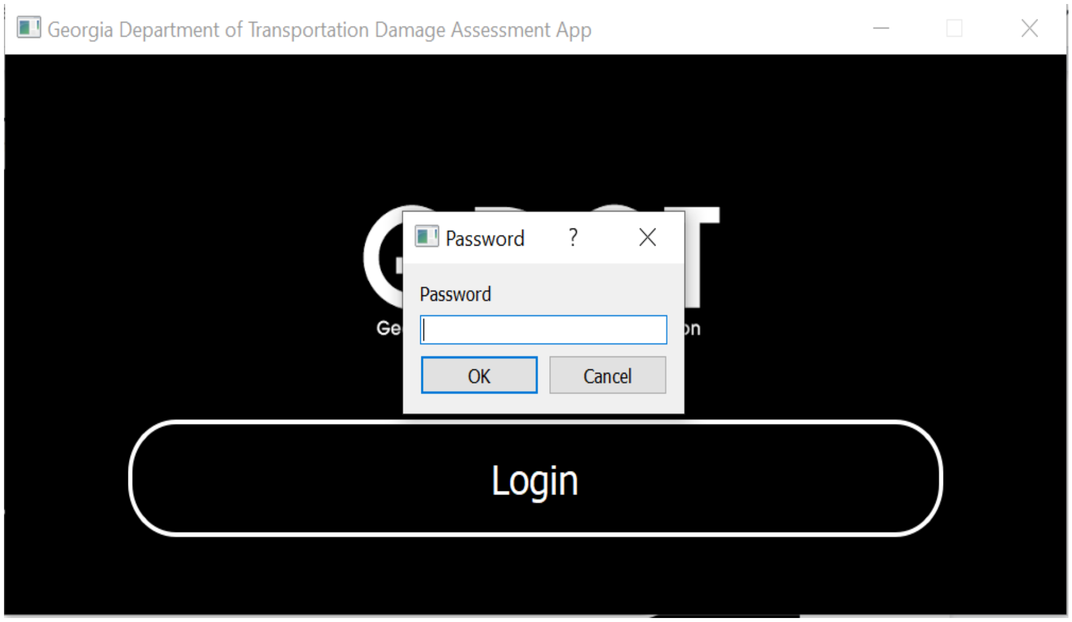

Figure 20. GUI Prompt Window for ArcGIS Account Password

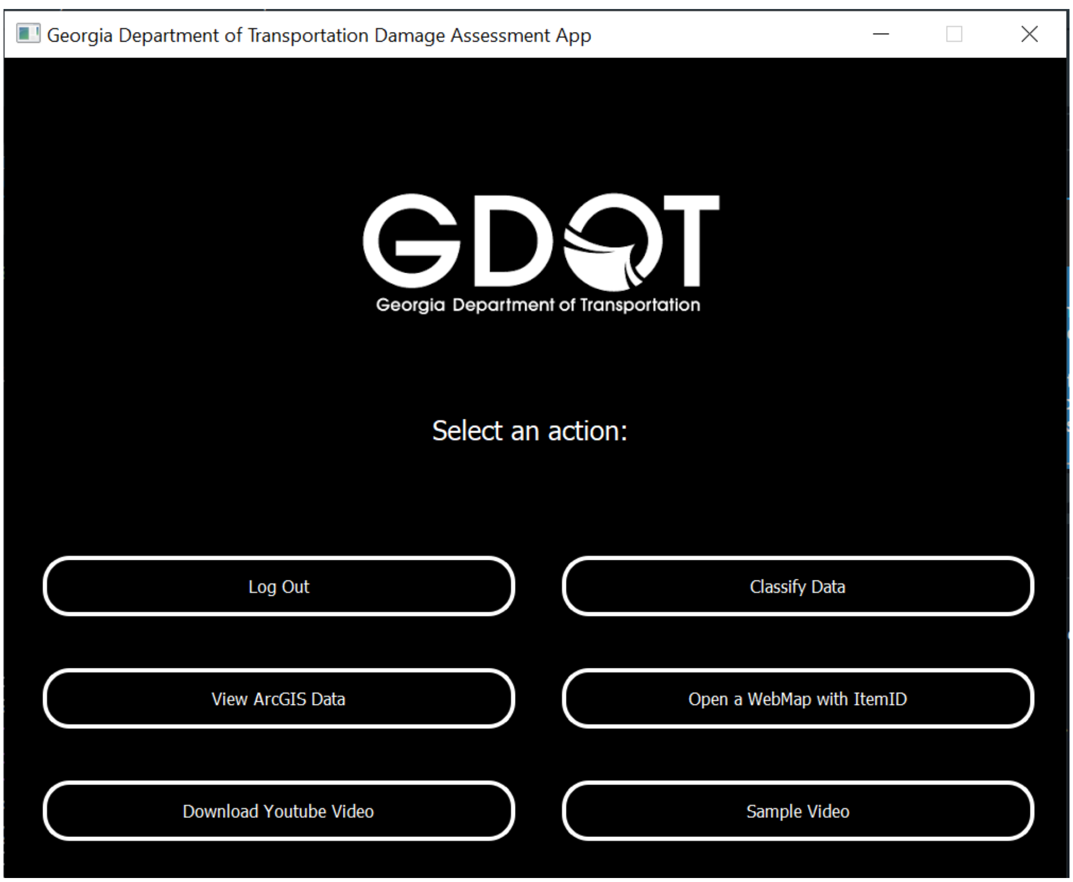

Figure 21. Application Main Window

The GUI implements many options for the user to develop disaster-damaged roads web maps neural network classification outputs. The GUI can access the data from the user's online account to showcase previously stored projects and information that is pertinent to the disaster-management system as in Figure 22. Users are also able to open online web maps from the GUI to visualize the maps, exemplified in Figures 23-25. The GUI can also initiate data classification with the Convolutional Neural Network without the user having to open MATLAB and work with multi-software. These are examples of the possible options the developed GUI provides to centralize the use of data classification with MATLAB and web map creation with ArcGIS. The centralization is a perk to the project since it has the potential to streamline natural disaster management time from first responders.

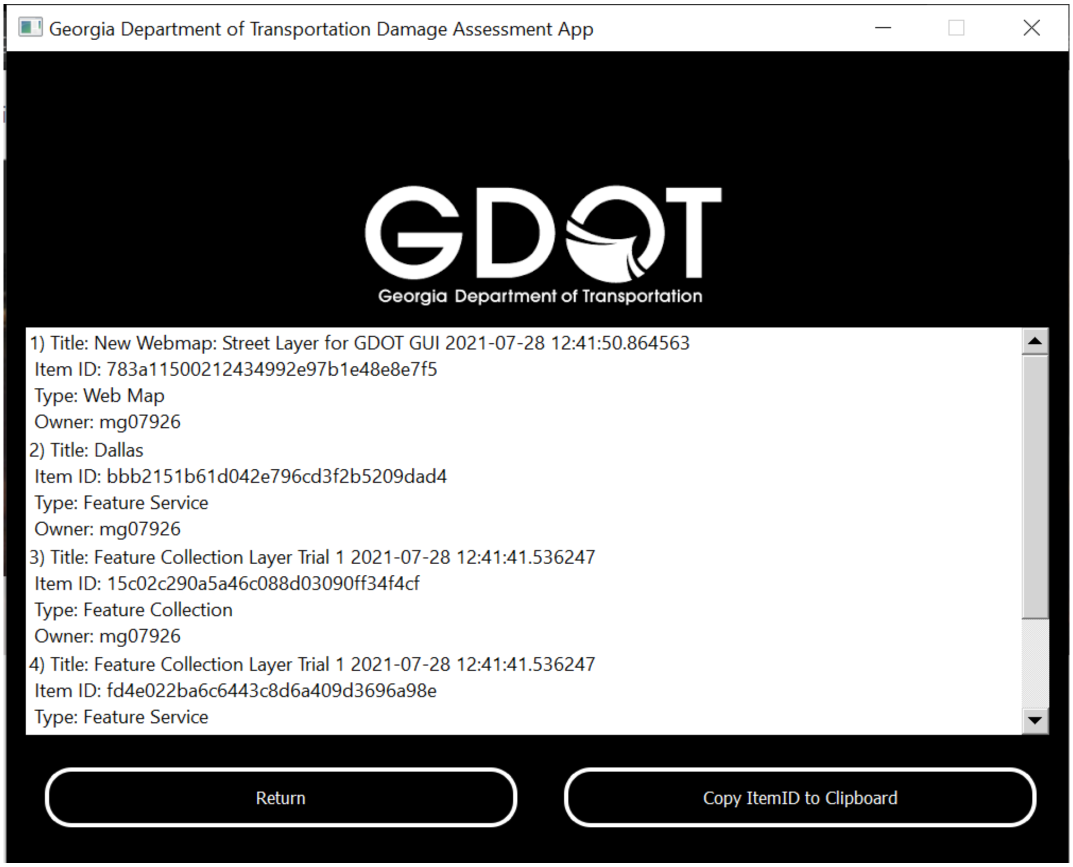

Figure 22. All ArcGIS Account Content Windowpane

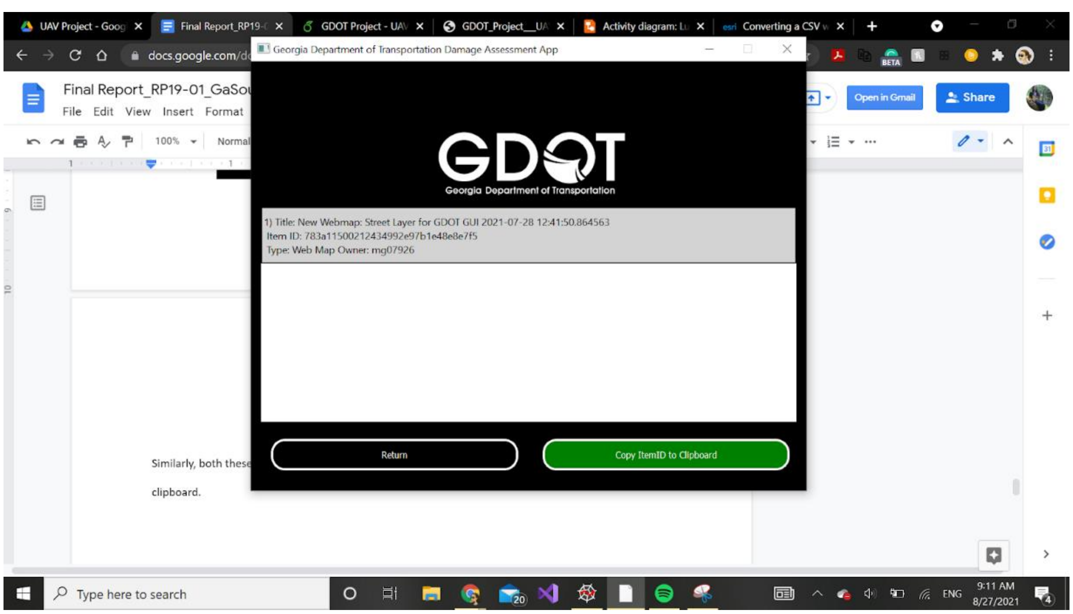

Figure 23. Copy ItemID to Clipboard from ArcGIS Account Content Window

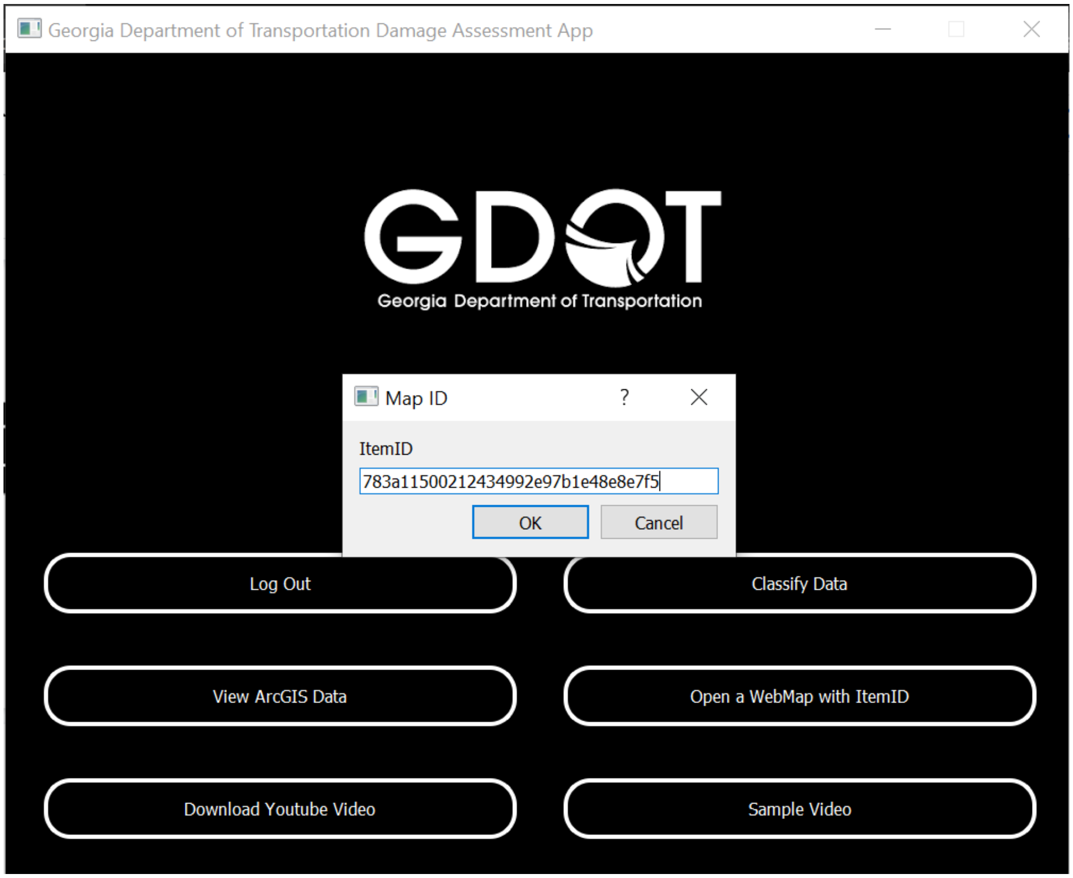

Figure 24. Paste ItemID from Clipboard to Map ID Prompt Window

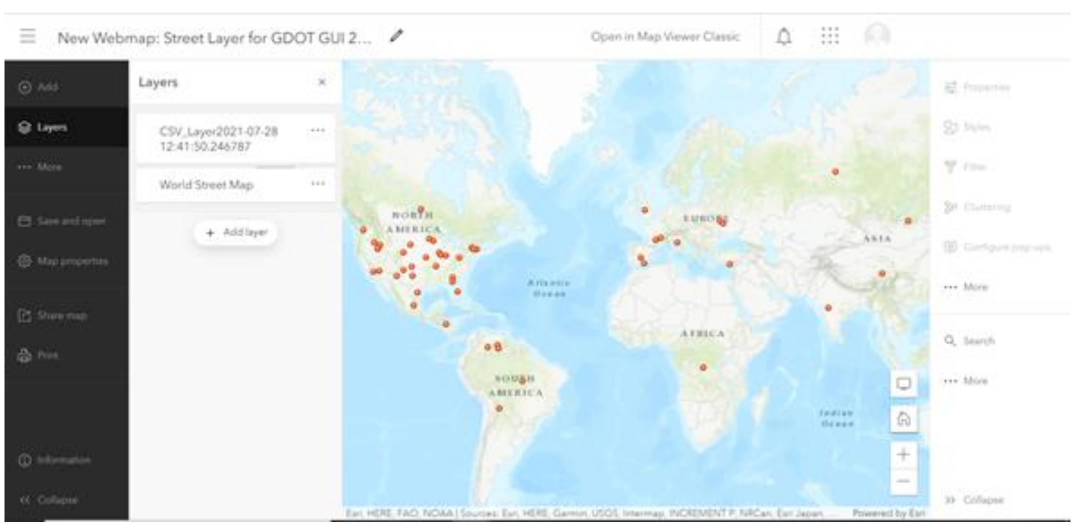

Figure 25. ArcGIS Web map Opened using the "*Open Web map with ItemID*" Button

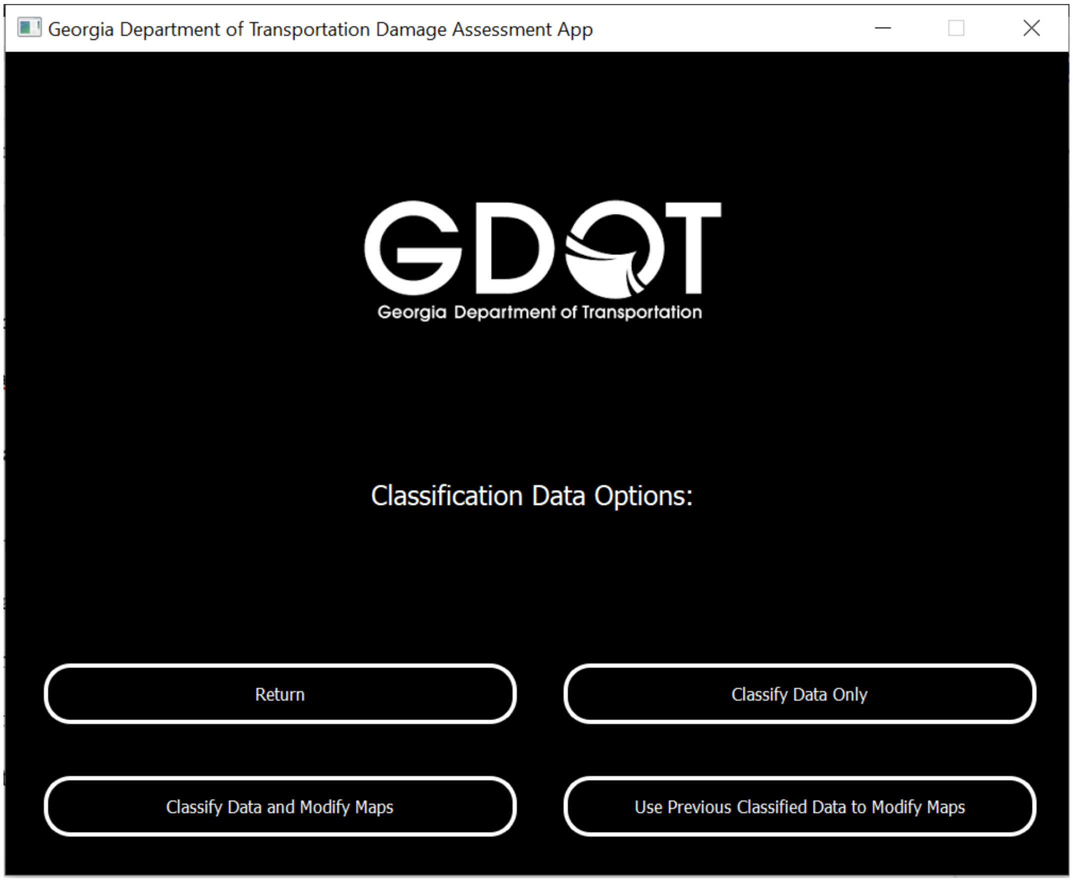

Figure 26. Data Classification and Mapping Window

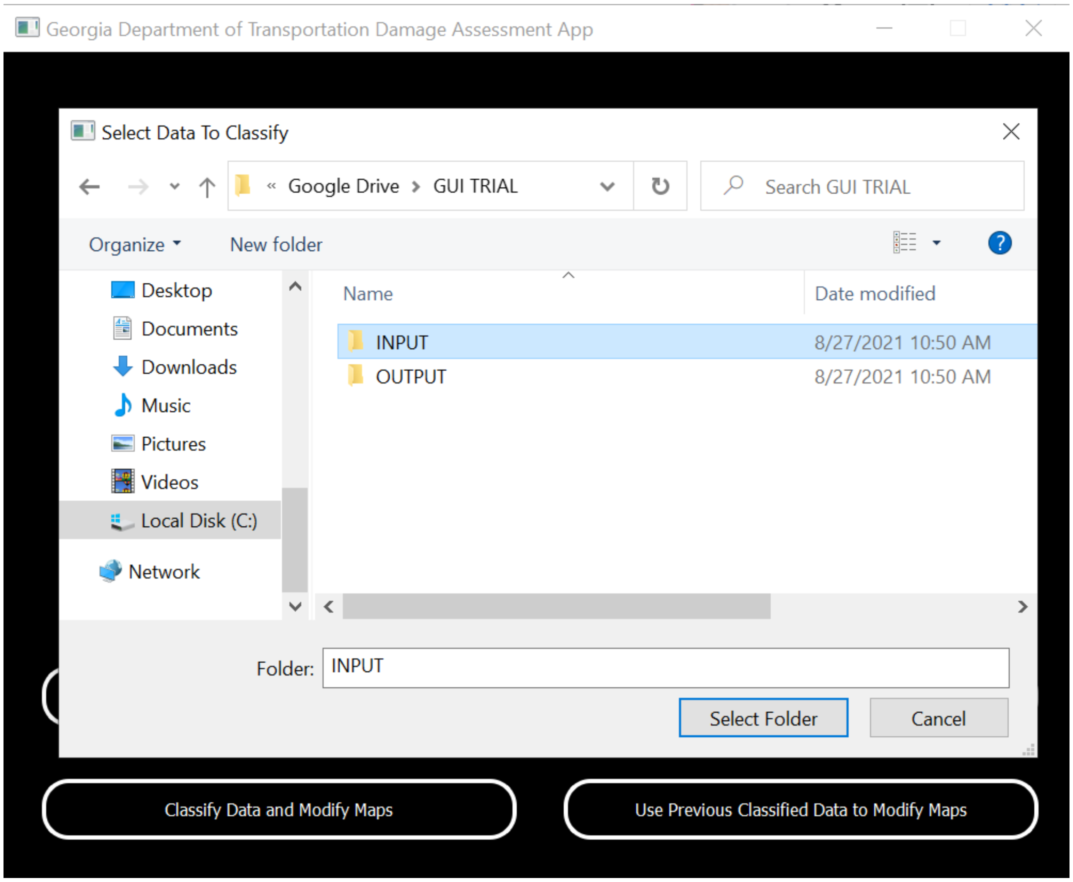

Figure 27. Data Selection for Classification

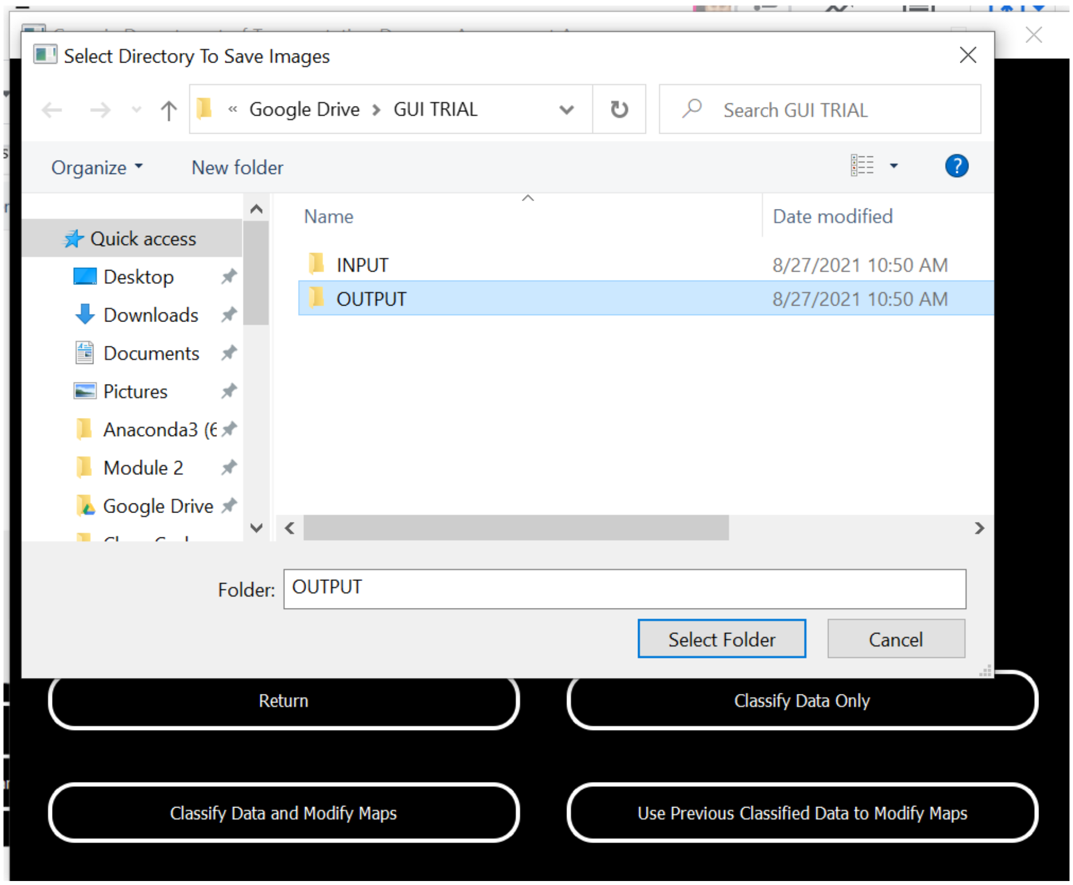

Figure 28. Folder Selection to Save Classification Output and Results

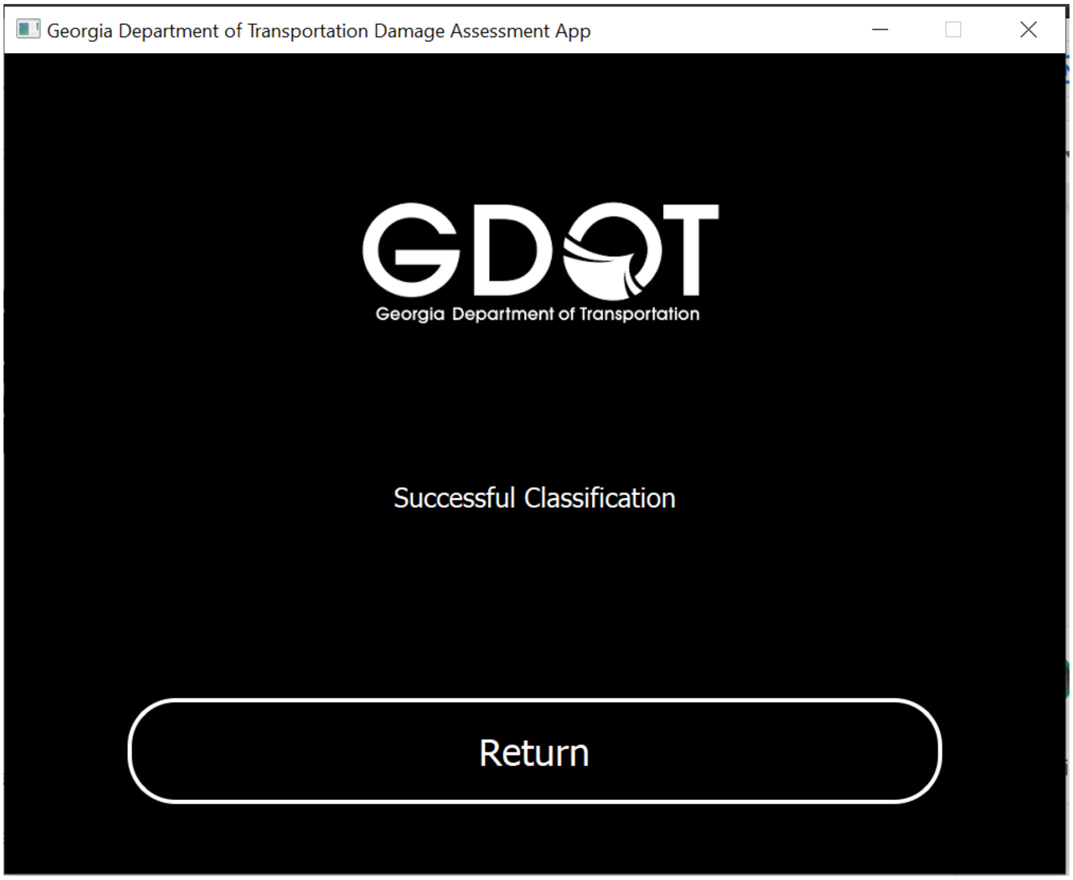

Figure 29. Successful Classification Windowpane

Chapter 4: Results

4.1 Convolutional Neural Networks' Results

The results presented in this section were obtained after training and testing three different Convolutional Neural Networks. To measure the network's performance the F1 score, precision, and recall metrics are leveraged. These metrics help understand where the model could have weaknesses. The metrics together will present a full picture of the networks' status and performance. It is worth noting that these metrics are commonly used and referenced across literature, which allows replicability and comparability.

4.1.1. Six Classes Classification

The classification accuracy is one of the main performance metrics to obtain from a network. After each network was trained and tested through K-Fold Cross-Validation, the mean of all folds per network was calculated. A total of three different CNNs (AlexNet, GoogLeNet, ResNet50) were investigated as previously established. The best network in terms of accuracy was AlexNet50 with 74.1% accuracy. The ResNet50 and GoogLeNet obtained an accuracy of 70.4% and 68.5%, respectively. Furthermore, the mean F1 score, recall, specificity, and precision were calculated as well. The classification metrics' results for each network are presented in Tables 1, 2, and 3.

Table 1. AlexNet Classification Results

Network Accuracy: 74.1%			
Classes	**Precision**	**Recall**	**F1**
Blocked Rd	0.375	0.300	0.333

Boat in Rd	1	1	1
Clear Rd	1	1	1
Damaged Rd	0.800	0.400	0.533
Flooded Rd	0.818	1	0.899
Power Lines	0.600	0.900	0.720

Table 2. GoogLeNet Classification Results

Network Accuracy: 68.5%			
Classes	**Precision**	**Recall**	**F1**
Blocked Rd	0.359	0..225	0.277
Boat in Rd	0.935	0.975	0.955
Clear Rd	0.845	1	0.916
Damaged Rd	0.609	0.500	0.549
Flooded Rd	0.731	0.527	0.613
Power Lines	0.568	0.925	0.704

Table 5-3. ResNet50 Classification Results

Network Accuracy: 70.4%			
Classes	**Precision**	**Recall**	**F1**
Blocked Rd	0.338	0.125	0.182
Boat in Rd	0.955	1	0.977
Clear Rd	0.803	1	0.891
Damaged Rd	0.733	0.550	0.628
Flooded Rd	0.866	0.723	0.788
Power Lines	0.534	0.975	0.690

Besides having the best accuracy, AlexNet was also the fastest network to train, thus requiring less computational power and time. The other deeper and more complex architectures were not able to produce better results. On the other hand, ResNet50 is capable of classifying with a better accuracy with larger testing and training datasets [35]. The size of the image library and the multiclass parameter limited the ability of these neural networks to converge and generalize.

When observing the Recall metric on each table, it is evident that the Neural Network was getting confused between the Blocked Road and Damaged Road categories. The low ratio suggests that the data the neural networks used to train is not enough. The expansion of the dataset could allow for networks to learn more features by having more examples.

The Precision metric across the networks also showed that the Blocked Roads and Damaged Roads have low ratios across the networks. This reinstates the fact that the networks need more training with more images for these classes to learn more features and correctly differentiate between them. Figures 31-33 showcase the confusion matrix for the first fold for each Convolutional Neural Network. Next. The Blocked Road category emerged as a weakness for the network and that to improve accuracy the dataset needs to be expanded.

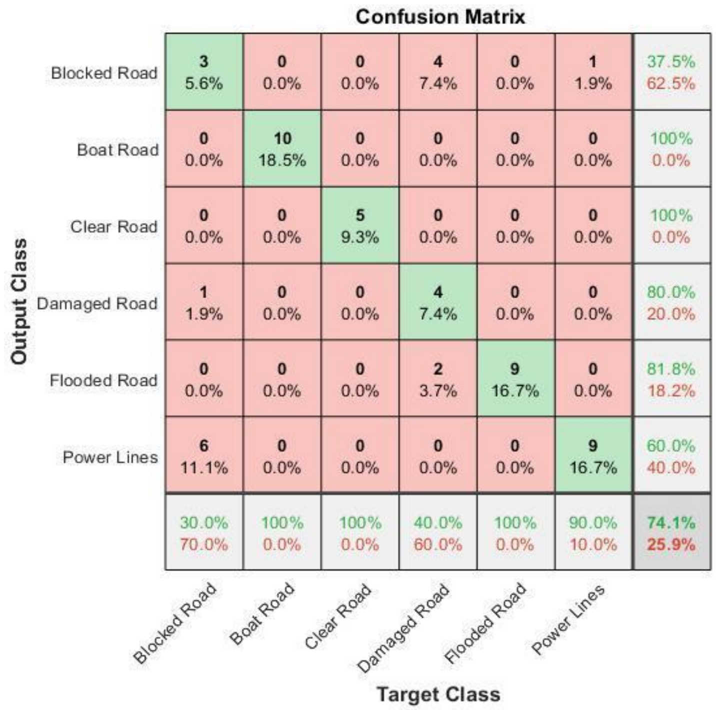

Figure 30. AlexNet Confusion Matrix

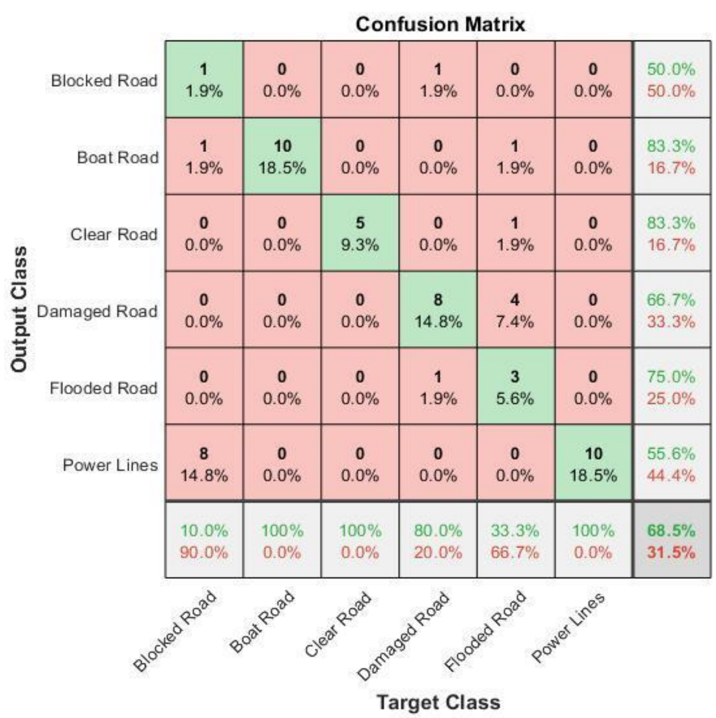

Figure 31. GoogLeNet Confusion Matrix

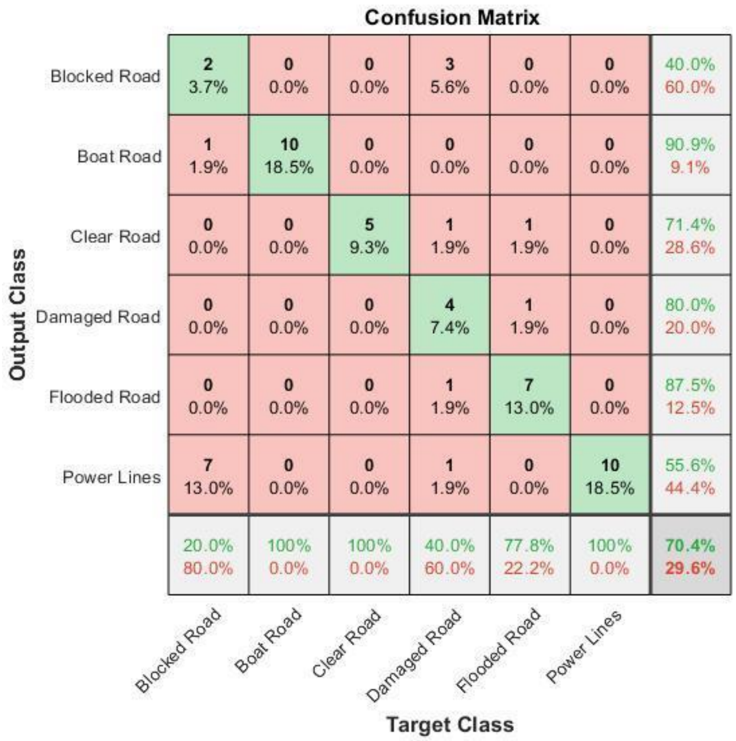

Figure 32. ResNet50 Confusion Matrix

4.1.2 Two Categories Classification

The network can also be trained to recognize if there is any type of damage or not. The dataset is split into Damaged Roads and Clean Roads. The first class, Damaged Roads, includes the previous five classes in it. While Clean Roads obtains more images into its group to balance the classes out and avoid imbalance. The classification accuracy for AlexNet with just two classes resulted in 99% accuracy. This results further prove the viability of leveraging pre-trained neural networks with Transfer Learning and K-Fold Cross-Validation. Figure 34 showcases and example of the classification results using this two-category classifier.

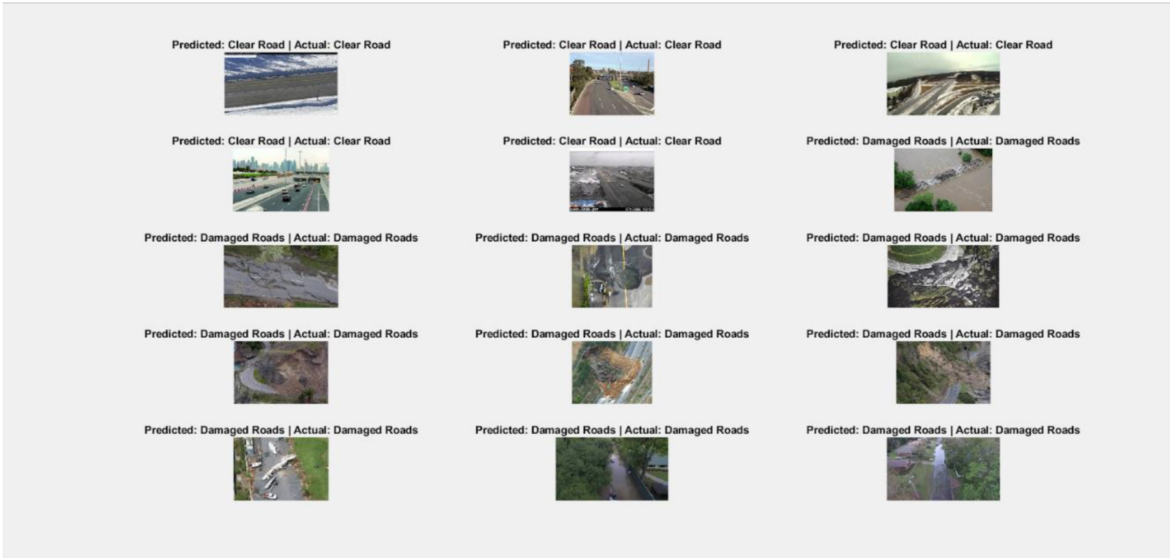

Figure 33. Two Category Neural Network Classification Output of Disaster Damaged Roads

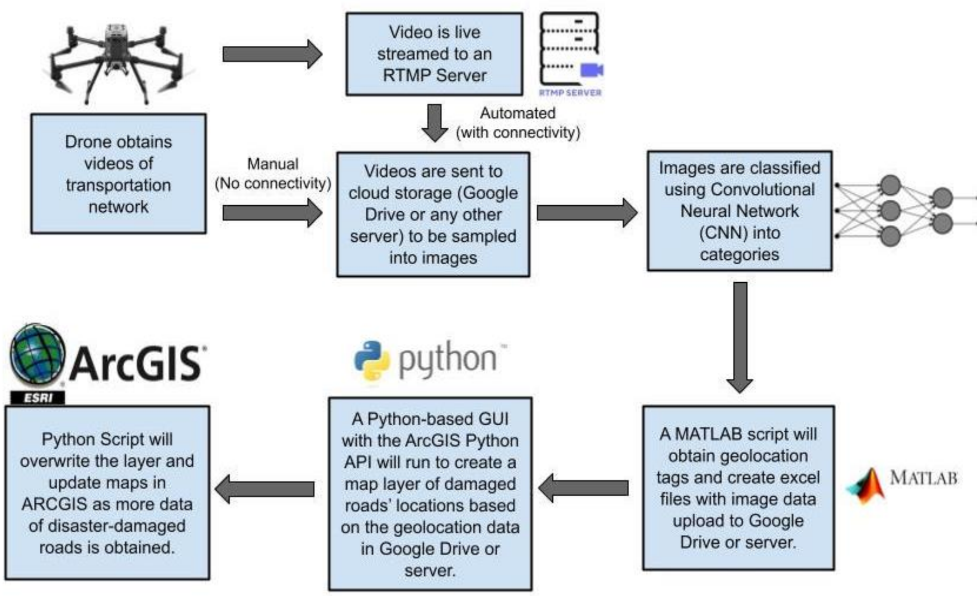

Figure 34. Complete System Flowchart

Chapter 5: Conclusion

5.1 Conclusion

This thesis sought to develop an unmanned aerial vehicle-based automated disaster assessment system with Convolutional Neural Networks to assess damages on roads caused by natural disasters. The DJI Matrice 300 RTK is leveraged to capture bird's eye view videos of roads after a natural disaster. The information is sent back to ground station, where the developed assessment system can be launched from the Graphical User Interface. The system implements a customized GUI application developed using Python 3 and MATLAB software. The GUI helps automate and centralize the operation of the classification of disaster damaged roads and the managing, sampling, and ArcGIS map tagging of the UAV-generated information.

The system was extensively simulated and tested to assess its effectiveness. The Classification Neural Networks tested were AlexNet, GoogLeNet, ResNet50. These networks were investigated after applying transfer learning and utilizing four-fold cross-validation for maximum learning efficiency. AlexNet achieved the highest accuracy of 74.1%. Moreover, the reduction of classification classes to just two improved the network's accuracy to 99%. The metrics used to analyze the network's performance also provided insight into the weaknesses the network had. The most explicit one being the classification confusion it showcases with

5.2 Recommendations

Some areas of consideration for future work are the improvement of the image library. The library size had a deep impact in the learning capacities of the Convolutional Neural Network. The library also contains varying degrees of image quality which can also affect the learning capacities of the Neural Network. Possible avenues to improve the multiclass accuracy include the expansion of the library as well as maintaining a stable balance across the classes.

References

[1] NOAA, National Centers for Environmental Information (NCEI) U.S. Billion-Dollar Weather and Climate Disasters (2021). https://www.ncdc.noaa.gov/billions , DOI: 10.25921/stkw-7w73.

[2] IFRC. (n.d.). *World disasters report 2020: Come heat or high water - tackling the HUMANITARIAN impacts of the climate Crisis Together [en/ar] - World*. ReliefWeb. Retrieved September 17, 2021, from https://reliefweb.int/report/world/world-disasters-report-2020-come-heat-or-high-water-tackling-humanitarian-impacts.

[3] Tetila, E. C., Machado, B. B., Menezes, G. K., Oliveira, A. D. S., Alvarez, M., Amorim, W. P., ... & Pistori, H. (2019). Automatic recognition of soybean leaf diseases using UAV images and deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 17(5), 903-907.

[4] Comfort, L. K., Ko, K., & Zagorecki, A. (2004). Coordination in rapidly evolving disaster response systems: The role of information. *American behavioral scientist*, 48(3), 295-313.

[5] Zhu, Y. J., Hu, Y., & Collins, J. M. (2020). Estimating road network accessibility during a hurricane evacuation: A case study of hurricane Irma in Florida. *Transportation research part D: transport and environment*, *83*, 102334.

[6] LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995.

[7] Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

[8] Prabhu. (2018, March 15). *CNN architectures - LeNet, Alexnet, VGG, GoogLeNet AND RESNET*. Medium. Retrieved September 16, 2021, from https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848.

[9] Whatmough, P. N., Zhou, C., Hansen, P., Venkataramanaiah, S. K., Seo, J. S., & Mattina, M. (2019). Fixynn: Efficient hardware for mobile computer vision via transfer learning. *arXiv preprint arXiv:1902.11128*.

[10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communication of ACM* Vol. 60, Issue 6 (June 2017), 84–90.

[11] Dwivedi, P. (2019, March 27). Understanding and coding a ResNet in Keras. Medium. Retrieved September 16, 2021, from https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33.

[12] NCDOT, 2018, PROJECT: Improved Approaches to Environmental Compliance During Highway Construction, TRID Database retrieved Sept. 1, 2021.

[13] CTECH, 2018, PROJECT: Tracking Shoreline Conditions to Protect, TRID Database as retrieved Sept. 1, 2021.

[14] MPC, 2017, PROJECT: Development of Unmanned Aerial Vehicle (UAV) Bridge Inspection Procedures, TRID Database as retrieved Sept. 1, 2021.

[15] TCSCS, 2017, PROJECT: Bridge Inspecting with Unmanned Aerial Vehicles R&D, TRID Database as retrieved Sept. 1, 2021.

[16] TRB, 2016, PROJECT: Railroad Bridge Inspections for Maintenance and Replacement Prioritization Using Unmanned Aerial (UAVs) with Laser Scanning Capabilities, TRID Database as retrieved Sept. 1, 2021.

[17] Bupe, P., Haddad, R. J., Rios, F., (2015). Relief and Emergency Communication Network Based on an Autonomous Decentralized UAV Clustering Network, *IEEE SoutheastCon* 2015.

[18] Ro, A., Oh, J., Dong, L. (2007). Lessons Learned: Application of Small UAV for Urban Highway Traffic Monitoring, *45th AIAA Aerospace Sciences Meeting, and Exhibit, Aerospace Sciences Meetings* 2007.

[19] “Eyes of the Army” U.S. Army Roadmap for UAS 2010-2035. https://irp.fas.org/program/collect/uas-army.pdf.

[20] Stewart, S.R., Berg, R., (2019) National hurricane center tropical cyclone report Hurricane Florence (Al062018) Nat. Hurricane Cent. (2019), p. 9.

[21] *Zenmuse H20 Series – unleash the power of one*. DJI. (n.d.). Retrieved Sept 2, 2021, from https://www.dji.com/zenmuse-h20-series.

[22] *MATRICE 600 Pro - DJI. DJI Official. (n.d.). Retrieved Sept. 2, 2021, from https://www.dji.com/matrice600-pro*.

[23] *MATRICE 300 RTK - DJI. DJI Official. (n.d.). Retrieved Sept. 2, 2021, from https://www.dji.com/matrice-300*.

[24] *PHANTOM 4 RTK - DJI. DJI Official. (n.d.). Retrieved Sept. 2, 2021, from https://www.dji.com/phantom-4-rtk*.

[25] *Yuneec H520 RTK - YUNEEC Official. (n.d.). Retrieved Sept. 2, 2021, from https://us.yuneec.com/h520-series/*.

[26] *Freefly Alta 8 Pro - ALTA Official. (n.d.). Retrieved Sept. 2, 2021, from https://freeflysystems.com/alta-8*.

[27] *Mavic 2 Pro - DJI. DJI Official. (n.d.). Retrieved Sept. 2, 2021, from https://www.dji.com/mavic-2*

[28] *MATRICE 210 V2 RTK - DJI. DJI Official. (n.d.). Retrieved Sept. 2, 2021, from https://www.dji.com/matrice-200-series-v2*

[29] Mariluz, O. (2019, May 27). *Earthquake in Loreto | What roads in the country are blocked or restricted after the earthquake?* RPP. Retrieved Sept. 8, 2021, from https://

https://rpp.pe/peru/loreto/terremoto-en-loreto-que-carreteras-del-pais-estan-bloqueadas-o-con-paso-restringido-tras-el-sismo-noticia-1199255.

[30] Sisson, P. (2020, January 22). *How the country's deadliest city for pedestrians plans to save lives with safer streets*. Curbed. Retrieved Sept. 8, 2021, from https://archive.curbed.com/2020/1/22/21064325/orlando-crash-cycling-pedestrian-traffic-safety.

[31] Shelton, C. (2021, August 30). *Waverly crews rushing to remove debris, Distribute TARPS ahead of IDA REMNANTS REACHING TN*. WZTV. Retrieved Sept. 8, 2021, from https://fox17.com/news/local/waverly-crews-rushing-to-remove-debris-distribute-tarps-ahead-of-ida-remnants-reaching-tn.

[32] Telegraph Media Group. (2011, March 18). *Japan earthquake: 30 pictures of boats and ships swept ashore by the tsunami*. The Telegraph. Retrieved Sept. 18, 2021, from https://www.telegraph.co.uk/news/picturegalleries/worldnews/8390718/Japan-earthquake-30-pictures-of-boats-and-ships-swept-ashore-by-the-tsunami.html?image=14.

[33] Everton Bailey Jr. | The Oregonian/OregonLive. (2012, November 20). *Heavy rain, winds leave thousands of Oregonians without power*. Oregonlive. Retrieved Sept 18, 2021, from https://www.oregonlive.com/pacific-northwest-news/2012/11/heavy_rain_winds_leave_thousan.html.

[34] Staff, N. B. C. (n.d.). Flooding shuts down Darlington Bridge until Early SAT., closes SCHOOLS. https://www.nbc15.com. Retrieved Sept. 18, 2021, from https://www.nbc15.com/content/news/Flooding-closes-Main-Street-bridge-schools-in-Darlington-562054731.html.

[35] Bocanegra, M. G., and Haddad, R. J., (2021) Convolutional Neural Network-Based Disaster Assessment Using Unmanned Aerial Vehicles, *SoutheastCon 2021*, pp. 1-6.

[36] Synced, Synced, About Synced Machine Intelligence | Technology & Industry | Information & Analysis, Synced, A., Machine Intelligence | Technology & Industry | Information & Analysis, & Name. (2017, May 9). *A guide to receptive field arithmetic for Convolutional Neural Networks*. Synced. Retrieved April 30, 2022, from https://syncedreview.com/2017/05/11/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks/

[37] S. M. A. Navid, S. H. Priya, N. H. Khandakar, Z. Ferdous and A. B. Haque, "Signature Verification Using Convolutional Neural Network," 2019 IEEE International Conference on Robotics, Automation, Artificial-intelligence and Internet-of-Things (RAAICON), 2019, pp. 35-39, doi: 10.1109/RAAICON48939.2019.19.

[38] *Quantum Transfer Learning*. PennyLane. (n.d.). Retrieved April 30, 2022, from https://pennylane.ai/qml/demos/tutorial_quantum_transfer_learning.html

Appendix A: Convolutional Neural Network Training MATLAB Code (AlexNet)

Retrain.m

```
function [rslt] = retrain(url1,url2)
%% Load Data
% This allows me to create labels for the
different roads.
allImages = imageDatastore(url1,
'IncludeSubfolders', true,...
    'LabelSource', 'foldernames')
%%
% Split data into training and test sets
[trainingImages, testImages] =
splitEachLabel(allImages, 0.7, 'randomize'); %
was 0.8
%% Load Pretrained Network (transfer learning)
% Load Pre-trained Network (GoogLeNet)
net = alexnet;
%%
answer = questdlg('Would you like to see the
training process?', ...
    'Training Process', ...
    'Yes','No','No');

switch answer
    case 'Yes'
        %% Use analyzeNetwork to display an
interactive visualization of the network
architecture and detailed information about the
network layers.
        analyzeNetwork(net)
        net.Layers(1) % check input layer
        inputSize = net.Layers(1).InputSize;

        %% Replace Final Layers
```

```
        % The convolutional layers of the
network extract image features that the last
learnable layer and the final classification
        % layer use to classify the input
image. These two layers, 'loss3-classifier' and
'output' in GoogLeNet, contain information
        % on how to combine the features that
the network extracts into class probabilities

        % Extract the layer graph from the
trained network. If the network is a
SeriesNetwork object, such as AlexNet, VGG-16,
or
        % VGG-19, then convert the list of
layers in net.Layers to a layer graph.

        if isa(net,'SeriesNetwork')
          lgraph = layerGraph(net.Layers);
        else
          lgraph = layerGraph(net);
        end

        %% Replace last layers
        % Find the names of the two layers to
replace. You can do this manually or you can
use the supporting function findLayersToReplace
        % to find these layers automatically

        [learnableLayer,classLayer] =
findLayersToReplace(lgraph);
        [learnableLayer,classLayer]
        %% Classes
        % In most networks, the last layer with
learnable weights is a fully connected layer.
Replace this fully connected layer with
        % a new fully connected layer with the
number of outputs equal to the number of
classes in the new data set
```

```
        numClasses =
numel(categories(trainingImages.Labels));

        if
isa(learnableLayer,'nnet.cnn.layer.FullyConnect
edLayer')
            newLearnableLayer =
fullyConnectedLayer(numClasses, ...
                'Name','new_fc', ...
                'WeightLearnRateFactor',10, ...
                'BiasLearnRateFactor',10);

        elseif
isa(learnableLayer,'nnet.cnn.layer.Convolution2
DLayer')
            newLearnableLayer =
convolution2dLayer(1,numClasses, ...
                'Name','new_conv', ...
                'WeightLearnRateFactor',10, ...
                'BiasLearnRateFactor',10);
        end

        lgraph =
replaceLayer(lgraph,learnableLayer.Name,newLear
nableLayer);

        %% Replace classification layers
        % The classification layer specifies
the output classes of the network. Replace the
classification layer
        % with a new one without class labels.
trainNetwork automatically sets the output
classes of the layer at
        % training time.
        newClassLayer =
classificationLayer('Name','new_classoutput');
```

```
        lgraph =
replaceLayer(lgraph,classLayer.Name,newClassLay
er);
        %% Check class layer connection

figure('Units','normalized','Position',[0.3 0.3
0.4 0.4]);
        plot(lgraph)
        ylim([0,10])

        %% Extract the layers and connections
of the layer graph and select which layers to
freeze.
        % The new layer graph contains the same
layers, but with the learning rates of the
earlier layers set to zero.
        layers = lgraph.Layers;
        connections = lgraph.Connections;

        layers(1:10) =
freezeWeights(layers(1:10));
        lgraph =
createLgraphUsingConnections(layers,connections
);

        %% Train Network
        % The network requires input images of
size 224-by-224-by-3, but the images in the
image datastore have different
        % sizes. Use an augmented image
datastore to automatically resize the training
images. Specify additional
        % augmentation operations to perform on
the training images: randomly flip the training
images along the
        % vertical axis and randomly translate
them up to 30 pixels and scale them up to 10%
horizontally and vertically.
```

```
        % Data augmentation helps prevent the
network from overfitting and memorizing the
exact details of the training
        % images.

        pixelRange = [-30 30];
        scaleRange = [0.9 1.1];
        imageAugmenter = imageDataAugmenter(
...
            'RandXReflection',true, ...
            'RandXTranslation',pixelRange, ...
            'RandYTranslation',pixelRange, ...
            'RandXScale',scaleRange, ...
            'RandYScale',scaleRange);
        augimdsTrain =
augmentedImageDatastore(inputSize(1:2),training
Images, ...
            'DataAugmentation',imageAugmenter);
        %% Validation set datastore size
processing
        % To automatically resize the
validation images without performing further
data augmentation,
        % use an augmented image datastore
without specifying any additional preprocessing
operations.
        augimdsValidation =
augmentedImageDatastore(inputSize(1:2),testImag
es);

        %% Specify the training options.
        % Set InitialLearnRate to a small value
to slow down learning in the transferred layers
that
        % are not already frozen. In the
previous step, you increased the learning rate
factors for the
```

```
        % last learnable layer to speed up
learning in the new final layers. This
combination of learning
        % rate settings results in fast
learning in the new layers, slower learning in
the middle layers,
        % and no learning in the earlier,
frozen layers.

        % Specify the number of epochs to train
for. When performing transfer learning, you do
not need to
        % train for as many epochs. An epoch is
a full training cycle on the entire training
data set.
        % Specify the mini-batch size and
validation data. Compute the validation
accuracy once per epoch.
        % Max Epoch was 6
        miniBatchSize = 10;
        valFrequency =
floor(numel(augimdsTrain.Files)/miniBatchSize);
        options = trainingOptions('sgdm', ...
            'MiniBatchSize',miniBatchSize, ...
            'MaxEpochs',10, ...
            'InitialLearnRate',3e-4, ...
            'Shuffle','every-epoch', ...
            'ValidationData',augimdsValidation,
...
            'ValidationFrequency',valFrequency,
...
            'Verbose',false, ...
            'Plots','training-progress');
        %% Train Network Command
        % Train the network using the training
data. By default, trainNetwork uses a GPU if
one is available
```

```
        % (requires Parallel Computing Toolbox™
and a CUDA® enabled GPU with compute capability
3.0 or higher).
        % Otherwise, trainNetwork uses a CPU.
You can also specify the execution environment
by using the
        % 'ExecutionEnvironment' name-value
pair argument of trainingOptions. Because the
data set is so small,
        % training is fast.

        net =
trainNetwork(augimdsTrain,lgraph,options);

        %% Classify Validation Images
        % Classify the validation images using
the fine-tuned network, and calculate the
classification accuracy.
        [YPred,probs] =
classify(net,augimdsValidation);
        accuracy = mean(YPred ==
testImages.Labels)
        %% Save the train network to a .mat
file
        cd(url2);
        FileName=['retrain_',datestr(now, 'dd-
mmm-yyyy-HH:MM')]
        save(FileName,'net');

        %% Display Sample Validation Images
        % Display four sample validation images
with predicted labels and the predicted
probabilities of the images
        % having those labels.
        idx =
randperm(numel(testImages.Files),4);
        figure
        for i = 1:4
```

```
                subplot(2,2,i)
                I = readimage(testImages,idx(i));
                imshow(I)
                label = YPred(idx(i));
                title(string(label) + ", " +
num2str(100*max(probs(idx(i),:)),3) + "%");
        end
        rslt = 'MATLAB Script Finished'
    case 'No'
        %% Use analyzeNetwork to display an
interactive visualization of the network
architecture and detailed information about the
network layers.
%         analyzeNetwork(net)
        net.Layers(1) % check input layer
        inputSize = net.Layers(1).InputSize;

        %% Replace Final Layers
        % The convolutional layers of the
network extract image features that the last
learnable layer and the final classification
        % layer use to classify the input
image. These two layers, 'loss3-classifier' and
'output' in GoogLeNet, contain information
        % on how to combine the features that
the network extracts into class probabilities

        % Extract the layer graph from the
trained network. If the network is a
SeriesNetwork object, such as AlexNet, VGG-16,
or
        % VGG-19, then convert the list of
layers in net.Layers to a layer graph.

        if isa(net,'SeriesNetwork')
          lgraph = layerGraph(net.Layers);
        else
          lgraph = layerGraph(net);
```

```
        end

        %% Replace last layers
        % Find the names of the two layers to
replace. You can do this manually or you can
use the supporting function findLayersToReplace
        % to find these layers automatically

        [learnableLayer,classLayer] =
findLayersToReplace(lgraph);
        [learnableLayer,classLayer]
        %% Classes
        % In most networks, the last layer with
learnable weights is a fully connected layer.
Replace this fully connected layer with
        % a new fully connected layer with the
number of outputs equal to the number of
classes in the new data set
        numClasses =
numel(categories(trainingImages.Labels));

        if
isa(learnableLayer,'nnet.cnn.layer.FullyConnect
edLayer')
            newLearnableLayer =
fullyConnectedLayer(numClasses, ...
                'Name','new_fc', ...
                'WeightLearnRateFactor',10, ...
                'BiasLearnRateFactor',10);

        elseif
isa(learnableLayer,'nnet.cnn.layer.Convolution2
DLayer')
            newLearnableLayer =
convolution2dLayer(1,numClasses, ...
                'Name','new_conv', ...
                'WeightLearnRateFactor',10, ...
                'BiasLearnRateFactor',10);
```

```
        end

        lgraph = 
replaceLayer(lgraph,learnableLayer.Name,newLear
nableLayer);

        %% Replace classification layers
        % The classification layer specifies 
the output classes of the network. Replace the 
classification layer
        % with a new one without class labels. 
trainNetwork automatically sets the output 
classes of the layer at
        % training time.
        newClassLayer = 
classificationLayer('Name','new_classoutput');
        lgraph = 
replaceLayer(lgraph,classLayer.Name,newClassLay
er);
        %% Check class layer connection

%figure('Units','normalized','Position',[0.3 
0.3 0.4 0.4]);
        %plot(lgraph)
        %ylim([0,10])

        %% Extract the layers and connections 
of the layer graph and select which layers to 
freeze.
        % The new layer graph contains the same 
layers, but with the learning rates of the 
earlier layers set to zero.
        layers = lgraph.Layers;
        connections = lgraph.Connections;

        layers(1:10) = 
freezeWeights(layers(1:10));
```

```
        lgraph =
createLgraphUsingConnections(layers,connections
);

        %% Train Network
        % The network requires input images of
size 224-by-224-by-3, but the images in the
image datastore have different
        % sizes. Use an augmented image
datastore to automatically resize the training
images. Specify additional
        % augmentation operations to perform on
the training images: randomly flip the training
images along the
        % vertical axis and randomly translate
them up to 30 pixels and scale them up to 10%
horizontally and vertically.
        % Data augmentation helps prevent the
network from overfitting and memorizing the
exact details of the training
        % images.

        pixelRange = [-30 30];
        scaleRange = [0.9 1.1];
        imageAugmenter = imageDataAugmenter(
...
            'RandXReflection',true, ...
            'RandXTranslation',pixelRange, ...
            'RandYTranslation',pixelRange, ...
            'RandXScale',scaleRange, ...
            'RandYScale',scaleRange);
        augimdsTrain =
augmentedImageDatastore(inputSize(1:2),training
Images, ...
            'DataAugmentation',imageAugmenter);
        %% Validation set datastore size
processing
```

```
        % To automatically resize the
validation images without performing further
data augmentation,
        % use an augmented image datastore
without specifying any additional preprocessing
operations.
        augimdsValidation =
augmentedImageDatastore(inputSize(1:2),testImag
es);

        %% Specify the training options.
        % Set InitialLearnRate to a small value
to slow down learning in the transferred layers
that
        % are not already frozen. In the
previous step, you increased the learning rate
factors for the
        % last learnable layer to speed up
learning in the new final layers. This
combination of learning
        % rate settings results in fast
learning in the new layers, slower learning in
the middle layers,
        % and no learning in the earlier,
frozen layers.

        % Specify the number of epochs to train
for. When performing transfer learning, you do
not need to
        % train for as many epochs. An epoch is
a full training cycle on the entire training
data set.
        % Specify the mini-batch size and
validation data. Compute the validation
accuracy once per epoch.
        % Max Epoch was 6
        miniBatchSize = 10;
```

```
        valFrequency =
floor(numel(augimdsTrain.Files)/miniBatchSize);
        options = trainingOptions('sgdm', ...
            'MiniBatchSize',miniBatchSize, ...
            'MaxEpochs',10, ...
            'InitialLearnRate',3e-4, ...
            'Shuffle','every-epoch', ...
            'ValidationData',augimdsValidation,
...
            'ValidationFrequency',valFrequency,
...
            'Verbose',false, ...
            'Plots','none');
        %% Train Network Command
        % Train the network using the training
data. By default, trainNetwork uses a GPU if
one is available
        % (requires Parallel Computing Toolbox™
and a CUDA® enabled GPU with compute capability
3.0 or higher).
        % Otherwise, trainNetwork uses a CPU.
You can also specify the execution environment
by using the
        % 'ExecutionEnvironment' name-value
pair argument of trainingOptions. Because the
data set is so small,
        % training is fast.

        net =
trainNetwork(augimdsTrain,lgraph,options);

        %% Classify Validation Images
        % Classify the validation images using
the fine-tuned network, and calculate the
classification accuracy.
        [YPred,probs] =
classify(net,augimdsValidation);
```

```
        accuracy = mean(YPred ==
testImages.Labels)
        %% Save the train network to a .mat
file
        cd(url2);
        FileName=['retrain_',datestr(now, 'dd-
mmm-yyyy-HH:MM')]
        save(FileName,'net');
        rslt = 'MATLAB Script Finished'

end

end
```

Appendix B: Convolutional Neural Network Testing MATLAB Code (AlexNet)

TestAI_AlexNet.m

```
% Load Training Images
% In order for imageDataStore to parse the
folder names as category labels,
% you would have to store image categories in
corresponding sub-folders.
allImages = imageDatastore('TrainingData2',
'IncludeSubfolders', true,...
    'LabelSource', 'foldernames');


% Split data into training and test sets
[trainingImages, testImages] =
splitEachLabel(allImages, 0.7, 'randomize');
%%
% Load Trained AI
load('disasterReliefAI_AlexNet.mat');
%%
% Test Network Performance
% test the performance of our new "snack
recognizer" on the test set.
testImages.ReadFcn = @readFunctionTrain2;
predictedLabels = classify(net, testImages);
accuracy = mean(predictedLabels ==
testImages.Labels)

% convert categorical labels to cell arrays so
that they can be displayed
% in title
displayPredicted = cellstr(predictedLabels);
displayActual = cellstr(testImages.Labels);

% strcat concatenates strings so that the
titles do not have to be typed
```

```
% manually. See Matlab Documentation for more
information

% dislay the Results in a single Figure Window.
figure(1)
for i = 1:(length(displayActual)) %% take off
/5 for smaller dataset
    subplot(ceil(length(displayActual)/80), 2,
i) %% was 3
    imshow(testImages.Files{i})
    title(strcat("predicted label: ",
displayPredicted{i}, " | Actual Label: ",...
        displayActual{i}))
end
```

Appendix C: Geolocation Data Retrieving MATLAB Code

Get_geotags.m

```
function [f] = getGeoTags(url,
pdest,displayPredicted, k)
%% Read how many files in a folder
% cd to inside the file where the images are
cd(url);
files = dir('*.jpg') ;
number = length(files); % how many files in the
folder
% [m,n] = size(files); %% get number of images
in the dir
%% Establish an data extracted array and loop
for each file
% T(1,:) = {'Name','Date-
Time','Latitude','Longitude'}; **issue with
cell
% to table T1,T2,T3,T4
for i = 1:number
    disp(files(i).name); % displays image/file
name
    imgName = files(i).name; % variable to
store image name
    info = imfinfo(imgName); % retrieves all
info of the images
    info.GPSInfo; % retrieves the GPS info from
the images

    latitude = info.GPSInfo.GPSLatitude; %
array of 3 float numbers
    longitude = info.GPSInfo.GPSLongitude; %
array of 3 float numbers
```

```
    Final_lat = latitude(1) + latitude(2)/60 +
latitude(3)/3600; % provides numerical value
for latitude
    Final_long = longitude(1) + longitude(2)/60
+ longitude(3)/3600; % provides numerical value
for longitude

     if info.GPSInfo.GPSLatitudeRef == 'S' %
makes the Final_lat negative if the picture is
located in the South
         Final_lat = Final_lat * (-1);
     end
     if info.GPSInfo.GPSLongitudeRef == 'W' %
makes the Final_long negative if the picture is
located in the West
         Final_long = Final_long * (-1);
     end
    %imgDate = info.DateTime;
    %fprintf('latitude = %f   longitude =
%f\n', Final_lat, Final_long);
    %add if you want to showcase the latitudes
and longitudes
    T(i,:) = {imgName, Final_lat, Final_long,
displayPredicted{i}, k{i}}; %T(i,:) = {imgName,
imgDate, Final_lat, Final_long}; % i+1 when
line eight executed
end
x = cell2table(T;
%% Gives the date and time of the creation of
the file and attaches it to the file name
Filename = sprintf('GDOT-location_%s.xlsx',
datestr(now,'mm_dd_yyyy_HH_MM'));
%% Save the file in google drive folder you
want
cd(pdest);
writetable(x,sprintf('%s.csv',Filename)); %%
creates csv file in the location of the google
drive
```

```
zip(sprintf('location_%s',
datestr(now,'mm_dd_yyyy_HH_MM')),'*.jpg', url);
%'C:\Users\User\Documents\COLLEGE
CLASSES\Research\UAV\TestImageOutput');
% zip(sprintf('newtry'),'*.jpg', url); % just
for newtry
x.Properties.VariableNames = {'Image Name'
'Latitude' 'Longitude', 'Classifiation',
'Path'};%x.Properties.VariableNames = {'Image
Name' 'Date and Time' 'Latitude' 'Longitude'};
f = x;
end
```

Appendix D: K-Fold Cross-Validation Dataset Segmentation MATLAB Code

NeuralNet_CrossVal.m

```
% Copyright 2017 The MathWorks, Inc.

% Deep Learning: Transfer Learning in 10 Lines
of MATLAB Code
% Transfer learning is a very practical way to
use deep learning by
% modifying an existing deep network(usually
trained by an expert) to work
% with your data.

% Problem statement
% The problem we tried to solve with transfer
learning is to distinguish
% between 5 categories of food - cupcakes,
burgers, apple pie, hot dogs and
% ice cream. To get started you need two
things:
%
% # Training images of the different object
classes
% # A pre-trained deep neural network (AlexNet)
% You can substitute these categories for any
of your own based on what
% image data you have avaliable.

% Load Training Images
% In order for imageDataStore to parse the
folder names as category labels,
% you would have to store image categories in
corresponding sub-folders.
cd('C:\Users\User\Documents\COLLEGE
CLASSES\Research\UAV\Code');
allImages = imageDatastore('TrainingData2',
'IncludeSubfolders', true,...
```

```
    'LabelSource', 'foldernames');
%% Cross Validation Datastores for Blocked
Roads
[datastore_BLK1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Blocked Road');
[datastore_BLK2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Blocked Road');
[datastore_BLK3, datastore_BLK4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Blocked Road');
blk_arr = {datastore_BLK1 datastore_BLK2
datastore_BLK3 datastore_BLK4};
%% Cross Validation Datastores for Flooded
Roads
[datastore_FL1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Flooded Road');
[datastore_FL2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Flooded Road');
[datastore_FL3, datastore_FL4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Flooded Road');
fl_arr = {datastore_FL1 datastore_FL2
datastore_FL3 datastore_FL4};
%% Cross Validation Datastores for Clear Roads
[datastore_Clear1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Clear Road');
[datastore_Clear2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Clear Road');
[datastore_Clear3, datastore_Clear4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Clear Road');
```

```
cl_arr = {datastore_Clear1 datastore_Clear2
datastore_Clear3 datastore_Clear4};
%% Cross Validation Datastores for Power Line
Roads
[datastore_PLR1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Power Lines');
[datastore_PLR2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Power Lines');
[datastore_PLR3, datastore_PLR4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Power Lines');
plr_arr = {datastore_PLR1 datastore_PLR2
datastore_PLR3 datastore_PLR4};
%% Cross Validation Datastores for Damaged
Roads
[datastore_DR1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Damaged Road');
[datastore_DR2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Damaged Road');
[datastore_DR3, datastore_DR4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Damaged Road');
dr_arr = {datastore_DR1 datastore_DR2
datastore_DR3 datastore_DR4};
%% Cross Validation Datastores for Boat Roads
[datastore_BR1, datastoreDummy75] =
splitEachLabel(allImages, 0.25, 'Include',
'Boat Road');
[datastore_BR2, datastoreDummy50] =
splitEachLabel(datastoreDummy75, 0.335,
'Include', 'Boat Road');
[datastore_BR3, datastore_BR4] =
splitEachLabel(datastoreDummy50, 0.5,
'Include', 'Boat Road');
```

```
br_arr = {datastore_BR1 datastore_BR2
datastore_BR3 datastore_BR4};
%% Split Data
for i = 1:1:4 %% for loop chooses the 'i' batch
of test images from each Label
    testImages =
imageDatastore(cat(1,blk_arr{i}.Files,
plr_arr{i}.Files, dr_arr{i}.Files,
br_arr{i}.Files, cl_arr{i}.Files,
fl_arr{i}.Files));
    testImages.Labels =
cat(1,blk_arr{i}.Labels, plr_arr{i}.Labels,
dr_arr{i}.Labels,  br_arr{i}.Labels,
cl_arr{i}.Labels, fl_arr{i}.Labels);
    x = 1; % dummy variable to create a
training images datastore cell based on the
batches not used in each label
    ti_arr = {}; % unused images in each label
cell (stores 3 datastores)
    for j = 1:1:4 % for loop to grab the
batches that are not the test images
        if j ~= i % makes sure we do not
include the test images
            ti_arr{x} =
imageDatastore(cat(1,blk_arr{j}.Files,
plr_arr{j}.Files, dr_arr{j}.Files,
br_arr{j}.Files, cl_arr{j}.Files,
fl_arr{j}.Files));
            ti_arr{x}.Labels =
cat(1,blk_arr{j}.Labels, plr_arr{j}.Labels,
dr_arr{j}.Labels,  br_arr{j}.Labels,
cl_arr{j}.Labels, fl_arr{j}.Labels);
            x = x+1;
        end
    end
    % joins the 3 unused image batches as a
datastore
```

```
    trainingImages = imageDatastore(cat(1,
ti_arr{1}.Files, ti_arr{2}.Files,
ti_arr{3}.Files));
    trainingImages.Labels = cat(1,
ti_arr{1}.Labels, ti_arr{2}.Labels,
ti_arr{3}.Labels);

    %% End for now
    % Load Pre-trained Network (AlexNet)
    % AlexNet is a pre-trained network trained
on 1000 object categories.
    % AlexNet is avaliable as a support package
on FileExchange.
    alex = alexnet;

    % Review Network Architecture
    layers = alex.Layers

    % Modify Pre-trained Network
    % AlexNet was trained to recognize 1000
classes, we need to modify it to
    % recognize just 5 classes.
    layers(23) = fullyConnectedLayer(6); %
change this based on # of classes
    layers(25) = classificationLayer

    % Perform Transfer Learning
    % For transfer learning we want to change
the weights of the network ever so slightly.
How
    % much a network is changed during training
is controlled by the learning
    % rates.
    opts = trainingOptions('sgdm',
'InitialLearnRate', 0.0001,...
        'MaxEpochs', 15, 'MiniBatchSize', 32);
    % learning rate 0.001
    % mini batch size 64
```

```
    % Set custom read function
    % One of the great things about
imageDataStore it lets you specify a
    % "custom" read function, in this case it
is simply resizing the input
    % images to 227x227 pixels which is what
AlexNet expects. You can do this by
    % specifying a function handle of a
function with code to read and
    % pre-process the image.

    trainingImages.ReadFcn =
@readFunctionTrain3;

    % Train the Network
    % This process usually takes about 5-20
minutes on a desktop GPU.
    myNet = trainNetwork(trainingImages,
layers, opts);


    % Test Network Performance
    % Now let's the test the performance on the
test set.
    testImages.ReadFcn = @readFunctionTrain3;
    predictedLabels = classify(myNet,
testImages);
    accuracy = mean(predictedLabels ==
testImages.Labels)
    % Save the Trained AI to a .mat file

save(sprintf('AlexNetdisasterReliefAI_%d',i),'m
yNet');

%plotconfusion(testImages.Labels,predictedLabel
s)
end
```

Appendix E: Image Resizing for Neural Network Training MATLAB CODE

readFunctionTrain2.m

```
% This function simply resizes the images to
fit in AlexNet
% Copyright 2017 The MathWorks, Inc.

function I = readFunctionTrain2(filename)
% Resize the images to the size required by the
network.
I = imread(filename);
I = imresize(I, [224 224]);
end
```

Appendix F: Graphical User Interface Python Code

GUI_1.py

```
from GPSPhoto import gpsphoto
import pyperclip as pycl
import sys
import arcgis
import webbrowser
import json
import matlab
import matlab.engine
import os
import cv2
import pandas as pd
from datetime import datetime
from arcgis.gis import GIS
from arcgis.features import FeatureLayerCollection
from arcgis.mapping import WebMap, WebScene
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QPixmap
from PyQt5 import QtGui, QtCore
from PyQt5.QtGui import QCursor
from PyQt5.QtCore import QDir
from tkinter import Tk     # from tkinter import Tk for
Python 3.x
from tkinter.filedialog import *
from os import path
from pathlib import Path
import youtube_dl
from googleapiclient.discovery import build



widgets = {
    "listWidget": [],
    "logo": [],
    "button": [],
    "question": [],
    "Return": [],
    "Classify Data": [],
    "View ArcGIS Data": [],
    "uname": [],
    "psw": [],
    "the_list": [],
    "All Content": [],
```

```
    "Search by Keyword": [],
    "Search by Title": [],
    "Select Data to Classify": [],
    "Open a Map with ItemID": [],
    "Create New Map":[],
    "Append Data to a Map":[],
    "Log out":[],
    "Overwrite a Map": [],
    "Copy ItemID": [],
    "Download and Sample Video":[],
    "Change Selected Map": [],
    "Download": [],
    "Use Previous Classified Data to Modify Maps": [],
    "Retrain Network": []
    }



app = QApplication(sys.argv)

window = QWidget()    #window widget
window.setWindowTitle("Georgia Department of Transportation
Damage Assessment App")
window.setFixedWidth(1000)

#window.move(2700, 200)
window.setStyleSheet("background-color: black;")

grid = QGridLayout()

global my_contt

################## Threads ###########################
import threading
from threading import Thread
from time import sleep
################# END THREADS ########################
def connectMatlab(url1, url2):
    try:
        eng = matlab.engine.start_matlab()
        x = eng.kk(url1, url2)
        print(x)
        sleep(1)
        showDialog("Successful Classification.")

    except:
```

```
        showDialog("Unsuccessful Classification. Try
again.")


    #sys.exit() # kill thread once function is done to
preserve computational power

def connectMatlab2(url1,url2):
    try:
        eng = matlab.engine.start_matlab()
        x = eng.retrain(url1,url2)
        print(x)
        sleep(1)
        showDialog("Successful Network Retraining.")


    except:
        showDialog("Unsucessfult Network Retraining. Try
again.")


    #sys.exit() # kill thread once function is done to
preserve computational power

def retrainChooseD():
    rfolder = QFileDialog.getExistingDirectory(window,
"Select Data To Retrain Network")
    sfolder = QFileDialog.getExistingDirectory(window,
"Select Where To Save Retrained Network")
    print(rfolder)
    print(sfolder)

    if (rfolder):
        if(sfolder):
            try:
                t =  threading.Thread(target =
connectMatlab2, args =(rfolder))
                t.start()

                show_frame3("Retraining in Progress",
show_frame5, "Return")

            except:
                show_frame3("Unsuccessful Network
Retraining. Try again.", show_frame5, "Return")
    else:
```

```
        show_frame3("Unsuccessful Network Retraining. Try
again.", show_frame5, "Return")


def chooseData():
    global folder
    global folder2
    #Tk().withdraw() # we don't want a full GUI, so keep
the root window from appearing
    folder = QFileDialog.getExistingDirectory(window,
"Select Data To Classify")
    print(folder)
    # Request for directory where to save images
    folder2 = QFileDialog.getExistingDirectory(window,
"Select Directory To Save Images")
    print(folder2)
    # Matlab Addition to run script / with data input
argument -- run classification
    if (folder):
        if (folder2):
            try:
                ### AQUIIIII
                t =  threading.Thread(target =
connectMatlab, args =(folder,folder2))
                t.start()
                # while(t.is_alive() == True):
                #     show_frame3("Classification in
Process", useless, "Wait")

                show_frame3("Classification in Progress",
show_frame5, "Return")

            except:
                show_frame3("Unsuccessful Classification.
Try again.", show_frame5, "Return")
    else:
        show_frame3("Unsuccessful Classification. Try
again.", show_frame5, "Return")

def useless():
    mj = 100010000


def showDialog(c):
   msgBox = QMessageBox()
   msgBox.setIcon(QMessageBox.Information)
   msgBox.setText(c)
```

```
   msgBox.setWindowTitle("Process Result")
   msgBox.setStandardButtons(QMessageBox.Ok)
   ret = msgBox.exec()


def chooseData2():
    global folder3
    global folder4
    #Tk().withdraw() # we don't want a full GUI, so keep
the root window from appearing
    folder3 = QFileDialog.getExistingDirectory(window,
"Select Data To Classify")
    print(folder3)
    # Request for directory where to save images
    folder4 = QFileDialog.getExistingDirectory(window,
"Select Directory To Save Images")

    print(folder4)
    # Matlab Addition to run script / with data input
argument -- run classification
    if (folder3):
        if (folder4):
            try:
                t =  threading.Thread(target =
connectMatlab, args =(folder3,folder4))
                t.start()
                # while(t.is_alive() == True):
                #      show_frame3("Classification in
Process", useless, "Wait")
                show_frame3("Classification in progress.
Please wait until it is finished.", show_frame7, "Next")

            except:
               show_frame3("Unsuccessful Classification.
Try again.", show_frame5, "Return")
    else:
        show_frame3("Unsuccessful Classification. Try
again.", show_frame5, "Return")

def chooseData3():
    global fname
    fname = QFileDialog.getOpenFileName(window, 'Choose CSV
File')
    print(fname)
    createFLayer(fname[0])
```

```
def chooseData4(wb_id):
    global fname
    fname = QFileDialog.getOpenFileName(window, 'Choose CSV
File')
    print(fname)

    try:
        t =  threading.Thread(target = createFLayer2, args
=(fname[0], wb_id))
        t.start()
        # while(t.is_alive() == True):
        #      show_frame3("Classification in Process",
useless, "Wait")
        show_frame3("Webmap Creation in progress. Please
wait until it is finished.", show_frame7, "Next")

    except:
        how_frame3("Unsuccessful Webmap Creation. Try
again.", show_frame5, "Return")

def chooseData5(wb_id):
    global fname
    fname = QFileDialog.getOpenFileName(window, 'Choose CSV
File')
    print(fname)
    try:
        t =  threading.Thread(target = createFLayer3, args
=(fname[0], wb_id))
        t.start()
        # while(t.is_alive() == True):
        #      show_frame3("Classification in Process",
useless, "Wait")
        show_frame3("Webmap Creation in progress. Please
wait until it is finished.", show_frame7, "Next")

    except:
        how_frame3("Unsuccessful Webmap Creation. Try
again.", show_frame5, "Return")


def mapData(x):
    global my_contt
    my_contt = gis.content.search(query="owner:" +
gis.users.me.username, item_type="Web Map")
    show_frame6(my_contt, x)

def my_title(title, x):
```

```
    global my_contt
    my_contt = gis.content.search(query="title:%s" %title,
item_type="Web Map")
    show_frame6(my_contt, x)

def my_keyword(keyword, x):
    global my_contt
    my_contt = gis.content.search(query="title:%s*"
%keyword, item_type="Web Map")
    show_frame6(my_contt, x)

def key_dial(x):
    text, ok = QInputDialog.getText(None, 'Keyword Search',
'Keyword')
    if ok == True:
        my_keyword(text, x)
        print(text)

def title_dial(x):
    text, ok = QInputDialog.getText(None, 'Title Search',
'Title')
    if ok == True:
        my_title(text, x)
        print(text)

def theDial():
    text, ok = QInputDialog.getText(None, 'Name',
'Username')
    text2, ok2 = QInputDialog.getText(None, 'Password',
'Password', QLineEdit.Password)

    if ok and ok2 == True:
        loginHandler(text, text2)

    else:
        clear_widgets()
        frame3()

def copyMyID(x):
    y = listWidget.currentRow()

    if x == 1:
        pycl.copy(my_contt[y].id)

    elif x == 2:
        chooseData4(my_contt[y].id)
```

```
    else:
        chooseData5(my_contt[y].id)

def copyMyID2(x):
    y = listWidget2.currentRow()
    show_frame11(x[y])

def MapIdEnter():
    text, ok = QInputDialog.getText(None, 'Map ID',
'ItemID')
    if ok == True:
        gis = GIS('https://www.arcgis.com', username,
password)
        #
https://mg07926.maps.arcgis.com/apps/mapviewer/index.html?w
ebmap=783a11500212434992e97b1e48e8e7f5#

webbrowser.open("https://"+username+".maps.arcgis.com/apps/
mapviewer/index.html?webmap="+text)


def dwld(text, play_id):
    now = os.getcwd()
    y = listWidget3.currentRow()
    # Request for directory where to save images
    new = QFileDialog.getExistingDirectory(window, "Select
Directory To Save Video")
    print(new)
    os.chdir(new)
    ydl_opts = {}

    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([text[y].strip()])
    print('done')
    os.chdir(now)
    show_frame3("Successful Download", lambda:
show_frame11(play_id), "Return")

def fetch(itemId):
    gis = GIS('https://mg07926.maps.arcgis.com', 'mg07926',
'Familia2016!') #username, password)

def get_pl():

    api_key = 'AIzaSyAgmPBCu4iUVg5t3vdQZCHSsCwu1AH0124'
    channel_id = 'UCCKprTOrntmBOhPr_boGxYw'
```

```
    # api_key =
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="/path/to/file
.json"

    youtube = build('youtube', 'v3', developerKey=api_key)

    request = youtube.channels().list(
            part='statistics',
            id = channel_id
        )

    response = request.execute()
    print(response)

    ######################################## PLAYLISTS

    youtube = build("youtube", "v3", developerKey =
api_key)
    request = youtube.playlists().list(
        part = "snippet",
        channelId = channel_id,
        maxResults = 150
    )
    response = request.execute()

    playlists = []
    playlists_id = []
    playlists_name = []
    playlists_snippet = []

    while request is not None:
        response = request.execute()
        playlists += response["items"]
        a_key = 'id'
        b_key = 'snippet'
        c_key = 'title'
        playlists_id = [a_dict[a_key] for a_dict in
playlists]
        playlists_snippet = [b_dict[b_key] for b_dict in
playlists]
        playlists_name = [c_dict[c_key] for c_dict in
playlists_snippet]
        request = youtube.playlists().list_next(request,
response)

    print(f"total: {len(playlists)}")
    print(playlists_id)
```

```
    print(playlists_name)
    return playlists_id, playlists_name

def get_pl2():

    api_key = 'AIzaSyAgmPBCu4iUVg5t3vdQZCHSsCwu1AH0124'
    channel_id = 'UCCKprTOrntmBOhPr_boGxYw'
    # api_key =
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="/path/to/file
.json"

    youtube = build('youtube', 'v3', developerKey=api_key)

    request = youtube.channels().list(
            part='statistics',
            id = channel_id
        )

    response = request.execute()
    print(response)

    ####################################### PLAYLISTS

    youtube = build("youtube", "v3", developerKey =
api_key)
    request = youtube.playlists().list(
        part = "snippet",
        channelId = channel_id,
        maxResults = 150
    )
    response = request.execute()

    playlists = []
    playlists_id = []
    playlists_name = []
    playlists_snippet = []

    while request is not None:
        response = request.execute()
        playlists += response["items"]
        a_key = 'id'
        b_key = 'snippet'
        c_key = 'title'
        playlists_id = [a_dict[a_key] for a_dict in
playlists]
        playlists_snippet = [b_dict[b_key] for b_dict in
playlists]
```

```
        playlists_name = [c_dict[c_key] for c_dict in
playlists_snippet]
        request = youtube.playlists().list_next(request,
response)

    print(f"total: {len(playlists)}")
    print(playlists_id)
    print(playlists_name)
    return playlists_id, playlists_name

def loginHandler(user, psw):
   # Log In ArcGIS
   try:
       global username
       global password

       username = user
       password = psw

       global gis
       gis = GIS('https://www.arcgis.com', username,
password)
       print(3)

       un = gis.properties.user.username
       print('Logged in as: {}'.format(un))
       show_frame2()
   except:
       show_frame3("Unable to login. Try again.",
show_frame1, "Return")
       print('why')


def smpl2():
    srt = QFileDialog.getOpenFileName(window, 'Choose SRT
File')
    outfile = open(srt[0],"r")
    data = outfile.readlines()
    gps_line = []

    for line in data:
        if 'latitude' in line:
            gps_line.append(line)


    latitude = []
    longitude = []
```

```
    for i in range(len(gps_line)):
        words = gps_line[i].split()
        latitude.append(words[23][:-1])
        longitude.append(words[26][:-1])


    time_f = []
    for line2 in data:
        if '-->' in line2:
            time_f.append(line2)

    final_t = []
    for i in range(len(time_f)):
        timing = time_f[i].split()
        final_t.append(timing[2])


    # Opens the Video file
    vid = QFileDialog.getOpenFileName(window, 'Choose Video
File')
    new = QFileDialog.getExistingDirectory(window, "Select
Directory To Save Frames")
    cap= cv2.VideoCapture(vid[0])

    i=0

    while(cap.isOpened()):
        ret, frame = cap.read()
        if ret == False:
            break
        if i % 1800 == 0: # this is the line I added to
make it only save one frame every 1800 frames = 30 fps of
the camera/ 1 frame every min
            cv2.imwrite(new + '/kang'+str(i)+'.jpg',frame)
            f2 = new + '/kang'+str(i)+'.jpg'
            photo = gpsphoto.GPSPhoto(f2)
            info = gpsphoto.GPSInfo((float(latitude[i]),
float(longitude[i])))
            photo.modGPSData(info, new +
'/kang'+str(i)+'.jpg')
        i+=1

    cap.release()
    cv2.destroyAllWindows()
    show_frame3("Successful Sampling", lambda:
show_frame2(), "Return")
```

```
def createFLayer(csv_Pfile): # need file path + name
    # Log In ArcGIS
    gis = GIS('https://www.arcgis.com', username, password)

    csv_df = pd.read_csv(csv_Pfile)
    csv_df = csv_df.rename(columns={'T1':'name', 'T2':
'Latitude', 'T3': 'Longitude', 'T4':
                                    'Classification', 'T5':
'Original Image Path'}, errors = "raise")

    # import as feature
    csv_featcol = gis.content.import_data(csv_df,
location_type = 'coordinates', latitude_field = 'Latitude',
                                          longitude_field =
'Longitude')

    # import json and convert the feature collection to a
JSON and
    # add it as a text based item to the GIS. The feature
collection
    # properties provides the layer definition and feature
set for a layer

    csv_featcol_dict = dict(csv_featcol.properties)
    csv_json = json.dumps({"featureCollection": {"layers":
[csv_featcol_dict]}})


    # add the featcol
    csv_item_properties = {'title': 'Feature Collection
Layer Trial 1 '+str(datetime.now()),
    'description':'Example demonstrating conversion of
pandas for GDOT Project' + \
    'dataframe object to a GIS item',
    'tags': 'arcgis python api, pandas, csv',
    'text':csv_json,
    'type':'Feature Collection'}
    csv_item = gis.content.add(csv_item_properties)

    butts = csv_item.publish()
    createWebMap(butts.id)

def createFLayer2(csv_Pfile, webmap_ID):
    # Log In ArcGIS
    gis = GIS('https://www.arcgis.com', username, password)
```

```
    csv_df = pd.read_csv(csv_Pfile)
    csv_df = csv_df.rename(columns={'T1':'name', 'T2':
'Latitude', 'T3': 'Longitude'}, errors = "raise")

    # import as feature
    csv_featcol = gis.content.import_data(csv_df,
location_type = 'coordinates', latitude_field = 'Latitude',
                                          longitude_field =
'Longitude')

    # import json and convert the feature collection to a
JSON and
    # add it as a text based item to the GIS. The feature
collection
    # properties provides the layer definition and feature
set for a layer

    csv_featcol_dict = dict(csv_featcol.properties)
    csv_json = json.dumps({"featureCollection": {"layers":
[csv_featcol_dict]}})


    # add the featcol
    csv_item_properties = {'title': 'Feature Collection
Layer Trial 1 '+str(datetime.now()),
    'description':'Example demonstrating conversion of
pandas for GDOT Project' + \
    'dataframe object to a GIS item',
    'tags': 'arcgis python api, pandas, csv',
    'text':csv_json,
    'type':'Feature Collection'}

    csv_item = gis.content.add(csv_item_properties)

    butts = csv_item.publish()

    wm_item = gis.content.get(webmap_ID)

    # create a WebMap object from the existing web map item
    wm = WebMap(wm_item)
    csv_layer = gis.content.get(''+str(butts.id))
    wm.add_layer(csv_layer,
options={'title':'CSV_Layer'+str(datetime.now())})

    # Publish the web map as an item to the portal
    web_map_properties = {'title':''+wm_item.title+' New
Data Appended',
```

```
                          'snippet':'This map service is
for GDOT GUI',
                          'tags':'ArcGIS Python API'}

    # Call the save() with web map item's properties.
    wm.save(item_properties=web_map_properties)
    # showDialog('Webmap Created.')

def createFLayer3(csv_Pfile, webmap_ID):
    # Log In ArcGIS

    gis = GIS('https://www.arcgis.com', username, password)

    csv_df = pd.read_csv(csv_Pfile)
    csv_df = csv_df.rename(columns={'T1':'name', 'T2':
'Latitude', 'T3': 'Longitude'}, errors = "raise")

    # import as feature
    csv_featcol = gis.content.import_data(csv_df,
location_type = 'coordinates', latitude_field = 'Latitude',
                                          longitude_field =
'Longitude')

    # import json and convert the feature collection to a
JSON and
    # add it as a text based item to the GIS. The feature
collection
    # properties provides the layer definition and feature
set for a layer

    csv_featcol_dict = dict(csv_featcol.properties)
    csv_json = json.dumps({"featureCollection": {"layers":
[csv_featcol_dict]}})


    # add the featcol
    csv_item_properties = {'title': 'Feature Collection
Layer Trial 1 '+str(datetime.now()),
    'description':'Example demonstrating conversion of
pandas for GDOT Project' + \
    'dataframe object to a GIS item',
    'tags': 'arcgis python api, pandas, csv',
    'text':csv_json,
    'type':'Feature Collection'}

    csv_item = gis.content.add(csv_item_properties)
```

```
    butts = csv_item.publish()

    wm_item = gis.content.get(webmap_ID)

    # create a WebMap object from the existing web map item
    wm = WebMap(wm_item)
    csv_layer = gis.content.get(''+str(butts.id))
    wm.remove_layer(wm.layers[0])
    wm.add_layer(csv_layer,
options={'title':'CSV_Layer'+str(datetime.now())})

    # Publish the web map as an item to the portal
    web_map_properties = {'title':''+wm_item.title+' Data
Overwritten',
                          'snippet':'This map service is
for GDOT GUI',
                          'tags':'ArcGIS Python API'}

    # Call the save() with web map item's properties.
    wm.save(item_properties=web_map_properties)
    # showDialog('Webmap Created.')

def createWebMap(csv_item_id):
    # Log In ArcGIS
    gis = GIS('https://www.arcgis.com', username, password)

    # Create an empty web map with a default basemap
    wm = WebMap()
    # Look for map street base layer
    search_result = gis.content.search("title:Street AND
owner:esri",
                                       item_type = "Map
Service", outside_org = True)
    # Choose first result (the one we are using) / apply it
to our base webmap "wm"
    street_layer = search_result[0]

    for lyr in street_layer.layers:
        wm.add_layer(lyr)

    csv_layer = gis.content.get(''+str(csv_item_id))
    wm.add_layer(csv_layer,
options={'title':'CSV_Layer'+str(datetime.now())})

    # Publish the web map as an item to the portal
    web_map_properties = {'title':'New Webmap: Street Layer
for GDOT GUI '+str(datetime.now()),
```

```
                            'snippet':'This map service is
for GDOT GUI',
                            'tags':'ArcGIS Python API'}

    # Call the save() with web map item's properties.
    wm.save(item_properties=web_map_properties)
    show_frame3('Web Map Created', show_frame7, "Return")

def overwrite_gis(itemId):
    gis = GIS('https://mg07926.maps.arcgis.com', username,
password)
    try:
        dataitem = gis.content.get(itemId)
        flayercol =
FeatureLayerCollection.fromitem(dataitem)
        #flayercol.manager.overwrite(newname) # name of the
file uploaded
        #flayercol.manager.overwrite(newname)

    except Exception as error:
        print(error)

def show_frame1():
    clear_widgets()
    frame1()

def show_frame2():
    clear_widgets()
    frame2()

def show_frame3(Question, showFrame, Action):
    clear_widgets()
    frame3(Question, showFrame, Action)

def show_frame4(x):
    clear_widgets()
    frame4(x)

def show_frame5():
    clear_widgets()
    frame5()

def show_frame6(my_contt, x):
    clear_widgets()
    frame6(my_contt, x)

def show_frame7():
```

```
    clear_widgets()
    frame7()

def show_frame8():
    clear_widgets()
    frame8()

def show_frame9():
    clear_widgets()
    frame9()

def show_frame10():
    clear_widgets()
    frame10()

def show_frame11(x):
    clear_widgets()
    frame11(x)

def clear_widgets():
    for widget in widgets:
        if widgets[widget] != []:
            widgets[widget][-1].hide()
        for i in range(0, len(widgets[widget])):
            widgets[widget].pop()

def create_buttons(answer):
        # button functions
        button = QPushButton(answer)

button.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
        button.setFixedWidth(485)
        button.setStyleSheet("*{border: 4px solid 'white';"
+
                        "border-radius: 25px;" +
                        "font-family: 'shanti';"
                        "font-size: 16px;" +
                        "color: 'white';" +
                        "padding: 15px 0;" +
                        "margin: 20px;}" +
                        "*:hover{background: 'green';}")
        return button

def frame1():
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
```

```
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter) # aligns text
inside the widget
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    # button widget
    button = QPushButton("Login")
    button.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
    button.setStyleSheet("*{border: 4px solid 'white';" +
                         "border-radius: 45px;" +
                         "font-size: 35px;" +
                         "color: 'white';" +
                         "padding: 25px 0;" +
                         "margin: 50px 100px;}" +
                         "*:hover{background: 'green';}"
                         )
    button.clicked.connect(theDial)

    widgets["button"].append(button)

    # place widget on the grid
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["button"][-1], 3, 0, 1, 2)

def frame2():
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    #question widget
    question = QLabel("Select an action: ")
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
```

```
    widgets["question"].append(question)


    # buttons
    button1 = create_buttons("Log Out")
    button2 = create_buttons("Classify Data")
    button3 = create_buttons("View ArcGIS Data")
    button4 = create_buttons("Open a WebMap with ItemID")
    button5 = create_buttons("Download Youtube Video")
    button6 = create_buttons("Sample Video")

    button1.clicked.connect(show_frame1)
    button2.clicked.connect(show_frame5)
    button3.clicked.connect(lambda: show_frame4(1))
    button4.clicked.connect(MapIdEnter)
    button5.clicked.connect(show_frame9)
    button6.clicked.connect(smpl2)

    # Append Buttons
    widgets["Return"].append(button1)
    widgets["Classify Data"].append(button2)
    widgets["View ArcGIS Data"].append(button3)
    widgets["Open a Map with ItemID"].append(button4)
    widgets["Download and Sample Video"].append(button5)
    widgets["uname"].append(button6)

    # place widget on the grid
    grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 2, 0)
    grid.addWidget(widgets["Classify Data"][-1], 2, 1)
    grid.addWidget(widgets["View ArcGIS Data"][-1], 3, 0)
    grid.addWidget(widgets["Open a Map with ItemID"][-1],
3, 1)
    grid.addWidget(widgets["Download and Sample Video"][-
1], 4, 0)
    grid.addWidget(widgets["uname"][-1], 4, 1)

def frame3(Question, showFrame, Action):

    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)
```

```
    #question widget
    question = QLabel(Question)
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
    widgets["question"].append(question)

    # buttons
    button = QPushButton(Action)
    button.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
    button.setStyleSheet("*{border: 4px solid 'white';" +
                         "border-radius: 45px;" +
                         "font-size: 35px;" +
                         "color: 'white';" +
                         "padding: 25px 0;" +
                         "margin: 50px 100px;}" +
                         "*:hover{background: 'green';}"
                         )

    button.clicked.connect(showFrame)

    # Append Buttons
    widgets["Return"].append(button)

    # place widget on the grid
    grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 3, 0, 1, 2)

def frame4(x):
    # Display Logo
        image = QPixmap("gdot.png")
        logo = QLabel()
        logo.setPixmap(image)
        logo.setAlignment(QtCore.Qt.AlignCenter)
        logo.setStyleSheet("margin-top: 100px;")
        widgets["logo"].append(logo)

        #question widget
```

```
        question = QLabel("Select Search Method: ")
        question.setAlignment(QtCore.Qt.AlignCenter)
        question.setWordWrap(True)
        question.setStyleSheet(
            '''
            font-family: Shanti;
            font-size: 25px;
            color: 'white';
            padding: 75px;

            '''
        )
        widgets["question"].append(question)


        # buttons
        button1 = create_buttons("Return")
        button2 = create_buttons("All Content")
        button3 = create_buttons("Search by Keyword")
        button4 = create_buttons("Search by Title")

        if x == 1:
            button1.clicked.connect(show_frame2)

        else:
            button1.clicked.connect(show_frame7)

        button2.clicked.connect(lambda: mapData(x))
        button3.clicked.connect(lambda: key_dial(x))
        button4.clicked.connect(lambda: title_dial(x))

        # Append Buttons
        widgets["Return"].append(button1)
        widgets["All Content"].append(button2)
        widgets["Search by Keyword"].append(button3)
        widgets["Search by Title"].append(button4)


        # place widget on the grid
        grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
        grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
        grid.addWidget(widgets["Return"][-1], 2, 0)
        grid.addWidget(widgets["All Content"][-1], 2, 1)
        grid.addWidget(widgets["Search by Keyword"][-1], 3,
0)
        grid.addWidget(widgets["Search by Title"][-1], 3,
1)
```

```
def frame5():
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    #question widget
    question = QLabel("Classification Data Options: ")
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
    widgets["question"].append(question)


    # buttons
    button1 = create_buttons("Return")
    button2 = create_buttons("Classify Data Only")
    button3 = create_buttons("Classify Data and Modify
Maps")
    button4 = create_buttons("Use Previous Classified Data
to Modify Maps")
    button5 = create_buttons("Retrain Network")

    button1.clicked.connect(show_frame2)
    button2.clicked.connect(chooseData)
#button2.clicked.connect(threading.Thread(target =
chooseData).start()) #
    button3.clicked.connect(chooseData2)
    button4.clicked.connect(show_frame7)
    button5.clicked.connect(retrainChooseD)

    # Append Buttons
    widgets["Return"].append(button1)
    widgets["Select Data to Classify"].append(button2)
#change name later
```

```
    widgets["Open a Map with ItemID"].append(button3) #
change name later
    widgets["Use Previous Classified Data to Modify
Maps"].append(button4)
    widgets["Retrain Network"].append(button5)


    # place widget on the grid
    grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 4, 0)
    grid.addWidget(widgets["Select Data to Classify"][-1],
2, 1) #change name later
    grid.addWidget(widgets["Open a Map with ItemID"][-1],
3, 0) #change name later
    grid.addWidget(widgets["Use Previous Classified Data to
Modify Maps"][-1], 3, 1)
    grid.addWidget(widgets["Retrain Network"][-1], 2, 0)

def frame6(my_contt, x):
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    global listWidget
    i=0
    listWidget = QListWidget()
    listWidget.setGeometry(50, 70, 150, 80)
    listWidget.setStyleSheet("QListWidget"
                                  "{"
                                  "border : 2px solid
black;"
                                  "background : white;"
                                  "}"
                                  "QListWidget QScrollBar"
                                  "{"
                                  "background : lightgrey;"
                                  "}"

"QListView::item:selected"
                                  "{"
                                  "border : 2px solid
black;"
```

```
                                    #"font-color: black;"
                                    "background : lightgrey;"
                                    "}"
                                    )

listWidget.setCursor(QtGui.QCursor(QtCore.Qt.IBeamCursor))

#listWidget.setTextInteractionFlags(Qt.TextSelectableByMous
e)

    while i < len(my_contt):
        listWidget.addItem(""+str(i+1)+") Title: "+
str(my_contt[i].title) + "\n Item ID: "+my_contt[i].id+
                           "\n Type: "+
str(my_contt[i].type) + "\n Owner: " +
str(my_contt[i].owner))
        i = 1 + i

    # Append Qlist
    widgets["listWidget"].append(listWidget)
    if x == 1:
        #Button
        button1 = create_buttons("Return")
        button2 = create_buttons("Copy ItemID to
Clipboard")

        button1.clicked.connect(lambda: show_frame4(x))
###### change this later
        button2.clicked.connect(lambda: copyMyID(x))

        widgets["Return"].append(button1)
        widgets["Copy ItemID"].append(button2)
        # place widget on the grid
        grid.addWidget(widgets["listWidget"][-1], 1, 0, 1,
2)
        grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
        grid.addWidget(widgets["Return"][-1], 2, 0)
        grid.addWidget(widgets["Copy ItemID"][-1], 2, 1)
    else:
        #Button
        button1 = create_buttons("Return")
        button2 = create_buttons("Change Selected Map")

        button1.clicked.connect(lambda: show_frame4(x))
        button2.clicked.connect(lambda: copyMyID(x))

        widgets["Return"].append(button1)
```

```
        widgets["Change Selected Map"].append(button2)

        # place widget on the grid
        grid.addWidget(widgets["listWidget"][-1], 1, 0, 1,
2)
        grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
        grid.addWidget(widgets["Return"][-1], 2, 0)
        grid.addWidget(widgets["Change Selected Map"][-1],
2, 1)

def frame7():
    # Display Logo
        image = QPixmap("gdot.png")
        logo = QLabel()
        logo.setPixmap(image)
        logo.setAlignment(QtCore.Qt.AlignCenter)
        logo.setStyleSheet("margin-top: 100px;")
        widgets["logo"].append(logo)

        #question widget
        question = QLabel("Select Data Modification: ")
        question.setAlignment(QtCore.Qt.AlignCenter)
        question.setWordWrap(True)
        question.setStyleSheet(
            '''
            font-family: Shanti;
            font-size: 25px;
            color: 'white';
            padding: 75px;

            '''
        )
        widgets["question"].append(question)


        # buttons
        button1 = create_buttons("Return")
        button2 = create_buttons("Overwrite a Map")
        button3 = create_buttons("Create New Map")
        button4 = create_buttons("Append Data to a Map")

        button1.clicked.connect(show_frame5)
        button2.clicked.connect(lambda: show_frame4(3)) #
chooseData5
        button3.clicked.connect(chooseData3)
        button4.clicked.connect(lambda: show_frame4(2)) #
chooseData4
```

```
        # Append Buttons
        widgets["Return"].append(button1)
        widgets["Overwrite a Map"].append(button2)
        widgets["Create New Map"].append(button3)
        widgets["Append Data to a Map"].append(button4)


        # place widget on the grid
        grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
        grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
        grid.addWidget(widgets["Return"][-1], 2, 0)
        grid.addWidget(widgets["Overwrite a Map"][-1], 2,
1)
        grid.addWidget(widgets["Create New Map"][-1], 3, 0)
        grid.addWidget(widgets["Append Data to a Map"][-1],
3, 1)

def frame8 ():
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    #question widget
    question = QLabel('Download Youtube Video')
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
    widgets["question"].append(question)

    # buttons
    button = QPushButton('Download')
    button.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
    button.setStyleSheet("*{border: 4px solid 'white';" +
                         "border-radius: 45px;" +
                         "font-size: 35px;" +
```

```
                              "color: 'white';" +
                              "padding: 25px 0;" +
                              "margin: 50px 100px;}" +
                              "*:hover{background: 'green';}"
                              )

    button.clicked.connect(yt_downl)

    button1 = QPushButton('Return')

button1.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
    button1.setStyleSheet("*{border: 4px solid 'white';" +
                              "border-radius: 45px;" +
                              "font-size: 35px;" +
                              "color: 'white';" +
                              "padding: 25px 0;" +
                              "margin: 50px 100px;}" +
                              "*:hover{background: 'green';}"
                              )

    button.clicked.connect(yt_downl) #######
    button1.clicked.connect(show_frame2)

    # Append Buttons
    widgets["Download"].append(button)
    widgets["Return"].append(button1)

    # place widget on the grid
    grid.addWidget(widgets["question"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 2, 1)
    grid.addWidget(widgets["Download"][-1], 2, 0)


def frame9():
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    #question widget
    question = QLabel("Select aa playlist: ")
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
```

```
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
    widgets["question"].append(question)
    k = get_pl()
    my_pl = k[1]
    my_pl_ID = k[0]

    # List
    global listWidget2
    i=0
    listWidget2 = QListWidget()
    listWidget2.setGeometry(50, 70, 150, 80)
    listWidget2.setStyleSheet("QListWidget"
                                        "{"
                                        "border : 2px solid
black;"
                                        "background : white;"
                                        "}"
                                        "QListWidget QScrollBar"
                                        "{"
                                        "background : lightgrey;"
                                        "}"

"QListView::item:selected"
                                        "{"
                                        "border : 2px solid
black;"
                                        #"font-color: black;"
                                        "background : lightgrey;"
                                        "}"
                                        )

listWidget2.setCursor(QtGui.QCursor(QtCore.Qt.IBeamCursor))

#listWidget.setTextInteractionFlags(Qt.TextSelectableByMous
e)

    while i < len(my_pl):
        listWidget2.addItem(""+str(i+1)+") Playlist Title:
"+ my_pl[i] + "\n Playlist ID: "+my_pl_ID[i])
```

```
        i = 1 + i

    # Append Qlist
    widgets["listWidget"].append(listWidget2)

    # buttons
    button1 = create_buttons("Return")
    button2 = create_buttons("Next")


    button1.clicked.connect(show_frame2)
    button2.clicked.connect(lambda: copyMyID2(my_pl_ID))


    # Append Buttons
    widgets["Return"].append(button1)
    widgets["Classify Data"].append(button2)


    # place widget on the grid
    grid.addWidget(widgets["listWidget"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 2, 0)
    grid.addWidget(widgets["Classify Data"][-1], 2, 1)

def frame10():
    print(10)


def vid_url(playlist_id):
    videos = []

    api_key = 'AIzaSyAgmPBCu4iUVg5t3vdQZCHSsCwu1AH0124'
    channel_id = 'UCCKprTOrntmBOhPr_boGxYw'
    # api_key =
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="/path/to/file
.json"

    youtube = build('youtube', 'v3', developerKey=api_key)
    nextPageToken = None
    while True:
        pl_request = youtube.playlistItems().list(
            part='contentDetails',
            playlistId=playlist_id,
            maxResults=50,
            pageToken=nextPageToken
        )
```

```
        pl_response = pl_request.execute()

        pl_request2 = youtube.playlistItems().list(
            part='snippet',
            playlistId=playlist_id,
            maxResults=50,
            pageToken=nextPageToken
        )

        pl_response2 = pl_request2.execute()
        items = pl_response2['items']
        vid_names = []

        vid_names =[lol['snippet']['title'] for lol in
items]

        vid_ids = []
        for item in pl_response['items']:

vid_ids.append(item['contentDetails']['videoId'])

        vid_request = youtube.videos().list(
            part="statistics",
            id=','.join(vid_ids)
        )

        vid_response = vid_request.execute()

        for item in vid_response['items']:
            vid_id = item['id']
            yt_link = f'https://youtu.be/{vid_id}'

            videos.append(yt_link)

        nextPageToken = pl_response.get('nextPageToken')

        if not nextPageToken:
            break
    return videos, vid_names


def frame11(play_id):
    # Display Logo
    image = QPixmap("gdot.png")
    logo = QLabel()
    logo.setPixmap(image)
```

```
    logo.setAlignment(QtCore.Qt.AlignCenter)
    logo.setStyleSheet("margin-top: 100px;")
    widgets["logo"].append(logo)

    #question widget
    question = QLabel("Select a video: ")
    question.setAlignment(QtCore.Qt.AlignCenter)
    question.setWordWrap(True)
    question.setStyleSheet(
        '''
        font-family: Shanti;
        font-size: 25px;
        color: 'white';
        padding: 75px;

        '''
    )
    widgets["question"].append(question)

    # List
    global listWidget3
    i=0
    listWidget3 = QListWidget()
    listWidget3.setGeometry(50, 70, 150, 80)
    listWidget3.setStyleSheet("QListWidget"
                                  "{"
                                  "border : 2px solid
black;"
                                  "background : white;"
                                  "}"
                                  "QListWidget QScrollBar"
                                  "{"
                                  "background : lightgrey;"
                                  "}"

"QListView::item:selected"
                                  "{"
                                  "border : 2px solid
black;"
                                  #"font-color: black;"
                                  "background : lightgrey;"
                                  "}"
                                  )

listWidget3.setCursor(QtGui.QCursor(QtCore.Qt.IBeamCursor))
```

```
#listWidget.setTextInteractionFlags(Qt.TextSelectableByMous
e)
    mh = vid_url(play_id)
    my_vid_url = mh[0]
    my_vid_name = mh[1]

    while i < len(mh[0]):
        listWidget3.addItem(""+str(i+1)+") Video Title: "+
my_vid_name[i] + "\n Video URL: "+my_vid_url[i])
        i = 1 + i

    # Append Qlist
    widgets["listWidget"].append(listWidget3)

    # buttons
    button1 = create_buttons("Return")
    button2 = create_buttons("Download Video")


    button1.clicked.connect(show_frame2)
    button2.clicked.connect(lambda: dwld(my_vid_url,
play_id))


    # Append Buttons
    widgets["Return"].append(button1)
    widgets["Classify Data"].append(button2)


    # place widget on the grid
    grid.addWidget(widgets["listWidget"][-1], 1, 0, 1, 2)
    grid.addWidget(widgets["logo"][-1], 0, 0, 1, 2)
    grid.addWidget(widgets["Return"][-1], 2, 0)
    grid.addWidget(widgets["Classify Data"][-1], 2, 1)


frame1()

window.setLayout(grid)

window.show()
sys.exit(app.exec())
```