Final Project Report

Title: Deep Neural Networks for Landcover Classification of the Colorado River Watershed
Notice: Dr. Bryan Runck
Author: Jake Ford
Date: 12/18/2022

Project Repository: https://github.com/ThisFord/GIS5571-arc1.git
Time Spent: 48hrs

Abstract

Machine Learning and the subset of algorithms known as Deep Neural Networks are increasingly used in image analysis and classification programs. This project uses the powerful computational advantages and predictive modeling capabilities of Deep Neural Networks (DNNs) in combination with the resources available through the Minnesota Supercomputing Institute (MSI) to produce a high-resolution Land Cover and Land Use dataset from multiple input data layers. The Chesapeake Bay Conservancy has produced a model trained on satellite, lidar and planimetric data from counties within the Chesapeake Bay Watershed; this project uses the CBC model in an automated deterministic workflow to create a reproducible model for a new input dataset from the Colorado River Watershed, focusing on Denver County. The DNN is trained on sentinel 2 imagery, and through transfer learning is fine tuned for the Denver County region. Data from Denver County is tested for accuracy against a manually classified subset of the input imagery, which is isolated from the testing workflow. The resulting dataset is a 1m resolution raster with land cover classification for the entire county. Producing an accurate dataset will demonstrate potential scalability; with future plans scaling the model up to a watershed wide dataset with automated updates.

Problem Statement

High resolution landcover data is essential to modern conservation efforts. The current National Land Cover Dataset has a 30m spatial resolution based on Landsat Imagery. Using deep neural network classification models on NAIP imagery, in combination with lidar and other planimetric data, can produce accurate 1m resolution classified data in a programmable workflow, making change analysis possible at a fine spatial scale. As climate change and drought conditions continue to impact the systems reliant on the Colorado River watershed this type of accessible and repeatable data becomes more and more necessary. This project will seek to replicate the Chesapeake Bay Conservancy Land Cover and Land Use workflow on data sourced from a county within the watershed with available lidar and NAIP imagery, producing a semantically segmented dataset with pixel-by-pixel classification for the entire county. Proving the workflow works on unique input data will allow for scaling to a watershed wide analysis. [1–3]

Table 1. Materials needed

#	Requirement	Defined As	(Spatial) Data	Attribute Data	Dataset	Prep
1	Satellite Imagery	NAIP 1m resolution imagery of target area	Georeferenced Raster		USGS	Clip to target area
2	County boundaries	Authoritative county boundary data	Boundary lines and coordinates		Denver CO	download
3	Lidar Groundcover Imagery	Point cloud data for surface and object analysis for county	Coordinate point cloud	Density, reflectivity, surface type, cover type	Denver CO	Clip to target area
4	Object Identification DNN Model	Deep Neural Network model created by the Chesapeake Bay Conservancy for object identification from lidar, NAIP, and planimetric data			Chesapeake Bay Conservancy Model	
5	ArcGIS Pro License	Industry standard GIS analysis software package				
6	Jupiter Labs	Python Notebook Interface				
7	Supercomputing Server Access	MSI servers to run DNN on large data set				DNN model and script plus data sets
8	eCognition License	segmentation and classification software			NAIP imagery	segment input layer
9	Planimetric data	County wide planimetric attribute data for quality control and LULC prediction	Geolocated polygons of structures and human made infrastructure	Labeled and classed objects ie structures, roads	Denver Co	

Input Data

The classifier will use input data from multiple sources to create accurate classifications, with initial training based on the Eurosat data, a labeled imagery dataset hosted on Tensorfow's Datasets api. The following data sources include planimetric, lidar, imagery, and areal information from the area of interest; Denver County Colorado (*Figure 1*). Denver was chosen as a representative county because it contains a mix of land cover classes, from urban to desert, and has an open data policy. The various data structures are used in preprocessing to create a data rich, multiband input raster that includes elevation, heights and vector boundaries along with digital numbers from the electromagnetic bands collected with the imagery.

Table 2. Data required

#	Name	Use	Link
1	NAIP Satellite Imagery	Raw input dataset for image object analysis	USGS[4]
2	County boundaries	Clipping mask for input data	ArcGIS[5]
3	colorado river basin county boundaries	Clipping mask for input data	Lincoln INstitute arcgis [6]
4	Lidar Ground cover Imagery	Object and surface attribute classification and identification, DEM production	Colorado Water Conservation Board
5	Denver regional lidar	Object and surface attribute classification and identification, 1ft resolution	Regional Lidar Project [7]
6	County Planimetric Data	QC and classification comparison	Denver Open Data [8]
7	Denver regional planimetric	Object and surface attribute classification and identification, layer for input data	Regional Planimetrics[8]
8	denver landuse landcover project	results verification and network modeling	Denver Project [9]
9	stactools-packages / **chesapeake-lulc**	chesapeake bay DNN model and training data	stactools-packages / **chesapeake-lulc** [10]
10	Azure / **pixel_level_land_classification**	Tutorial for planning and modeling	Azure /[11]
11	Tensorflow Datasets	Eurosat Data for Model Training	https://www.tensorflow.org/datasets/catalog/eurosat [12]
12	ArcGIS World Imagery	Visualization base map	https://www.arcgis.com/home/item.html?id=10df2279f9684e4a9f6a7f08febac2a9#! [13]

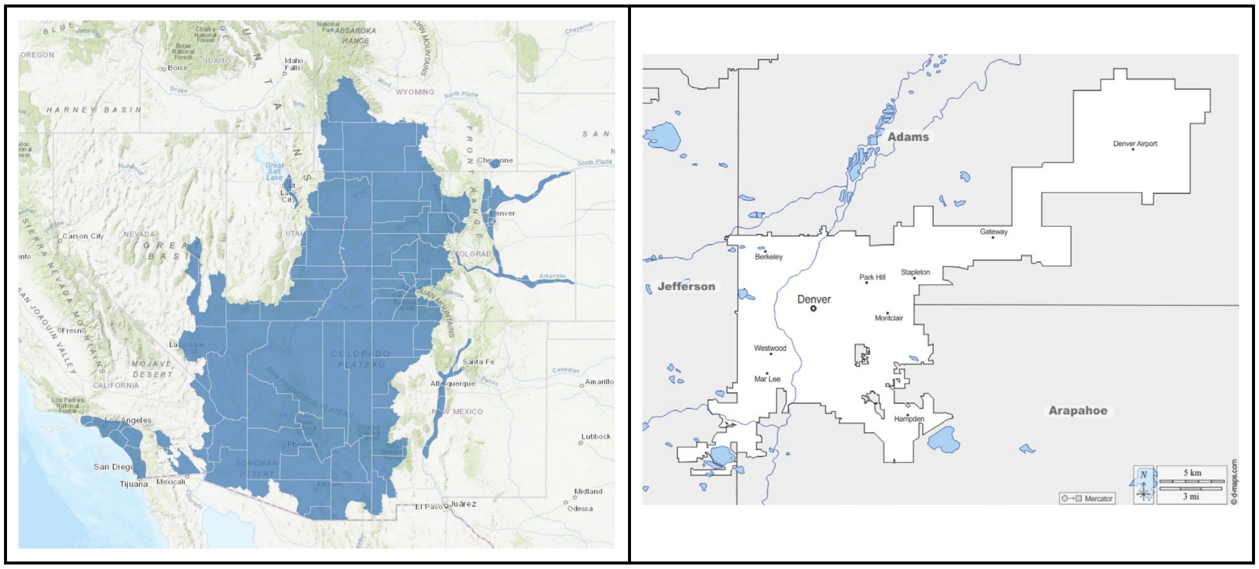


Figure 1, showing the Area of Interest counties in the Colorado River Watershed on the left and a close up of Denver County on the right.

Methods

The Chesapeake Bay Conservancy has developed a methodology to develop high resolution 1m Land Use/Land Cover data for the Chesapeake Bay watershed from NAIP imagery using a predictive DNN model.(*Chesapeake Bay Program Land Use/Land Cover Data Project*, n.d.) This project will be replicating the CBC methodology for a single county in the Colorado River watershed, Denver County, defined as the area of interest (AOI) (*Figure 1)*. The process includes assembling lidar and planimetric data to train the DNN model and set classification attributes, combining this data with 1m NAIP imagery and classifying pixels with the DNN, based on predetermined land cover classifications.[1]

Overview

The classifier is created using tensorflow on the code sharing platform google colab using the Keras Sequential model trained on Tensorfow Dataset's Eurosat imagery.[14,15] The project starts with data wrangling, gathering data from multiple sources and preprocessing the Denver County data. We end up with a 6d multidimensional raster which is sliced into 64x64 pixel chunks for processing. That set is randomly sampled and manually labeled to create a transfer learning subset. The labeled subset is used to fine tune the model that has been trained on a much larger labeled set of images from the sentinel 2 satellite (Eurosat,) accessed through tensorflow datasets. After the transfer learning process, the second step of model training is complete and the full multidimensional raster dataset is run through the classifier. The end product is a classified land cover data set; the model is also set to run cross validation with a subset of of the new training data to compile accuracy statistics.[16,17] *(Figure 2.)*

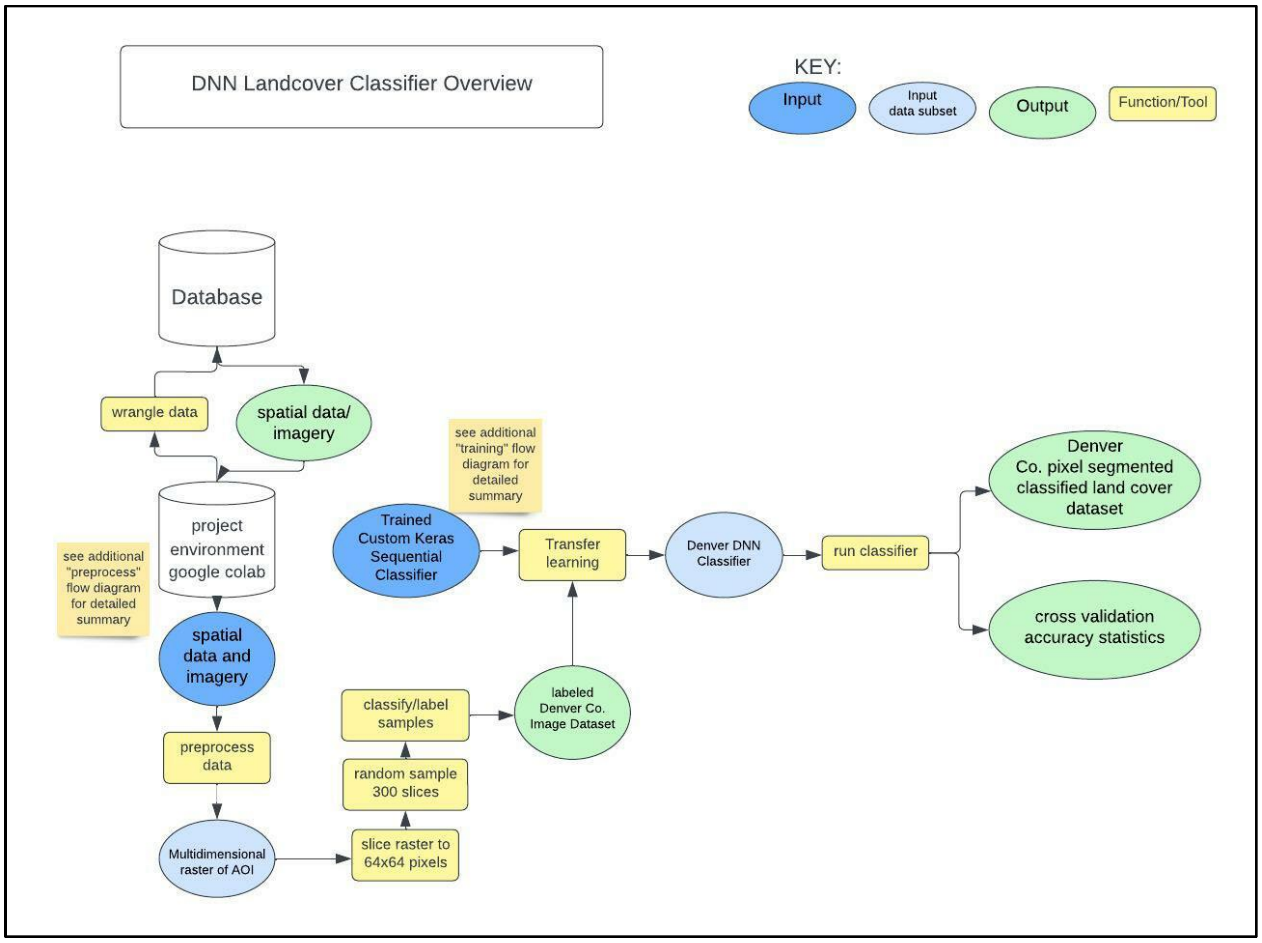

DNN Landcover Classifier Overview
KEY:
Input
Input data subset
Output
Function/Tool
Database
wrangle data
spatial data/ imagery
project environment google colab
see additional "preprocess" flow diagram for detailed summary
spatial data and imagery
preprocess data
Multidimensional raster of AOI
slice raster to 64x64 pixels
random sample 300 slices
classify/label samples
labeled Denver Co. Image Dataset
see additional "training" flow diagram for detailed summary
Trained Custom Keras Sequential Classifier
Transfer learning
Denver DNN Classifier
run classifier
Denver Co. pixel segmented classified land cover dataset
cross validation accuracy statistics

Figure 2. High level overview data flow diagram of a machine learning workflow for image classification.

Preprocessing

The first step is to wrangle all the data, format it for use and create a multidimensional raster of the entire Denver Co dataset. At the end of this phase, the data has been clipped to our study area boundaries, the lidar data has been rasterized and used to create a nDSM, and vector data has been rasterized and standardized to create building and road footprints to help identify urban infrastructure and road classes. The data are fused together in a 6 dimensional raster, with layers for each of the NAIP imagery EM bands (R, G, B, NIR) a layer for the nDSM raster and a layer for the footprints raster.[18,19] (*Figure 3.)*

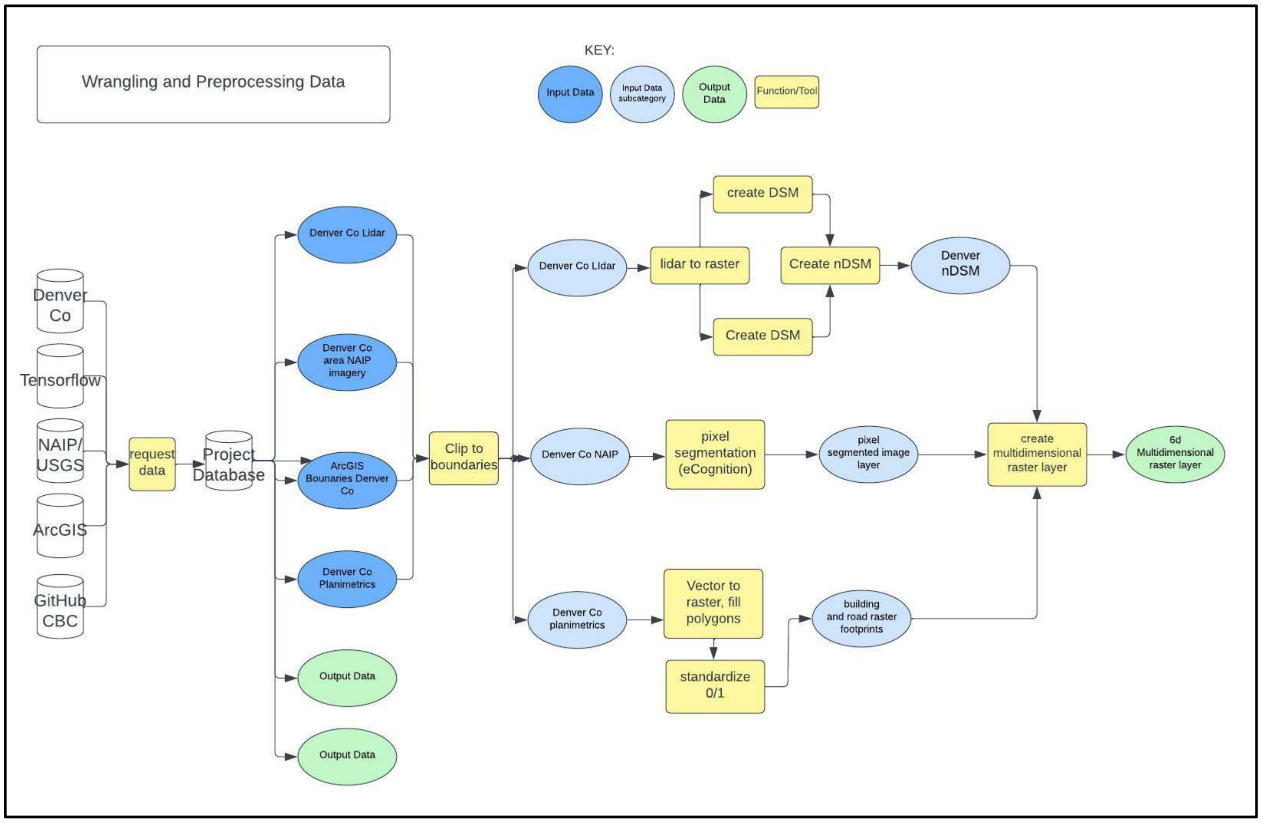


Fig 3. A flow diagram describing the preprocessing phase of the project.

The DNN Model

I used the Keras Sequential algorithm for modeling, with the layers each normalizing the input, performing a convolution, then passing the resulting data on to the next layer. The depth of each convolutional layer increases as the information becomes more dense through the pooling phase. Max Pooling, the final step of each convolutional layer, consolidates pixels values to maximize meaningful outputs, making each layer more effective at identifying class categories from characteristics of the image. The DNN operates with a built in feed-back loop—called back propagation, that uses iterative computing power to make many small adjustments to the weights in the convolutional layers, and pass on successful settings to the next round of calculations in the following epoch. *(Figure 4, 5.)* The DNN classifier is trained like this until it reaches a high level of accuracy with its predictions, then, the weights are frozen so they no longer change, and it's ready to be used to process new input imagery.[17,20,21]

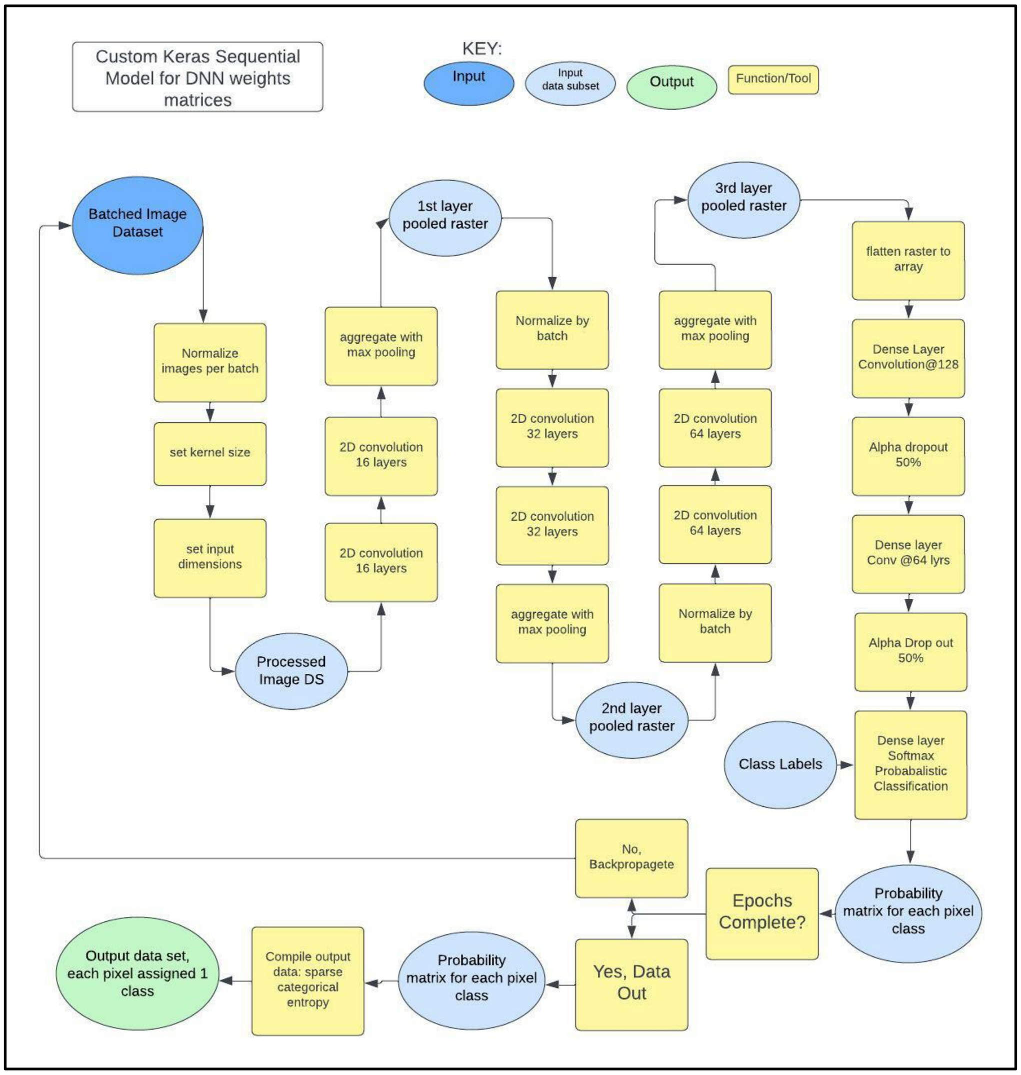
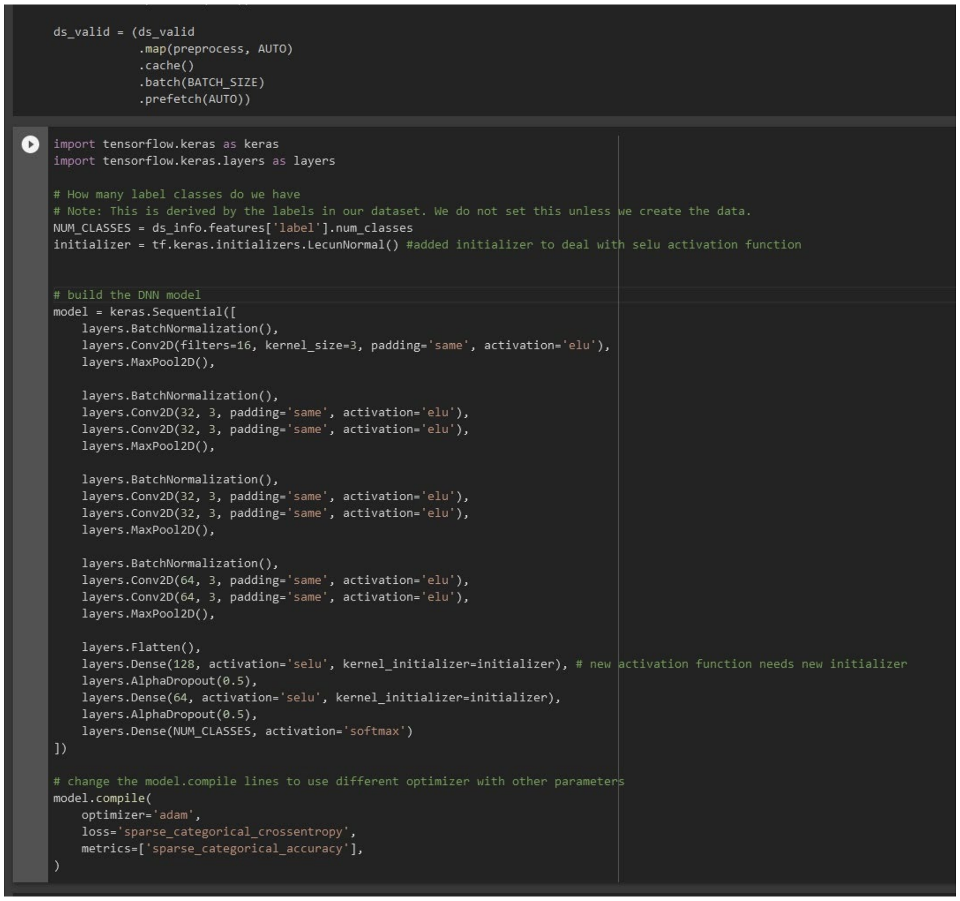


Fig 4. Detailed flow diagram of the Keras Sequential model, corresponds to the code snippet in figure 5.



```
ds_valid = (ds_valid
            .map(preprocess, AUTO)
            .cache()
            .batch(BATCH_SIZE)
            .prefetch(AUTO))
```

```
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

# How many label classes do we have
# Note: This is derived by the labels in our dataset. We do not set this unless we create the data.
NUM_CLASSES = ds_info.features['label'].num_classes
initializer = tf.keras.initializers.LecunNormal() #added initializer to deal with selu activation function


# build the DNN model
model = keras.Sequential([
    layers.BatchNormalization(),
    layers.Conv2D(filters=16, kernel_size=3, padding='same', activation='elu'),
    layers.MaxPool2D(),

    layers.BatchNormalization(),
    layers.Conv2D(32, 3, padding='same', activation='elu'),
    layers.Conv2D(32, 3, padding='same', activation='elu'),
    layers.MaxPool2D(),

    layers.BatchNormalization(),
    layers.Conv2D(32, 3, padding='same', activation='elu'),
    layers.Conv2D(32, 3, padding='same', activation='elu'),
    layers.MaxPool2D(),

    layers.BatchNormalization(),
    layers.Conv2D(64, 3, padding='same', activation='elu'),
    layers.Conv2D(64, 3, padding='same', activation='elu'),
    layers.MaxPool2D(),

    layers.Flatten(),
    layers.Dense(128, activation='selu', kernel_initializer=initializer), # new activation function needs new initializer
    layers.AlphaDropout(0.5),
    layers.Dense(64, activation='selu', kernel_initializer=initializer),
    layers.AlphaDropout(0.5),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

# change the model.compile lines to use different optimizer with other parameters
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy'],
)
```

Fig 5. The code block which defines the parameters of the Keras Sequential model.

Model Training

Here is an overview of how the Keras Sequential model is trained and used to identify land cover classes from images. In the top tier of *Figure 6,* the model's parameters are set, the loss scheme is chosen (sparse categorical entropy is a wordy way of saying each pixel can belong to one and only one class, the model attempts to minimize this loss parameter) and the number of iterations the model will perform, called epochs, is set.[22,23] In the lower tier, the model takes input from a pre-existing labeled training set, separates out data to reserve for validation, and runs the data through the model in batches for the set number of epochs. For each epoch the entire training set of images is passed through the network model, then the validation data is passed through, and through *back propagation*, the model uses the results from each batch to change the weights of the convolution to improve accuracy. *(Figure 4.)*

Once the training is complete and the model is accurately predicting cover classes from the training data, the layer settings are saved and frozen, and the model is ready for use on unlabeled data. When the convolutional layers of the DNN are frozen, they no longer adjust with the feedback. In a process called *transfer learning* (*Figure 3.*) a subset of the convolutional layers (usually the final dense layer) are allowed to continue to adjust as a new labeled data set is run through the classifier, allowing the model to improve in identifying regional land cover types.[24]

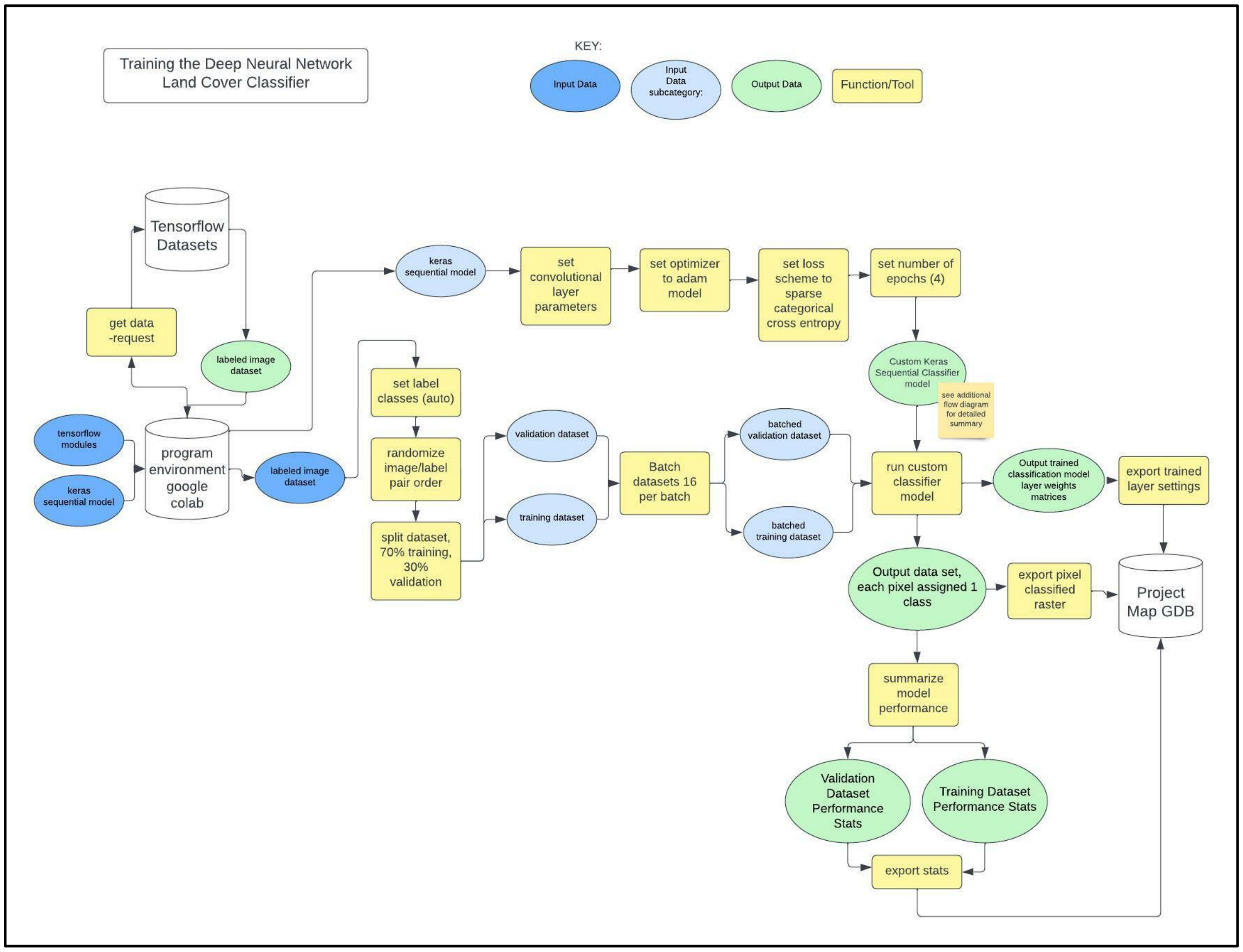
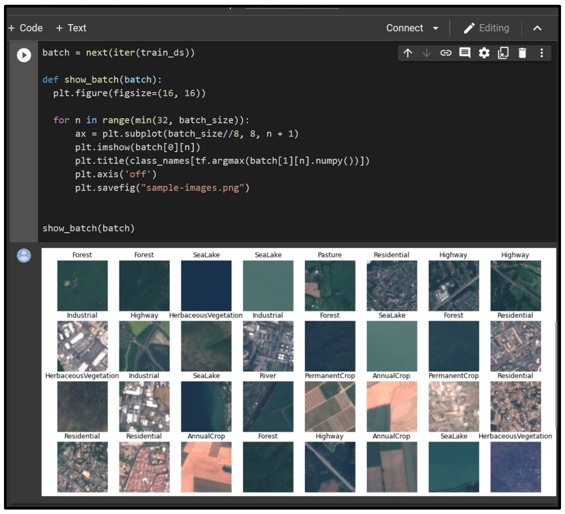

Training the Deep Neural Network Land Cover Classifier
KEY:
Input Data
Input Data subcategory:
Output Data
Function/Tool
Tensorflow Datasets
get data -request
labeled image dataset
tensorflow modules
keras sequential model
program environment google colab
labeled image dataset
keras sequential model
set convolutional layer parameters
set optimizer to adam model
set loss scheme to sparse categorical cross entropy
set number of epochs (4)
Custom Keras Sequential Classifier model
see additional flow diagram for detailed summary
set label classes (auto)
randomize image/label pair order
split dataset, 70% training, 30% validation
validation dataset
training dataset
Batch datasets 16 per batch
batched validation dataset
batched training dataset
run custom classifier model
Output trained classification model layer weights matrices
export trained layer settings
Output data set, each pixel assigned 1 class
export pixel classified raster
Project Map GDB
summarize model performance
Validation Dataset Performance Stats
Training Dataset Performance Stats
export stats

Figure 6. How the DNN model is trained, validated, and saved.

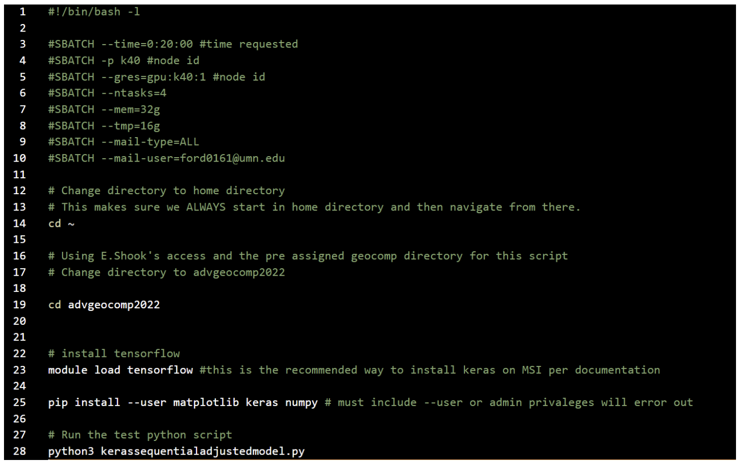


```
batch = next(iter(train_ds))

def show_batch(batch):
  plt.figure(figsize=(16, 16))

  for n in range(min(32, batch_size)):
      ax = plt.subplot(batch_size//8, 8, n + 1)
      plt.imshow(batch[0][n])
      plt.title(class_names[tf.argmax(batch[1][n].numpy())])
      plt.axis('off')
      plt.savefig("sample-images.png")


show_batch(batch)
```

Figure 7. A sample of the training dataset, visualized using Google Colab and Tensorflow with Eurosat data.

MSI

MSI is the University of Minnesota’s supercomputing cluster. Training the image classification model can be computationally intensive, and require many iterations, supercomputers are designed to handle this kind of computation very quickly. MSI has the Mesabi cluster’s k40 GPU, a very powerful graphics processor, great for the map algebra involved in the image/raster functions. The remote environment must be configured, and jobs created, queued and executed through the command line interface. I’ve written a bash script to set everything up on the university’s server and queue a job and set it to run for many epochs *(Figure 8.)* So far the Keras module is not installed correctly. Once this issue is fixed, I can run multiple jobs at once and potentially parallelize the process. In short the MSI resources are a very fast computer to speed up the training and classification process.[25,26]



```bash
#!/bin/bash -l

#SBATCH --time=0:20:00 #time requested
#SBATCH -p k40 #node id
#SBATCH --gres=gpu:k40:1 #node id
#SBATCH --ntasks=4
#SBATCH --mem=32g
#SBATCH --tmp=16g
#SBATCH --mail-type=ALL
#SBATCH --mail-user=ford0161@umn.edu

# Change directory to home directory
# This makes sure we ALWAYS start in home directory and then navigate from there.
cd ~

# Using E.Shook's access and the pre assigned geocomp directory for this script
# Change directory to advgeocomp2022

cd advgeocomp2022


# install tensorflow
module load tensorflow #this is the recommended way to install keras on MSI per documentation

pip install --user matplotlib keras numpy # must include --user or admin privaleges will error out

# Run the test python script
python3 kerassequentialadjustedmodel.py
```

Figure 8, an example of a submit BASH script to run the classifier on the MSI clusters.

Results

This model's training phase was executed successfully with input data from labeled Sentinel 2 imagery sourced from Tensorflow's online Datasets.[12] Over several iterations of shifting values in the convolutional layers, I achieved about 83% accuracy on labeling by class with a small number of 4 epochs *(Figure 9, 10.)* I now have a model with decent accuracy, but it is in need of more training with a much larger number of epochs. I have yet to initiate the transfer learning phase to work with all the Denver County data, that’s the next step.

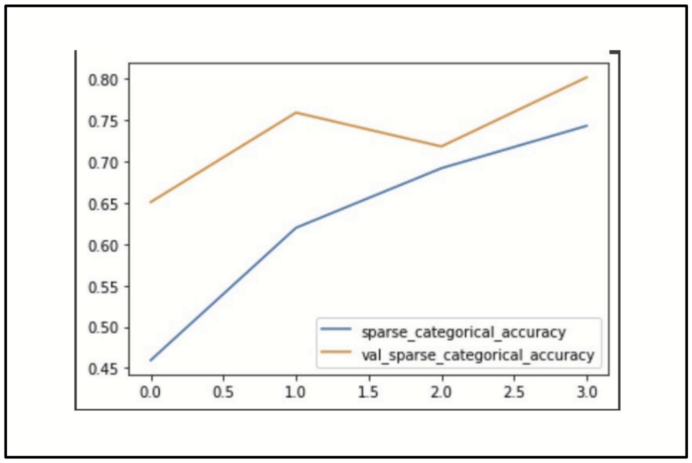
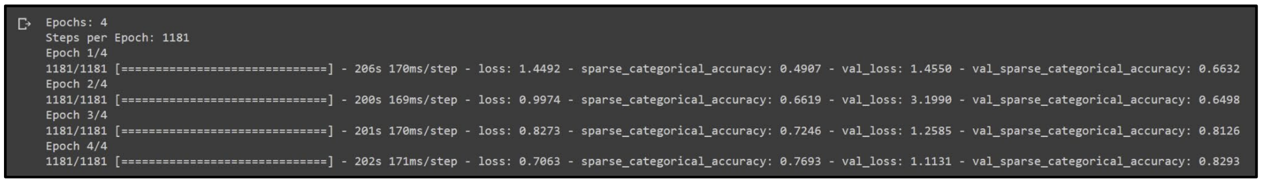


Figure 9. Chart of model accuracy over four epochs with performance on the training set in blue, and accuracy on the validation set in orange.



```
Epochs: 4
Steps per Epoch: 1181
Epoch 1/4
1181/1181 [==============================] - 206s 170ms/step - loss: 1.4492 - sparse_categorical_accuracy: 0.4907 - val_loss: 1.4550 - val_sparse_categorical_accuracy: 0.6632
Epoch 2/4
1181/1181 [==============================] - 200s 169ms/step - loss: 0.9974 - sparse_categorical_accuracy: 0.6619 - val_loss: 3.1990 - val_sparse_categorical_accuracy: 0.6498
Epoch 3/4
1181/1181 [==============================] - 201s 170ms/step - loss: 0.8273 - sparse_categorical_accuracy: 0.7246 - val_loss: 1.2585 - val_sparse_categorical_accuracy: 0.8126
Epoch 4/4
1181/1181 [==============================] - 202s 171ms/step - loss: 0.7063 - sparse_categorical_accuracy: 0.7693 - val_loss: 1.1131 - val_sparse_categorical_accuracy: 0.8293
```

Figure 10. Loss and accuracy stats for each epoch of the DNN model.

Results Verification

After each epoch the model performs a validation cross check, running the novel validation data through the model. This helps to identify overfitting, and gives an insight into the overall accuracy the model will have on images it has not been trained on.[24] The validation accuracy of my model is plotted in orange on the graph above *(Figure 9, 10.)* This is done in a cross validation scheme; where a separate labeled validation set with the labels removed are fed through the model, and the output labels compared to the predefined labels. An accuracy assessment plan of the Denver County data requires cross validation with a manually labeled validation set of randomly chosen images to be compared against the predicted output. The Chesapeake model was able to achieve a 91% average accuracy across its study area. Similar results are expected. [1]

Discussion and Conclusion

I now have a model with decent accuracy, trained on the Eurosat input data, but it is in need of more training with a much larger number of epochs. Increasing the number of epochs drastically increases computational cost. I have access to the university's MSI super computer clusters, but have yet to be successful running the Tensorflow script on the MSI GPUs. The tensorflow modules install successfully with the submit script, which sets up the programming environment on the supercomputer node, but they don't appear to contain the correct Keras sub modules needed to run the sequential convolution model.

I ran into stumbling blocks pretty much every direction I turned at this point. I started scripting in a jupyter lab python environment in order to use tensorflow through google colab. I ran into trouble with access to data (arc online has access to the Chesapeake Bay data, but I am locked out of it in the Jupyter Lab environment, as it requires a license to access.) I then tried the same approach in ArcPro, following a tutorial using Esri's proprietary classification tools, but it seemed these were old modules and dependencies had changed. It appears they are designed specifically for NVIDIA GPUS, and won't work with other hardware configurations. I also ran into conflicts between arcpy and the esri GIS module, where the GIS tools wouldn't work in the ArcPro notebook environment, and the arcpy tools wouldn't work in the standard jupyter notebook environment. I consulted with Eric Shook about this and he explained that Esri updates modules often and dependency changes can cause these kinds of compatibility issues from year to year. As a result I haven't yet got to the transfer learning phase. Further, the datasets themselves are enormous and are a challenge to work with; the Denver CO Lidar Data was over 200GB for the county. The NAIP data is also large and detailed, and I would like to experiment with using Planet imagery to increase the temporal resolution of the classifications. At this point I've started to investigate the Microsoft Azure platform, which I have access to through a student account which is built around the newer NVIDIA GPUs. I am in the process of building a new model using the Chesapeake Bay data on that platform, though it appears the student license may be limited.

In summary, the challenges presented in working with the volume and diversity of data types in the relatively new domain of Deep Neural Networks has been an immense challenge given my experience level. I am looking forward to using the working Keras model on a GPU in the future, getting the classification model up and running through the Azure platform, and continuing to develop the DNN model in colab. Proving that the CBC model is adaptable for other watersheds at the county scale will open the door for further work on the entire Colorado River watershed*,* allowing for change detection and monitoring at high spatial resolutions.[11] The reproducibility of the process will allow for further development of multiple time periods, data essential for monitoring the impact of the continuing drought and climate change on a valuable and fragile watershed.

References

1. Claggett, P. *et al.* Chesapeake Bay Program's One-meter Resolution Land Use/Land Cover Data: Overview and Production. 61.
2. Marcos, D., Volpi, M., Kellenberger, B. & Tuia, D. Land cover mapping at very high resolution with rotation equivariant CNNs: Towards small yet accurate models. *ISPRS J. Photogramm. Remote Sens.* **145**, 96–107 (2018).
3. Zhu, X. X. *et al.* Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources. *IEEE Geosci. Remote Sens. Mag.* **5**, 8–36 (2017).
4. Earth Resources Observation And Science (EROS) Center. National Agriculture Imagery Program (NAIP). (2017) doi:10.5066/F7QN651G.
5. Colorado County Boundaries. https://data-cdphe.opendata.arcgis.com/datasets/colorado-county-boundaries/explore?location=38.973275,-105.550600,7.55.
6. Colorado River Basin GIS Open Data Portal. https://coloradoriverbasin-lincolninstitute.hub.arcgis.com/.
7. LidarDownload - CO Hazard Mapping & RiskMAP Portal. https://coloradohazardmapping.com/LidarDownload.
8. Regional Planimetric Data Project. *DRCOG* https://drcog.org/services-and-resources/data-maps-and-modeling/regional-planimetric-data-project (2018).
9. Regional Land Use Land Cover Project. *DRCOG* https://drcog.org/services-and-

resources/data-maps-and-modeling/regional-land-use-land-cover-project (2019).
10. stactools-chesapeake-lulc. (2022).
11. Azure/pixel_level_land_classification: Tutorial demonstrating how to create a semantic segmentation (pixel-level classification) model to predict land cover from aerial imagery. This model can be used to identify newly developed or flooded land. Uses ground-truth labels and processed NAIP imagery provided by the Chesapeake Conservancy. https://github.com/Azure/pixel_level_land_classification.
12. eurosat | TensorFlow Datasets. *TensorFlow* https://www.tensorflow.org/datasets/catalog/eurosat.
13. ArcGIS. https://www.arcgis.com/home/item.html?id=10df2279f9684e4a9f6a7f08febac2a9#.
14. Starter: EuroSAT. https://kaggle.com/code/ryanholbrook/starter-eurosat.
15. Helber, P., Bischke, B., Dengel, A. & Borth, D. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. Preprint at https://doi.org/10.48550/arXiv.1709.00029 (2019).
16. Prabhu. Understanding of Convolutional Neural Network (CNN) — Deep Learning. *Medium* https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (2019).
17. Pramoditha, R. Coding a Convolutional Neural Network (CNN) Using Keras Sequential API. *Medium* https://towardsdatascience.com/coding-a-convolutional-neural-network-cnn-using-keras-sequential-api-ec5211126875 (2022).
18. Sohn, G. & Dowman, I. Data fusion of high-resolution satellite imagery and LiDAR data for automatic building extraction. *ISPRS J. Photogramm. Remote Sens.* **62**, 43–63 (2007).
19. National Agriculture Imagery Program (NAIP) Data Dictionary | U.S. Geological Survey. https://www.usgs.gov/centers/eros/science/national-agriculture-imagery-program-naip-data-dictionary#acquisition_date.
20. Team, K. Keras documentation: The Sequential model. https://keras.io/guides/sequential_model/.
21. Team, K. Keras documentation: Keras FAQ. https://keras.io/getting_started/faq/#why-is-my-training-loss-much-higher-than-my-testing-loss.
22. Team, K. Keras documentation: Probabilistic losses. https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class.
23. Ananthram, A. Keras Transfer Learning For Beginners. *Medium* https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e (2018).
24. Team, K. Keras documentation: Training & evaluation with the built-in methods. https://keras.io/guides/training_with_built_in_methods/.
25. msi_planet/README.md at master · uspatial/msi_planet. *GitHub Enterprise* https://github.umn.edu/uspatial/msi_planet.
26. Distributed training with TensorFlow | TensorFlow Core. *TensorFlow* https://www.tensorflow.org/guide/distributed_training.

Self-score

Fill out this rubric for yourself and include it in your lab report. The same rubric will be used to generate a grade in proportion to the points assigned in the syllabus to the assignment.

Category	**Description**	**Points Possible**	**Score**
Structural Elements	All elements of a lab report are included **(2 points each)**: Title, Notice: Dr. Bryan Runck, Author, Project Repository, Date, Abstract, Problem Statement, Input Data w/ tables, Methods w/ Data, Flow Diagrams, Results, Results Verification, Discussion and Conclusion, References in common format, Self-score	28	**28**
Clarity of Content	Each element above is executed at a professional level so that someone can understand the goal, data, methods, results, and their validity and implications in a 5 minute reading at a cursory-level, and in a 30 minute meeting at a deep level **(12 points)**. There is a clear connection from data to results to discussion and conclusion **(12 points)**.	24	**24**
Reproducibility	Results are completely reproducible by someone with basic GIS training. There is no ambiguity in data flow or rationale for data operations. Every step is documented and justified.	28	**28**
Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.	20	**20**
		100	**100**