-Independent Work Report Spring 2023-

Analyzing Misclassifications of Architectural Images via Simple Machine Learning Models

Sydney Berry ‘24
Advisor: Brian Kernighan

Abstract

Machine learning models that can identify building types would be useful for improving urban planning and population density mapping. However, very little research has been done to create these models. In this project I created two simple machine learning models and asked them to distinguish between two different types of buildings. My first model was a linear classifier that uses stochastic gradient descent and a softmax loss function. My linear classifier model had the easiest time identifying stores, which had letters printed on the store front. In future research, it could be beneficial to develop models that focus on searching for distinct features that are traditionally present in a particular type of building. My second model was a CNN that uses convolution, pooling, ReLu, flatten, linear, and softmax layers. This model never achieved a training accuracy above random guessing. Over the course of doing this project, I learned that completing this research will involve developing a new large dataset of architectural images with stereotypical examples of various types of buildings. In addition, I learned that this research will require optimized packages and coding environmentals with a lot of computational resources in order to make runtimes feasible.

1. Introduction

As computers have become more powerful and accessible, they have quickly infiltrated various fields of research to serve as a tool. One of these fields is architecture. However, the introduction

of computers into the field of architecture was not welcomed with open arms. When digital tools were first introduced, they were criticized for stifling creativity. Several architects felt that it caused people to “make decisions too fast” [1]. However, the functionality of new tools allows people to address new goals. Computers can quickly do calculations that are painstakingly slow to do by hand. While it requires some time to become familiar with new software, their benefits can enrich and open new possibilities.

The incorporation of computers into architectural work has taken several different forms. People have developed software such as Rhino and AutoCAD to allow architects to design their projects on computers rather than hand drawing. There has also been ongoing research on using machine learning to identify architectural elements using CNNs (as is currently being done by the National University of Singapore with Chen et al.’s research) [2] and auto-generating new designs for buildings through projects such as DALL-E [3]. Both of these projects require that machines learn the similarities and differences between various architectural features in order to tell what the feature is and where it might go in a design.

Creating a model that can identify different types of buildings could be used “for urban utility planning and population density mapping at a finer level of urban intrinsic scale” [2]. Since the model could be used to predict buildings' purposes, it could use images of areas rather than human input labels to map out what type of buildings dominate particular areas. However, research on how to create models that can identify a building’s purpose from a photograph is currently extremely limited. Jielin Chen and her team began working on this problem by compiling their own dataset and using it to train 2 different CNN architectures [2]. However, some of their findings might be linked to how they structured their dataset using various class sizes, rather than the CNN’s ability to learn.

The goal of my project was to explore simple machine learning models' misclassifications of building types to gain a better insight into the challenges that developing building identification models will have to resolve. Due to the time it takes to develop and run these models, this project focuses on two models. The first model uses a stochastic gradient descent linear classifier on the pixels of architectural images. The second model uses a CNN with multiple convolution and pooling layers. The original plan was to analyze the misclassifications of these models to learn the similarities and differences between types of buildings. In addition, this project aimed to examine the differences in misclassifications between the two models to see what features work well with each model.

However, I encountered challenge after challenge when attempting to complete this project. One of the biggest issues was computational power. For future work in this realm, I recommend exploring optimized packages and coding environments with high computational abilities to help deal with the large number of inputs and parameters needed to construct these models. In order to combat these issues, I created models that only had to differentiate between two types of buildings at one time. In doing so, I found that it was relatively easy for the linear classifier to identify stores. Ultimately, I was not able to adequately train a large enough CNN to identify between two types of buildings using the small datasets I created from Chen et al.'s research. It is necessary to use complex training techniques such as data augmentation or transfer learning in order to train a CNN to identify different types of buildings using a relatively small dataset. For future research, it would be useful to have a larger and more standard set of images to train a CNN rather than the one Chen et al. created.

2. Related Work

When I originally began this project, I wanted to determine similarities and differences in various architectural forms by writing a program that analyzed the annotations of an Annotated Image Database of Architecture for common words and phrases. In order to do this project, I needed to find a dataset that included annotations of the architectural elements, such as whether the building pictured included columns and bay windows. In searching for a dataset that would meet my needs, I quickly found that the availability of datasets of architectural images that were already organized and labeled and could be easily adapted to use as input for a program was very limited. Previous researchers have assembled several data sets relating to building styles. However, these datasets and the models they can train cannot fulfill the same purposes of aiding in urban planning mapping as a model that could identify different types of buildings. When doing their research, Jielin Chen and her team found very limited prior attempts to design and test "individual building instance classification" models. In order to aid other researchers, Chen et al. developed an Annotated Image Database of Architecture [2]. However, Chen et al.'s dataset only contains basic labels describing the building pictured that do not extend as far as annotating architectural features within the image. In searching for datasets to use for this project, I was unsuccessful in finding any datasets that included annotations of individual features. In addition, I could not find any datasets besides Chen et al.'s that provided well-labeled architectural images of various building types that could be easily used for a program's input. Thus, I decided to modify my project to use Chen et al.'s dataset to explore simple machine learning models' misclassifications of building types in the hope that these misclassifications would give me more insight into the similarities and differences in various building types and how models can recognize them.

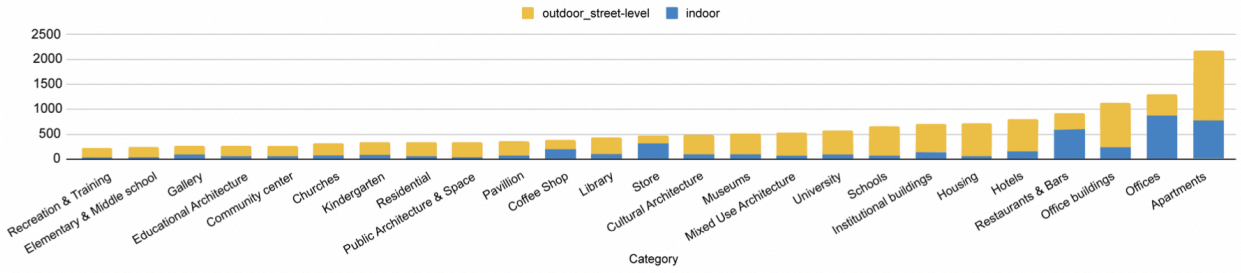
Chen et al.'s dataset consists of 256 by 256 pixel images from ArchDaily which Chen et al. states probably has "the largest online repository of architecture projects worldwide." Chen et al. filtered the images to make sure every image was a real-world photo that focused on a piece of architecture. They used information from ArchDaily's website to construct all of their labels, and they developed their dataset to have various levels of annotations. They provided a string of binary 0 and 1 labels to indicate whether a photograph was or was not an aerial shot, was taken indoors or outdoors, and what type of building was pictured in the image. The string contained a 0 for every category the building in the picture did not relate to and a 1 for every category it belonged to. Chen et al. also compiled a CSV that contains the file name of every image, its architectural label (building type), scene label (indoor or outdoor), ArchDaily label, file number, the name of the building photographed, the building's architect, the building's city, the country in which the building is located, the building's area, the year the building was built, the image URL, the image source, and the project URL. For this project, I chose to focus on Chen et al.'s building-type labels. In order to deal with issues of class imbalance, Chen et al. decided to discard types of buildings that they deemed to have too few or too many images associated with them. However, their dataset still contains a large disparity between the number of images for each type of building. The number of images per category in Chen et al.'s dataset is shown in Figure 1. One way Chen et al. analyzed their data was by examining weighted F1 scores at various epochs. An epoch is one iteration through the training algorithm. The F1 scores were calculated using this formula: $F1 = 2(rp)/(r + p)$ where

$p = true\ positive\ /\ (true\ positive\ +\ false\ positive)$ and

$r = true\ positive\ /\ (true\ positive\ +\ false\ negative)$. When training their models, Chen

et al. found there to be fluctuations in F1 scores in early epochs that they thought could be related to the varying class sizes [2].

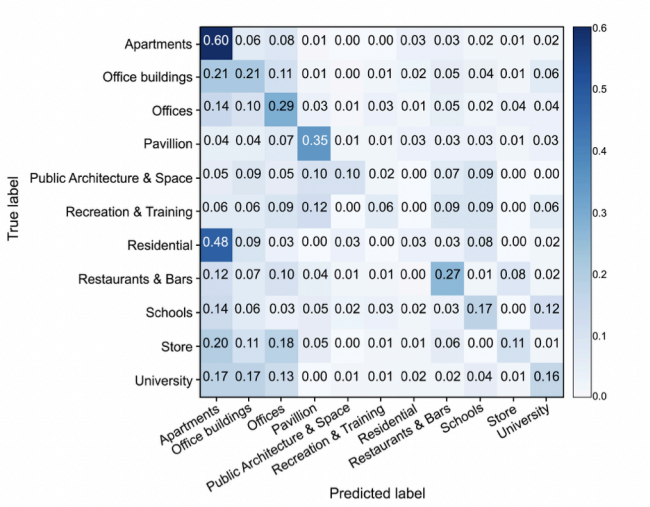
Figure 1. "Number of images per category in AIDA, sorted in ascending order; AIDA contains 14,659 images from 25 architecture categories." (Reprinted from ref 2. Copyright 2021.)



After assembling their dataset, Chen et al. used it to conduct their own research in which they developed a CNN with a multi-label classification network. Their CNN framework produces two labels: one to identify the building scene (indoor and outdoor) and one to identify the building type. Chen et al. experimented using two different CNN architectures: ResNeXt and DenseNet. For the output layer, Chen et al. used a sigmoid function since it allows them to do multi-label classification. Chen et al. used PyTorch in order to create and test their CNNs. Chen et al. deemed their dataset as not sufficiently large enough to train their CNN from scratch, so they used the transfer learning approach. As a result, both of Chen et al.'s CNN architectures were pre-trained on the ImageNet database. When training their network using their own dataset, they increased their input by using data augmentation using three different settings. The first setting was to create a random crop of 224 by 224 pixels and then resize the crop to be 256 by 256 pixels. The second setting was to rotate the image by a random angle between -15 and 15 degrees. The third setting was to flip the images horizontally 50% of the time. Data augmentation and the transfer learning approach allowed Chen et al. to train their large CNN with a relatively small dataset [2].

When evaluating the two different CNN networks Chen et al. was testing their dataset on, they concluded that the CNN using the DenseNet architecture was slightly more accurate. In Figure 2, you will find the normalized confusion matrix for 11 select classes in Chen et al.'s database. The confusion matrix shows how accurate the DenseNet CNN was in making predictions after 70 epochs. According to Chen et al.'s matrix, the category DenseNet identified correctly most often was Apartments. DenseNet got the true label for Apartments 60% of the time. However, part of this accuracy could have been skewed by the model guessing Apartment often. DenseNet incorrectly labeled 48% of the residential buildings as apartments [2]. One reason the model could be more keen to predict apartments is the disparity in class size. Chen et al.'s dataset contains over 2,000 images of apartments and less than 500 images of residential buildings. When a program guesses that an image is an apartment without knowing any other information, it is more likely to be correct than if it had guessed a residential building. When performing my own experiments, I restructured the data to examine equal class sizes to eliminate this concern and see if any of the results differed.

Figure 2. "Normalized confusion matrix for architectural category prediction with DenseNet-161 trained until epoch 70 (showing only 11 selected categories), demonstrating the convoluted inter-class relationships between different architectural categories." (Reprinted from ref 2. Copyright 2021.)



3. Approach

When I initially started experimenting with my linear classifier and CNN programs, I tried running the entirety of Chen et al.'s dataset and asking the programs to create models that could distinguish between all 25 of their categories. However, I quickly realized that due to how my code was written, the runtimes for these programs were not practical. I ran one of the programs for 8 hours before it crashed. Extrapolating how many training iterations were left when it crashed, I calculated that it would take 200 hours for this program to run one time. From these experiments, I concluded that I needed to design an approach to this project that would minimize the amount of data my programs needed to examine.

I first attempted to reduce the quality of the pictures from 256 by 256 pixels to 32 by 32 pixels. However, I quickly realized that this obscured the buildings in the photograph beyond recognition. Thus, the models would no longer be able to examine architectural features in the photographs as they would all be blurred.

If I could not reduce the quality of the images, then I decided I needed to reduce the number of images that the programs would be examining at one time. Since Chen et al.'s dataset was relatively small compared to the number of parameters they thought were necessary to create their models, they had to train their models with ImageNet before using them with their own dataset. Knowing this, I thought it was extremely impractical that I could reduce the training set even further and continue to create a model that could discern between 25 types of buildings. Thus, I decided I would create models whose tasks were to distinguish between two different types of buildings. I ran my programs on various combinations of different types of buildings. I then recorded the test accuracies and the confusion matrices for the test set. I was then able to

analyze how the test accuracies differed between the pairings to determine which buildings were easier for the models to identify.

I decided to focus on five different types of buildings that were included in Chen et al.'s dataset: apartments, houses, churches, stores, and universities. There is an example image from Chen et al.'s dataset for each type of building that is shown in Figure 3 below. I picked these five categories because I felt that some of these buildings would have similar architectural elements, while others would have very different structures. These guesses were based on my prior knowledge of these buildings. For example, I thought apartments and houses would both contain architectural elements that alluded to them being residential buildings. However, I expected apartments to be relatively bigger than most houses as single family houses are normally one to three stories while mid-rise apartment buildings have nine to twelve stories. I expected several of the churches to include distinct symbols such as crosses to allude to it being a place of worship. Stores were likely to have signs with advertising and displays of products being sold inside. I was curious where university buildings would fall as there are a wide variety of different types of buildings found on university campuses.

Figure 3. Example Images from my test sets

Originally, I was planning to analyze the output from both my linear classifier and CNN programs to answer the following questions: Is one model consistently more accurate? Are the

most accurate pairings consistent between models? What does examining the images reveal about the categories that were easy and hard to identify? However, some of these questions became harder to answer when I was unable to obtain usable output from my CNN. Despite this, I managed to gain a greater appreciation for the complexity of the challenge of identifying building types from photographs. In addition, I learned better approaches to take from the numerous challenges I faced, which would hopefully lead to more concrete answers to these questions if they were used in the future.

4. Implementation

4.1 Organizing Data

I obtained the images I used as data from Jielin Chen's team's Annotated Image Database of Architecture. All of Chen et al.'s pictures were 256 by 256 pixels with filenames describing the type of building shown in the picture. I created my datasets by randomly selecting 100 training, 50 validation, and 25 testing exterior architectural images for each type of building I was experimenting with. After doing some initial experiments with this distribution, I reorganized my data to have 140 training images, 18 validation images, and 17 testing images for each building type. This redistribution made my data have an 80-10-10 percent split between the different sets, which is typically used by computer science researchers.

When randomly selecting exterior pictures from Chen et al.'s dataset, I purposely did not include any interior images. I chose to focus on pictures of the buildings' exteriors because I wanted my project to focus on architectural elements that provided information of the building's purpose. Including interior images would have given the model information about the furniture and items inside the building. For example, shelves and piles of items such as clothing would

indicate a store. However, these were not the items I wanted my model to be focusing on. Exterior images included less of this information even though some interior spaces were occasionally visible through windows. Thus, my model could focus on features architects designed to help signify the intended purpose of the building.

I stored these 15 sets of images in their own separate folders. Before running my models, I would combine the training, validation, and testing sets for the two categories I was currently examining into temporary folders. When I was done collecting data for that pairing, I would move the files back to their separate folders for future use.

4.2 Pre-processing data

The first thing my code does is prepare my data to be input into my models. All my data is sorted into folders. I use Python's global package to read the file list for the folders from my local computer into my program. I then read all the images in each file list using cv2, which is an open source computer vision library. I store the images in an array. I kept all the red, green, and blue values for each pixel and my models use all these values to train and predict what type of building is in the image. Since the file names provided by Chen et al. all contain the building type they labeled the image as, I wrote a function that examines the file name and creates an array of 0s and 1s to signify which category the image pertains to. The CNN model I am using needs the data from my images in a different order than it was initially read in as. Because of this, I used temporary arrays and for loops to rearrange the data into the appropriate order without overwriting data in the original array.

4.3 Linear Classifier Model

I based my linear classifier model on an implementation I worked on in Princeton's COS 429: Computer Vision course taught by Olga Russakovsky [5]. My linear classifier model starts by flattening the data by reshaping the images' pixel values into vectors. It normalizes the data by subtracting the mean training image from all images. It also adds a basis term of 1 to every image. The flattened image data is passed into my linear classifier, which uses stochastic gradient descent. The linear classifier does 100 training iterations in which it randomly chooses 200 training images to examine in each iteration. In each iteration, the program calculates the gradient for the softmax loss function and then updates the model by subtracting the gradient times the learning rate of 0.001. The softmax loss function uses a softmax activation function plus a cross-entropy loss. The softmax activation function assigns a probability of occurrence to each class. The cross-entropy loss sums the negative logarithm of all the probabilities.

The code is written to use the validation set to set a regularization strength hyperparameter. However, running this part of the program takes significant time for each regularization strength. Thus, I tested multiple regularization strengths on the Apartments and Churches' validation set. My results from these tests are shown in Table 1 on the following page. From these results, I decided to use a regularization strength of 0.002 to train all my linear classifier models, because it had the highest validation accuracy on the set of images for Apartments and Churches. Since Apartments and Churches ended up having one of the lowest test accuracies in my initial experiments, I do not think this decision biased my results to this pairing. Since I set a standard validation accuracy, I no longer needed validation sets. Thus, I used all my validation sets as a second test set for all my remaining experiments with my linear

classifier. My code outputs are training accuracy, validation accuracy (or a second test accuracy), test accuracy, and the confusion matrix. I use this output to analyze the misclassifications.

Table 1. Apartments and Churches' validation accuracies for various regularization strengths when trained with my linear classifier.

Regularization Strength	Validation Accuracy
0.0002	0.68
0.002	0.76
0.02	0.59
0.2	0.68
2	0.68
20	0.68

4.4 CNN Model

I based my CNN on a CNN I developed in Princeton's COS 429: Computer Vision course to distinguish between a set of images with 10 categories, some of which were human, dog, and airplane [4]. My original network had a convolution layer followed by a pooling layer. These are followed by another convolution and pooling layer. Then there is a ReLu layer, a flatten layer, a linear layer, and a softmax layer. Over the course of this project, I experimented with adding and removing convolution and pooling layers. I found very little difference in the outputs when I made these changes.

While my CNN worked well for its original COS 429 task, my building training images for this project were significantly larger than the images that this CNN was designed for. In addition, I had less training data than I used to train the original CNN. Thus, my model had more parameters within the model that needed to be trained (especially for the linear layer) and less training images to examine. Optimistically, I originally thought that having to distinguish between two classes instead of 10 would lessen the amount of training data I needed because I

was planning to use less convolution and pooling filters in order to learn details in the images. Initially, I was using learning rates of 0.1 and 0.01. However, all my training losses were Not a Number (NaN), a numeric data type for values that are undefined or cannot be represented. Thus, I concluded my learning rate was too high. When I tried running my program with a learning rate of 0.001, half the time I would get a learning rate of NaN and half the time my program was able to correctly learn to 100% accuracy a set of 10 training images. However, the model identified all the images in the test set as only one category even when I increased the training set of images to 280. When I tried running my program with learning rates less than 0.001, the program appeared to be learning nothing. Even with a training set of 10 images, the CNN network kept the same guesses for all 10 images for 50 iterations. Ultimately, I had to conclude that my dataset was not large enough to train a CNN to distinguish between two different buildings pictured in Chen et al.'s dataset without using advanced training methods such as data augmentation and the transfer learning approach.

4.5 Issues with Computational Power

At all stages of implementing this project, I kept encountering issues with the computational power my computer had and the demands of my programs. The files that I adapted from COS 429 in order to do this project contained lots of Numpy objects and operations. Numpy arrays exist in the Random Access Memory of the host and not the GPU memory. Thus, using Colab Pro's GPU could not drastically speed up my code. If I had the time, I would have liked to become more familiar with packages such as PyTorch since PyTorch tensors would work well on a GPU.

In terms of finishing this project on time, I designed the approach of this project to reduce the amount of data I need to run on the program in order to reduce the run times. In addition, I abandoned a Spatial Pyramid Matching model that I had been developing along with my other two models. A Spatial Pyramid Matching model examines features at various levels in an image while weighing features at higher levels more than features at lower levels. The philosophy behind this model is that higher levels localize features more precisely. Since Spatial Pyramid Matching takes into account spatial information within an image by studying various regions of it block by block, it tends to perform well on scene classification problems. Thus, I initially thought it could be a promising model to test Chen et al.'s images on. However, needing to examine my large 256 by 256 images multiple times (one time for each level) proved to be extremely computationally demanding. Thus, I was unable to complete test runs for this model.

5. Evaluation

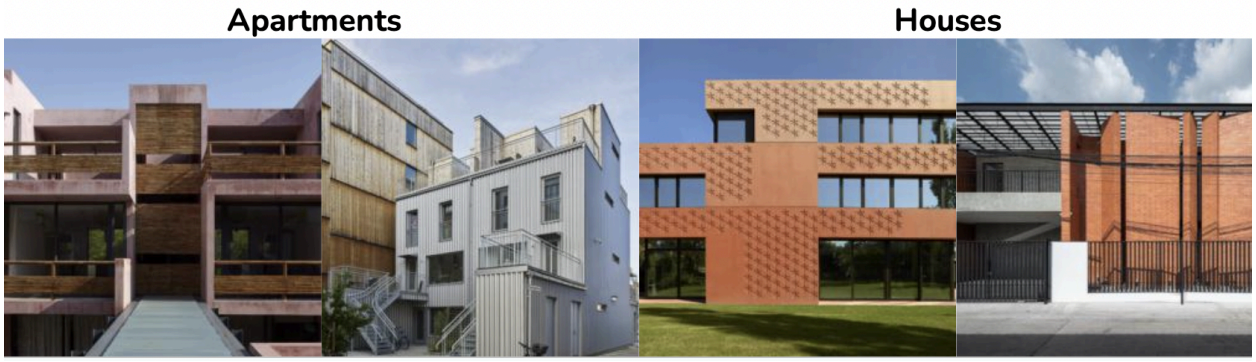
The following 11 tables display all the data I collected from my linear classifier model. Table 2 shows my initial results with 100 training, 50 validation, and 25 testing exterior architectural images for each type of building. In my program, I gave each category the program was currently examining a label of 0 or 1. Here I am showing the models' test accuracy as well as the information provided by the confusion matrix. A confusion matrix shows how many predictions were correct and incorrect for each class being examined. For the second row of the table, Apartments were labeled as 0 and Churches were labeled as 1. The final test accuracy of the model created by the linear classifier was 56%. The program correctly identified 9 images that were apartments as apartments. It correctly identified 19 images as churches. It incorrectly identified 6 churches as apartments. It also incorrectly identified 16 apartments as churches.

I used Table 2 to verify that my linear classifier model was learning from the training data and was able to discern between two different types of buildings with a greater than random chance test accuracy. The only test accuracy worse than random guessing was for the pairing of Apartments and Houses. Since the program is trying to differentiate between two categories, random guessing would have a 50% accuracy. The linear classifier created a model that had a 40% test accuracy. However, looking at pictures in my test sets, the distinction between an apartment and a house is sometimes hard for me to recognize as well. Select images from my test set that are hard for me to distinguish between are shown in Figure 4. Since this is the only pairing with a test accuracy worse than random guessing, I am confident that the model is capable of learning to distinguish between building types.

Table 2. Initial Results from Linear Classifier before redistributing data to have 80-10-10 split.

Category 0	Category 1	Test Accuracy	True 0	True 1	False 0	False 1
Apartments	Houses	40%	10	10	15	15
Apartments	Churches	56%	9	19	6	16
Apartments	Universities	54%	8	19	6	17
Apartments	Stores	62%	13	18	7	12
Stores	Universities	74%	19	18	7	6
Stores	Churches	76%	19	19	6	6
Stores	Houses	80%	21	19	6	4
Houses	Churches	62%	17	14	11	8
Houses	Universities	64%	18	14	11	7
Churchs	Universities	70%	13	18	7	12

Figure 4. Example Images of Apartments and Houses from my test sets



After I was confident the linear classifier was learning, I ran my program twice for each pairing and recorded the accuracies of both runs for both test sets (one of which was originally made to be a validation set) and the confusion matrices for test set 2 (the set of images originally made to be the test set). I decided to run the program twice for each category because sometimes there was a large variance in test accuracies between runs. For example, the test accuracy for test set 2 for Apartments and Churches was 0.53 for the first run and 0.71 for the second run. In order to eliminate some of the variation, I took the average of the two runs for every pairing. Still, some of the pairings had a lot of variation in the average test accuracies for the two test sets. This could be due to how small the test sets are. Test Set 1 contains 36 images and Test Set 2 contains 34 images. Thus, a few images that were randomly chosen but are harder to identify could skew the accuracy more than would occur for a larger test set. In order to get a better idea of each pairing's test accuracy, I averaged both test sets' average accuracy for both runs. This data is displayed in Table 13.

Table 3. Linear Classifier's Output for Apartments (0) and Houses (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.79	0.79	0.79
Test Set 2 Accuracy	0.65	0.53	0.59
Test Set 2 True 0	8	10	9
Test Set 2 False 1	9	7	8
Test Set 2 False 0	3	9	6
Test Set 2 True 1	14	8	11

Table 4. Linear Classifier's Output for Apartments (0) and Churches (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.69	0.64	0.67
Test Set 2 Accuracy	0.53	0.71	0.62
Test Set 2 True 0	6	10	8
Test Set 2 False 1	11	7	9
Test Set 2 False 0	5	3	4
Test Set 2 True 1	12	14	13

Table 5. Linear Classifier's Output for Apartments (0) and Universities (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.58	0.67	0.63
Test Set 2 Accuracy	0.74	0.71	0.72
Test Set 2 True 0	12	11	11.5
Test Set 2 False 1	5	6	5.5
Test Set 2 False 0	4	4	4
Test Set 2 True 1	13	13	13

Table 6. Linear Classifier's Output for Apartments (0) and Stores (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.89	0.89	0.89
Test Set 2 Accuracy	0.59	0.62	0.60
Test Set 2 True 0	9	9	9
Test Set 2 False 1	8	8	8
Test Set 2 False 0	6	5	5.5
Test Set 2 True 1	11	12	11.5

Table 7. Linear Classifier's Output for Stores (0) and Universities (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.83	0.83	0.83
Test Set 2 Accuracy	0.85	0.76	0.81
Test Set 2 True 0	14	12	12
Test Set 2 False 1	3	5	5
Test Set 2 False 0	2	3	3
Test Set 2 True 1	15	14	14

Table 8. Linear Classifier's Output for Stores (0) and Churches (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.83	0.86	0.85
Test Set 2 Accuracy	0.65	0.65	0.65
Test Set 2 True 0	9	8	8.5
Test Set 2 False 1	8	9	8.5
Test Set 2 False 0	4	3	3.5
Test Set 2 True 1	13	14	13.5

Table 9. Linear Classifier's Output for Stores (0) and Houses (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.89	0.89	0.89
Test Set 2 Accuracy	0.74	0.74	0.74
Test Set 2 True 0	11	14	12.5
Test Set 2 False 1	6	3	4.5
Test Set 2 False 0	3	6	4.5
Test Set 2 True 1	14	11	12.5

Table 10. Linear Classifier's Output for Houses (0) and Churches (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.61	0.61	0.61
Test Set 2 Accuracy	0.62	0.59	0.60
Test Set 2 True 0	11	8	9.5
Test Set 2 False 1	6	9	7.5
Test Set 2 False 0	7	5	6
Test Set 2 True 1	10	12	11

Table 11. Linear Classifier's Output for Houses (0) and Universities (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.64	0.50	0.57
Test Set 2 Accuracy	0.59	0.65	0.62
Test Set 2 True 0	10	10	10
Test Set 2 False 1	7	7	7
Test Set 2 False 0	7	5	6
Test Set 2 True 1	10	12	11

Table 12. Linear Classifier's Output for Churches (0) and Universities (1)

	Run 1	Run 2	Average
Test Set 1 Accuracy	0.72	0.61	0.67
Test Set 2 Accuracy	0.47	0.47	0.47
Test Set 2 True 0	6	10	8
Test Set 2 False 1	11	7	9
Test Set 2 False 0	7	11	9
Test Set 2 True 1	10	6	8

Table 13. Averaged Test Sets' Average Accuracy for 2 runs for Linear Classifier

Building Types	Averaged Test Sets' Average Accuracy for 2 runs
Apartments & Houses	0.69
Apartments & Churches	0.64
Apartments & Universities	0.67
Apartments & Stores	0.75
Stores & Universities	0.82
Stores & Churches	0.75
Stores & Houses	0.81
Houses & Churches	0.61
Houses & Universities	0.59
Churchs & Universities	0.57

The highest average test accuracy was 82%, which was obtained when the model was run on Universities and Stores. This was followed closely by Stores and Houses, which had an 81% average test accuracy. All four of the pairings with the highest average test accuracies contained Stores. Thus, it appears that the linear classifier model is having an easy time identifying stores.

When looking through the test images, I noticed that a lot of the images of stores included words on the building. I think this could be one feature that is helping the model. However, there are still images in this set that I would have a hard time identifying by hand. For example, the leftmost picture in Figure 5 was labeled by Chen et al. as a store and the rightmost was labeled as a house.

Figure 5. Example Images of Stores and Houses from my test sets

Stores being the easiest for the linear classifier to identify differs from Chen et al.'s finding that Apartments were the easiest to identify. There are several possible explanations for this deviation. The first is that Chen et al.'s results are potentially biased towards Apartments because their training and testing sets contain significantly more Apartments than other building types. If Chen et al.'s results were not skewed by the various class sizes, then these results suggest that linear classifier and CNN models might have significant differences in their strengths and weaknesses when it comes to building identification, as they appear to have different types of buildings that they are better at identifying. However, more testing would be necessary to make sure Chen et al.'s results were not skewed and that these differences between linear classifiers and CNNs persist.

6. Conclusions

In completing this project, I gained a better understanding of the challenges that are persistent in creating machine learning models that are able to identify different types of buildings. One of the takeaways from my research has been realizing the lack of datasets that exist that can help aid this research. While Chen et al.'s work attempted to provide a jumping-off point, it still has many flaws. First off, the disparity in the number of images for each class makes it hard to tell if the model is learning how to identify particular building types or learning to guess more often categories that are seen more frequently. Second off, lots of images are ambiguous even to the human eye. Thus, the differences in various building types are very subtle. It could be beneficial to create a dataset with images of these building types that are more distinct for beginning research. For example, researchers could work on developing models that can distinguish between suburban houses and highrises. Once these models are developed, then they could work on fine-tuning their models with more ambiguous images such as the ones provided in Chen et al.'s dataset. Lastly, more data is needed to properly train and test machine learning models than the amount present in Chen et al.'s dataset.

Another takeaway from this research is that when developing models to tackle these tasks, researchers should spend time learning about packages that will optimize their code as well as coding environments that will provide the computational power needed. Due to the large training sets and the number of parameters needed to complete this task, programs associated with it will be demanding of machine's resources which can lead to very long and unfeasible run times.

With all these complications, I was still able to observe that my linear classifier model had the easiest time identifying stores. One possible reason for this is that the store images

contained distinct architectural elements such as words on the exterior of the building. For future research, it could be useful to try to identify distinct features that normally show up in particular types of buildings and develop models that search for these particular features in order to make an identification.

7. Acknowledgements

I would like to thank Sierra Eckert and Professor Brian Kernighan for advising me through the entire process of doing this project. I would also like to thank my IW 01: Digital Humanities peers (Alice Lee, Archie McKenzie, Eva Vesely, Liam Esparraguera, Louis Larsen, Mary Tsahas, Naomi Farkas, and Rafael Collado) for their weekly feedback on my progress reports. Lastly, I would like to thank Professor Brian Kernighan, Abbitha Berry, and Morgan Michicich for reading drafts of my paper and providing feedback. All of these people were instrumental in helping me complete this project.

8. Honor Code

This paper represents my own work in accordance with University regulations. /s/ Sydney Berry

9. References

[1] Zavoleas, Yannis. “Afterword: In the Architect's Mind: Drawing (,) Architecture's Future” in Computational design : from promise to practice / Nicole Gardner, M. Hank Haeusler,Yannis Zavoleas, av edition, 2020. 158-173
[2] Chen, Jielin & Stouffs, Rudi & Biljecki, Filip. “Hierarchical (multi-label) architectural image recognition and classification.” 2021. 10.52842/conf.caadria.2021.1.161.
[3] Florian, Maria-Cristina. “Can Artificial Intelligence Systems like DALL-E or Midjourney Perform Creative Tasks?” ArchDaily, February 17, 2023. https://www.archdaily.com/987228/can-artificial-intelligence-systems-like-dall-e-or-midjourney-perform-creative-tasks.

[4] Russakovsky, Olga. “Assignment 2.” Class Assignment, COS 429: Computer Science, Princeton University, Princeton, NJ, Spring 2023.
[5] Russakovsky, Olga. “Assignment 3.” Class Assignment, COS 429: Computer Science, Princeton University, Princeton, NJ, Spring 2023.