

Design Mining for *Minecraft* Architecture

Euisun Yoon and Erik Andersen and Bharath Hariharan and Ross Knepper

Department of Computer Science, Cornell University
ey222@cornell.edu, eland@cs.cornell.edu, bharathh@cs.cornell.edu, rak@cs.cornell.edu

Abstract

3D construction sandbox games such as *Minecraft* have provided new opportunities for people to express their creativity. However, individual players have few tools to help them learn about architectural style or how to improve the structure they are building. Ideally, players could utilize tools that capitalize on the large numbers of 3D models built by others to offer guidance for their particular project. We trained a neural network to classify a large collection of *Minecraft* models from various websites in terms of style (Ancient, Asian, Medieval, or Modern). We present experimental results demonstrating that our model can classify the user-indicated style of a structure with 55% accuracy. We further demonstrate use of this model to highlight nearest neighbors to a specific query structure. We have integrated these tools into a *Minecraft* Mod that allows players to classify their structure's style and view nearest neighbors in real-time.

Introduction

Minecraft is the second most-selling game of all time (Peckham 2016). This game enables players to create 3D structures out of blocks, which they frequently upload to websites such as www.minecraft-schematics.com. *Minecraft* provides a substantial educational and creative opportunity to learn about architectural modeling. While players often create buildings that mimic real-world architectural styles, they lack resources for improving their building skills and learning about architecture. Players typically receive no concrete feedback until they show their model to other people.

Ideally, users would have tools that can leverage the myriad structures constructed by others and provide specific analysis and suggestions. Inspired by work in *design mining* that automatically analyzes large corpora of website designs (Kumar et al. 2013) to provide website-specific suggestions and recommendations, we collected a large collection of *Minecraft* models and used a deep learning neural network to analyze the style of a model and retrieve similar models created by other users as suggestions. We present experimental results demonstrating that our model can classify the user-indicated style of a structure with 55% accuracy. Additionally, our technique allows users to view other 3D

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

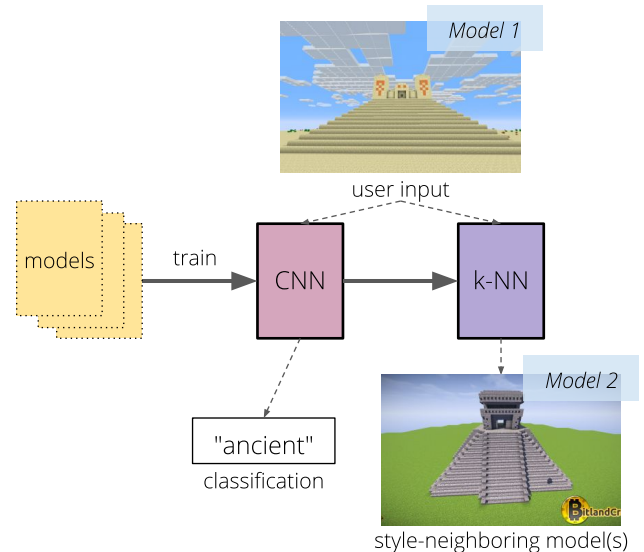


Figure 1: Overview of interface workflow. Scraped intermediates are used to train a convolutional neural network, where its intermediate output is used to train a k-nearest neighbor algorithm. The neural network classifies the style of the player's input model, while the k-NN algorithm provides style suggestions through outputting its stylistic nearest neighbors. See *Minecraft References* for image and model sources for each indicated *Model ID* in all figures.

structures produced by other players that are similar in style to their model. We integrated these tools into a *Minecraft* Mod that allows players to classify their structure's style and view nearest neighbors in real-time.

This paper makes the following contributions:

- A deep learning classifier of *Minecraft* models that can determine with 55% accuracy whether the structure conforms to one of a number of popular categories: Ancient, Asian, Medieval, or Modern.
- An interface that allows players to build structures within *Minecraft* and witness the classification.
- A visualizer module that examines a player's model and displays similar structures built by other players adjacent to the player's model for comparison.

Related Work

Design mining

Kumar et al. (2011) explored semantic properties of websites, enabling retargeting the content of one website into the style of another. Kumar et al. (2013) then introduced the idea of mining websites for design patterns. They used a web crawler to gather 100,000 websites and analyzed the Document Object Model structure of these websites. They also built on the work of Lim et al. (2012) by using machine learning to classify the websites in terms of various attributes. This enabled interesting applications such as querying the database of websites for nearest neighbors of a website. This work demonstrated that automatic techniques can learn design principles by gathering and analyzing many exemplars, and make this information useful to designers. However, this work is limited to the specific domain of website designs. In our work, we extend this idea to 3D voxel structures and a video game domain.

3D structure mining

In the past few years there has been significant progress in automatic understanding and classification of 3D shapes (see (Xu et al. 2017) for a review). Early approaches relied on defining *descriptors* of shapes (Knopp et al. 2010; Bronstein et al. 2011). Following dramatic progress in image classification (Krizhevsky, Sutskever, and Hinton 2012), Su et al. classified 3D shapes by rendering them in multiple views and using image classification models (Su et al. 2015). Instead, Wu et al. design machine learning models that work directly with voxel grids (Wu et al. 2015). While other representations of 3D shapes exist, such as point clouds (Qi et al. 2017), we use the voxel-based representation because it matches the representation used by *Minecraft* itself.

Design

We are interested in providing creative assistance to *Minecraft* players. Presumably, the player has some target in mind: a particular kind of structure that they wish to create. For example, they might be trying to create a Buddhist temple. We want to provide them with feedback that tells them if their goal is reached, and what they can change to do better. In other words, the system should tell a player if the structure being built is of a particular kind, and provide suggestions for improvement.

Unfortunately, the set of things that people might want to build is large and varied. Specifying by hand the key elements for every kind of structure that one might possibly build is tedious and requires a lot of expert knowledge. It is even harder to provide suggestions for improvement, since these suggestions depend a lot on the exact structure the player is building and have to be explicitly tailored to each specific instance.

However, there is now a growing community of *Minecraft* players, and there are online forums where people share their creations, along with tags indicating what they created. We can leverage this “wisdom of the crowd” in two ways. First, we can use these collections of player-created models to *automatically learn* what a particular style or class of models

means. Concretely, we can use machine learning techniques to build a mapping from *Minecraft* models to styles, so that we can tell a player if the model they created matches a particular style. This is a hard classification problem because of the large variation within certain styles, as well as the subtle differences between styles. Second, these collections provide a large space of *examples*, and one way to provide a player with suggestions or ideas is to surface similar structures that other players might have built. Doing so requires a meaningful notion of “similarity” that correlates with the players’ perceptions of similarity.

Below, we show how we solve these challenges. We first show how we can automatically organize online collections of *Minecraft* models and collect a curated dataset of different styles. We next use this curated dataset in conjunction with modern machine learning techniques to train a classifier for style. Finally, we show that in the process, the classifier learns a perceptually meaningful notion of similarity.

Design mining

We focused on architectural styles because buildings are the most common kind of structures people create. We mined a community-driven website that hosts a collection of *Minecraft* creations. Specifically, we focused on gathering data for schematics files, a file format created by the community for storing components of the 3D *Minecraft* world. This format allows flexibility in design and is ideal for capturing architectural structures as it allows the use of third-party programs which allow editing and/or simulation. While there exists various web forums that host a collection of schematics files, we use *Minecraft-Schematics* (<http://www.minecraft-schematics.com/>) as our source of training data as it provides a set of *themes* based on the schematic’s architectural style, where users label their creations with the fitting *theme*. In order to broadly classify architectural styles in a way that matched the source data, we chose the most popular user-selected themes (Ancient, Asian, Medieval, and Modern) and defined each theme as a separate class. We then scraped a total of 857 schematic files across these classes, utilizing *Scrapy*, a Python scraping and web crawling framework. It is worth noting that the categorization of architectural styles based on the user-selected themes is nebulous and potentially culturally insensitive. A curated set of classes would be ideal; our approach itself is orthogonal to this choice.

Pruning the dataset: Out of the creations that were labeled, many structures were found to not be a part of the labeled category and were thus eliminated from the dataset. For instance, architectural structures such as the Colosseum was eliminated from the Modern category. A constraint that was introduced by our hardware capabilities was the size of the schematic files - only schematic files of dimensions $66 \times 66 \times 66$ and smaller were used due to memory size constraints. In total, 391 schematic files were identified as well-classified and of the appropriate size.

Building a classifier for *Minecraft* models

Next, we turn to the problem of using this dataset of models to train a classifier for architectural styles. The goal here is

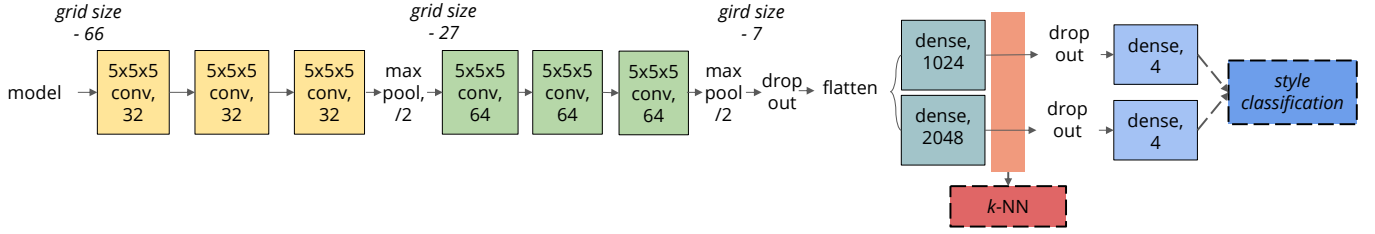


Figure 2: Architecture for convolutional neural network. Our network models differ in the output dimensions of its first set of dense layers: 1024 and 2048 respectively. The output of this set of layers is used for our k-nearest neighbors algorithm in exploring design neighborhoods, while the output of the last set of dense layers is used for style classification.

for this classifier to be used by *Minecraft* players to judge if their particular creation captures their intended style.

Note that this is a difficult classification problem. Different models might look very different but still be of the same style, while similar structures may appear across styles: for example, towers are ubiquitous in both ancient and medieval architectures. Furthermore, the differences between styles might be subtle and localized: for instance, the curvature of the roof might be defined by just a couple of blocks. The classifier therefore needs to automatically identify local structural motifs that are indicative of style and use them to make a decision. This leads to the final challenge: these structural motifs are domain-specific, and no generic, off-the-shelf shape representation captures them. Instead, the classifier must figure these out on its own.

Predicting style using convolutional networks: We leverage recent advances in computer vision and graphics, and use convolutional networks (LeCun et al. 1998; Wu et al. 2015) operating on top of *voxel grids*.

A voxel grid is a 3-dimensional grid, where each grid cell is called a voxel. Any *Minecraft* creation can be represented as a voxel grid, and indeed, building structures in *Minecraft* involves changing the material of a given voxel. Thus, in *Minecraft* structures, each voxel is associated with a particular material. We start by first representing this material in binary (traditional one-hot representations were too memory-inefficient). That is, we use an M -dimensional vector to represent 2^M materials. $M = 8$ in our experiments. Given a $d \times d \times d$ voxel grid, this representation of materials allows us to represent any creation as a $d \times d \times d \times M$ tensor.

Our task is to build a classification model that takes this tensor representation as input and outputs the architectural styles. Convolutional networks are especially suited for this problem because they are built out of convolutions: an operation that searches for local patterns in a grid. These models have also led to astounding progress in image recognition (Krizhevsky, Sutskever, and Hinton 2012). We briefly explain the structure of these networks below.

Convolutional networks can be thought of as a series of learnable functions composed with each other. In networks operating on voxel grids, the input and output of each function or *layer* is a 4-dimensional tensor representing the voxel grid, often called a feature map. Intuitively, each layer looks at local neighborhoods in its input voxel grid and tries to identify patterns by *convolving* its input with a filter, whose

weights are learnable parameters. Mathematically, for input tensor \mathbf{x} and filter \mathbf{w} , the output tensor \mathbf{y} is given by:

$$y_{i,j,k} = \sum_{l=-K}^K \sum_{m=-K}^K \sum_{n=-K}^K \sum_d x_{i-l,j-m,k-n,d} w_{l,m,n,d} \quad (1)$$

Here K is the kernel size and determines the size of the local neighborhoods. A convolutional layer has multiple filters, and all the outputs are stacked together into another 4-dimensional tensor.

Convolutional networks also frequently subsample their inputs as they go deeper and deeper into the network. This has the effect of increasing the *size* of the local neighborhood that is considered in subsequent layers, while also increasing their *invariance* to fine-grained changes. Thus, the earlier layers might identify fairly local patterns, such as the sizes of windows, while later layers might be looking at global properties such as the presence of towers. This gradation from object parts to objects to scenes as one goes deeper into the convolutional network is central to their representative power, especially in our problem domain: subtle local differences in style can be detected early on, and later layers can keep track of the presence of these differences while being invariant to their precise location.

Our convolutional network architecture is shown in Figure 2. The architecture consists of seven convolutional layers with max pooling after the fourth and seventh layers, followed by a fully connected layer and a softmax classifier. The input is a $66 \times 66 \times 66 \times 8$ tensor as described above. Note that our focus here is on providing a proof-of-concept design aid, and not on creating the best style classifier. As such, we did not experiment with other architectures; there might be better choices.

The learnable parameters of convolutional networks are the weights of the filters in each convolutional layer. These weights are set to minimize a loss function on the training set by using variants of (stochastic) gradient descent: one starts with a random initialization of the parameters and then iteratively takes steps along the negative gradient of the loss with respect to the parameters. Since our problem is one of classification, we use the cross-entropy loss. Given predicted class probabilities $p_i, i = 1, \dots, C$ for a given example x where C is the number of classes, and the true label y for this example, the cross entropy loss is simply the negative log probability of the true class: $L(\mathbf{p}, y) = -\log p_y$. Our




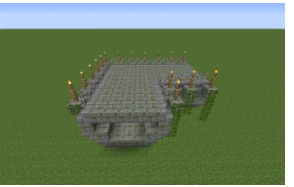




	Seed model	Nearest neighbors			
Model ID	3	4	5	6	
					
Tag	Ancient	Ancient	Ancient	Ancient	
Prediction	Ancient	Ancient	Ancient	Ancient	
Distance	-	0.019	0.019	0.027	
Model ID	7	8	9	10	
					
Tag	Asian	Asian	Asian	Asian	
Prediction	Asian	Asian	Asian	Asian	
Distance	-	0.058	0.077	0.08	

Figure 3: Good examples of classifications and nearest neighbor matches. The left column shows seed models, which could be user-provided but here are taken from a held-out test set drawn from the same *Minecraft* model repository as the training set. Each example is annotated with the user-selected style tag, the classifier’s style prediction, and the cosine distance between the seed model and its nearest neighbors. In these examples, the model appears to recognize architectural structures. The closest two neighbors of Seed Model 3 are ancient wells, and the three neighbors of Model 7 are Asian pagodas. See *Minecraft References* for image and model sources for each *Model ID*.









	Seed model	Nearest neighbors			
Model ID	11	12	13	14	
					
Tag	Medieval	Medieval	Medieval	Medieval	
Prediction	Medieval	Medieval	Medieval	Medieval	
Distance	-	0.051	0.053	0.055	
Model ID	15	16	17	18	
					
Tag	Modern	Modern	Modern	Modern	
Prediction	Modern	Modern	Modern	Modern	
Distance	-	0.028	0.086	0.115	

Figure 4: A set of classifications and nearest neighbor matches that show different types of structures that have a similar style. Row 1 contains figures which all demonstrate a cottage-like medieval house, while row 2 figures illustrate international style modern architecture. Note that along with examples in Figure 3, most well-behaving examples have cosine distances of magnitude less than 0.1 from the seed model. See *Minecraft References* for image and model sources for each *Model ID*.

model follows fairly standard protocol and uses off-the-shelf libraries (Abadi et al. 2016), making it easily extensible.

Exploring design neighborhoods

As discussed above, we want the system to not only tell the player if their structure is of the target style, but also pro-

vide suggestions for improvement. For example, they may want ideas about additional structural motifs to incorporate in their creation, or they may want to change the style of their creation, or incorporate features from other styles. We want to provide such suggestions by leveraging our mined dataset to surface *similar* models that other players have built. For this, we need a notion of similarity that coincides with our intuitions.

Our convolutional-network-based style classifier provides a way of computing this similarity. As discussed above, convolutional networks can be thought of as a series of learnable functions composed with each other. The very last layer of our convolutional network is a linear layer that produces 4 scores, one for each style, with the predicted style being the one with the largest score. The score for each style is thus a *linear* function of the inputs to this layer (or the outputs of the penultimate layer). Thus the output of the penultimate layer can be considered as an intermediate *style space*, in which different styles such as the “Asian-ness” of a design correspond to simple linear functions. This is in contrast to the space defined by other intermediate layers, in which style is a more complex, non-linear function. After the convolutional-network-based classifier has been trained, we can chop off the last layer, and use the rest of the network as a mapping from *Minecraft* voxel grids to this meaningful style space. We hope that, as in other domains such as visual recognition (Donahue et al. 2014), this intermediate space will contain rich semantic information beyond the original task the convolutional network was trained for.

This style space allows us to explore the neighborhood of any given *Minecraft* model. Concretely, we embed the user model into this style space and retrieve the nearest neighbors (using cosine distance: $d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$) in this space. Our experiments below show that these nearest neighbors indeed surface style elements that the convolutional network was not explicitly trained for.

Evaluation

Evaluating classification accuracy We trained two convolutional networks with different architectures on the task of classifying *Minecraft* structures as one of Ancient, Asian, Medieval, or Modern. The two architectures are both equally deep, but vary in the number of channels in the last few fully-connected (or dense) layers and therefore in the number of learned parameters (a dense layer with c_{in} input channels and c_{out} output channels has $c_{in}c_{out}$ parameters). Each network was trained with Adagrad for 200 iterations with an initial learning rate of 1.

We achieve the following accuracy for model types:

Model	Test Accuracy (%)
Material-count baseline	45
CNN-1024	50
CNN-2048	55

The best accuracy we achieve is 55% on the CNN 2048 model, which is significantly better than the chance performance of 25%, and also much better than a baseline logistic regression model using a histogram of material counts as features, which has an accuracy of 45%.



Figure 5: In-game screenshot of real-time *Minecraft* interface. The interface generates a glass cage in which the user can build their structure in, and its nearest neighbors are generated around the cage as seen on the structure on the left. The generated neighbor in this image is Model 31, as cited in *Minecraft References*.

Nevertheless, the fact that the test accuracies are not very high can be attributed to two reasons. First, the task itself is challenging: style judgements are hard to make, and sometimes the user annotations that we are relying on can be noisy. Second, we find that the training accuracies ($\sim 99.5\%$) are almost twice the test accuracies for both models, indicating overfitting. This problem is likely the result of our small training set, and will be mitigated as online communities of players grow.

Exploring the nearest neighbors Next, we retrieve the nearest neighbors in style space as discussed in the previous section. Figures 3 through 6 show examples of retrieved nearest neighbors, with the first column being the “seed” or the “query”. A comprehensive set of nearest neighbor models can be found on <https://minecraft-design.github.io/>. Below we discuss noteworthy observations which may assist users in creating stylized models.

Figures 3 and 4 show some examples of meaningful nearest neighbors. Particularly, Figure 3 shows cases where the retrieved neighbors capture the same architectural structure category (“well” and “pagoda” respectively), even though the convolutional network was not explicitly trained to recognize these structures. Models in Figure 4 ostensibly correspond to different kinds of structure, but the similar visual appearance indicates that the network is in fact latching on to some implicit style. In all these cases, looking at these nearest neighbors can provide a user for additional ideas on how to improve their creation.

Figure 6 shows examples where our model performed less well. In the top row, the cosine distance between the seed and the neighbors is *much larger* than the average distance between seed and neighbor models, which is 0.073. This suggests that the seed model is in a sparsely-populated part of the style space and there are not many similar models in the dataset. As such, the “nearest neighbors” are actually fairly far from the seed model. The middle row shows an example of a misclassified seed model. Notably, most misclassified seed models tend to have nearest neighbors from the category they are misclassified into, as the network appears to falsely extrapolate information that they are of a similar


	Seed model	Nearest neighbors		
Model ID	19	20	21	22
				
Tag	Ancient	Modern	Ancient	Ancient
Prediction	Modern	Modern	Ancient	Ancient
Distance	-	0.333	0.34	0.341
Model ID	23	24	25	26
				
Tag	Asian	Medieval	Medieval	Medieval
Prediction	Medieval	Medieval	Medieval	Medieval
Distance	-	0.04	0.051	0.057
Model ID	27	28	29	30
				
Tag	Medieval	Medieval	Medieval	Medieval
Prediction	Medieval	Medieval	Medieval	Medieval
Distance	-	0.112	0.372	0.115

Figure 6: **Top Row:** Poor quality nearest neighbors. Note the large cosine distances, suggesting that the seed model is in a sparsely-populated part of the style space. The first nearest neighbor shares the seed model’s incorrectly predicted style, *Modern*, and the remaining neighbors are of the correct style, *Ancient*, but none of them are similar. **Middle Row:** Misclassified seed model with close neighbors from the incorrectly predicted class. **Bottom Row:** Similar models that were not classified as neighbors: the cosine distances (distances from Model 27 are shown) are large in spite of the perceptual similarity. See *Minecraft References* for image and model sources for each *Model ID*.

architectural style. The bottom row shows 3D models that subjectively appear to have similar features yet their *semantic* similarity is not correctly identified because our neural network classified them into different styles.

Real-time *Minecraft* Interface

We modified *Glowstone*, an open source *Minecraft* server, to create an experimental interactive in-game interface that allows people to classify buildings they created. In our interface, users can build structures within a $66 \times 66 \times 66$ cage where we use our neural network model to predict the architectural style the user has built, as shown on Figure 5. In addition, users can explore the nearest neighbors generated by the interface after they have completed their model.

Conclusions

In this paper, we demonstrate the ability to “design mine” *Minecraft* models of buildings to help a *Minecraft* player.

We trained a neural network that links features of a model’s design (materials and structure) to the user-specified style. We also use the neural network to explore similarity within the learned space of designs. The user can query the neural network with an arbitrary user-generated model and obtain the nearest neighbor models provided by the crowd. This allows a user to identify and modify design cues to cause their model to be classified as a certain style. All code, data and models will be available at <https://minecraft-design.github.io/>.

In the future, we hope to identify specific model features (windows, material etc.) that contribute most strongly to its classification. Our style classification in conjunction with identification of these features is a way of mathematically capturing style and is a first step towards the eventual goal of procedurally generating cities of certain styles. We believe that tools such as those presented in this paper can help people to unleash their creativity in pursuits like *Minecraft* by presenting design choices more explicitly to users.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1526035. We are grateful for this support.

Minecraft References

The following is a list of *Minecraft* model creations from *Minecraft Schematics* (www.minecraft-schematics.com/) used in this paper, corresponding to the *Model ID* listed in each figure. Each model is cited with its model name, creator account name, and link to its model creation, where each link needs to be appended to the following prefix url: www.minecraft-schematics.com/schematic. For example, Model 1 listed below can be accessed at www.minecraft-schematics.com/schematic/2538/.

1. Default Desert Temple Supreme, KarmicForager, /2538/
2. Maya temple, Bitconion, /9319/
3. Desert Village Well, lunchboxxx19, /9057/
4. Village Well 3x3, lordolaf, /8175/
5. Village Well 3x3 Sandstone, lordolaf, /9764/
6. Ancient Skyway Modified with sewer(Corner), mrnada, /3132/
7. Japanese Pagoda, firemonkey, /1220/
8. large pagoda (just house), CoonClaws, /1722/
9. – The Bagel Pagoda –, w4rl0, /125/
10. Pagoda, Bender, /2812/
11. Medieval Storage, CeepreCaleb, /9257/
12. Rohan stable (it's a harda world schematica project: Rohan), spokiechris, /8175/
13. first farm, runaddict2, /9764/
14. Old Church, DadCANN, /8792/
15. Garage with office, pratenik1, /5893/
16. Modern House, PrincesseRena, /11704/
17. Modern House 2, PrincesseRena, /11676/
18. Modern 2 story house above small pond, splinteredvoyagers, /197/
19. House One, Elarson, /8924/
20. Modern House, SuperNovaXII, /11180/
21. Mountain House, mikerspiker, /1402/
22. Standard Temple of quartz, ZAZA, /975/
23. Pagoda of LIGHT, beefmouth, /1411/
24. Rustic House(Unfurnished), ChocoKyle, /9658/
25. Town Wall, BASsFI3ND, /5112/
26. MC tower, RicksPlumbus, /10629/
27. Bonetown Windmill, EdCr0w, /8922/
28. SmaKHouSE, Oleg_BoG, /9566/
29. Papps Mill, sammf, /9948/
30. Sandstone Winmill (Medieval), DadCANN, /8808/
31. haunted mansion by zerte, zerte, /2922/

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, 265–283.

Bronstein, A. M.; Bronstein, M. M.; Guibas, L. J.; and Ovsjanikov, M. 2011. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics (TOG)* 30(1):1.

Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; and Darrell, T. 2014. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML*.

Knopp, J.; Prasad, M.; Willems, G.; Timofte, R.; and Van Gool, L. 2010. Hough transform and 3d surf for robust three dimensional classification. In *European Conference on Computer Vision*, 589–602. Springer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Kumar, R.; Talton, J. O.; Ahmad, S.; and Klemmer, S. R. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2197–2206. ACM.

Kumar, R.; Satyanarayan, A.; Torres, C.; Lim, M.; Ahmad, S.; Klemmer, S. R.; and Talton, J. O. 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3083–3092. ACM.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE*.

Lim, M.; Kumar, R.; Satyanarayan, A.; Torres, C.; Talton, J.; and Klemmer, S. 2012. Learning structural semantics for the web. Technical report, Tech. rep. CSTR 2012-03. Stanford University.

Peckham, M. 2016. ‘minecraft’ is now the second best-selling game of all time. <http://time.com/4354135/minecraft-best-selling/>. Accessed: 2018-06-03.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1(2):4.

Su, H.; Maji, S.; Kalogerakis, E.; and Learned-Miller, E. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, 945–953.

Wu, Z.; Song, S.; Khosla, A.; Yu, F.; Zhang, L.; Tang, X.; and Xiao, J. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1912–1920.

Xu, K.; Kim, V. G.; Huang, Q.; and Kalogerakis, E. 2017. Data-driven shape analysis and processing. In *Computer Graphics Forum*, volume 36, 101–132. Wiley Online Library.