Scene Recognition with the MiniPlaces Dataset
6.869 / 6.819: Advances in Computer Science

Zijin Shi
Massachusetts Institute of Technology
Cambridge, MA
zijinshi@mit.edu, 6.819

Gerardo Bledt
Massachusetts Institute of Technology
Cambridge, MA
gbledt.mit.edu, 6.869

Abstract

This project dealt with attempting to achieve good scene classification accuracy for a given set of images as part of the MiniPlaces Challenge. As a rather open-ended challenge, we researched many methods commonly used to recognize scenes as well as explored possible modifications to currently existing techniques. Mainly we will be describing our research findings, the approaches taken and attempted, our attempts at improving classification, and our results. Unfortunately, many of our more novel planned approaches weren't able to be fully realized due to implementation difficulties with the basic setup which held us up for a rather long time because of inexperience. However, these novel ideas are outlined and could be worth exploring.

1. Introduction

The MiniPlaces Challenge is designed to be an image classification benchmark dataset. There are many common datasets used as benchmarks for classification in Computer Vision and MiniPlaces is a subset of the larger Places2 dataset as described in [11]. Generally, researchers wishing to show improvements in classification over previous methods will run them on one of these standardized sets and report accuracy comparisons. Two common comparisons are if the image was correctly labeled as with the most probable result (top-1 accuracy) or if it was correctly labeled in the most probable 5 results (top-5 accuracy). In this project we are aiming to have a high top-5 accuracy, while competing with classmates for the best classification.

Convolutional Neural Networks (CNN) have been shown to be very successful in classifying images and objects. Some approaches are now combining the benefits of feature extraction from CNNs with image clustering techniques to recognize scenes in unsupervised learning [2]. In one paper, the authors take a look specifically at CNNs used for both the ImageNet and Places datasets and make note that the image scales are different for each and therefore parameters should be scaled accordingly to get better results [4]. A paper by the creator of the MiniPlaces dataset talked about the performance of using CNNs on the full Places dataset and found that they were able to get good performance in part due to the CNN used and in part due to the richness of the actual dataset [12]. An example of a CNN that is known to perform well in scene recognition tasks is ResNet, which won the ImageNet challenge in several categories [3].

In all of these papers, it seems that the common theme is to use the CNN along with some other method that the authors are researching themselves. This shows the power, versatility, and ease of use that makes CNNs an attractive option for image recognition tasks. However, a lot of the success depends on the structure of the network, as well as the chosen parameters. Through this project, we intend to explore some of the methods that lead to improvements of scene recognition with this data through the use of CNNs. After some research into previously proven methods as well as common techniques, we attempted to implement some of the more promising methods.

The main CNN backbone we chose to use was the AlexNet with some modified parameters as well as some adaptive parameters during the training based on results. We also found that many times, results would have top-5 groups with only 3 or 4 similar scene categories, but another related scene category would be close in the top-10 and may sometimes be the correct answer. As such, we decided to calculate the semantic difference between all of the scenes and store them in a matrix that we could use to adjust the resultant scores to drop out some outliers that happened to score well and maybe boost some of the scenes that were related to other high scoring scenes.

2. MiniPlaces: General Setup

The first task before even attempting to train a model to recognize images is to understand and manage the given data and create a workflow for dealing with the code. It

turned out to be nontrivial as there was a lot of setup that was required to be able to effectively run the model.

2.1. Image Dataset

Through the given dataset, we were provided with a subset of the larger Places2 dataset. The full dataset contains over 10 million images in more than 400 unique scene categories. However, the MiniPlaces data will be a more manageable size for the scope of the project with 100 scene categories, 100,000 training images, 10,000 validation images, and 10,000 testing images. In the training data, there are 1000 images in each scene category, and 100 in the validation and test datasets. The training data is labeled with its corresponding scene and may also contain some object annotations that mark objects commonly found in the scene.

2.2. AWS Instance Setup

For our implementations we used an EC2 instance on Amazon Web Services (AWS) with AWS Educate credit to run the programs. This allowed us access to an NVIDIA GK104GL GRID K520 GPU for significantly faster computation time as compared to our laptop CPUs. The basic setup with AWS took a rather long time to implement however as neither of us had any experience using AWS or any server for remote GPU calculations. After about a week of waiting for the AWS account approval, AWS Educate approval, transfer and linking of Educate credits with AWS account support ticket response, and finally the increase in instance limit to above 0, we were able to create an instance of a g2.2xlarge as recommended for the problem.

3. Approaches

Our goal was to use the provided training data in order to get good results on the test dataset. Several approaches were implemented in order to improve the baseline performance and classify the new scene images. This included parameter tuning on the baseline network, adding and removing layers, as well as trying different CNN structures.

3.1. AlexNet: Tensorflow

As part of the problem data, we were provided with an implementation of AlexNet as a baseline implementation of a Deep Learning network for classifying images. AlexNet is a CNN that was developed by Alex Krizhevsky at the University of Toronto and which is described in [5], where a top-1 and top-5 error rate of 37.5% and 17.0% were achieved on the ImageNet ILSVRC-2010 dataset including over 1.2 million training images, 50,000 valication images, and 150,000 testing images. They also compared their CNN with other state-of-the-art techniques used for image classification such as Sparse coding and SIFT with Fisher Vectors [8] which had a reported top-1 and top-5 accuracy almost 10% lower than AlexNet.

Table 1. AlexNet Layer Structure

Layer Group	Individual Layers		
1	Conv $224 \rightarrow 55$	ReLU 55	Pool $55 \rightarrow 27$
2	Conv 27	ReLU 27	Pool $27 \rightarrow 13$
3	Conv 13	ReLU 13	-
4	Conv 13	ReLU 13	-
5	Conv 13	ReLU 13	Pool $13 \rightarrow 6$
6	FC 6	ReLU 6	Dropout 6
7	FC 6	ReLU 6	Dropout 6

This CNN has 7 distinct layer groups, each of which may have some combination of a convolution, batch normalization, Rectified Linear Unit (ReLU) activation, pooling layer, dropout, and fully connected layers. The exact structure of the AlexNet, along with the size changes as a result of each layer can be seen in Table 1. The arrow signifies that a size change has occurred as a result of passing through the layer. Each layer has an associated weighting that acts on the data as well. Figure 1 shows the graphical representation of the CNN. The image was taken from [5], which is the paper that describes the CNN in detail.

We tried some parameter tuning, but generally did not see any heavy improvement. As it took too long to see any meaningful training data results, we decided small parameter tuning was not the best way to improve our classification accuracy. We were able to see some big performance deterioration rapidly though for large changes in some parameters.

Therefore, taking some of the advice from [5], we implemented and modified a couple of their tricks to reduce overfitting and improve classification accuracy. The first modification was to check for stagnant top-5 validation accuracy and reduce the learning rate by some factor that we changed around if the average accuracy had not improved over a given iteration window. We saw that at the beginning, the learning rate was fluctuating wildly and therefore the learning rate was reduced before a true plateau in accuracy was found. Therefore, we added a threshold to begin checking for plateau only after a certain about of iterations, where we generally saw the accuracy stagnate in the original case.

Eventually, we also added an extra layer to normalize the results. We achieved this by adding a SoftMax layer at the output that made sure our results were a number between 0 and 1 rather than whatever the scale resulting from the

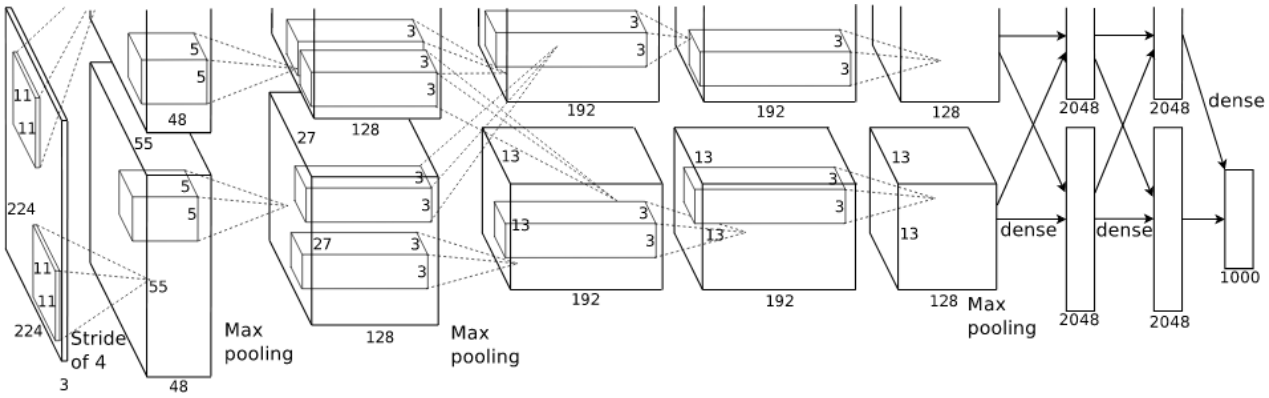

Figure 1. **AlexNet CNN.** The structure of the AlexNet CNN used for baseline testing on the MiniPlaces2 dataset.

training data is currently. It was easier to think of the results as a pseudo-certainty likelihood metric between 0 and 1. This also helped condition the numbers to work better with our other approach ideas.

3.2. ResNet: PyTorch

As we found that ResNet was generally preferable to AlexNet for image recognition and had outperformed may other CNNs during the challenges, we decided to implement it as well once performance on AlexNet stagnated. Figure 2 shows the basic building block for the residual learning network. The full network will have many layers of these chained together and using different parameters. Some variations of this building block are also described in the paper from Microsoft Research [3]. PyTorch lent itself

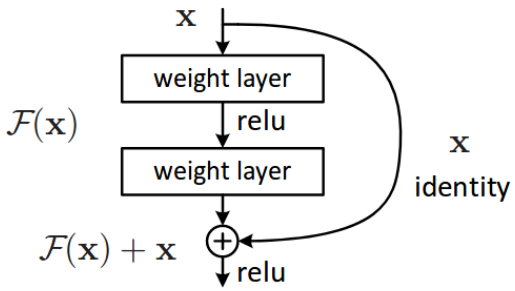

Figure 2. **ResNet Block.** A simple building block for the residual learning process.

very naturally to implement ResNet and some starter code was modified to work with our dataset. We ran it with the same AWS setup on the GPU as the AlexNet. The base code was setup with the provided parameters.

3.3. Related Scene Semantic Correlation

While looking at the results for our validation set we noticed that many times, scenes that were correctly classified would often have one or two classifications in the top-5 that made no sense with the scene or others in the top-5. Therefore, we looked at trying to implement some semantic similarity metric that could drop out outliers and pull in a scene that was maybe top-10, but is closely related to 3 or 4 of the top-5. We attempted to use a Semantic Similarity checker between two words from UMBC that took in the two words and scored them from 0 to 1 which meant from no relation to exactly the same [10]. For example, ”tree” and ”bush” had a relation score of 0.8405, whereas ”tree” and ”mountain” had a score of ”0.1911” and ”tree” and ”car” had a score of 0.0 as they are not similar.

We used the 100 scene category list and compared it with all other words on the list by calling the website with the two words. Figure 3 shows the resulting matrix of relation values between the various scene categories. If we analyze it

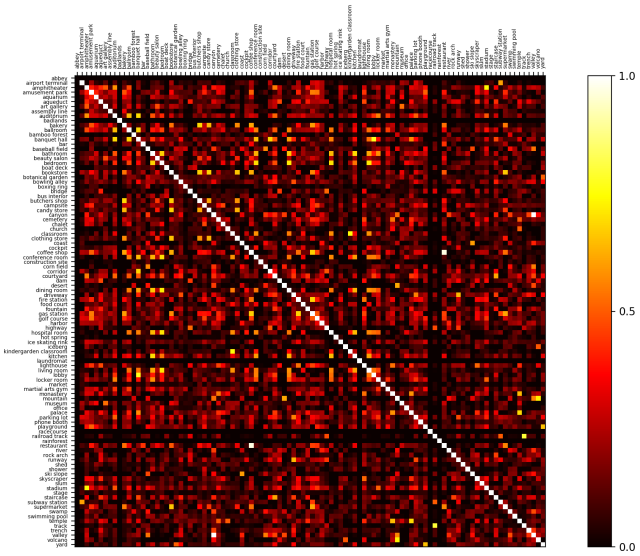

Figure 3. **Semantic Relation Matrix.** A matrix visualization of the semantic relation between the various scene categories. The 100 categories make up the two axes and their semantic relationship to the other scenes is shown according to the scale denoted by the color bar.

we can see that certain scenes, such as ”canyon” and ”val-

ley" have a high relation score in the matrix. We can use this matrix of scores between scenes to rescore the scenes and perhaps find a grouping of more closely related categories as our top-5. This would increase the probability that we find a correct match in top-5 by reducing false-positive outliers. To prepare the matrix into a meaningful form that fairly adds and subtracts from the scene probability we performed the following operations

$$\mathbf{M}_r = \mathbf{M}_0 - \mathbf{I}_N \tag{1}$$

where $\mathbf{M}_0$ is our original matrix, $\mathbf{M}_r$ is a reduced matrix that removes the self relations that are always 1. Next we make sure that the rows are zero mean as

$$\mathbf{M}_{r,\bar{i}=0} = \mathbf{M}_r - \mathbf{M}_{r,\bar{i}} \tag{2}$$

where $\mathbf{M}_{r,\bar{i}}$ is the row-mean matrix of $\mathbf{M}_r$ and $\mathbf{M}_{r,\bar{i}=0}$ is the zero row-mean matrix of $\mathbf{M}_r$. Finally we add back the self relation entries as

$$\mathbf{M} = \mathbf{M}_{r,\bar{i}=0} + \mathbf{I}_N \tag{3}$$

The relating classification scores as a result of our model are given as a vector

$$\mathbf{\Phi}(S) = \begin{Bmatrix} \Phi(S_0) \\ \vdots \\ \Phi(S_{N-1}) \end{Bmatrix} \tag{4}$$

where S_i is a scene category, such as forest and $\Phi(S_i)$ is the classification output form our model. Now we can use the two to create a modified score

$$\hat{\mathbf{\Phi}}(S) = \mathbf{M}\mathbf{\Phi}(S) \tag{5}$$

with $\hat{\mathbf{\Phi}}(S)$ being the modified semantic relation adjusted classification score.

4. Results

A training was run with the original AlexNet setup and achieved a top-1 accuracy of 35% and a top-5 accuracy of 64.7%. To improve results, we had access to various parameters experiments were run with large changes to these to observe any effects on performance. In general, a large learning rate tended to see big fluctuations in accuracy, while a small learning rate did not improve in accuracy even throughout many iterations. Increasing the batch size seemed to make training faster and better, but would get high top-5 accuracy of about 90% on the training data and lower of about 70% on the validation set suggesting that we might be suffering from overfitting.

We went through many iterations of parameter tuning, and saw mostly worse results until we tried some of the overfitting reduction methods discussed in section 3.1. We decided to reduce the learning rate by a factor of 10 if the average of the 500 latest top-5 accuracies was less than the average of the previous set of 500 top-5 accuracies. This got us over the next milestone to a top-1 accuracy of 36.97% and a top-5 accuracy of 67.29%. This was a solid increase, but ultimately not enough to claim it was not due to variability in how the data was randomized.

The next set of experiments dealt with trying modifications to the CNN structure. The first was the addition of the normalization SoftMax layer. This again yielded top-5 accuracies of about 65-70%. However, we then added a duplicate of layer group 5 right before the original layer group 5 and removed the pooling layer. This improved results by several percent and got us a top-5 accuracy of XXXXXXXXXXX% and a top-1 accuracy of XXXXXXXXXXXXXXXXX%. Removing the pooling layer meant that we were able to integrate it with the current network without the need to worry about sizing issues.

The Semantic Relation Matrix (SRM) for semantic adjusted scores as described in section 3.3 was then implemented. We first ran a model that got us validation accuracies of 38.1% and 67.0% for top-1 and top-5. We took that same data and applied the SRM to see if result improved. Unfortunately, it turns out that this method does not improve accuracy as we got 38.0% top-1 and 61% top-5 accuracies. There were a few instances where the correct answer was in the top-10 and would be pulled up to the top-5 with the method. However, e realized that semantic similarity doesn't necessarily correlate with image similarity. For instance, "train station" and "racetrack" are similar semantically, but look quite different. We tried several different normalization schemes, but none seemed to improve accuracy. We do not believe it is a completely lost venture, but rather one that needs more research.

Finally, the ResNet implementation was run on the training data as well. Without any modifications, it seemed to learn at a slower rate than the AlexNet around the same iterations while the training was running. Most of our efforts were spent attempting to modify and improve AlexNet until it became clear that only heavy modification would allow any more improvement. Therefore, we did not get as much of a chance to explore the ResNet structure, but did get an instance of it running the training on the data. This resulted in a top-5 error of XXXXXXXXX% and a top-1 error of XXXXXX%.

5. Ideas for Scene Classification Accuracy Improvements

While our recognition accuracy may not have been as high as we wanted it to be, there are several ideas that we had for improving the accuracy. Unfortunately, we had a lot of trouble getting the simple basic training working and

we were not able to implement these for this mini-project. However, with the experience gained through this, it would be possible to implement them in a future attempt to use CNNs for training now that we are familiar with the whole workflow and code.

5.1. Data Augmentation

For the mini-project, one of the requirements was to use only the provided training data images. The means that we are limited to train on the current orientation and size of each image. This does not mean we cannot manipulate the current data to extract a more rich dataset. For example, If you only train with images of trucks facing to the left and are then given an image of a truck facing to the right, your model might completely not classify it as a truck depending on the robustness. However, if you take the original images and flip them over the central vertical axis, you've now trained your model to recognize images facing both left and right without the need to retake a picture. This is appropriate, because most naturally found scenes are not direction dependent.

Similar to this, most pictures taken are subject to orientation and scale variations. Therefore, it may be a good idea to take the data and rotate it by some arbitrary angle since people will likely not take pictures with the camera's x axis exactly parallel to the ground. The angle choice may depend on the scene in question, but it is not likely that a large portion of the images will be upside-down or heavily rotated, so perhaps $20 - 30^\circ$ may be a good amount. Then we can take subsamples of the rotated image to train on. Overall, the point of this data augmentation is to take the current available data and increase the variance of the training set for a more robust model.

5.2. Object Recognition

As part of the dataset, we were provided with a set of files that contained object annotations for certain scenes. There are many ways we could have made use of this data. For one, we could have ran an object detection algorithm such as the one described in [6] which uses examples of objects and SVMs to recognize similar objects in new pictures. This would have allowed us to detect specific features in the scene that would raise the probability of it belonging to a category. However, this would have required us to implement some sort of object recognition. It sounds good in theory and this was one of our early ideas, but implementing an effective object recognition algorithm and integrating it with the training would have been a project in itself.

5.3. Probabilistic Kalman Filter for Scene Certainty Correction

Kalman Filters are generally used in dynamic systems to optimally fuse measurements and models together in order to improve accuracy in the states. It is commonly used in IMU's to fuse imperfect measurements to give a very good estimate for the IMU's orientation in 3D space. In this case, we our states are the discrete classification of a scene, but if we define a probability that the image is a specific scene as the state, we can use our training data as a set of predictions and measurements for the Kalman Filter as

$$\hat{\mathbf{x}} = \begin{Bmatrix} P(S_0) \\ \vdots \\ P(S_{N-1}) \end{Bmatrix} \tag{6}$$

where S_i is a scene category as it was before. In a paper submitted to the Robotics Conference this year by one of the team members on this project, it is shown how you can use various probabilities as states rather than an actual physically measured quantity and achieve much higher certainty from the KF [1]. We believe the same theory could be applied to classification probabilities to increase recognition certainty.

At every step, we would use the result of one run of the CNN to create a vector of probabilistic state prediction variables

$$\hat{\mathbf{u}} = \begin{Bmatrix} P(S_0|I) \\ \vdots \\ P(S_{N-1}|I) \end{Bmatrix} \tag{7}$$

where I is the training data from the image. The resultant validation set classification accuracy as our covariance matrix denoting the certainty we assign to how much we trust that our prediction is correct. Now we can use the semantic scene relation as well as the object recognition from sections 3.3 and 5.2 as two separate measurements.

$$\tilde{\mathbf{z}}_1 = \begin{Bmatrix} P(S_0|O) \\ \vdots \\ P(S_{N-1}|O) \end{Bmatrix} \qquad \tilde{\mathbf{z}}_2 = \begin{Bmatrix} P(S_0|R) \\ \vdots \\ P(S_{N-1}|R) \end{Bmatrix} \tag{8}$$

where O is the list of detected objects and R is the weighted semantic relation score given the resultant rankings. The KF provides an "optimal" way to fuse various pieces of uncertain, imperfect data together resulting in a much more certain set of top-5 classifications that should improve the overall accuracy.

6. Conclusion: Reflections and Lessons Learned

We were both going into MiniPlaces with no experience in any of CNNs, AWS, PyTorch, or Tensorflow, as well as limited familiarity of Python which meant a lot of small tasks for an experienced user required us to research how to effectively (and sometimes not even effectively) get them done. However, experience with these common tools is precisely what we hoped to gain from taking the class. As a

result of the project, we are much more familiar with the actual implementations and setup of these tools rather than simply understanding how they work in theory. For us, this was the biggest challenge, even with a good understanding of how to apply CNNs and their role in image recognition. It was a bit frustrating as our lack of experience and challenges with the learning curve held us up so long in the early stages and after long training runs only to find a bug. We felt we had some possibly good innovations as outlined in 3.3 and 5.3, but didn't have a good enough grasp on the project early enough to implement them effectively.

After finally getting some decent results, we were interested in getting an extremely crude sense for how this compares to humans learning to recognize scenes and objects. According to a paper in the journal of Attention, Perception, & Psychophysics [7], the human brain can process an image in about 13ms, which is about 75 Hz. According to research published in the journal Child Development babies are able to recognize images about 9 months after birth [9]. Roughly then, a child processes about 1.2 billion images before they start being able to recognize some simple objects. We also note that these images are not discrete single images of an object but go through all of the variations and more that were discussed in section 5.1 as it is a continuous image stream. If we also assume that the brain is some sort of complicated Neural Net, which would be a gross over simplification of the complexity of the human brain, then it is likely safe to also assume that the NN involved in training object recognition is huge and adaptable, certainly much larger than our 7 layer group AlexNet with hardcoded parameters.

With this in mind, our result of 32.71% top-5 error with only a small discrete image dataset and approximately 15000 iterations with the AlexNet and handtuned parameters is actually rather encouraging. However, while humans are notoriously some of the slowest animals developmentally, they end up being the most intelligent and are capable of recognizing images that may be far from anything they've ever been "trained" with previously. This robustness is something that computers are yet to come close to achieving and is truly amazing. As a bonus, human babies do not have to deal with the implementation details of setting up an AWS account to learn from their images as this was taken care of through millions of years of evolution.

7. Division of Labor

For this mini-project, both team members contributed equally. Ideas were talked about together as we worked in the same room as each other and discussions were had about best ways to implement various approaches. Various code modifications and parameter choices were made together. Where code or report needed to be written, an equal portion was done by both teammates, either as a collaboration or through a separate task of equal magnitude on the report or the code. Overall, work distribution was fair and chosen as needed.

References

[1] G. Bledt, P. Wensing, S. Ingersoll, and S. Kim. Contact model fusion for event-based locomotion in unstructured terrains. *(submitted) 2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[2] J. Guérin, O. Gibaru, S. Thiery, and E. Nyiri. CNN features are also great at unsupervised classification. *CoRR*, abs/1707.01700, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] L. Herranz, S. Jiang, and X. Li. Scene recognition with cnns: Objects, scales and dataset bias. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 571–579, 2016.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] A. Mohan, C. Papageorgiou, and T. Poggio. Example based object detection in images by components. *IEEE Trans. Pattern Anal. and Machine Intell*, 23:349–361, 2001.

[7] M. C. Potter, B. Wyble, C. E. Hagmann, and E. S. McCourt. Detecting meaning in rsvp at 13 ms per picture. *Attention, Perception, & Psychophysics*, 76(2):270–279, Feb 2014.

[8] J. Sanchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1665–1672, Washington, DC, USA, 2011. IEEE Computer Society.

[9] J. L. Shinskey and L. J. Jachens. Picturing objects in infancy. *Child development*, 85 5:1813–20, 2014.

[10] UMBC. Umbc phrase similarity service, 2013.

[11] B. Zhou, A. Khosla, À. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *CoRR*, abs/1610.02055, 2016.

[12] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc., 2014.