650 Project Final Report
Team member: Jing Cao, Jieping Yang
Title: Lego brick image search engine

Introduction

LEGO usually sells building bricks in sets to build a specific object like a Lamborghini or the White House. Though those parts are in different shapes, sizes, and colors, there are a lot of common parts that can be used in different sets. If you have some scattered parts, in which sets can you use them? Even further, if you try to build something new with your lego bricks on hand, what could you build? There is a website called Rebrickable(https://rebrickable.com/) that provides a search function with a similar purpose as ours, and one of our datasets is actually collected from Rebrickable API. However, they don't provide an image search. Our project is to build an image search engine especially for the users who have no idea what the name of each part is. The user can get information about the name of this part, as well as which lego sets the part could be used for, after uploading the part image.

Related Work

Objects in realistic settings exhibit considerable variability. So to learn to recognize them, it is necessary to use much larger training sets. Traditional machine learning methods are no longer suitable for huge amounts of realistic data with the relatively small labeled dataset. Neural networks, specifically Convolutional Neural Networks (CNNs) and Graph Convolutional Networks (GCNs), are perfect for large-scale image recognition or classification. However, GCNs usually suffer from a huge computational cost. That's why we chose CNN methods (Krizhevsky et al, 2017) trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into 1000 different classes. A number of attempts after that have been made to improve the original architecture of Krizhevsky et al. in order to achieve better accuracy. The best-performing submissions to the ILSVRC2013 (Zeiler & Fergus, 2014; Sermanet et al., 2013) utilized smaller receptive window sizes and smaller stride of the first convolutional layer. Another line of improvements dealt with training and testing the networks densely over the whole image and over multiple scales (Sermanet et al., 2013); Howard, 2013). VGG(He et al., 2016) pushed the depth to 16–19 weight layers by adding more convolutional 3x3 layers. The Microsoft team presented a residual learning framework (ResNet)(Simonyan et al., 2014) to ease the training of networks that are substantially deeper than those used previously, for example, VGG nets (He et al., 2016).

Data

Lego sets data: https://www.kaggle.com/rtatman/lego-database

We have gotten several csv files from this website, just as Fig. 1 shows, and we need to clean them up to move forward.

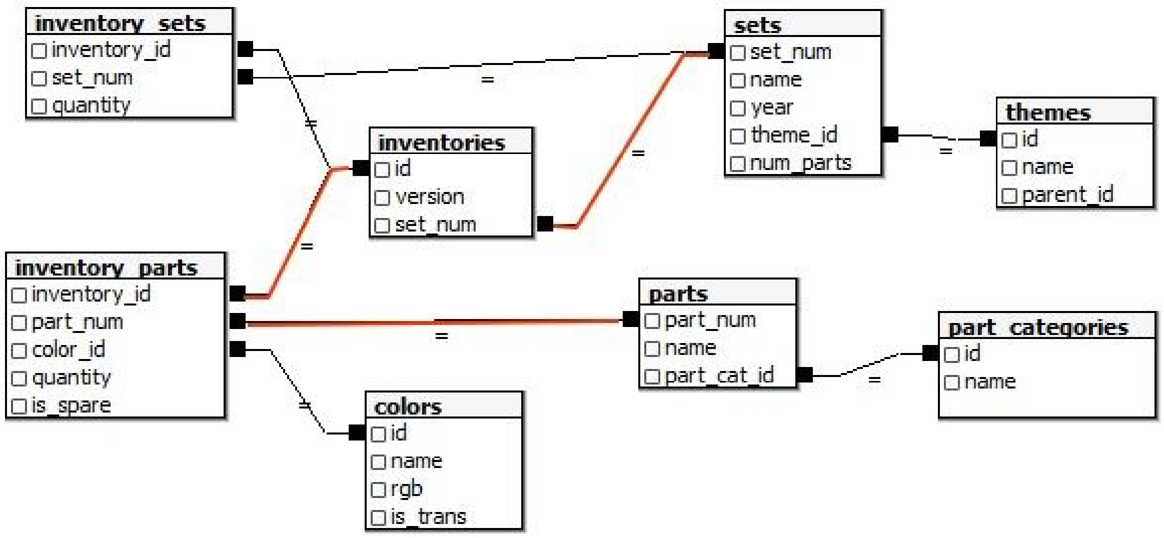
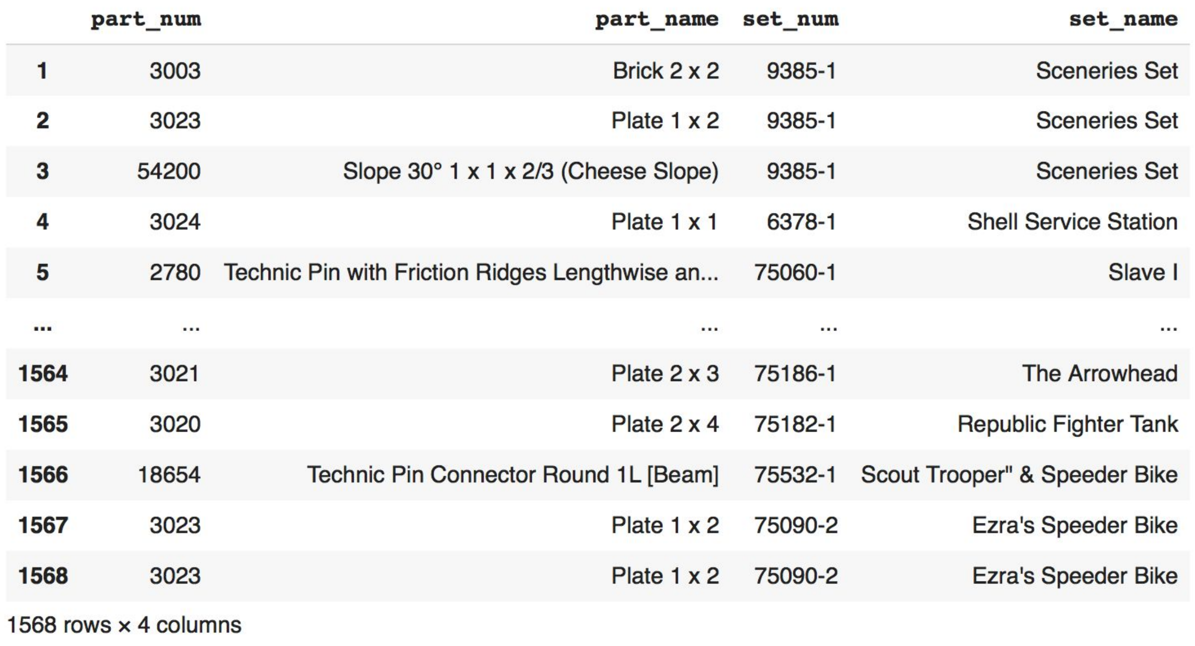


Fig.1 The Schema of the LEGO parts information database

Because one of the aims of this project is to check whether a part is in a set or not, then we need to collect them into one database which includes part_num, part_name, set_num, set_name. In that way, if we know a part name or number, we can retrieve it to a set, a specific object. Our final dataset is a DataFrame including just 4 columns, as the picture shows. Based on both the lego sets data and the image dataset, we simplified this database. The final dataset only contains 1568 objects, just as Fig. 2 shows.



	part_num	part_name	set_num	set_name
1	3003	Brick 2 x 2	9385-1	Sceneries Set
2	3023	Plate 1 x 2	9385-1	Sceneries Set
3	54200	Slope 30° 1 x 1 x 2/3 (Cheese Slope)	9385-1	Sceneries Set
4	3024	Plate 1 x 1	6378-1	Shell Service Station
5	2780	Technic Pin with Friction Ridges Lengthwise an...	75060-1	Slave I
...	...	...	...	...
1564	3021	Plate 2 x 3	75186-1	The Arrowhead
1565	3020	Plate 2 x 4	75182-1	Republic Fighter Tank
1566	18654	Technic Pin Connector Round 1L [Beam]	75532-1	Scout Trooper" & Speeder Bike
1567	3023	Plate 1 x 2	75090-2	Ezra's Speeder Bike
1568	3023	Plate 1 x 2	75090-2	Ezra's Speeder Bike

1568 rows × 4 columns

Fig. 2 The LEGO database of part number, part name, set number and set name

Image data: https://www.kaggle.com/joosthazelzet/lego-brick-images?
The image dataset contains 50 classes of lego parts. After combining it with the lego sets data, we finally have 32000 images that includes 38 kinds of parts with 800 images for each. This dataset is still very limited on the shapes and colors of lego bricks. We split the data into training, validation, and test with a rough ratio of 7:1.5:1.5. Therefore, for each class, we have 560 images to train the models.

Methods

Cleaning the database

The first step is to check whether the parts we need or not. Because we have a list of the parts with photos, we will check those names and numbers firstly. And we simplified the database. There are only 38 different parts we need in total.

After this step, the second step is to merge those useful databases together. From the database "parts" to "inverntory_parts", check if the number of those parts is the same then merge them as a new database. Using the same method to merge it with "inventories", then by "sets". During the first merge from parts to inventory_parts, we only extract the parts whose color is light gray, because the images we have are light gray LEGO parts. After 3 merges, we got the final database. The result is a new database with "part_name", "part_num", "set", and "set_number" variables to help us do information classification.

Building models

We tried two models for classification. One is Resnet50. The other one is our customized model. The architecture of the customized model is shown in Fig. 3. We use ReLU as the activation function for all the layers except the last one that uses the Sigmoid function. In order to reduce computational cost and prevent overfitting, we also added a Dropout layer with the probability of 0.5. There are 42506726 parameters to train. We used Adam with a learning rate of 0.00001 as the optimizer, categorical cross-entropy as our loss function, and accuracy as the evaluation metric. (We didn't use other metrics in this project but other metrics such as error rate at top 1 or top 5 could be used to evaluate the models).

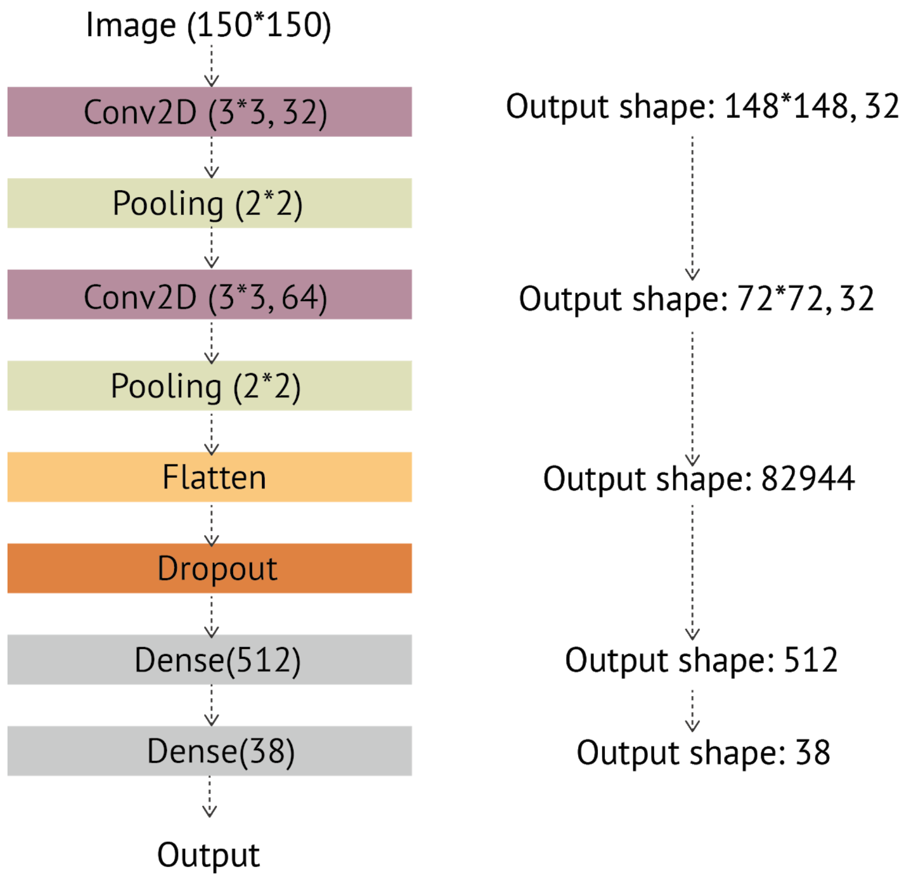


Fig. 3 The architecture of our customized model

We run those two models on two sets of data. One is the original image data that is rescaled by 1/225 so that every pixel is between 0 and 1. The other one is the image data with augmentation, more specifically, data warping in this project. The reason for data warping is to increase the amount of data by adding modified pictures. This method is proved to be an effective way for regularization that prevents overfitting (Wong et al., 2016). Noted that we only augmented the training data. A sample of data warping is shown in Fig. 4.

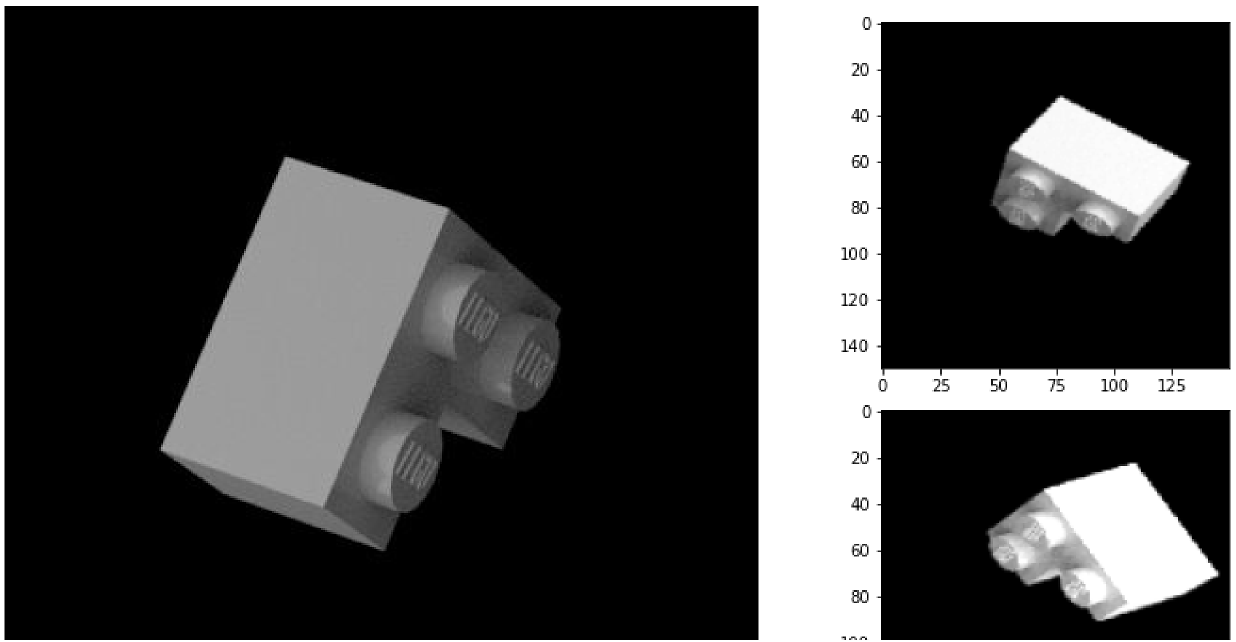


Fig. 4 Left is the original image for "2357 brick corner 1x2x2 166R". The right two images are the random samples of modified copies. Parameters for data augmentation: rescale = 1/225, rotation_range = 40, width_shift_range = 0.1, height_shift_range = 0.1, brightness_range = (0, 2), shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True, and fill_mode = "nearest".

Predicting new lego pieces

Once the model is workable, we can get a number of the parts from an image uploaded by the user. We defined a new function whose input data is the information of the part, and output data will be the name of the sets that can be builded by this specific part. By this system, the users will upload images of the parts they have, then our model will get the number of this specific part. After that, the function we defined can export the information of the sets which could be built by this part. In this way, we can achieve the original goal that is to check if the users try to build something new with their lego bricks on hand, what they could build.

Evaluation and Results

From Table 1, we can see that the Resnet50 achieved a very low accuracy, which is a surprise for us. We also tried it on augmented data in order to prevent the overfitting problem and it turned out to be even lower accuracy. Our customized model has a much better result on the original data with an accuracy of 0.6147 on the original data and an accuracy of 0.3972 on the augmented data within 30 epochs.

Table 1 Classification result for Resnet50 and customized models on test set.

	Accuracy	**Loss**
Resnet50 (baseline)	0.1336	3.2417
Resnet50 w/ augmentation	0.0825	3.3061
Customized model	0.6147	1.2206
Customized model w/ augmentation	0.3972	1.7350

Looking at the training and validation accuracy through time in Fig. 5 and 6, the interesting thing is that ResNet50 doesn't show severe overfitting problems on the original data, but it suffered from overfitting when it was trained on the augmented data. The customized model, on the other hand, follows the instinct. It shows overfitting problems on the original data but not on the augmented data (underfitting under 30 epochs). Within 30 epochs, the customized model fitted on the original data achieved better accuracy than the one fitted on the augmented model. But the latter model could achieve better results if we have time to run more epochs without overfitting.

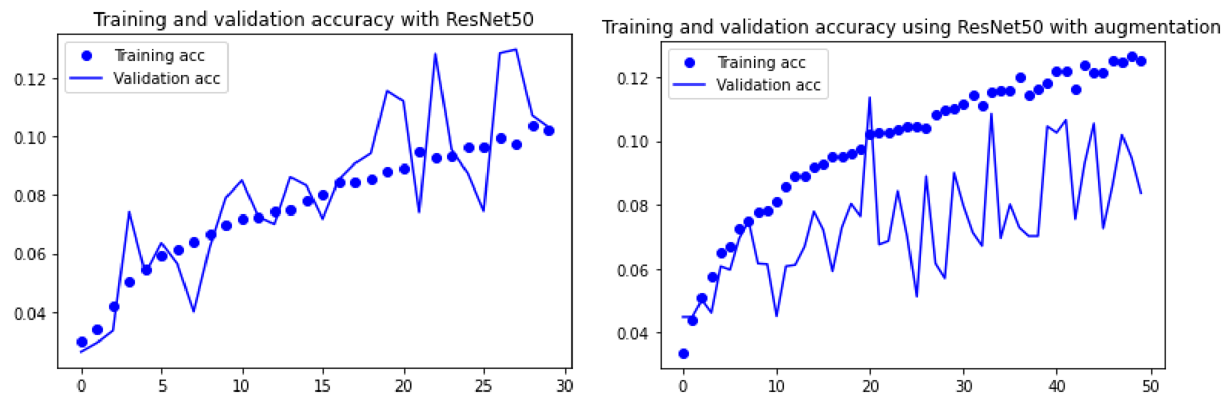


Fig. 5 Training and validation accuracy with ResNet50 on the original data (left) and the augmented data (right).

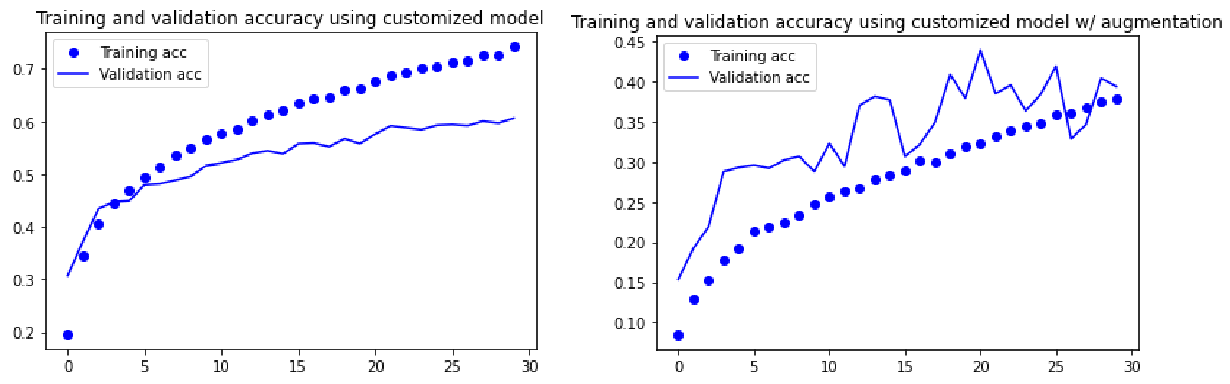


Fig. 6 Training and validation accuracy with customized model on the original data (left) and the augmented data (right).

Discussion

In general, all of the models performed not well on lego image classification, especially with the data with augmentation. Our customized model, as a much simpler model, works better than Resnet50 in this project. We think the main reason is the lack of original data. The original training data in one class includes 560 pictures of one lego piece with different angles. Even though more original data with more details could help this complex convolutional neural network to improve the result in general, twisting the images to generate more data may not be useful in this case because some of the twisted images lose information due to the brightness changes (imitating real pictures).

Besides, Resnet50 is pre-trained on ImageNet dataset (Russakovsky et al., 2015) that includes animals (Lion, cat, bird, etc), foods (strawberry, orange, etc), objects (minivan, racket, mug, etc.). The dataset (last checked at 19:00 pm on Dec. 16th, 2020) has very limited data related to building materials (96) and even fewer pictures of similar building pieces (e.g., bricks) to lego pieces. Therefore, it's reasonable to have a very low accuracy based on the parameters trained on non-relevant data to lego pieces. In the future, if we still benefit from those kinds of pre-trained models, we can unfreeze the last few layers in the base model and train both these layers and the customized part to fine-tune the neural network based on our own data.

Conclusion

In general, both ResNet50 and our simple customized model perform not very well on this lego image classification due to limitation of data and computational power. Data augmentation could prevent overfitting problems under certain circumstances (simple model on small dataset in this project).

Other Things We Tried

We also tried the Xception pre-trained model that was introduced by Chollet (2017) from Google. We saw this model in one Kaggle notebook created by Artem Nechitaylo (https://www.kaggle.com/artemnechitaylo/lego-bricks-classification-keras-cnn). The author used this model combined with three more dense layers on only 10% of the data and got an accuracy of 59.91% on the test data. However, after a few epochs running on the data with augmentation, we found that the model didn't learn much on the validation set with the validation accuracy of only about 23% and severe overfitting as shown in Fig. 7. We think the huge difference between the results may be due to the difference in data augmentation. The author augmented all of the data, including training, validation, and test set. But we only augmented the training data. Therefore, we dropped this method.

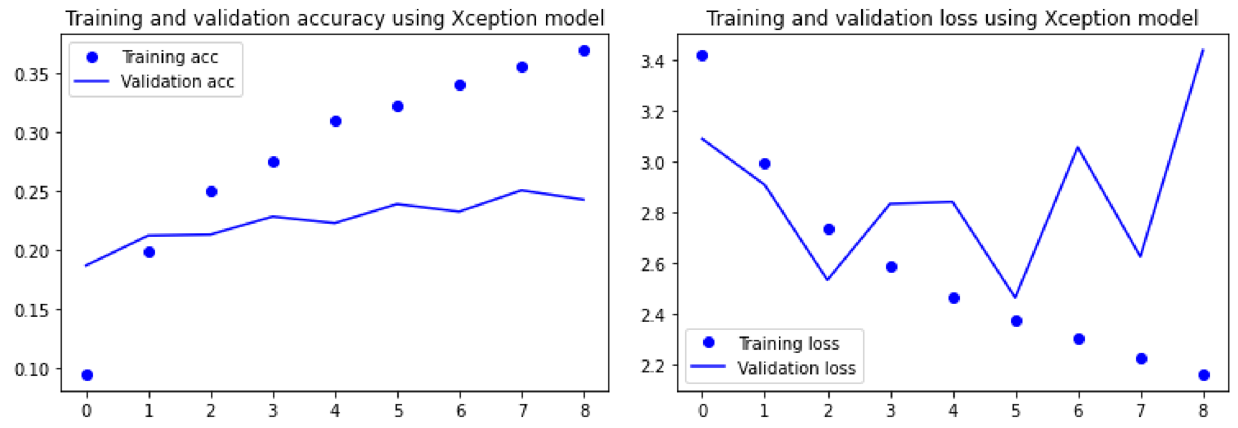


Fig. 7 Xception model for data with augmentation within 9 epochs.

Another thing we tried is that we run the notebook both on Jupyter Notebook and Colab. Surprisingly, Jupyter Notebook runs faster than Colab without any disconnection problem. Even so, we still need almost 10 hours to run 30 epochs.

What we would like to do next
Overall, this project is a pilot trial for lego part classification with limited data and low computational power. There are a lot of places that can be improved in future work.

We can generate many more images so that we can train the model so that even without data augmentation, we can still reduce our avoidable bias. We can also take real lego piece pictures to fit our model. Real pictures are actually more fit for this project because our goal is to interact with real-world users.

Besides, we didn't consider color information in this project. However, many lego pieces have the same structure but different colors. We can deal with color information in two ways according to the users' needs. The first way is that our model can ignore the color and only consider the structural information, so users can create a different color palette as they desire. The second is that our model strictly classifies the pieces with color as one of the features so that users can rebuild the original set.

Moreover, from the model aspect, we only use two convolutional layers since we want to keep the model simple for the consideration of computational cost. However, to decrease the high bias in our current models, we could increase the size of our neural network by adding more layers or neurons. Besides, we can also tune the parameters such as the size of the pooling layer, the number of epochs, the learning rate, etc.

Our task is fine-grained image classification, so we could also try other pre-trained models for the same task, such as TBMSL-Net (Zhang et al., 2020) that is trained on FGVC Aircraft (Maji et al., 2013), and DAT (Ngiam, 2018) that is trained on Stanford Cars (Krause et al., 2013).

At last, we can create a web page that allows users to upload pictures so that our system can better interact with users.

References

Chollet, F. (2017). Xception: Deep learning with depth wise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Howard, A. G. (2013). Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*.

Krause, J., Deng, J., Stark, M., & Fei-Fei, L. (2013). Collecting a large-scale dataset of fine-grained cars.

Wong, S. C., Gatt, A., Stamatescu, V., & McDonnell, M. D. (2016, November). Understanding data augmentation for classification: when to warp?. In *2016 international conference on digital image computing: techniques and applications (DICTA)* (pp. 1-6). IEEE.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

Maji, S., Rahtu, E., Kannala, J., Blaschko, M., & Vedaldi, A. (2013). Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*.

Ngiam, J., Peng, D., Vasudevan, V., Kornblith, S., Le, Q. V., & Pang, R. (2018). Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

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

Zhang, F., Li, M., Zhai, G.,, & Liu, Y. (2020). Multi-branch and Multi-scale Attention Learning for Fine-Grained Visual Categorization. arXiv:2003.09150v3