Classification of Brick Patterns

Huan KUANG, huan2015@ufl.edu

Abstract—**In this project, I presented an application of Convolutional Neural Network (CNN) for multi-class image classification. Specifically, the goal of this project was to predict the brick bond pattern from an image, where each image can belong to only one pattern. I implemented the CNN with PyTorch. The CNN used in this study has two convolutional layers, two pooling layers, and three fully connected layers. The similarities among the brick bond patterns were a substantial challenge in this study. Evaluations on validation sets and a benchmark set for testing purposes showed a relatively inadequate performance result. I got an accuracy of 61.2% on the validation set, and an accuracy of 60.5% on the test set.**

I. INTRODUCTION

MULTI-CLASS image classification is a valuable task, in this regard, has been well discussed in a large number of existing literature. A variety of architectures have been used on the ImageNet Large Scale Visual Recognition Challenge (LSVRC) dataset, which contains 1.2 million images from the 1000 classes. Seminal contributions have been made by Krizhevsky and his colleagues on utilizing Convolutional Neural Networks (CNN) to classify the images in the LSVRC dataset [1]. They achieved the top-5 error rates of 17.0% for the multi-class image classification. Krizhevsky and his colleagues also won the ILSVRC-2012 competition with a top-5 test error rate of 15.3% by using a variant of this model [1].

Recently, CNN becomes a popular method for image classification. The most popular pre-trained CNN architectures are GoogLeNet, AlexNets, and ResNet50. Regarding classifying construction material images, Bunrit and her colleagues applied the AlexNet and GoogleNet to classify 3,600 images of bricks, wood, and concrete [2]. They attained 95.5% as the highest classification accuracy by fine-tuning GoogleNet for distinguishing the three classes of construction materials. Besides the CNN, other methods such as Support Vector Machine (SVM), Multilayer Perceptron (MLP), etc. were also employed to classify the multi-class images. Rashidi and his colleagues compared the performance of SVM, MLP, and Radial Basis Function (RBF) in terms of classifying images of concrete, red bricks, and OSB boards [3]. They found SVM outperformed the MLP and RBF, and it was not surprising that they found MLP was performing poorly. Comparing to CNN, there were two major drawbacks of using MLPs for image processing. Firstly, MLP cannot preserve the pieces of information about the relative positions between pixels. If the objects in an input image change their positions, the MLP will respond differently to that image. Secondly, MLP needs more weight parameters than CNN as it uses one perceptron for each pixel in an image. Take the brick image used in this project as an example. I have to train 120,000 weights for one image with MLP since each image has 200 x 200 pixels with three color channels.

As far as I know, no previous research has investigated and classified brick images. In this study, I propose to classify the brick bond patterns in images via CNN. CNN has been known as a special structure of MLP. One of the major differences between MLP and CNN is that CNN has convolutional layer(s). The convolutional layer with filters extracts features as a matrix of pixels from an input image [1]. Therefore, CNN could preserve the relationship between pixels and requires fewer weight parameters.

II. IMPLEMENTATION

In this section, I will explain the specific implementation details for my project. The CNN architecture (see Figure 1) of this project contained seven layers, among which two are convolutional layers, two are pooling layers, and the rest three are fully connected (FC) layers. Based on the functions, the layers can be separated into the feature-engineering and classification groups. Particularly, convolutional and pooling layers correspond to feature-engineering which prepared data for FC layers in a way that extracted the most important features in an efficient manner [4]. Likewise, the FC layers corresponded to the classification which classified the input features into five classes in this project.

The model was implemented using the PyTorch library, and the learning rate is set to 0.0001 together with epoch as 2,000 and patience as 10. (The learning rate was set by trying 11 values among 0.000001, 0.000005, 0.00005, 0.00001, 0.0005, 0.0001, 0.005, 0.001, 0.05, 0.01, 0.1. Regarding the time spent on the training process and the accuracy, 0.0001 was the best learning rate for this project. Moreover, while using the validation set to counter overfitting, I deployed the mechanism that continues the training process for several iterations to avoid falling into a local minimum in the early stage of training. The number of iterations allowed is called patience. This value of patience was learned from trial and error. I tried setting different patience values such as 3, 5, 10, and 15. I found the best value was 10 for this project.)

A. Convolutional Layers

As mentioned before, the convolutional layer(s) played a role in extracting features in CNN. In this section, I focused on illustrating two fundamental operations, namely, convolution operation and non-linear operation.

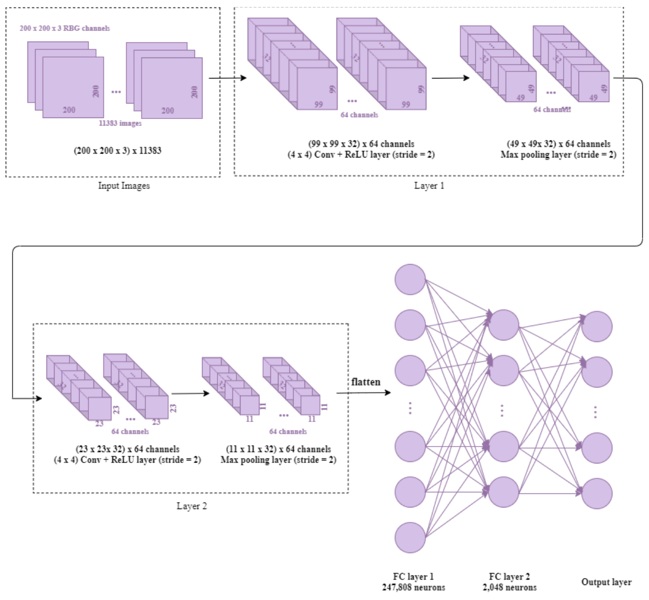
200 x 200 x 3 RBG channels
200
200
11383 images
(200 x 200 x 3) x 11383
Input Images
32
99
64 channels
(99 x 99 x 32) x 64 channels
(4 x 4) Conv + ReLU layer (stride = 2)
49
64 channels
(49 x 49x 32) x 64 channels
Max pooling layer (stride = 2)
Layer 1
23
64 channels
(23 x 23x 32) x 64 channels
(4 x 4) Conv + ReLU layer (stride = 2)
11
64 channels
(11 x 11 x 32) x 64 channels
Max pooling layer (stride = 2)
Layer 2
flatten
FC layer 1
247,808 neurons
FC layer 2
2,048 neurons
Output layer

Fig. 1. CNN Architecture

1) *Convolution operation*

The role of convolution operations was to reduce the size of the input images, meanwhile holding the critical features for classification. The convolution operations in my CNN convoluted the 3D (width x height x dimension) input with a 2D filter. After all the pixels passing the filter in a form of sliding windows, I obtained a matrix called the feature map, which is smaller than the input matrix. The equation for computing the feature map values with a 2D filter is defined in equation (1):

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \quad (1)$$

where *i* and *j* are the indexes of rows and columns of the result matrix, *m* and *n* are the indexes of rows and columns of the filter/kernel matrix, the input image is denoted by I, and the filter/kernel is denoted by K [5].

For the 3D input image in this project, the CNN applied the 2D filter on each of R, B, G color channels separately. Both convolution operations in layer 1 and layer 2 had 64 channels with a convolution window size of 4 x 4. The stride is set as 2, thus the filter moves 2 pixels at a time. (I also tried to set the convolution window size as 3 x 3 and 5 x 5. Regarding the training accuracy, this was not different when used a different window size. Regarding the time spent on the training process, the 3x3 option took 10 more mins to train the model. Similarly, I tried setting stride as 1, and there was no difference regarding training accuracy but required longer training time.)

2) *Non-linear operation*

The non-linear operation followed each convolution operation. The researchers suggested that the non-linear operation helped derive the features' non-linear property [6]. Moreover, it is extremely important to have a non-linear operation since the forwarding pass in the neural network is a linear function in nature. Without the non-linear operation, the structure of multiple layers will disappear, and eventually, there will only be one linear layer [7]. The non-linear operation had its active function and will not have any impact on the size of the feature map. In this project, I used the Rectified Linear Unit (ReLU) function. See the equation for ReLU in equation (2). RuLU converted all negative pixel values to zero and kept the non-negative pixel values as the original.

$$\phi\,(\mathrm{x}) = \max\,(0, \mathrm{x}) \quad (2)$$

B. *Pooling Layers*

The pooling layers were applied after each of the non-linear operations in the convolutional layers. The pooling layers helped extract the most important features that were positional and rotational invariant [1]. They were expected to reduce the size of feature maps but maintain the significant features for classification at the same time. The benefit of reducing the size of feature maps was that it could decrease the needed computational power. In this project, I adopted the max-pooling strategy which returned the maximum value from a specific region in a feature map. The maximum value was a representation of its region after breaking down the feature map into various regions by a kernel. I used zero-padding and a stride size of 2 for the max pooling.

C. *Fully Connected Layers*

I used the fully connected (FC) layers to generate the final classification results by learning the possible non-linear combinations of the features provided by convolutional layers. There were three FC layers in this CNN. Strictly speaking, each neuron in one layer is connected to all neurons in the previous layer. The first and second FC layers were hidden layers with 247,808 neurons in the first FC layer, and 2,048 neurons in the second FC layer. The last FC layer was the output layer and it had 5 neurons which equate to the number of classes. As shown in figure 1, the first FC layer received the features that were extracted by the second max pooling layer. The output matrix of feature maps in the convolutional layers had been flattened to column vectors as input for FC layers. The FC layers trained a neural network in a similar manner as MLP did. Expressly, the FC layers computed the weights for the two hidden layers and the output layer with forward paths, and updated the weights based on the backpropagation iteratively.

I used the Softmax function at the output layer (the last FC layer) for multi-class classification, where each image can belong to only one class. Softmax generated a 5-dimensional vector, which corresponding to the five desired brick patterns in this project. A probability between 0 and 1 was associated with each dimension of the vector. The class (dimension of the vector) which corresponded to the highest probability will be assigned as the predicted class of an input image. The Softmax function is shown in equation (3).

$$S(\vec{z_i}) = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}} \quad (3)$$

where $\vec{z_i}$ donate the input vector to the softmax function, C donate the number of classes in the multi-class classifier.

For instance, if the Softmax result of an input image is [0.83, 0.04, 0.06, 0.05, 0.02], the predicted class for that input image will be Class 0. In this project, I forced CNN to put classes that were not in the training data into Class 0 because it corresponded to the category of images other than bricks.

D. Dropout Layer

To prevent potential over-fitting, I added a dropout layer after the first FC layers, which is a hidden layer. Dropout was a regularization technique that imitated training several models that had "different architectures or be trained with different data [8]". Dropping certain neurons could help with reducing redundancy in the weight matrix in the training process, since neurons were mutually dependent on each other in an FC layer [9]. I set the probability of dropout as 0.2. It means that some of the input neurons will be randomly zeroed during training with a probability of 0.2 using samples from a Bernoulli distribution in PyTorch [10].

E. GPU

To train the CNN in this project, I used an NVIDIA GeForce RTX 2080 GPU with 8GB of memory. The purpose of using GPU was to improve the efficiency of the training process. Average GPU utilization when train the CNN is about 20%, and it took about 30 mins to complete the model training.

III. EXPERIMENTS

A. Data and Normalization

In this project, the total number of brick images was 11,383, and the number of classes was 5. There were 1949, 2112, 2575, 3420, and 1327 images in Class 0 (images other than bricks), Class 1 (Flemish Stretch Bond), Class 2 (English Bond), Class 3 (Stretcher Bond), and Class 4 (Other Brick Patterns) respectively.

Instead of viewing the image as a whole, the computer treated each image as the pixels in matrices which corresponds to red, green, and blue color channels. The original pixel values in the input images were unsigned integers between 0 and 255. I normalized the pixel values to the range from 0 to 1 prior to training the CNN by calculating the mean value per image and subtracting the mean value from the pixel values. (I also normalized the pixel values to the range from -1 to 1. There was no difference between the two normalization methods for this project regarding the training accuracy and the time spent on training.) The primary reason for having image normalization was to avoid the potential gradients exploding and to increase the convergence speed [4].

B. Training, Validation and Test Data

First of all, I created a benchmark set for testing purposes. This test set was not used in the process of training weights or cross-validation, so it is reasonable to test the performance of a trained model with it. Since the five classes had unbalanced sample sizes (more images in Class 3 and fewer images in Class 4), I randomly split each class into training and test sets separately. Specifically, within each class, 60% of the images were randomly selected as training data, and the rest 40% were used as the test set. This approach was to ensure that each class will have similar representability in both training and test sets. Table I presented the number of images of each class in training and test sets.

TABLE I.
Summary of sample size

	All	**In Training Set**	**In Test Set**
Class 0	1949	1168	781
Class 1	2112	1267	845
Class 2	2575	1545	1030
Class 3	3420	2050	1370
Class 4	1327	797	530
Total	**11383**	**6827**	**4556**

Within the training set, three different cross-validation schemes were adopted to validate the stability of the model and to evaluate the performance of my CNN while training. The three cross-validation schemes were three-fold, five-fold, and ten-fold. The fundamental consideration for having three different cross-validation schemes was the unbalanced class issue. Theoretically speaking, randomly splitting the unbalanced class into a large number of folds, such as 10, cannot guarantee equal representation of each class in training and validation sets. There was no clear guideline about how to identify the appropriate number of folds. Therefore, I tried three-fold, five-fold, and ten-fold cross-validation schemes.

C. Performance on the Training Set

This CNN achieved a model accuracy of 67.7 % on the training set with the five-fold cross-validation scheme. This result was consistent across what has been found with three-fold and ten-fold cross-validation schemes. Class 0 which was the images other than bricks has the highest precision, recall, and F1 score. In other words, the trained CNN could accurately classify the brick images and non-brick images.

TABLE II.
Confusion Matrix on Training Set

Predicted	**Ture Class**				
Class	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	**949**	16	27	35	63
Class 1	11	**587**	202	88	87
Class 2	12	161	**614**	149	71
Class 3	16	213	310	**1212**	117
Class 4	13	52	44	51	**283**

TABLE III.
Precision, Recall, and F1 Score for the Five Classes on Training Set

Class	**Truth overall**	**Prediction overall**	**Precision (%)**	**Recall (%)**	**F1 Score**
Class 0	1001	1092	87.06	94.81	0.91
Class 1	1029	975	60.21	57.05	0.59
Class 2	1197	1007	60.97	51.30	0.56
Class 3	1535	1868	64.88	78.96	0.71
Class 4	621	443	63.88	45.57	0.53

However, this model had difficulty in distinguishing the brick patterns. Especially, this model performed poorly on predicting Class 1 which was Flemish Stretch Bond, Class 2 which was

English Bond, and Class 4 which was the other brick pattern. The similarities among the brick bond patterns were a substantial challenge in this study. The confusion matrices of this model on the training set were presented in Table II, and the precision, recall, and F1 score were shown in Table III. It was possible that the model slightly overfits this training set. Thus, I got better performance on the training set and relatively unsatisfactory performances on validation and test sets.

D. Performance on the Validation Set

Firstly, regarding the five-fold cross-validation scheme, the model accuracy on the validation set was 61.2 %. Similar to what I found on the training set that the trained CNN performed well on predicting the brick images vs non-brick images. However, it performed insufficiently on predicting each of the brick bond patterns. The confusion matrices on the validation set were presented in Table IV, and the precision, recall, and F1 score were shown in Table V.

Moreover, the model accuracy on the validation set was 58.3% for the three-fold cross-validation scheme, while the model accuracy is 51.7% on the validation set based on the ten-fold cross-validation schema. The reason for having a lower model accuracy rate on the ten-fold cross-validation schema was that there were a smaller number of images in the validation set. As mentioned before, randomly splitting the training data into 10 folds cannot guarantee equal representation of each class in training and validation sets.

TABLE IV.
Confusion Matrix on Validation Set

Predicted	**Ture Class**				
Class	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	**149**	9	16	12	8
Class 1	6	**137**	73	44	22
Class 2	5	45	**183**	63	40
Class 3	4	37	71	**345**	37
Class 4	3	10	5	51	**69**

TABLE V.
Precision, Recall, and F1 Score for the Five Classes on Validation Set

Class	**Truth overall**	**Prediction overall**	**Precision (%)**	**Recall (%)**	**F1 Score**
Class 0	167	194	76.80	89.22	0.83
Class 1	238	282	48.58	57.56	0.53
Class 2	348	336	54.46	52.59	0.54
Class 3	515	494	69.83	66.99	0.68
Class 4	176	138	50.00	39.21	0.44

E. Performance on Test Set

The test set was not used in the process of training weights or cross-validation, and the trained CNN achieved a model accuracy of 60.5 % on this set. Overall, the findings on the test set were in accordance with findings reported on training and validation sets. This model correctly classified the majority of Class 0, which indicates the images other than bricks. This result suggested that the trained CNN had a certain power to distinguish the images other than bricks from the brick pattern images.

However, the model had limited capability in distinguishing and predicting the given brick bond patterns. Specifically, this model inaccurately identified most of Class 1 (Flemish Stretch Bond). Half of the Class 1 images were misclassified as Class 3, and the other 15.8% of the Class 1 images were mislabeled as Class 2. Moreover, half of the Class 2 were also mislabeled as Class 3. Class 3 were overestimated. The trained CNN had an extremely low power of correctly labeling the images in Class 4. Class 4 was always underestimated. 18% of Class 4 were mislabeled as Class 0, 15% of Class 4 were misidentified as Class 1, 13% of Class 4 were inaccurately classified as Class 2, and 26 % of Class 4 were misclassified as Class 3. The confusion matrices on the test set were presented in Table VI, and the precision, recall, and F1 score were shown in Table VII.

The potential explanation for the unanticipated inadequate performance could be that the brick patterns were too similar to each other. Moreover, certain images in the original dataset were taken from different orientations which add some noise to the classification.

TABLE VI.
Confusion Matrix on Test Set

Predicted	**Ture Class**				
Class	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	**713**	25	55	47	94
Class 1	8	**369**	164	99	78
Class 2	9	90	**423**	94	67
Class 3	22	317	368	**1097**	136
Class 4	29	44	20	33	**156**

TABLE VII.
Precision, Recall, and F1 Score for the Five Classes on Test Set

Class	**Truth overall**	**Prediction overall**	**Precision (%)**	**Recall (%)**	**F1 Score**
Class 0	781	934	72.26	91.41	0.83
Class 1	845	718	51.54	43.67	0.47
Class 2	1030	683	61.84	41.03	0.49
Class 3	1370	1939	56.55	80.13	0.66
Class 4	530	282	55.52	29.38	0.38

IV. CONCLUSIONS

In conclusion, I trained and evaluated a CNN with two convolutional layers, two pooling layers, and three fully connected layers for classifying input images to five classes of brick bond patterns. The model obtained an accuracy of 59.1 % on the test set. The CNN predictions were relatively more accurate when it comes to distinguishing the brick images from images other than bricks. The results showed that training the proposed CNN model using current data has limited power to accurately classify the Flemish stretcher bond, English bond, stretcher bond, and other brick patterns.

REFERENCES

[1] I. S. Alex Krizhevsky and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[2] S. Bunrit, N. Kerdprasop and K. Kerdprasop, "Evaluating on the Transfer Learning of CNN Architectures to a Construction Material Image

Classification Task." *Int. J. Mach. Learn. Comput*, vol. 9, no. 2, pp. 201-207, 2019.

[3] A. Rashidi, M.H. Sigari, M. Maghiar and D. Citrin, "An analogy between various machine-learning techniques for detecting construction materials in digital images". *KSCE Journal of Civil Engineering*, vol. 20, no. 4, pp.1178-1188, 2016.

[4] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors". *arXiv preprint arXiv:1207.0580,* 2012.

[5] R.N. Bracewell. "Chapter 3: Convolution," in *The Fourier Transform and Its Applications.* 3rd Edition, New York: McGraw-Hill, 2000, pp. 24-54.

[6] H. Li, W. Ouyang and X, Wang, "Multi-bias non-linear activation in deep neural networks". In *International conference on machine learning*, pp. 221-229, 2016.

[7] H. Lee, R. Grosse, R. Ranganath, and A. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM, pp. 609–616, 2009.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp 1929-1958, 2014.

[9] D. Simard, P.Y. Steinkraus and J.C. Platt, "Best Practices for Convolutional Neural Networks", *Proc. Seventh Int'l Conf. Document Analysis and Recognition*, 2003.

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Desmaison, A. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems* (pp. 8026-8037). Curran Associates, Inc.