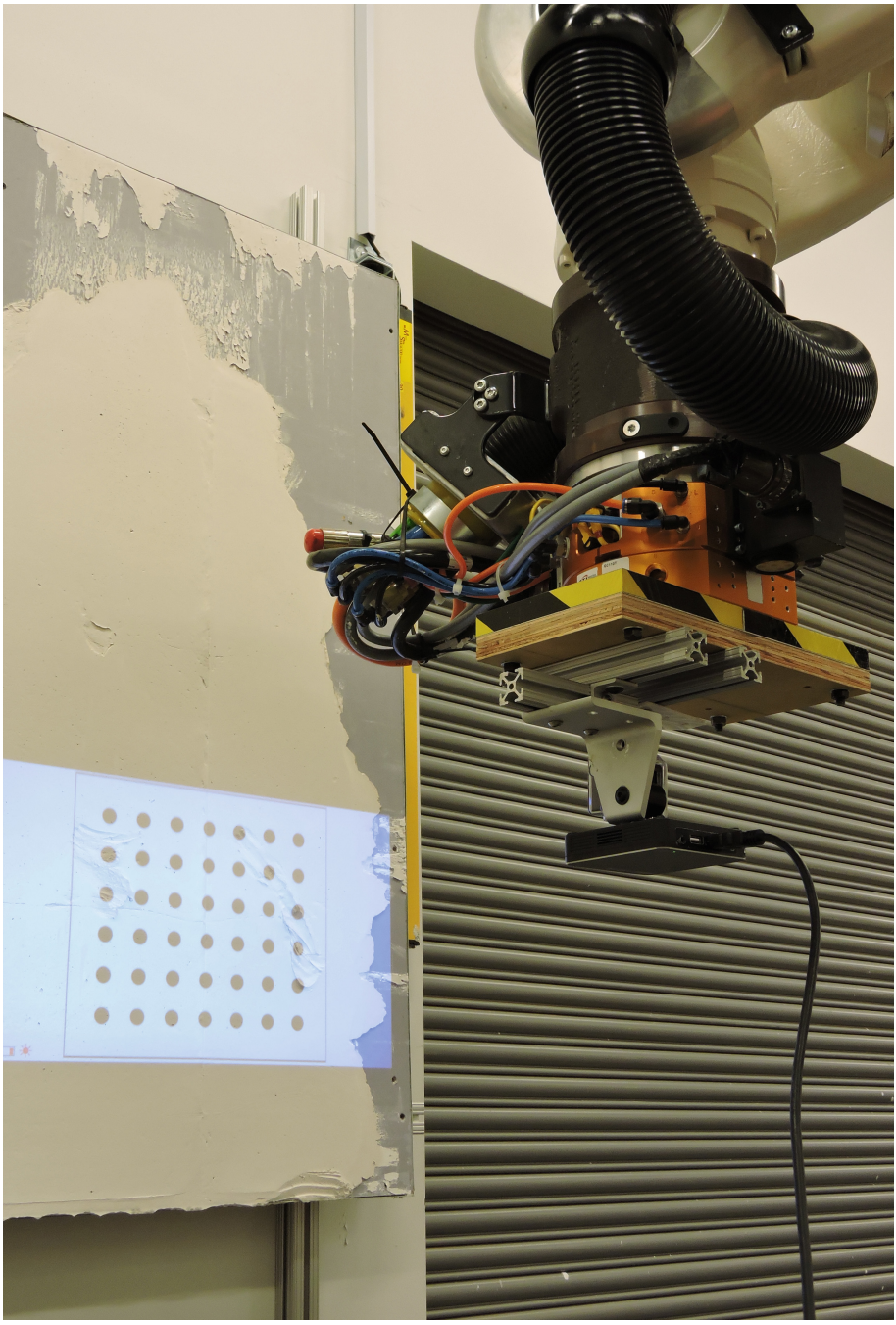

Image Classification for Robotic Plastering with Convolutional Neural Network

Joshua Bard(✉), Ardavan Bidgoli, and Wei Wei Chi

Carnegie Mellon University, Pittsburgh, PA 15213, USA
jdbard@cmu.edu, {abidgoli,weiweic}@andrew.cmu.edu

Abstract. Inspecting robotically fabricated objects to detect and classify discrepancies between virtual target models and as-built realities is one of the challenges that faces robotic fabrication. Industrial-grade computer vision methods have been widely used to detect manufacturing flaws in mass production lines. However, in mass-customization, a versatile and robust method should be flexible enough to ignore construction tolerances while detecting specified flaws in varied parts. This study aims to leverage recent developments in machine learning and convolutional neural networks to improve the resiliency and accuracy of surface inspections in architectural robotics. Under a supervised learning scenario, the authors compared two approaches: (1) transfer learning on a general purpose Convolutional Neural Network (CNN) image classifier, and (2) design and train a CNN from scratch to detect and categorize flaws in a robotic plastering workflow. Both CNNs were combined with conventional search methods to improve the accuracy and efficiency of the system. A web-based graphical user interface and a real-time video projection method were also developed to facilitate user interactions and control over the workflow.

Keywords: Architectural robotics · Machine learning
Convolutional neural networks · Image classification

1 Motivation

Surface finishing is an essential domain in the architectural construction practice, which requires high-skilled workers and demand accurate quality control procedures. By way of example, the authors have developed a robotic workflow to use industrial robots for decorative plastering techniques (Bard et al. 2016a, b). One of the remaining challenges in this workflow is to implement an automated, precise, and reliable quality control pipeline to guarantee satisfying results through a touch-up scenario. The touch-up procedure would let the user automatically inspect the surface and detect any unwanted fabrication artifact and command the robot to correct it.

Researchers have developed a wide range of scanning systems using different combination of sensory data, including, but not limited to, multiple RGB cameras/view (Vasey et al. 2014), RGB cameras combined with pattern projection (Rocchini et al. 2001; Zhang et al. 2002), RGB-D sensory data (Amtsberg et al. 2015), and depth data (Bard et al. 2016a, b) to reconstruct a digital representation of the physical models that can be used in the feedback loop.

J. Willmann et al. (Eds.): ROBARCH 2018, *Robotic Fabrication in Architecture, Art and Design 2018*, pp. 3–15, 2019.
https://doi.org/10.1007/978-3-319-92294-2_1

Construction tolerances standards may vary from rough to finish application resulting in different level of accuracy in each of these phases (Bard et al. 2016a, b). The achievable level of accuracy by the above-mentioned techniques with respect to the construction tolerances might not be desirable for such a delicate task as surface finishing and touch-up tasks.

Our approach requires a vision-based solution to detect texture flaws (i.e., scratches, bubbles, …) and small-scale 3D finishing issues (i.e., holes, unfinished patches). It proposes a single-camera solution without 3D reconstruction as the main input for the quality check workflow. This will result in simpler hardware setup, faster workflow, and lower costs. This approach can also be useful for other fabrication workflows, for example subtractive and deforming manufacturing.

The proposed system takes advantage of a state-of-the-art computer vision method based on *Convolutional Neural Network* (CNNs or *ConvNets*) for image classification and object detection.

Recently CNNs have dominated the image processing field, outperforming other image processing and computer vision methods by a large margin. Since 1990's CNNs have been used for different applications, including, but not limited to optical character recognition, medical image processing, feature extraction, object recognition, image understanding, and optimization (Egmont-Petersen et al. 2002; LeCun et al. 1989), thanks to their robust and real-time performance even in noisy spaces (Pal and Pal 1993).

The breakthrough advancements in computational hardware (efficient GPU architectures, possibility of distributed/multi-core/cloud-based/parallel processing, and dramatic cut in the hardware and service prices), alongside the open-source and widely accessible software platforms for machine learning, simplified and enhanced the implementation of CNNs in different contexts. CNNs can now be deployed with a reasonable budget and fewer technical challenges. These factors render CNNs as the method of choice for image classification and object detection in the past few years.

2 Methodology

The authors tested two approaches to implement CNN for this task, (1) transfer learning on *Google*'s *Inception v3* model and (2) design and training a CNN from scratch. The resulting CNN were tested for accuracy and efficiency in a robotic plastering workflow as a vision-based feedback loop for surface touch-ups.

Since its public release, Inception v3 has been used as an almost off-the-shelf image classifier for different use-case scenario. Several research teams leveraged this architecture for scientific studies, possibly most notably, for example detecting skin cancer classification (Esteva et al. 2017). In its purest form, it only requires users to organize the training dataset in a folder structure and run the provided script for a desired number of epochs. This doesn't necessitate any substantial machine learning knowledge or advanced programming skills. On the other hand, design and training a model from scratch demands for both but may provide simple and optimized results.

2.1 Apparatus Setup

Hardware. The hardware setup consists of two different end-of-arm-tool assembly (EOAT) and the connected computer unit. The first EOAT was designed to apply plaster and the second one was dedicated to the quality control and feedback loop. It includes a camera and a video projector to capture images and project results on the probing surface (see Fig. 1).
Despite the resiliency and scalability, CNN algorithms are computationally complex and expensive. While embedding computing solutions from *nVIDIA*, i.e. *Jetson* series, are capable of running such CNNs, authors decided to centralize all the computational process on a connected computer, leveraging GPU acceleration.

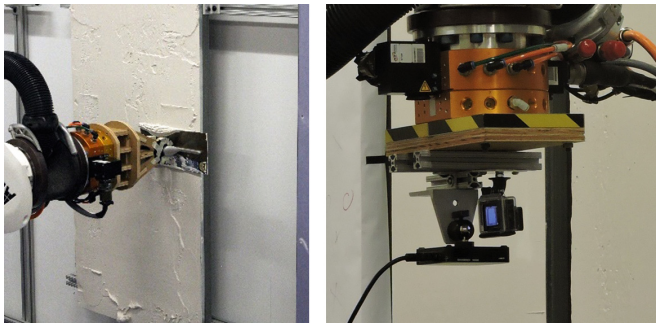

Fig. 1. End-of-arm-tools, left: plastering tool, right: camera and projector

Software.[1] A user interface was developed as a web-based application using *Django* framework. This architecture makes it possible to use *HTML*/*JavaScript* interactions at the front-end while leveraging *Python* scripts on the back-end. User can interact with the robot for motion commands, triggering the image processing workflow, and monitor the results.

To control the cameras, an in-house library was developed leveraging GoPro's built-in Wi-Fi protocol that provides full control over the camera functionalities.

The robotic control module was developed based on project *Open-ABB* (Dawson-Haggerty, n.d.). It communicates with the *IRC5* controller to transfer motion commands and inquire robot's status (see Fig. 2).

2.2 Data Set

The data set consists of images taken from a series of plaster finishes applied by a robot on drywall test panels. To collect the training samples, the GoPro camera was used to take 5 mega-pixel images of available plastered panels. Due to the GoPro camera significant lens distortion an image calibration method was applied using *OpenCV*, and

[1] Code for this project are available on GitHub (https://github.com/Ardibid/RoboticPlasteringCNN).

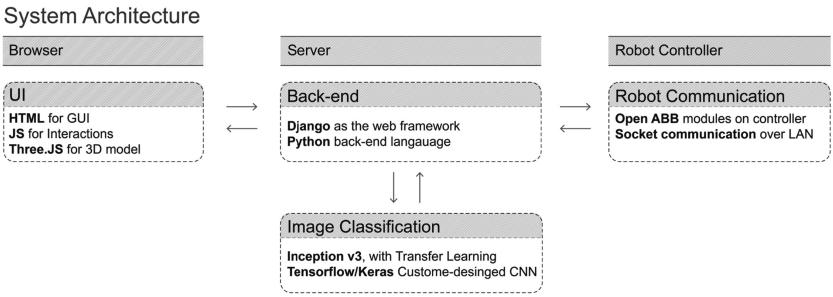

Fig. 2. System architecture

only the central 1024 px × 1024 px region of each image was used. Images were manually cropped and labeled into smaller sections as one of the three main classes: (1) pass, (2) fail, (3) markup (see Table 1). Then the same data set was categorized in five classes; perfect or near-perfect plaster regions were labeled as (1) pass, while images containing fabrication flaws including: (3) holes, (4) scratches, and (5) unfinished surfaces were categorized as fail. The markup class was left intact (see Table 2).

The markup class was dedicated to hand drawn characters that users could sketch on the work surface to communicate with the robot. Markup training samples were taken from hand drawn marks on a white surface in the same lighting condition as the plastered panels.

Table 1. Training data set for three classes

Class	Training samples	Validation samples	Test samples
1 Pass	77	23	26
2 Fail	135	42	52
3 User markup	91	28	36
Total = 510	303 (∼%60)	93(∼%18)	114(∼%22)

Table 2. Training data set for five classes

Class	Training samples	Validation samples	Test samples
1 Bad finish	26	9	10
2 Holes	29	8	11
3 Rough finish	78	22	29
4 User markup	87	29	36
5 Pass	68	23	27
Total: 492	288 (∼%59)	91(∼%18)	113(∼%23)

2.3 Image Classification Methods

In addition to designing and training a CNN from scratch, it is a well-established practice to repurpose currently available CNN architectures and their pre-trained models for a new task. Shin et al. categorized three methods to repurpose CNNs for image detection as: (1) training CNN from scratch, (2) using available CNNs without training the network, and (3) using unsupervised pre-training and fine-tuning (Shin et al. 2016).

Transfer learning on Inception v3. In 2014, *Google*'s entry for the *Large-Scale Visual Recognition Challenge* (ILSVRC2014), titled *GoogleNet* demonstrated astonishing performance (Russakovsky et al. 2015; Szegedy et al. 2015; Szegedy et al. 2016). The core model behind *GoogleNet* was called *Inception* (Szegedy et al. 2016) (Fig. 3) which adopted a relatively simpler architecture compared with other competitors and was computationally less expensive. Since then, the inception model has been used in cutting-edge research for object classification in different contexts, notably medical image processing (Esteva et al. 2017).

What makes the *Inception* model an ideal platform for object categorization in different contexts is its flexibility to be retrained for relatively similar tasks. This approach, called *transfer learning*, is a well-practiced method to fine-tune and repurpose CNN models for new tasks with very small training data set to train a deep CNN (Donahue et al. 2014).[2]

By only modifying the second to last layer of the model, transfer learning on inception v3 eliminates the need for training a whole new model from scratch for every new set of classes. Transfer learning could be a proper choice for this use case scenario since (1) it has already been trained to detect a wide range of features and there is no need to train it from scratch, theoretically this will save substantial amount of time; (2) it performs well when the size of training dataset is relatively small; (3) it requires a

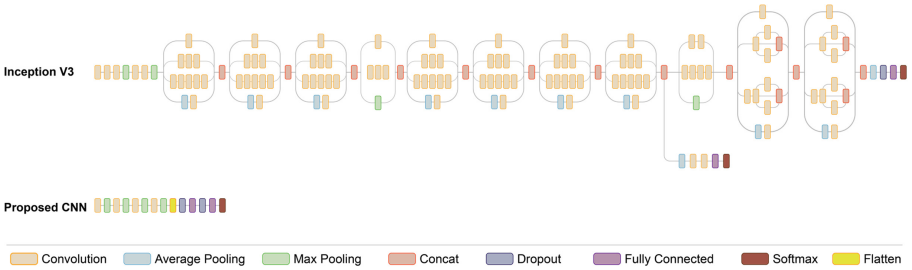

Fig. 3. Inception v3 architecture compared with the proposed architecture (Inception architecture diagram is reproduced from ("Inception v3," 2018))

[2] In the same year, at CVPR2014, Oquab et al. and Sharif et al. also addressed transfer learning and representation. For further information please look at (Oquab et al. 2014; Sharif Razavian et al. 2014).

simple workflow to repurpose the model. In light of these three advantages, we decided to exploit transfer learning technique as a possible back-end methods for our feedback loop system.

Proposed CNN. A significant trade-off of using transfer learning is the heavy model that it entails. Trained to classify one thousand classes of objects, the CNN trained model occupies hundreds of megabytes on the system storage and requires expensive computation to process a single image.

However, in our case, most of the captured images are of low contrast with primarily white backgrounds and subtle changes in color. This color space requires different feature layers for an efficient classification. Accordingly, the authors designed and trained a sequential multi-layer CNN. This architecture has already been proved its performance in several state-of-the-art models, including *AlexNet* (Krizhevsky et al. 2012) and later *VGGNet* (Simonyan and Zisserman 2014).[3] The proposed architecture is significantly simpler than of the *Inception*, resulting in a speed boost.

The authors designed and tested a series of CNNs using *Keras* with *Tensorflow* back-end to find an optimum architecture. Several combinations of convolutional, dropouts, and fully connected layers have been tested. In each architecture, all models have been trained for a fixed number of epochs and the model with the highest f1 score were selected. The results from each architecture were then compared with each other to select the optimum architecture. The selected architecture demonstrated the highest f1 score on both 5 and 3-class classification, while the others failed to demonstrate same f1 score or took longer epochs to converge to the same score.

The proposed architecture consists of four convolutional layers (3×3 kernel) paired with *Relu* activation function, and *maxPooling* (2×2), followed by three fully connected layers and *softmax* at the end. To reduce the effects of overfitting, it also leverages *dropout* to prevent inter-dependencies between hidden layer nodes (see Fig. 4 and Table 3).

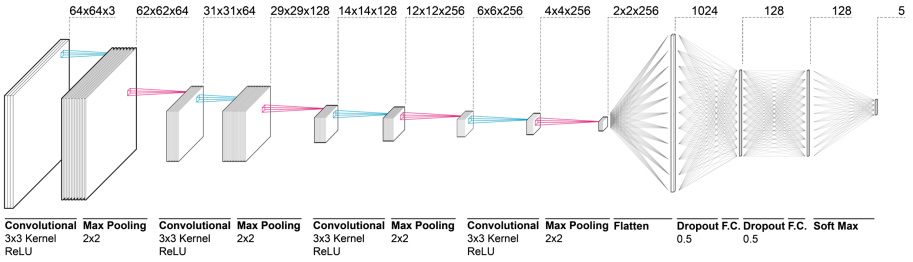

Fig. 4. Our CNN architecture for 5-class model

[3] AlexNet architecture might be confusing at the first sight since it has two parallel pipelines. However, the reason behind this dual pipeline is to train the model on two separate GPU simultaneously.

Table 3. Proposed CNN layers

Layer (type)	Output shape	Param #
conv2d_1 (Conv2D)	(None, 62, 62, 64)	1792
max_pooling2d_1 (MaxPooling2)	(None, 31, 31, 64)	0
conv2d_2 (Conv2D)	(None, 29, 29, 128)	73856
max_pooling2d_2 (MaxPooling2)	(None, 14, 14, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_3 (MaxPooling2)	(None, 6, 6, 256)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_4 (MaxPooling2)	(None, 2, 2, 256)	0
flatten_1 (Flatten)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 128)	131200
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 5)	645

3 Training Results

3.1 Transfer Learning

We used the model from the Tensorflow GitHub repository and followed the steps described in Tensorflow documentation ("Image retraining Tutorial," n.d.). The result after 4000 epochs are reported in Table 4.

Table 4. Training results, transfer learning (The training results in this chart directly result from the script provided on TensorFlow's GitHub repository after introducing new training samples. No fine-tuning, modification, or custom loss function has been applied to the re-training process. F1 score is measured on a test data set after the training phase.)

Three-class model	Ave. time to process batch of 114 patches of 64 × 64
Train acc.: 1.000 | Val. acc.: 1.00 (N = 100) Test. acc.: 0.964 (N = 112) Test F1 score: 0.9469	10–11 s
Five-class model	
Train acc.: 0.990 | Val. acc.: 0.79 (N = 100) Test. acc.: 0.904 (N = 104) Test F1 score: 0.8938	12–13 s

3.2 Our Model

In this model, the authors leveraged data augmentation to increase the data set size and improve the model's resiliency against small variations of the input data. Training and test samples where reshaped to the same size (28 × 28 × 3) beforehand. The model

Table 5. Training results, our model

Three-class model | F1 Score	Ave. time to process batch of 114 patches of 64 × 64
Train: 0.9826 | Validation: 0.9892 | Test: 0.9736	19 ms
Five-class model | F1 Score	
Train: 0.8559 | Validation: 0.9333 | Test: 0.9292	19 ms

was trained in two scenarios, one with (1) pass, (2) fail, and (3) markup labels[4] and the second one trained to define different types of fail including (1) bad finish, (2) hole, (3) rough finish (see Table 5 and Fig. 5).

3.3 Method Comparison

Comparing the speed and accuracy of the two approaches, signifies that the lighter and less complex architecture of the proposed CNN is on par and even better than what we could obtain using transfer learning. Transfer learning is significantly less complicated method, from the user point of view, that doesn't require significant understanding of machine learning. However, it is not an efficient method for fast image classification which is essential in this use-case-scenario.

On the other hand, designing, fine-tuning, and testing a CNN from scratch requires additional skills and substantial amounts of time in advance. But it pays off with the accuracy and efficiency it brings to the inspection process. Accordingly, we decided to continue with this proposed CNN.

4 Testing Implementations Scenarios

The proposed CNN was used as the image-classification back-end for a robotic plastering feedback loop, which consist of classification tool, user interface, and user interactions.

4.1 Image Surveying Methods

To survey each image, the algorithm divides it into a gird of 64 × 64 px patches that could be fed into the classifier in batches.[5] With the hardware setup described above, it took 50 ms on average to process a 256 × 64 × 64 batch of data, equivalent of a 1024 × 1024 image.

[4] Markups are simple user-defined drawings, i.e. circles and crosses, that can be used to communicate with the system.

[5] Although the authors first implemented Quad Tree search algorithm to compensate for the possible slow classification pipeline, the final model performance was good enough to provide near real-time experience. Accordingly, we opted for a grid search algorithm and avoided potential challenges that a Quad Tree search would introduce. The biggest drawback being its tendency to ignore small features in the initial steps of the search process when surveying large areas of the given image.

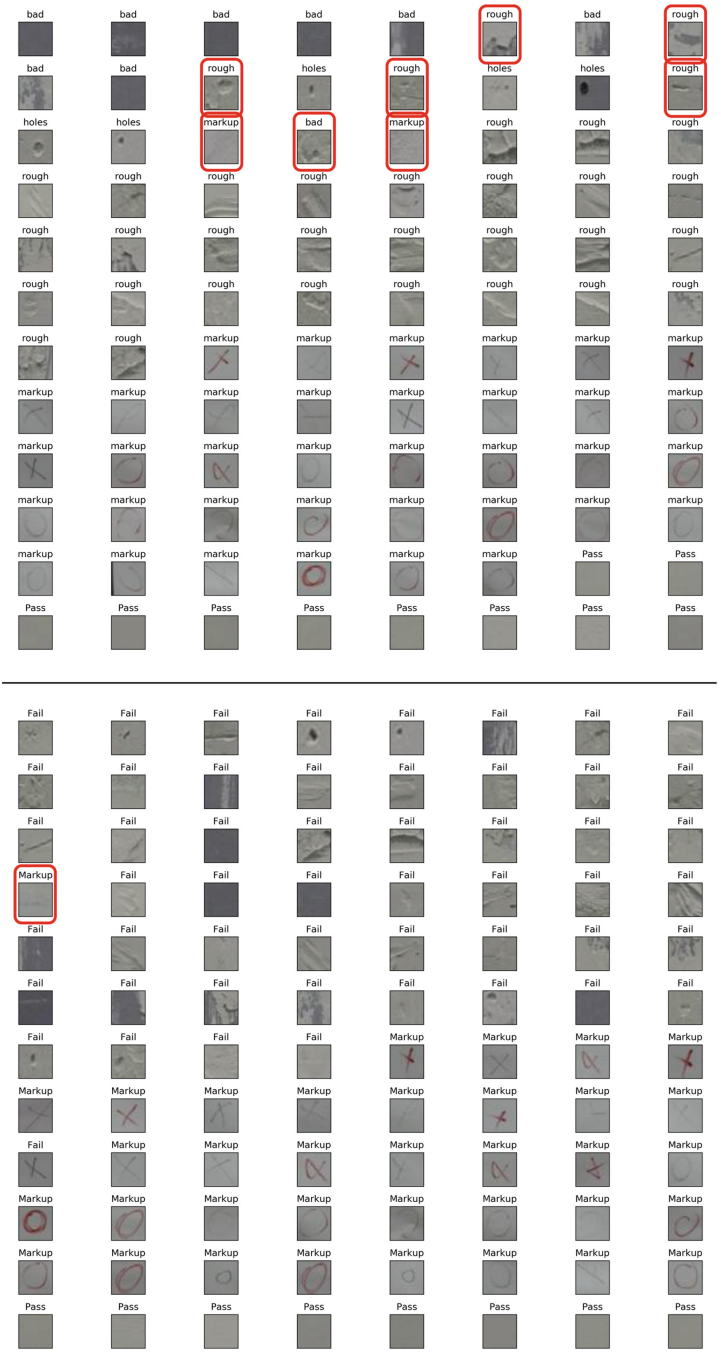

Fig. 5. Test cases for five-class (Top), three-class (Bottom) models. Errors are highlighted in red

4.2 User Interface/Interaction

Two models of interaction were designed and implemented for user interaction, first a web-based graphical user interface (see Fig. 6). In this interface, the user can send motion commands, make queries on the robot status and trigger the feedback loop process. Moreover, the user can command the robot to trowel a new layer of plaster and/or compensate for the detected flaws in a localized area of the surface.

Second interaction model consists of a real-time image projection over the working area. Users can jog the robot to a specific region of the surface and trigger the feedback process. In this case, a mounted projector highlights the detected flaws directly on the surface in real-time (see Fig. 7). In this model of interaction, users do not need to use a separate computer unit, making it more useable for on-site applications. Users can also interact with the robot by drawing a series of pre-defined mark ups on the surface. Using such markups, user can override the feedback loop results and force the robot to add/remove specific regions to error/pass cases (se Fig. 8).

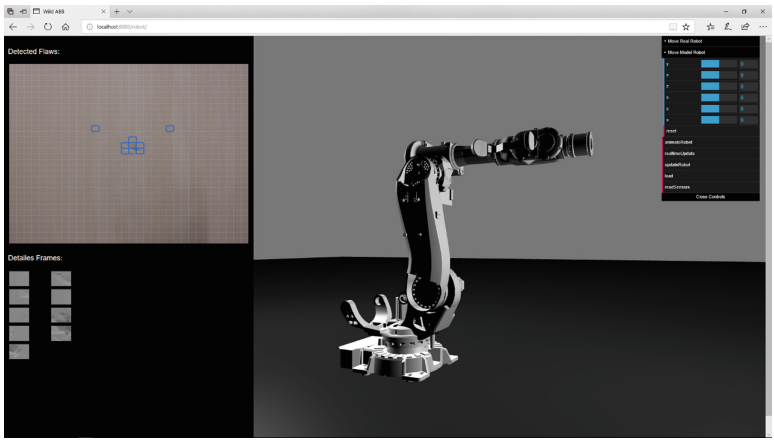

Fig. 6. User interface as running on a web browser

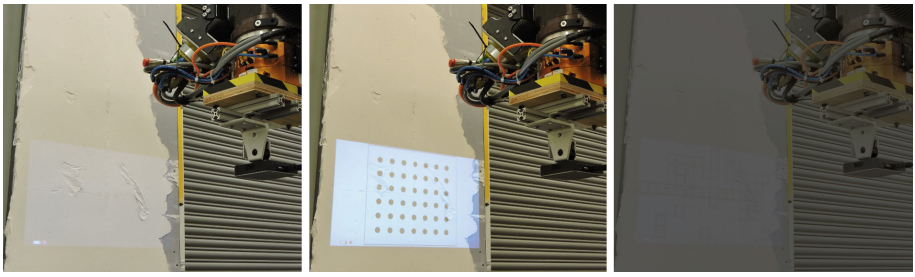

Fig. 7. Feedback-loop highlights on the finished surface (Left: scanning area, Center: camera/projector calibration, right: projected results, highlighting fail regions)

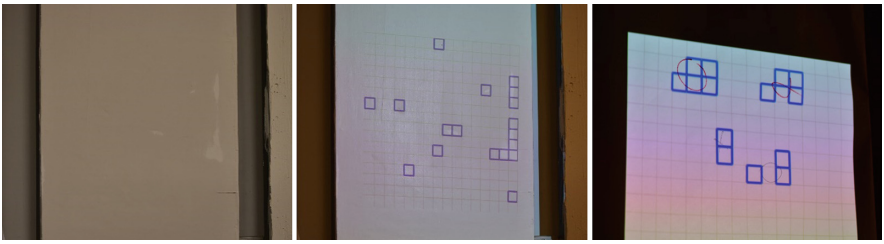

Fig. 8. Projection results on a surface with fine finish (Left: Original surface, Center: Highlighted flaws, Right: Markup detection)

5 Conclusions and Next Steps

In this work, we proposed two vision-based feedback loops based on CNNs and measured their performance in classification accuracy and speed. Due to the specific visual properties of the scanned surfaces, a custom-made shallow model could successfully accomplish the task in almost real-time fashion, outperforming state-of-the-art deep CNN architectures in this application. The proposed models performed well on both rough surfaces and fine finishes, using only one RGB camera.

As additive manufacturing techniques for construction mature, there will be an increased demand for more robust finishing approaches when manufacturing viable architectural components. In automated robotic finishing and touchup scenarios, inspection feedback loops with a high level of hardware simplicity, accuracy, and capability of categorizing detected flaws are of great importance. As the authors look to develop this work, we anticipate extending the application of the proposed model to more complex component geometries (i.e. (Bidgoli and Cardoso Llach 2015)) and different finishing/touch up scenarios using a variety of building materials (e.g., concrete, metal, or wood).

References

Amtsberg, F., Raspall, F., Trummer, A.: Digital-material feedback in architectural design. In: Ikeda, Y., Kaijima, S., Herr, C., Schnabel, M.A. (eds.) Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia: Emerging Experience in Past, Present and Future of Digital Architecture, CAADRIA 2015, pp. 631–640. Daegu, South Korea (2015)

Bard, J., Blackwood, D., Sekhar, S., Brian, N.: Reality is interface: two motion capture case studies of human–machine collaboration in high-skill domain. Int. J. Architectural Comput. **14**(4), 398–408 (2016a)

Bard, J., Tursky, R., Jeffers, M.: RECONstruction. In: Reinhardt, D., Saunders, R., Burry, J. (eds.) Robotic Fabrication Architecture, Art and Design 2016, pp. 262–273. Springer, Switzerland (2016b)

Bidgoli, A., Cardoso Llach, D.: Towards a motion grammar for robotic stereotomy. In: Ikeda, Y., Kaijima, S., Herr, C., Schnabel, M.A. (eds.) Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia: Emerging Experience in Past, Present and Future of Digital Architecture, CAADRIA 2015, pp. 723–732. Daegu, South Korea (2015)

Dawson-Haggerty, M. (n.d.): Open ABB Driver. https://github.com/robotics/open_abb. Accessed 5 Oct 2017

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: DeCAF: a deep convolutional activation feature for generic visual recognition. In: Proceedings of the 31st International Conference on Machine Learning, PMLR, vol. 32(1), pp. 647–655. ACM, New York (2014)

Egmont-Petersen, M., de Ridder, D., Handels, H.: Image processing with neural networks—a review. Pattern Recogn. **35**(10), 2279–2301 (2002)

Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S.: Dermatologist-level classification of skin cancer with deep neural networks. Nature **542**(7639), 115–118 (2017)

Image retraining Tutorial (n.d.). https://www.tensorflow.org/tutorials/image_retraining#training. Accessed 20 Nov 2017

Inception v3 (2018). https://www.kaggle.com/pytorch/inceptionv3. Accessed 20 Feb 2018

Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems, NIPS 2012, vol. 1, pp. 1097–1105. Lake Tahoe, CA (2012)

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)

Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), pp. 1717–1724. Columbus, OH (2014). https://doi.org/10.1109/CVPR.2014.222

Pal, N.R., Pal, S.K.: A review on image segmentation techniques. Pattern Recogn. **26**(9), 1277–1294 (1993)

Rocchini, C., Cignoni, P., Montani, C., Pingi, P., Scopigno, R.: A low cost 3D scanner based on structured light. Comput. Graph. Forum **20**(3), 299–308 (2001)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Bernstein, M.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)

Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf: an astounding baseline for recognition, In: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR-W 2014), pp. 806–813. Columbus, OH (2014). https://doi.org/10.1109/CVPRW.2014.131

Shin, H.-C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Summers, R.M.: Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Trans. Med. Imaging **35**(5), 1285–1298 (2016)

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

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V, Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), pp. 1–9. Boston, MA (2015)

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), pp. 2818–2826. Las Vegas, NV (2016)

Vasey, L., Maxwell, I., Pigram, D.: Adaptive part variation. In: McGee, W., Ponce de Leon, M. (eds.) Robotic Fabrication in Architecture, Art and Design 2014, pp. 291–304. Springer International Publishing Switzerland (2014)

Zhang, L., Curless, B., Seitz, S.M.: Rapid shape acquisition using color structured light and multi-pass dynamic programming. In: Proceedings of First International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2002), pp. 24–36. Padova, Italy (2002)