3D Model-Based Data Augmentation for Hand Gesture Recognition

Ben Limonchik
Stanford University
benlimon@stanford.edu

Guy Amdur
Stanford University
gamdur@stanford.edu

Abstract

This project focuses on classifying hand gestures and improving the testing accuracy using virtual 3D models to augment the original limited dataset. An existing work done on this classification problem and dataset uses a constrained generative model to predict hand gestures. The goal of this project was to explore how generating new virtual data from 3D models can augment the original dataset to improve CNN models' performances. We experimented with various custom CNN architectures as well as pre-trained models such as VGG-16 and Inception-ResNet-v2 in order to optimize our classification. We used learning visualizations techniques in order to inform how we generated new virtual 3D data. By using the original real training dataset and the new virtual 3D training dataset we were able to outperform the original work done on this problem, reaching over 96% testing classification accuracy.

1. Introduction

Hand Gesture recognition has a wide variety of applications. Sign language is the primary way more than 70 million deaf people communicate with non-deaf people. Being able to automatically translate hand gesture to text can facilitate deaf to non-deaf communication. In this project we are investigating the problem of identifying six hand gestures representing letters and numbers in the sign language.

Furthermore, as Virtual Reality is increasingly gaining popularity, the ability to carefully recognize what hand gesture the user is using is crucial to ensure a reasonable and smooth user experience. Hand interaction is a main component in the VR/AR industry and hand tracking is essential for this kind of virtual interaction.

To tackle this problem our neural networks take in a set of hand images with a variety of backgrounds and outputs the scientific name of the hand gesture. Since our dataset included only 4872 training images, we attempted to generate virtual data from realistic 3D hand models using the Unity game engine in order to outperform networks trained on real data only. Since data collection is often an expensive and very time consuming process in deep learning studies, we attempted to test whether generating fake images on Unity in a quick and relatively cheap manner could be used to outperform networks that rely only on real images. Please note that the testing set was kept the same (i.e. only real images) to ensure that any accuracy improvements are consistent with real life examples.

2. Related Work

Interesting work has been done to solve the hand gesture recognition problem as documented in references [2], [3] and [17].

One paper by Sebastien Marcel [7] uses a neural network approach to classify hand gestures. Marcel first creates the dataset by segmenting the hands from a full-body images using a space discretisation based on face location and body anthropometry. The body-face space is built using the anthropometric body model expressed as a function of the total height itself calculated from the face height.

Next, Marcel uses a neural network model which had already been applied to face detection: the constrained generative model (Figure 1). The constrained generative learning tries to fit the probability distribution of the set of hands using a non-linear compression neural network and non-hand examples. Each hand example is reconstructed as itself and each non-hand example is constrained to be reconstructed as the mean neighborhood of the nearest hand example. Then, the classification is done by measuring the distance between the examples and the set of hands.

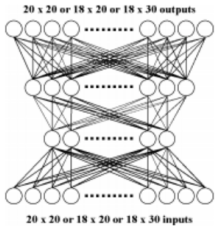


Figure 1: Constrained generative model

Most of the hand gesture database was used for training

and the remainder was used for testing. Although the images with complex backgrounds are more difficult to learn, the CGM achieves a good recognition rate and a small false alarm rate (Table 1 and 2 in Figure 2). Our goal was to achieve better results on the same dataset used in Marcel's paper using more advanced CNN architectures.

Table 1 : Mean results on our test database with uniform backgrounds

Postures	Number of images	Mean detection rate	Mean false alarm rate
A,B,C,V	241	93.8 %	1 / 11,111
A to V	382	93.4 %	1 / 11,904

Table 2 : Mean results on our test database with complex backgrounds

Postures	Number of images	Mean detection rate	Mean false alarm rate
A,B,C,V	165	74.8 %	1 / 18,867
A to V	277	76.1 %	1 / 14,084

Figure 2: Accuracies per model for the Marcel experiment

Other works we reviewed in preparation of this paper include NVIDIA's Hand Gesture Recognition with 3D Convolutional Neural Networks [8], Althoff's Robust multimodal hand-and head gesture recognition [1] and E. Ohn-Bar's [11]. These researches informed our approach to customizing CNN networks however they used much larger datasets of which not all of them were public and they were specific to car drivers and thus were irrelevant to our data augmentation problem.

Finally, in preparation of this paper we have either referred to or skimmed the following relevant works: [4], [5], [6], [13], [10], [9], [20], [12] [15] [19]

3. Methods

3.1. Customized CNN Architectures

After conducting a literature review, we came up with a few CNN architectures that we wanted to test. These architectures differ in the type and number of layers chained together. We use several combinations of convolutional, ReLU, batch-normalization and fully connected layers.

Using repeated convolutional layers with a small filter size allows us to increase the effective receptive field for each neuron whilst keeping the computational expense manageable. The ReLU activations introduce non-linearities and offer sparse activation and efficient gradient back-propagation. The batch-normalization forces the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. The fully connected layer, usually at the end of the network, introduces additional parameters and also allows us to reshape the output of the convolutional layers into the desired number of classes. Furthermore, in every few convolutional layers we added 2-by-2 maxpooling layers in order to reduce the dimensionality of the following layers and especially reduce the number of parameters of the last few fully connected layers.

Finally, we used a softmax loss also sometimes referred to as negative log-likelihood loss. The softmax loss (Figure 3) uses the raw scores of the input image in order to measure the probability of the image being classified as one of the six possible hand gestures.

$$Loss_{softmax} = \sum_{i=1}^{N} L_i$$

Where:

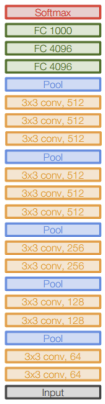
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Figure 3: Softmax loss

3.2. VGG-16 Network

VGG is a relatively new 16-layer CNN architecture developed at the University of Oxford in 2014 [14]. During training, the input to the network is a fixed-size RGB image. The only preprocessing done is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional layers, where the network uses filters with a very small receptive field of 3x3 (which is the smallest size to capture the notion of left/right, up/down, center). The convolution stride is fixed to 1 pixel; the spatial padding of the convolutional layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3x3 convolutional layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers (not all the convolutional layers are followed by max-pooling). Max-pooling is performed over a 2x2 pixel window, with stride 2.

A stack of convolutional layers (which has a different depth in different architectures) is followed by three fully-connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. See Figure 4 for a VGG-16 network diagram.



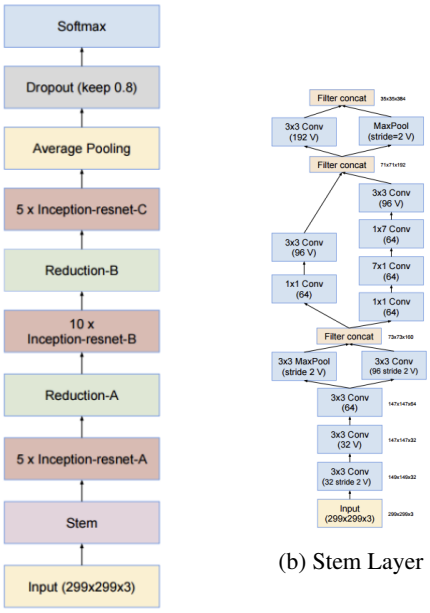
VGG16

Figure 4: VGG-16 Network Diagram

3.3. Inception-ResNet-v2 Network

The Inception-ResNet-v2 network is a CNN network architecture developed at Google in 2016 [16]. The network combines two of most recent ideas: Residual connections introduced by He et al. and the latest revised version of the Inception architecture suggested by Google. It is argued that residual connections are of inherent importance for training very deep architectures. Since Inception networks tend to be very deep, it is natural to replace the filter concatenation stage of the Inception architecture with residual connections.

The residual version of the Inception networks uses cheaper Inception blocks than the original Inception. Each Inception block is followed by filter-expansion layer (1x1 convolution without activation) which is used for scaling up the dimensionality of the filter bank before the addition to match the depth of the input. Another small technical difference between the residual and non-residual Inception variants is that the Inception-ResNet uses batch-normalization only on top of the traditional layers, but not on top of the summations. Finally, the Inception-ResNet-v2 network scales down the residuals before adding them to the previous layer activation to stabilize the training.



(a) Network Diagram

(b) Stem Layer

Figure 5: Inception-ResNet-v2 Architecture

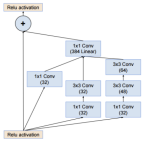


Figure 6: Inception-ResNet-A layer

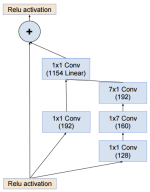


Figure 7: Inception-ResNet-B layer

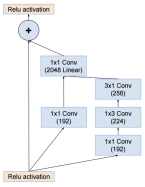
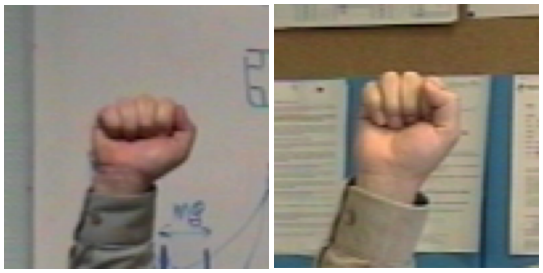


Figure 8: Inception-ResNet-C layer

4. Datasets

4.1. Marcel Dataset

The hand gesture dataset we used from Marcel's paper is composed of six different hand gestures called: a, b, c, point, five, and v (see image samples below). The dataset was collected from ten different people, eight of which were used for training data and the other two are used for testing data. All images are of size 77x66 pixels. The testing set is further broken down into two groups: 'uniform' images and 'complex' images. In the 'uniform' testing images the image of the hand gesture is taken with a white background in order to simplify the classification problem. On the other hand, in 'complex' testing set the images of the hand gestures are taken on top of a variety of colorful backgrounds.



(a) Uniform background (b) Complex background

Figure 9: Testing images

The complex test set is much more difficult to classify since the neural network needs to learn the pattern of the hand itself despite the complexity of the colorful background. As can be seen in table 1, there is an unequal amount of sample images from each category:

gesture/data-set	Training	Testing (uniform)	Testing (Complex)
A	1329	58	39
B	487	61	41
C	572	65	47
Five	654	76	58
Point	1395	65	54
V	435	57	38
Total	4872	382	277

Table 1: Number of data points of each Gesture

A training dataset of 4872 images is clearly not enough for this type of deep learning task. Therefore we attempted to generate additional images using artificial methods as discussed below. Note that for as our validation set we used 300 randomly samples images from the training dataset.

4.2. 3D Model-Based Dataset

Since data collection is an expensive part of deep learning, we had decided to take another approach for additional data collection. By generating additional data from 3D representations we got a cheap way of collecting additional, albeit virtual data. There were a few key conditions that the generated images had to fulfill.

- Be faster and cheaper to generate than actual photos.
- Look as close to the actual photos as possible.
- The photos must vary in lighting conditions, hand pose, skin color, finger placement etc.

4.2.1 Rendering with Unity3D

Unity3D is a game engine capable of creating realistic 3D graphics in realtime. Compared to non-realtime 3D rendering software such as 3Ds Max or Blender it produces less realistic imagery. However, this was not an issue since the output images would have a resolution of 76x66, which means that quality improvements of the non-realtime renderer would be barely visible anyway. Therefore, a realtime renderer is a better choice in this case because of its immense rendering speed compared to the alternatives.

Figure 10: The Unity editor window

4.2.2 Realistic Scene Setup

The base scene in Unity was set up to resemble the general layout of the real dataset as close as possible. The scene consisted of the hand model in the center and a plane covering the entire field of view at the back. The background plane had an actual photograph as a texture to improve realism. A single directional light source resembling the sun or an interior ceiling light was placed pointing down toward the hand in the scene.

The hand model was chosen for its realistic mesh and textures downloaded from Blend Swap [18]. Also, the model was rigged, meaning that the actual mesh can be controlled indirectly by manipulating a skeleton structure and its joints instead of the vertices of the mesh.

The hand model was posed to line up with the six different gesture classes already present in the Marcel dataset.

4.2.3 Varying Photo Conditions

For each new virtual image, a number of parameters were manipulated in the scene.

- The joints in the hand were randomized within a small range of angles, making the hand pose look slightly different for each image.
- The placement of the hand (as well as its rotation) within the image frame was randomized, increasing the variance of the dataset and hopefully improving the CNN's robustness to hand placement.

- The background image was randomly chosen from a set of interior photographs with various furniture, lighting conditions and color.

- The intensity and angle of the sunlight was randomized, increasing variance of lighting conditions within the virtual dataset. Note: light was the variable we played most with when generating virtual training data. We did so as a result of our findings using Saliency maps (see more explanation in section 5.1.1).

(a) Gesture A (b) Gesture B

Figure 11: Virtual images generated using 3D software

5. Experiment

5.1. Building customized architectures

In order to classify the test hand gestures we began by experimenting with a variety of custom neural network architectures in order to find an optimal network to fit our data. Table 2 summarizes the different architectures we tested:

Architectures	**Average testing accuracy**
[FC]-[FC]-[Softmax Loss]	18.2%
[Conv-Relu]-[FC]×2-[Softmax Loss]	37.9%
[Conv-Relu]×2-[FC]×3-[Softmax Loss]	52.3%
[Conv-Relu]×3-[FC]×3-[Softmax Loss]	41.2%
[Conv-Relu]×4-[FC]×1-[Softmax Loss]	39.2%
[Conv-Relu-BN-pool]×2-[FC]×1-[Softmax Loss]	56.9%
[Conv-Relu-BN-dropout-pool]×2-[FC]×1-[Softmax Loss]	46.0%

Table 2: Average testing accuracy per model

Given the small real dataset we have, we expected that networks with less fully connected layers and more convolutional layers would perform better on our classification problem. This expectation was confirmed by the first few models we trained which had multiple fully connected layers. Therefore, in later versions we tried to use more convolutional layers and smaller fully connected layers. We further introduced pooling layers to our models in order to decrease the dimensionality of the last fully connected layers and thus reduce the overall number of parameters in our network. Nevertheless, we observed that our testing accuracy didn't increase as we added more and more convolutional layers. We tested networks with one to four convolutional layers and found that the testing accuracy drops below two and above two convolutional layers. Therefore, moving forward we developed our next networks with two convolutional layers.

5.1.1 Data Visualization

In order to better understand what our network learned we visualized the weights of the first convolutional layer. Most of the visualization were similar to those on figure 12 which do indicate that some body-colored blobs are learned but they were not very informative beyond that.

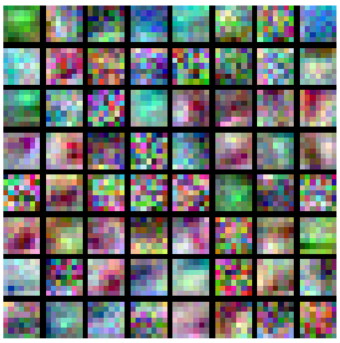


Figure 12: Weights Visualization

On the other hand, we used saliency maps in order to find the most important regions in our training images for the classification problem. As seen by figure 13 we found out as we expected that the region of the hand in the image was very important for the classification task. Interestingly we noticed that light was also extremely important (i.e. intense red regions) to the classification problem. This is the reason why we decided to generate virtual 3D training datasets where we moved around the location of the light source while keeping the hand still. We did so in order to be able to feed the network images of the same hand gesture with shadows in different directions. We expected that this change will make our network more robust to light an-

gle changes. This hypothesis was confirmed as seen by the accuracy improvements.

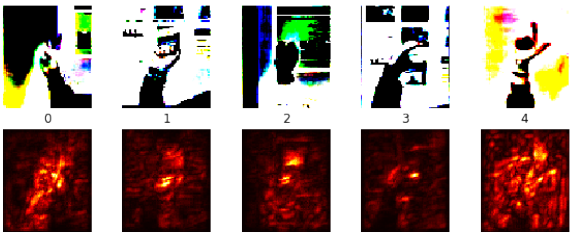


Figure 13: Raw images fed to the network and their saliency maps

This led to the idea of generating many virtual images using Unity where the hand gesture would remain still but the position of the light source will rotate around. We guessed correctly as shown in Section 6, that by using fake virtual data with a variety of light sources orientations, we were able to fool a network to perform even better on the real testing dataset.

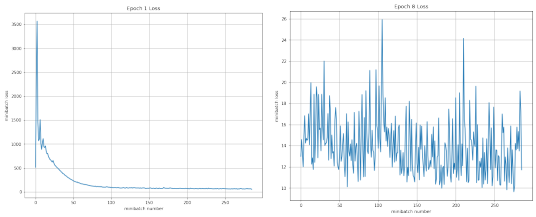
5.1.2 Hyper Parameter Tuning

The best custom network we built was a two layer convolutional network with the following architecture:

$$[Conv-Relu-BN-pool]\times 2-[FC]\times 1-[SoftmaxLoss]$$

Once we stopped exploring new custom networks we spent considerable time hyper tuning the parameters of our model. Initially we tried different optimizers including RMSProp and Momentum but we finally settled on using Adam optimizer due to its multi-dimension rate adaptability as well its ability to build momentum in gradient descent. As for our minibatch size we experimented with batch sizes between 8 and 124 input images. Small batch size led to a lot of fluctuation in our loss versus iteration plots. However, while larger batches result in smaller spikes, due to the small dataset we have the epoch is completed quite quickly making it hard to observe the loss trend over the epoch. Finally, we settled on using 32 images as our batch size to minimize spikes on our loss plot but still maintain a readable curve to inform our learning. Such loss versus iteration plots can be seen in figure 14.

Finally, given our chosen minibatch size, optimizer and architecture, we grid searched (see figure 15) for the optimal learning rate and the optimal regularization factor (note: we used L2 regularization). The optimal combination we found as seen by figure 15 was: Learning rate: 0.0009, Regularization: 0.000802. Despite the various customization we made to the network we were unable to pass the 60% testing



(a) Uniform background (b) Complex background

Figure 14: Loss vs. Iteration Plots

threshold and thus we chose to experiment with customizing pretrained networks such as VGG-16 and Inception-ResNet.

Regularization/Learning rate	0.0001	0.0002	0.0003	0.0004	0.0005	0.0006	0.0007	0.0008	0.0009	0.001
1.00E-05	0.353	0.398	0.453	0.476	**0.534**	0.39	0.403	0.479	0.497	0.445
0.000208	0.401	0.353	**0.521**	0.408	0.461	0.458	0.482	0.395	0.403	**0.529**
0.000406	0.317	0.461	0.44	**0.526**	0.487	0.351	**0.521**	0.432	0.463	0.487
0.000604	0.461	0.414	0.416	0.469	0.382	0.408	0.49	0.437	0.427	0.458
0.000802	0.403	0.38	0.33	0.466	0.353	0.411	0.458	0.461	**0.57**	0.466
0.001	0.327	0.369	0.267	**0.565**	0.416	0.429	0.466	**0.521**	**0.503**	**0.558**

Figure 15: The Unity editor window

5.2. VGG customization

We downloaded an existing VGG-16 model from TensorFlow models library to transfer learning to a new VGG model. The VGG-16 model's weights have already been trained on ImageNet, achieving accuracy of around 92% in the ImageNet Large Scale Visual Recognition Challenge.

We then replaced the last output layer with a custom fully connected layer of size [1000x6] to output the desired number of classes in our dataset. We fine-tuned the network's hyper-parameters in the following way:

- Batch size: 32
- Learning rate: 1e-3
- Dropout probability: 0.5
- Weight decay: 5e-4

After setting up these parameters, we trained only the last fully-connected output layer on 10 epochs. Next, we re-trained the full model on 10 more epochs, with the following hyper-parameters:

- Batch size: 32
- Learning rate: 1e-5
- Dropout probability: 0.5
- Weight decay: 5e-4

We trained the customized model on 3 training datasets composed of: 1) original data, 2) virtual data that we generated from 3D models, and of the same size of the original data, 3) a combined dataset containing both original and virtual data.

5.3. Inception-resnet-v2 customization

In a similar fashion we downloaded an existing Inception-resnet-v2 model from TensorFlow's models library to transfer learning to a new Inception-resnet-v2 model. The Inception-resnet-v2 model's weights have already been trained on ImageNet, achieving accuracy of around 95% in the ImageNet Large Scale Visual Recognition Challenge.

We then replaced the last output layer with a custom fully connected layer of size [2048 x 6] to output the desired number of classes in our dataset. We fine-tuned the network's hyper-parameters in the following way:

- Number of epochs = 10
- Batch size: 8
- Initial learning rate: 2e-4
- Learning rate decay = 0.7
- Number of epochs before decay = 2

After setting up these parameters, we trained the entire customized model on 3 training datasets composed of: 1) original data, 2) virtual data that we generated from 3D models, and of the same size of the original data, 3) a combined dataset containing both original and virtual data.

5.4. Results

We evaluated each of the three trained models per method on the same test datasets composed of hand gestures images with uniform and complex backgrounds separately as well as both sets combined. (see Figure 16).

5.4.1 VGG-16

We can see a large gap in accuracies (around 30%) for the VGG model trained on the real original data. This gap suggests that the original dataset was not generalized enough for capturing different image background settings. This model clearly overfits the uniform data.

The VGG model trained on the virtual data performed the worst with lowest accuracies on both uniform and complex test sets.

The VGG model trained on both real and virtual data (labeled as combined in the table) achieved lower accuracy on the uniform test set than the first model (trained on real data) did. This model, however, surpassed the accuracy of the first model by 14% on the complex set, closing the uniform-complex accuracy gap to 2%. This interesting result suggests that the new virtual dataset provides some generalization of the data, which leads to a model that better classifies complex images.

5.4.2 Inception-ResNet-v2

The Inception-ResNet-v2 models performed significantly better than the VGG models did on every train-test sets combination. Here we can see an interesting result that the models trained on virtual datasets achieved accuracies higher than the first model (trained on real dataset) by more than 10%. In fact, the highest accuracies for both uniform and complex test sets outperformed those in Marcel's paper, surpassing 96%.

In this case, we see a clear advantage for adding virtual data to our training dataset, and even completely replacing it.

Model	Train	Test	Accuracy
VGG-16	real	uniform	0.80
		complex	0.501
	virtual	uniform	0.620
		complex	0.45
	combined	uniform	0.642
		complex	0.624
Inception-ResNet-v2	real	uniform	0.8204
		complex	0.8479
	virtual	uniform	0.9382
		complex	0.9662
	combined	uniform	0.9624
		complex	0.9577

Figure 16: Accuracies per model

As we trained the Inception-ResNet-v2 models we monitored the confusion matrix to ensure that the model was not over-fitting for particular gestures but that the accuracy was relatively uniform across classes. Table 3 shows the confusion matrix for the optimal Inception-ResNet-v2 model trained with combined data on complex testing dataset.

We can see from the table that our models misclassified the gesture A (i.e. a fist) as a Point (i.e. one finger pointing). This behavior can be explained by particular backgrounds that have vertical objects behind the hand, which confuse the model to interpret them as fingers. An example for such misclassification is shown in Figure 17.

Truth/ Label	A	B	C	point	five	V
A	35	0	1	3	0	0
B	0	41	0	0	0	0
C	2	0	44	1	0	0
point	0	0	0	52	0	2
five	0	0	0	0	58	0
V	0	0	0	2	0	36

Table 3: Confusion matrix for combined model on complex dataset

Figure 17: Misclassification

6. Conclusion

As we can see from the results, 3D model-based data can significantly augment scarce datasets to improve models' accuracies. Virtual datasets generation can be controlled to compensate for ungeneralized data by applying various linear transformations, backgrounds and lighting.

Another conclusion that can be drawn from the data visualization section is that saliency maps can point at important network behaviors and potential drawbacks in the dataset. Such insights from exploring a network's saliency map can be then used to generate customized virtual data to better fit the network's behavior and help it generalize for the unseen test dataset. Overall, due to the use of virtual data, customization and fine-tuning pretrained models such as Inception-ResNet, we were able to surpass the original accuracies of 93.0% for uniform backgrounds and 84.4% for complex backgrounds and achieve instead 96.62% and 96.24% respectively.

7. Future Work

There are several experiments that can be done to farther test the advantages of virtual data augmentation. One experiment would be generating a larger amount of virtual data to see if it can improve accuracy even more.

Furthermore, given a larger dataset it would be interesting to test whether our networks can learn to distinguish between dozens of different gestures instead of only six gestures.

Finally, we would like to test virtual data augmentation on other datasets and classification tasks to see if this technique can improve other models as well.

References

[1] Frank Althoff, Rudi Lindl, Leonhard Walchshausl, and S Hoch. Robust multimodal hand-and head gesture recognition for controlling automotive infotainment systems. *VDI BERICHTE*, 1919:187, 2005.

[2] Lucas Bonansea. 3d hand gesture recognition using a zcam and an svm-smo classifier. 2009.

[3] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. Robust 3d hand pose estimation in single depth images: from single-view cnn to multi-view cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3593–3601, 2016.

[4] Ho-Joon Kim, Joseph S Lee, and Jin-Hui Park. Dynamic hand gesture recognition using a cnn model with 3d receptive fields. In *Neural Networks and Signal Processing, 2008 International Conference on*, pages 14–19. IEEE, 2008.

[5] Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen. Human hand gesture recognition using a convolution neural network. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 1038–1043. IEEE, 2014.

[6] Cristina Manresa, Javier Varona, Ramon Mas, and Francisco J Perales. Hand tracking and gesture recognition for human-computer interaction. *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, 5(3):96–104, 2005.

[7] Sbastien Marcel. Hand posture recognition in a body-face centered space. `http://www.idiap.ch/resource/gestures/papers/marcel-chi-99.pdf`, May 1999.

[8] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz. Hand gesture recognition with 3d convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–7, 2015.

[9] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*, pages 342–347. IEEE, 2011.

[10] Steven J Nowlan and John C Platt. A convolutional neural network hand tracker. *Advances in Neural Information Processing Systems*, pages 901–908, 1995.

[11] Eshed Ohn-Bar and Mohan Manubhai Trivedi. Hand gesture recognition in real time for automotive interfaces: A multimodal vision-based approach and evaluations. *IEEE transactions on intelligent transportation systems*, 15(6):2368–2377, 2014.

[12] Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen. Sign language recognition using convolutional neural networks. In *Workshop at the European Conference on Computer Vision*, pages 572–578. Springer, 2014.

[13] Toby Sharp, Cem Keskin, Duncan Robertson, Jonathan Taylor, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, Alon Vinnikov, Yichen Wei, et al. Accurate, robust, and flexible real-time hand tracking. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3633–3642. ACM, 2015.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] Ekaterini Stergiopoulou and Nikos Papamarkos. Hand gesture recognition using a neural network shape fitting technique. *Engineering Applications of Artificial Intelligence*, 22(8):1141–1158, 2009.

[16] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[17] Jonathan Tompson, Murphy Stein, Yann Lecun, and Ken Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, 33(5):169, 2014.

[18] Thomas Walentin. Simple hand rig — blend swap. `https://www.blendswap.com/blends/view/75824`, 2014. (Accessed on 06/04/2017).

[19] Deyou Xu. A neural network approach for hand gesture recognition in virtual reality driving training system of spg. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 519–522. IEEE, 2006.

[20] Xiaoming Yin and Ming Xie. Hand gesture segmentation, recognition and application. In *Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on*, pages 438–443. IEEE, 2001.