CopyCat: Real-time Hand Gesture Recognition using a Convolutional Neural Network

Julia Arnardottir
Department of Mechanical Engineering
Stanford University
julia93@stanford.edu

Solveig Einarsdottir
Department of Electrical Engineering
Stanford University
einarsd@stanford.edu

Abstract

Hand gestures are an intuitive way of communicating, independent of age and nationality. As robots of all sort become more frequent in our everyday lives, it is important to explore ways to communicate with them. In this paper, using computer vision and neural networks to classify hand gestures is studied, and the results visualized on an animated robot that performs different movements based on the predicted hand gestures. Two different model architectures, CNN and CNN-LSTM, were implemented in python and compared. The results indicate that neural networks are a promising way of carrying out this task, but we achieved 65.5% accuracy on 5 stacked 64x64 resolution RGB-images using a 2D-CNN trained on a dataset with about 292 samples of each gesture.

1 Introduction

Our world is becoming more automated every year, and robots of all sorts are becoming widely used. An increasing number of manufacturers rely on robots to stay competitive and the number of industrial robots worldwide in 2019 was estimated to become 2.6 million [1]. Collaborative solutions based on humans and robots work together are entering the market, with robots handling repetitive jobs that require speed and accuracy while humans can focus on more diverse cognitive jobs. After safety, the most important thing to ensure when it comes to human and robot collaboration is ease of interaction. There are many ways to communicate with a robot, including sending direct inputs, using speech, touch or gestures. In our project we focused on interpreting gestures through computer vision.

Sending an input to a robot traditionally requires the use of a separate interface such as a control panel. However, direct communication between human and robot, such as through speech or gestures, creates various opportunities for different kinds of collaboration. As opposed to spoken language, hand gestures are most often independent of where in the world you come from and more intuitive. Although general hand gestures don't provide means for nearly as deep communication as spoken language, it still allow us to give a variety of useful commands.

For human robot collaboration to be useful the robot needs to be able to react in real time. To save memory, the computer can store a few frames per second instead of every frame of the video, and pass those into the model. The input to our algorithm is a set of five RGB-images that can be frames from a video, or image captures from a camera. We then use either a convolutional neural network (CNN) or a combined CNN and recurrent neural network (RNN) to output a predicted hand gesture. To visualize the output, the label of the predicted hand gesture is used to give commands to an animated robot (implemented as a computer game using free character animations from Mixamo [2], and activated using PyAutoGUI [3]) displayed on the computer monitor to mimic or react to the hand gesture. For real-time classification, images are captured through a webcam (using the OpenCV library [4]) at a rate of 3 frames per second, and fed to a trained model.

2 Related Work

Video Classification: Classifying hand signals and movements is a problem many have experimented with [11], for instance for interpreting sign language [5], which in many ways is similar to our project. The main difference is the point of view, but in our case the camera is egocentrically placed. The benchmark for that sort of hand gesture recognition is the one associated with the dataset we used, EgoGesture [6], where different models were tested and a eight layer 3D-CNN model performed the best and reached a accuracy of 86.4% using 16 RGB-images and 88.1% accuracy using depth images. Despite better results for depth images, we will be focusing on RGB-images since depth cameras are not as readily available. 3D-CNNs are commonly used for video classification [7, 8], and have been found to perform well. Other methods include RNN with LSTM units [10], and other algorithms that not all were covered in this course. This task is however rarely performed by hand.

2D-Convolutions: Due to limitations of data, computational power and memory, we are more interested in experimenting with 2D-convolution and fewer frames. VGG16 is a well known 2D-CNN, and has been used both on its own [9] and combined with LSTM units to perform hand gesture recognition [6, 7]. Those sort of methods were found to have accuracies up to about 65%.

3 Dataset and Features

We used the EgoGesture Dataset [6], which contains 24,161 gesture samples from 83 classes of hand gestures. The data was collected as multiple videos, where one video shows one subject performing about 12 hand gestures in a row. Ten frames per second (fps) were saved as images, and stored in separate folders for each video. The labels for the data are documented in CSV files, where one file correspond to the images from one video. Out of the 83 hand gestures in the EgoGesture dataset, 10 are used for this project. For each gesture, there are 292 video segments and 8-30 corresponding images. Five images (using every three frame) are used to classify each gesture. The number of images used and frames per second to collect was based on visual assessment under the assumption that human performance is close to ideal. The gestures are equally distributed throughout the dataset, and 80% of the data is used for training and 20% for development.
The original image sizes are 320 x 240, but they are resized down to 64 x 64 to speed up the training. The hand gestures are clearly visible by the human eye in that size, as shown in the sample on Figure 1. No data augmentation was performed on the data, since the distortion, rotation and flipping might alter the meaning of the signs. If data augmentation was to be performed, color alternations would be the most useful in this case.

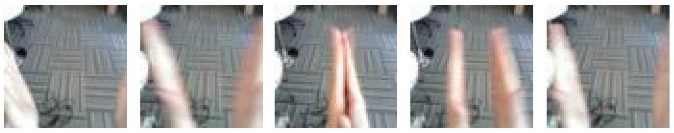


Figure 1: One set of resized images used as a sample for the class 'Applaud'.

4 Methods

Python code for both models is published on GitHub [18].

4.1 Approach 1: CNN

Using a 2D-CNN instead of 3D-CNN is more computationally inexpensive, since it requires fewer parameters to learn and since the outputted dimension is 2D instead of 3D and there are fewer computations to be performed. Hence, we wanted to explore if moving the convolution along only 2 spatial dimensions would be enough, and therefore decided to use 2D-convolution.

The stacked arrays of decoded images are passed through a 2D-convolution layers with 3x3 filters, followed by a batch normalization, a ReLU activation and a max pooling, this is repeated $N = 5$ times and number of channels is doubled in each step. Next it is flattened before it enters 2 fully connected layers ($M = 1$), and finally a softmax activation (argmax), as shown in figure 2. This network was created using Tensorflow [12] in Python, and based upon a GitHub repository from CS230 for a computer vision example [13].

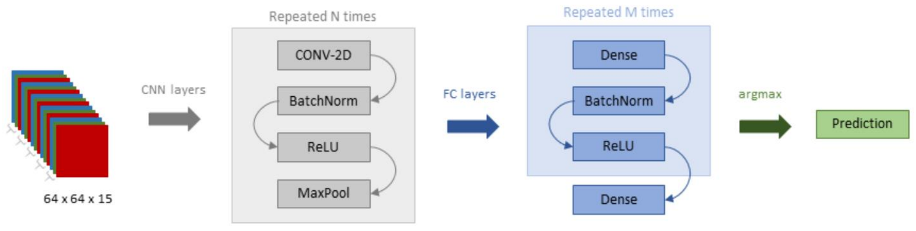


Figure 2: Structure of CNN model.

For training, the cross-entropy loss function in Eq. (1) is used along with gradient descent and the Adam optimizer in Tensorflow, with a momentum of $\beta = 0.9$.

$$L(\hat{y}, y) = -\sum_{j=1}^{c} y_j log(\hat{y}_j) \tag{1}$$

4.2 Approach 2: CNN-RNN combination

After learning about sequence models, an alternative method was proposed. Instead of stacking the images and feeding into a 2D-CNN to predict the output, the images were first passed through a 2D-CNN and the results from that were passed into a recurrent neural network (RNN) as shown in figure 3. For the CNN model, the convolution layers are repeated $N = 8$ times and only some of those include max pooling. For the many-to-many RNN model, a single LSTM block with $M = 256$ units was used. This network was created using Keras [14] in Python, and based upon a GitHub repository showing five different methods for classifying videos [15]. For this architecture, the Adam optimizer and cross-entropy loss were used as before to train the model, but now with Keras. For the optimizer, a learning rate decay of 10^{-6} was used along with the typical momentum values $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Dropout and early stopping were added for regularization effects and avoid overfitting.

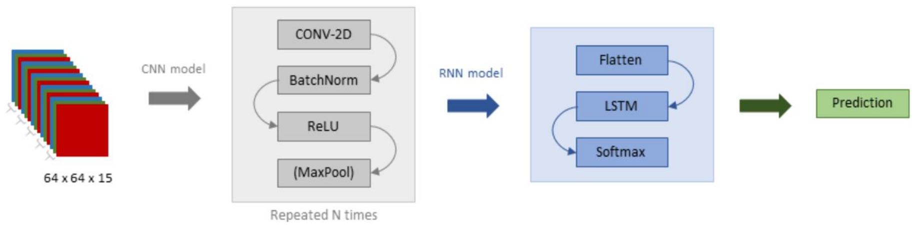


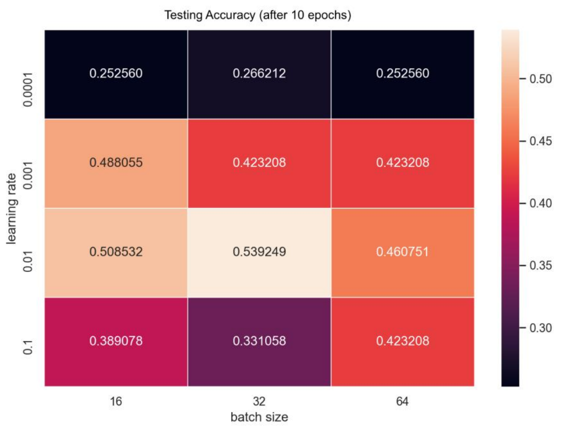
Figure 3: Structure of CNN-RNN model.

5 Experiments/Results/Discussion

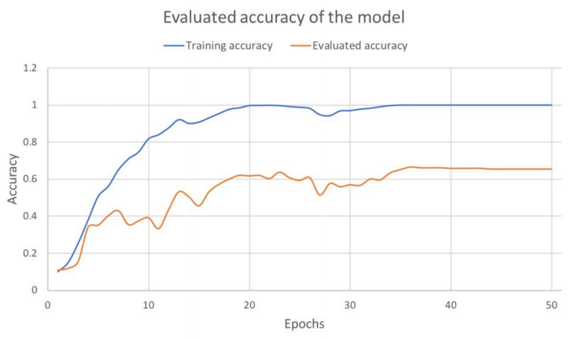
The primary metric used to evaluate the performance of the models is accuracy. The dataset is balanced, and based on the use case, accuracy is determined to be an appropriate metric.

5.1 Approach 1: CNN

The two main hyperparameters that were tuned, and most significantly impacted the results, were learning rate and batch size. Figure 4a shows the values tried and the resulting accuracy after 10 epochs. Moving forward a learning rate of $\alpha = 0.01$ and batch size of 32 were chosen. Other hyperparameters that were explored include number epochs, number of convolution layers, N, and number of fully connected layers, M. Figure 4b shows how the accuracy develops with epochs. After about 40 epochs, the model stops improving, and a development set accuracy of 0.655 was reached. By the difference in training set and development set accuracy, the model appears to be overfitting to the training data.



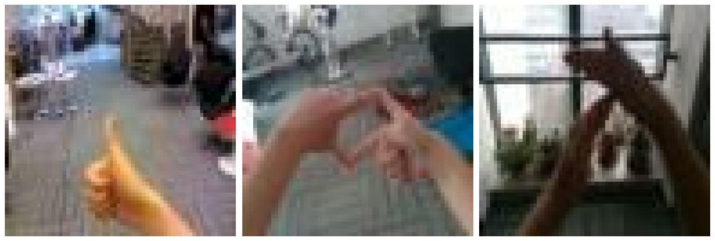
(a) Heatmap



(b) Accuracy

Figure 4: Performance of 2D-CNN.

Error analysis: The incorrectly labeled samples were explored, but no noticeable trend was detected. A similar number of each class had a wrong prediction, and the images have different brightness, colors, and so on. Three examples of incorrectly labeled samples are shown in figure 5.



(a) Thumbs up (b) Frame (c) Pause

Figure 5: Examples of incorrectly labeled samples (center image out of 5 stacked images shown).

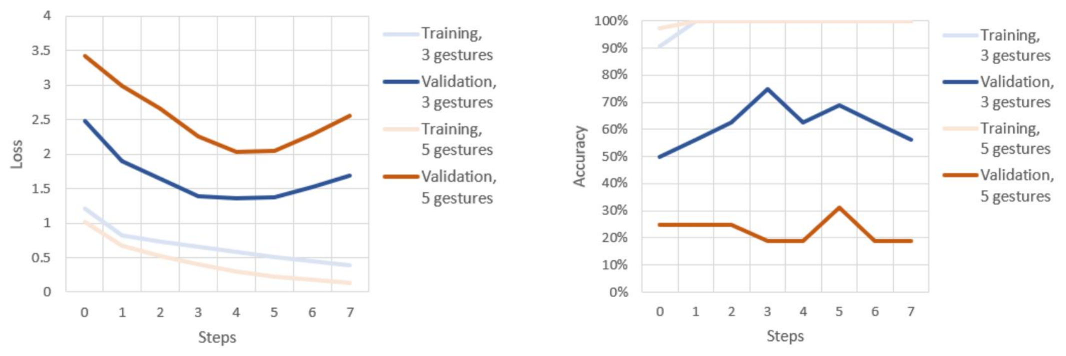
5.2 Approach 2: CNN-RNN combination

Training over a maximum of 12 epochs was deemed sufficient, since the loss had almost always stopped decreasing by then. Early stopping with a patience of three epochs without improvement in loss was used. In the LSTM layer, 50% dropout was incorporated to help prevent the model from overfitting to the training data. Based on researches about how learning rate decay might slow down training [16], we explored putting the learning rate decay to zero. However, we found that in our case this resulted in faster improvement in accuracy to begin with but significantly worse accuracy by the end of training. Hence, learning rate decay was set to a small non-zero value. This was explored for a few different learning rates, but to accommodate zero learning rate decay, decreasing the learning rate helps since it reduces the risk of divergence. Furthermore, based on computational power and memory, we limited our batch size to a maximum of 16. For learning rate, number of convolution layers, and number of hidden units in the LSTM layer, hyperparameter tuning was performed for the different combinations of these parameters since they are a significant part of the model. The optimal values, based on development set accuracy, are listed in table 1. Learning rate turned out to be the parameter that the model was most sensitive to.

Table 1: Optimal hyperparameters for CNN-LSTM architecture.

Learning rate	No. Convolution layers	No. LSTM hidden units
0.001	8	256

The performance of the neural network was much better when training on fewer gestures, but fails to generalize as more gestures were added. The difference in performance for 3 and 5 gestures (4 and 6 classes) is shown in figure 6. Decreasing cross entropy loss does not imply an increase in accuracy [17], as is visible through the results.



(a) Loss (b) Accuracy

Figure 6: Performance of NN when classifying 3 or 5 gestures.

6 Conclusion/Future Work

Since hand gestures, like the ones in our dataset, were formed by humans beings to relay information to other human beings it goes without saying that humans are very good at identifying them and the Bayes optimal error is therefore very close to zero. For the 2D-CNN model we can see from figure 4b that the training error is close to 0% but our evaluated error is over 30% for 11 classes, and for the CNN-RNN model we can see that from figure 6 the training error is also about 0% whereas the evaluated error is over 70% for only six classes. We thus have a variance problem in both our models, and more so for the CNN-RNN model. This is most likely due to overfitting to our dataset since it is quite small. The error could possibly be reduced by using additional regularization techniques, like data augmentation. More data in general would also be ideal. Increasing the resolution might also help improve the performance, but that would be more computationally expensive and require more memory. For the RNN-CNN model, the network fails to generalize as more classes are added, but better performance might be achievable for more computational power.

The results indicate that using 2D-CNN is a promising method for predicting hand gestures, and accuracy of about 65% is computationally inexpensive to obtain.

7 Contributions

Julia worked on programs for collecting image data and labels from the dataset, building the models (the CNN model and the CNN-RNN model), and programs that allow for real-time classification of hand gestures through a webcam. Hyperparameter tuning for the CNN-RNN model.

Solveig Asta worked on a program to run hyperparameter searches and for creating confusion matrices, and performed the hyperparameter search and other experiments on the models.

Both participated in writing the report.

Additional, we would like to thank our CS230 project mentor Patrick Cho for the useful discussions and advise regarding the project.

References

[1] International Federation of Robotics (IFR). Executive summary world robotics 2016 industrial robots. Technical Report, 2016.

[2] Adobe Systems Incorporated. Animated 3D characters. On `https://www.mixamo.com`, 2019.

[3] Python Software Foundation (PSF). PyAutoGUI package. Accessed at `https://pypi.org/project/PyAutoGUI/` on 6/7/2019

[4] G. Bradski. The OpenCV Library. Published in *Dr. Dobb's Journal of Software Tools*, 2008.

[5] H. Huang, W. Zhou, Q. Zhang, H. Li, W. Li. Video-Based Sign Language Recognition without Temporal Segmentation. *AAAI Conference*, 2018.

[6] Y. Zhang, C. Cao, J. Cheng and H. Lu, EgoGesture: A New Dataset and Benchmark for Egocentric Hand Gesture Recognition. In *IEEE Transactions on Multimedia (T-MM),Vol.20, No.5,* pp.1038-1050, 2018.

[7] O. K$\ddot{o}p\ddot{u}$kl$\ddot{u}$, A. Gunduz, N. Kose and G. Rigoll. Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks. *Institute for Human-Machine Communication*, TU Munich, Germany. 2019.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. *CVPR*, 2014.

[9] G. Strezoski, D. Stojanovski, I. Dimitrovski, G. Madjarov. Hand Gesture Recognition using Deep Convolutional Neural Networks. *ICT Innovations Conference Paper*, 2016.

[10] A. Sarkar, A. Gepperth, U. Handmann, and T. Kopinski. Dynamic Hand Gesture Recognition for Mobile Systems Using Deep LSTM. In *International Conference on Intelligent Human Computer Interaction* pp. 19-31. Springer, Cham. 2017.

[11] A. Fatale and N. Arkalgud. Recognizing Hand Gestures. Syracuse University, 2019.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] O. Moindrot and G. Genthial. Hand Signs Recognition with Tensorflow. Published at `https://github.com/cs230-stanford/cs230-code-examples/tree/master/tensorflow/vision`, 2018.

[14] F. Chollet and others. Keras. Published at `https://keras.io`, 2015.

[15] M. Harvey. Five video classification methods. Published on *GitHub*, 2017.

[16] S.L. Smith, PJ. Kindermans, C. Ying, and Q.V. Le (from Google). Don't decay the learning rate, increase the batch size. Published as a *conference paper at ICLR*, 2018.

[17] J. Huotari. Why Loss and Accuracy Metrics Conflict? Posted on *Jussi Huotari's Web*, 2018.

[18] J. Arnardottir and S.A. Einarsdottir. Copycat Project. `https://github.com/juliaar/CS230_Final`, 2019.