Hand Gesture Recognition

Group 7
Arshia Zafari, Erik Seetao, Joni De Guzman, Hayk Hovhannisyan
A11167578, A10705834, A53212113, A12466074
{azafari, eseetao, j5deguzm, hhovhann}@ucsd.edu

***Abstract*— Gesture recognition is an active field of research which has a wide range of applications including hand gesture recognition, human machine interaction, and immersive game technology. From the Leap Motion Hand Gesture Recognition dataset comprised of 20,000 images of ten distinct hand gestures, we train a 2D convolutional neural network to classify these hand gestures. Our model is able to achieve a success rate of 90.2%.**

I. INTRODUCTION

Given a collection of hand gesture training images, we design a classification model that learns to distinguish between the different gestures when given new images. This problem has widespread application in areas such as ASL translation, virtual gaming, and driver/pedestrian hand recognition for safe and smart driving. As of recently, deep learning has seen tremendous growth in different computer vision and machine learning applications from image classification and segmentation to object detection and recognition. Therefore, we decided to apply deep learning methods to the problem of hand gesture recognition. We implement a convolutional neural network that takes in images of 10 different hand gestures and outputs the predicted gesture type. Although our model is more simple in that it learns to recognize only 10 hand gestures, it can easily be scaled up to learn a larger number of hand gestures and signs.

II. RELATED WORK

A hand gesture recognition system that utilizes depth and intensity channels is fed into a 3D CNN (convolutional neural network) in [1]. Their 3D CNN consists of two parallel networks that are capable of operating on both high-resolution and low-resolution inputs. The two subnetworks learn different features, and their outputs are merged to classify an input into one of 19 gestures. A 3D CNN functions the mostly same as a 2D CNN, but it allows for spatial and temporal data where the third dimension of time is absent in a 2D CNN. This model is a more sophisticated version of our model as we simplify ours to use only spatial data. This work also mentions the use of heavy data augmentation to improve the robustness of their network. We draw ideas from their data augmentation when implementing ours on our data.

In [2], a multi-layered random forest model is used to recognize and classify sign language signals from depth images. The data undergoes augmentation to make the features scale, translation, and rotation invariant since a hand sign can have a very different appearance when seen from a different view. The use of a multi-layered random forest aids in discriminating between hand signs that look very similar, but the authors mention the data augmentation being more of a contributing factor in improving the results significantly.

On the other hand, [3] combines a CNN with a sliding window approach to perform hand gesture recognition in real-time on raw video streams. Their network consists of two separate models: one is a CNN to detect the presence of a gesture and the second is a CNN classifier to classify the detected gestures. This work is different than other works as it operates on dynamic data rather than static images. Our approach uses static images in the form of individual still images of gestures rather than a full sequence. Since we do not use dynamic data, our network does not need to a second model to detect the presence of a gesture.

Rather than using neural networks, [4] design handcrafted spatio-temporal features with a traditional SVM (support vector machine) classifier to perform hand gesture recognition in an in-vehicle setting. Due to the recent success of deep learning, using a traditional approach is outdated as deep learning models continue to report higher performance and accuracy for gesture recognition.

Most similar to our work is [5], which also uses a 2D CNN to implement hand gesture recognition. As opposed to using a readily available dataset, this implementation creates their own dataset of gestures. This work offers an even simpler CNN model than ours, yet it still retains good performance. It differs from our model in that we build a deeper model with more convolutions. A deeper neural network allows for learning more higher level features, so we decide to include more layers in our model.

III. DATASET

We use the Hand Gesture Recognition Database from Kaggle[1]. The database contains 20,000 infrared images (of size 640×240) of hand gestures captured by a Leap Motion sensor. The images are split into 10 distinct gestures from 10 different users, comprised of 5 males and 5 females. All hand gesture images are taken with the right hand. The 10 gestures are: palm, L, fist, fist side, thumb, index, OK, palm side, C, and down. Figure 1 shows samples of the different gestures in the dataset.

[1] https://www.kaggle.com/gti-upm/leapgestrecog

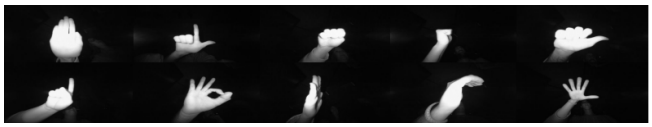

Fig. 1. Top row: palm, L, fist, fist side, thumb. Bottom row: index, OK, palm side, C, down

A. Preprocessing & Data Augmentation

For preprocessing, all images are converted to grayscale, normalized, centered, cropped, and downsampled to square images of size 120×120. This is accomplished by first employing `findContours()` via OpenCV to perform a blob detection on the hand in each image like in Figure 2. For each hand blob, we find the center of the hand and crop it to a square 240×240. If an image has a hand too far to the left or right that cropping goes out of bounds, the image is zero padded and then cropped. Finally, the image is downsampled by a factor of 2 to yield a size of 120×120.

Fig. 2. findContours() used to find hand center

Due to the limited amount of images in the dataset, we apply further augmentation by mirroring and rotating the images, effectively doubling our dataset to 40,000 samples. In doing so, we ensure right and left hand representation which also aids in improving the robustness of our model. Figure 3 below gives an example of a mirrored image and Figure 4 gives a rotated mirror of 15 degrees counterclockwise.

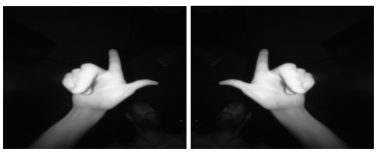

Fig. 3. Mirrored image of 'L' gesture

Fig. 4. Rotated image of 'palm' gesture

It is important for the dataset to undergo enough augmentation in order for the images provide a consistent and clear amount of information, as well as am adequate representation of different perspectives. This ensures that our model performs better in varying circumstances and is capable of distinguishing a larger variety of images [6].

IV. METHODS

We exploit deep learning techniques for our hand gesture recognition task. For our model, we build a 2D CNN with dilated convolutions, batch normalization, pooling, dropout, and fully-connected layers. The model is built using the Keras v.2.1.6 back-end of Tensorflow.

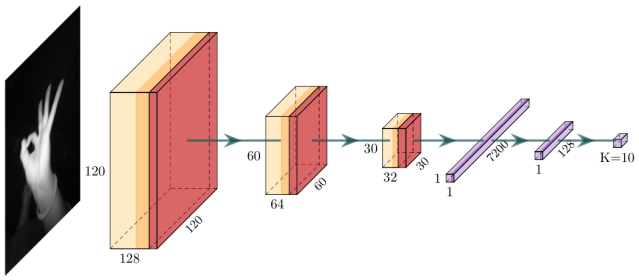

Fig. 5. 2D CNN network architecture [7]

A. Network Architecture

Our architecture consists of 3 dilated convolutional layers with ReLU (rectified linear unit) activation, each followed by additive Gaussian noise, batch normalization, and pooling layers. This results in a pyramid-like architecture as the pooling layers decrease the spatial resolution of the inputs to half their size. Following the pyramid architecture, we add 2 fully-connected layers with 7200 and 128 units, respectively. The final layer of the model is fully-connected with 10 units and a softmax activation to appropriately classify the input image into one of the 10 hand gestures. A diagram of the described model is shown in Figure 5.

We go into more detail about each of the different layers in our network below.

- **Dilated convolutions:** The three convolutional layers use 128, 64, and 32 filters, respectively. All convolutional layers use a 3×3 kernel, a stride of 1, and a dilation rate of 2. We use dilated convolutions because increasing the dilation rate effectively increases the receptive fields of the output. What this means is that our 3×3 kernel functions the same as a 7×7 kernel but without the same number of parameters as a 7×7 kernel. This results in an increased receptive field meaning and less computation. Figure 6 shows an illustration of a 2 dilated convolution that we utilize in our model.
- **Gaussian noise:** Layers of additive Gaussian noise follow the convolutional layers to increase robustness of the network. Since pixel values of the image were a factor of 10^{-2}, we chose Gaussian noise with variance of 0.0035.

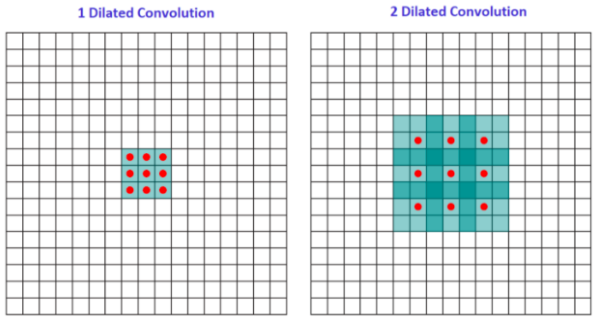

Fig. 6. Dilated convolution of a 3x3 kernel with a dilation rate of 2

- **Batch normalization:** Batch normalization is added between convolutional and pooling layers to help prevent the model from overfitting and to allow better generalizability of the network.
- **Pooling:** Pooling layers reduce the network's output size by downsampling the inputs. Specifically, we use max pooling which takes the maximum value about a 2×2 window.
- **Dropout:** A dropout layer (with keep_ratio = 0.5) is applied before the last fully-connected layer, so during training 50% of the nodes are dropped from the network. This thereby prevents certain nodes from influencing the output and allows the network to learn more robust and specific features.
- **Fully-connected:** As mentioned previously, we include 3 fully-connected layers with the following number of output units: 7200, 128, and 10.
- **Softmax:** Softmax is used as the activation for the last fully-connected layer since it is the primary activation for classification with more than two classes.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- **ReLU:** ReLU is nowadays the standard non-linear activation function for neural networks because it avoids the vanishing gradient problem. The ReLU equation and its gradient is defined below.

$$f(x) = ReLU(x) = max(x, 0)$$

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & otherwise \end{cases}$$

V. EXPERIMENTAL SETTING

For our experiments, we employ a training, validation, and testing split of 60%, 20%, and 20% respectively. This results in 32,000 images for training, 4,000 for validation, and 4,000 for testing.

A. Training

For training our model, we choose the hyperparameters specified in Table I, where the training structure follows a similar implementation to [8]. The original images were downsampled to 120×120 to accelerate computation time, and an initial downsampling by a factor of 2 does not result in much loss of features. The RMSProp optimizer is chosen since it is a fast and popular optimizer that performs well on large datasets. The batch size we use is 64. Using a small size, we are able to regularize the training and have a lower generalization error. The model uses a standard learning rate of 0.01 to prevent the gradient from descending too slow or failing to converge.

num_classes	10
num_epochs	25
batch_size	64
image_size	120 ×120
learning_rate	0.01
optimizer	RMSProp
criterion	Cross Entropy
weight_initialization	Xavier

TABLE I
MODEL SPECIFICATIONS

We can visualize the outputs of the trained network to try to explain how it is manipulating the pixel data in order to extract features used to classify them [9]. Looking at Figure 7 we see the output of the activation layer of the first convolution layer and see an example of a sideways fist gesture representation after being passed through the network. We also analyze deeper and view the same image representation after the last activation of the third convolution layer in Figure 8.

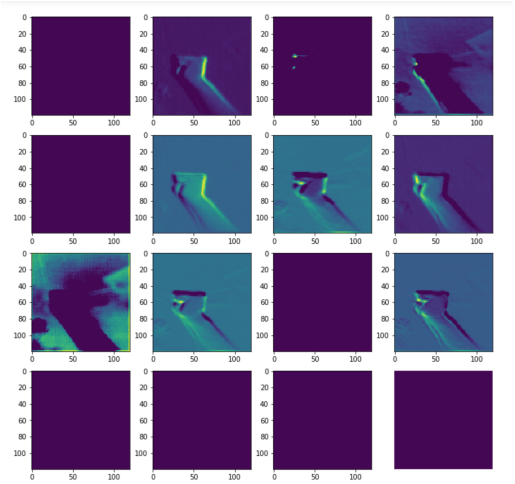

Fig. 7. Output of layer 3

Loss: Since our task is essentially multi-class classification, we appropriately use cross-entropy loss as the loss function.

$$L = -\sum_{x,d \in T} \sum_{k} d_k log(y_k) \quad (1)$$

B. Testing

The test images had equal representation of all 10 gesture classes as seen in Figure 9. Using the three main evaluation

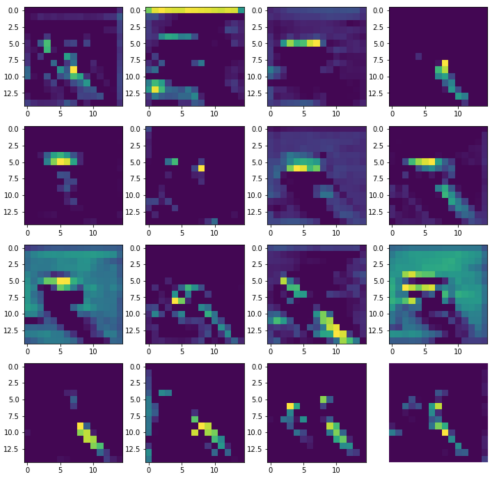

Fig. 8. Output of layer 12

metrics that we chose (explained in the next section), we predicted the labels of these test images and compared them to the true labels. With this, we were able to count the number of successful and failure cases.

Fig. 9. Frequency of test samples

C. Evaluation Metrics

In order to evaluate the performance of our model, our primary metric is accuracy which is the percentage of correct classifications.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

In addition, we evaluate the total accuracy across the test data with top-2 and top-3 accuracy. Top-2 accuracy illustrates the accuracy of our predicted label being within the top two sets of guesses among the 10 possible classes an image could be. The same works for top-3 where the prediction is within the top three guesses.

Secondary metrics used are precision, recall, and f1-score. Precision is the fraction of positive predictions that are correctly guessed divided by the total number of positive predictions be it correct or incorrect. Whereas recall is the fraction of correctly guessed positive predictions divided by the total amount of actual positive values. F1-score is the harmonic average of the precision and recall, where an F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

VI. Results & Discussion

A. Results

The results show a wide set of conclusions. In general, the model performs quite well. The test accuracy came out to be 90.2%, the top-2 accuracy was 97.7%, and the top-3 accuracy was 99.7% shown in Table II.

Test loss	1.21636
Test accuracy	90.2%
Top-2 accuracy	97.7%
Top-3 accuracy	99.7%
Correctly labelled	3,607

TABLE II
Overall Test Results

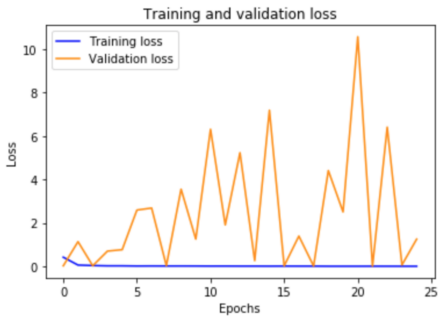

Fig. 10. Training results

Out of the 4,000 test images, 3,607 images were correctly classified. Figure 11 shows examples of the correctly classified images, whereas Figure 12 shows examples of the incorrectly classified images. The normalized confusion matrix in Figure 13 shows how most each class prediction aligns with the true prediction except for the fist and index gestures. The classification report in 14 show all secondary evaluation metrics for the various classes, showing the support for each.

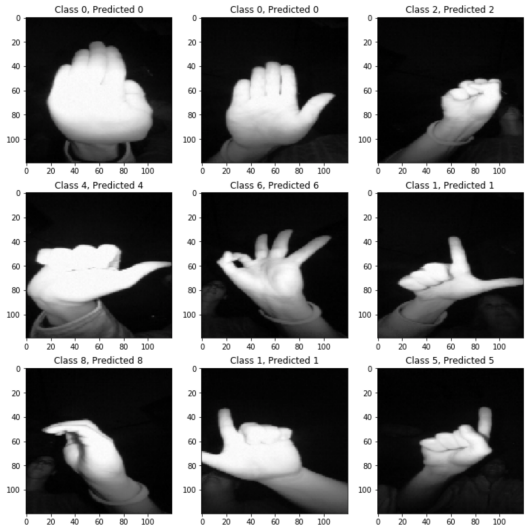

Fig. 11. Success cases where our network correctly labelled images

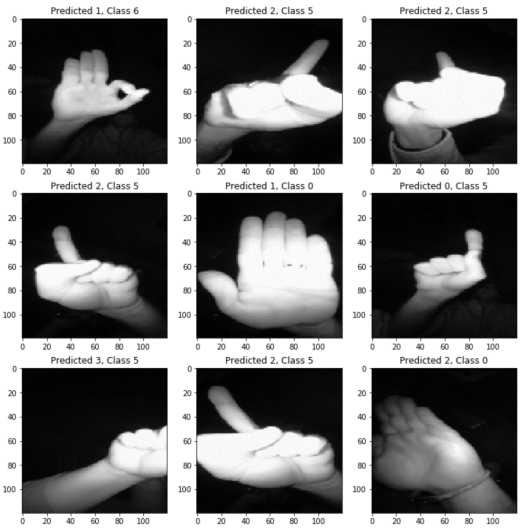

Fig. 12. Failure cases where our network incorrectly labelled images

B. Discussion

The training results from Figure 10 show the validation loss fluctuating while the training loss decreases and remains very small. This is most likely an indication of the model overfitting on the training data. Simply put, overfitting occurs when your model has learned to fit the specifics of the training set and fails to generalize well when presented with unseen data. This may affect validation accuracy more than training accuracy because at training time, the small changes made to the parameters are specifically designed to optimize a training objective which is a good proxy for training accuracy; so the effect is less like a random perturbation of the parameters, more something pushing classifications for training examples in a consistent direction. For the validation samples, because the update was not computed with them in mind, the way it acts is closer to a random perturbation. This could be due to the small size of the validation set, so

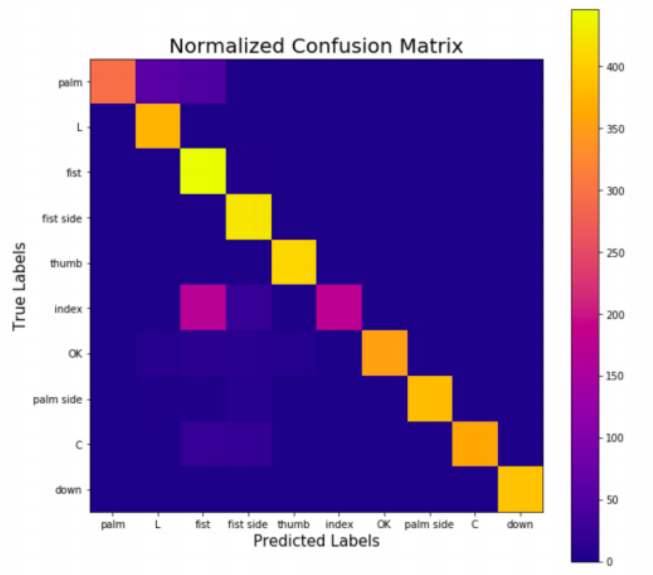

Fig. 13. Confusion matrix for our model. The x-axis represents the predicted labels and the y-axis represents the true labels for the test images. The format is as such: if you predicted A when the actual label was B, you get an entry in the (A,B) cell of the matrix.

	precision	recall	f1-score	support
palm	1.00	0.74	0.85	399
L	0.85	1.00	0.92	374
fist	0.64	1.00	0.78	448
fist side	0.87	1.00	0.93	423
thumb	0.99	1.00	0.99	406
index	1.00	0.48	0.65	376
OK	1.00	0.91	0.95	387
palm side	1.00	0.97	0.98	395
C	1.00	0.90	0.94	402
down	1.00	1.00	1.00	390
micro avg	0.90	0.90	0.90	4000
macro avg	0.93	0.90	0.90	4000
weighted avg	0.93	0.90	0.90	4000

Fig. 14. Classification report

increasing the size can reduce the variance of the validation accuracy changes that result from parameter changes because they are averaged over more samples.

The outputs of the activation layers are hard to interpret. Looking at an example of the sideways fist hand gesture as seen by the network, we can try to understand the types of features the network tries to extract at a varying levels of the architecture. Figure 7 shows the output of the image through the different filters of the first convolutional layer. The network does well in extracting edges (i.e. low level features) as the hand gesture becomes more distinct against the background. As we move deeper into the network we can analyze the output of the same image through the third and last convolutional layer, shown in Figure 8. The image becomes more pixelated, showing the various shapes and higher level features the network is trying to extract.

Analyzing the model's performance on the test set allows us to obtain a better understanding of the results. Out of the 4,000 test images, 3,607 were correctly classified, corresponding to 90.2% accuracy. Both the top-2 and top-3 accuracy are quite high. Studying the confusion matrix in Figure 13, we see that most images are correctly classified

as the predicted labels and true labels are aligned across the diagonal. However, many images from the index class were incorrectly labeled as a fist. We can see such examples in Figure 12 with many images of the index gesture (class 5) being predicted as a fist (class 2). The reasoning behind this is that the gesture for an index figure is essentially a fist with a single finger raised, which makes the fist a subset of the index gesture. For many images of the index gesture, the index finger is difficult to make out and is not as prominent as the fist. This resulted in many misclassifications and is also reflected in the classification report of Figure 14. The gesture that had the best accuracy was the down gesture, with 100% precision, recall and F1-score.

VII. CONCLUSION & FUTURE WORK

Training the network on the original dataset gave almost perfect accuracy since there was not any drastic variability between the image samples, which is why layers of noise were added. Although this decreased accuracy, this made the model more robust to different gesture perspectives. However, the model still experienced some overfitting during training. The variance in the validation loss is also likely due to the small batch size since there are less images to compute the error for each epoch. Misclassifications of the index gesture as a fist is likely due to the fact that the index is essentially the fist gesture with one finger raised, so the single finger can be hard to recognize. The layer visualizations are difficult to interpret but it give us insight on the features that the network is trying to learn, like gradients and shapes.

For future work, we plan to expand our model to incorporate several image sequences to allow short videos of hand gestures. More complex 3D convolutional network architectures enables both spatial and temporal data as input. By implementing 3D CNN models as a future work, this will allow us to translate and recognize live motion of dynamically changing gestures.

CONTRIBUTIONS

- Arshia Zafari (team lead): Wrote notebook for training, testing, evaluation, displaying results, the DEMO code, as well as the analysis and other sections of the report. Also made the mid-quarter PowerPoint and designed the poster. Organized regular group meetings.
- Erik Seetao: Created data loader to read from dataset, hand blob localizer, data augmentation with mirrored and rotated images for supplement, downsampling and visualization segments.
- Joni De Guzman: Worked on the network architecture which included implementation of the dilated convolutions; wrote the related work and methods section of the report, as well as other sections; and maintain the Github repository of our code.
- Hayk Hovhannsiyan: Worked on data pre-processing used for training and testing of the model, including scripts for cropping, re-scaling, transferring and loading the data. Created data for DEMO, and took part of regular group discussions and interpretation of results.

REFERENCES

[1] P. Molchanov, S. Gupta, K. Kim, and J. Kautz. Hand gesture recognition with 3d convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–7, 2015.

[2] A. Kuznetsova, L. Leal-Taix, and B. Rosenhahn. Real-time sign language recognition using a consumer depth camera. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 83–90, 2013.

[3] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, and Gerhard Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. *CoRR*, abs/1901.10323, 2019.

[4] E. Ohn-Bar and M. M. 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.

[5] Abhishek Signh. Cnn gesture recognizer: Cnn gesture recognizer, 2017.

[6]

[7] W. Ji L. Liebel H. Iqbal, P. Fernandez. Plotneuralnet, 2019.

[8] A. Sharma. Autoencoder as a classifier using fashion-mnist dataset, 2018.

[9] A. Kumar. Understanding cnn with keras, 2018.