CS230 - Winter 2019 Doodle Recognition Challenge

Arsh Buch, Sophia Chen, John Yu
Department of Electrical Engineering
Stanford University
{arshbuch, schen10, johnyu2}@stanford.edu

Abstract

In this paper, we present a thorough analysis of the performance of traditional algorithms on human generated images. Specifically, we perform classification from images in the Quick, Draw! Doodle Recognition Kaggle Competition. We use models such as VGG-16, MobileNet, and Residual Networks for inference on these grayscale images and obtain an accuracy of 71.8%, which is comparable to state-of-the-art given our resource constraints.

1 Introduction

In class, we have mostly used neural networks on rendered images for image recognition, but we have not seen how these models perform on human drawings. Though we are only looking at doodles, this problem is extremely important for applications where human input reading can be automated such as cashing electronic checks or essay reading for standardized tests.

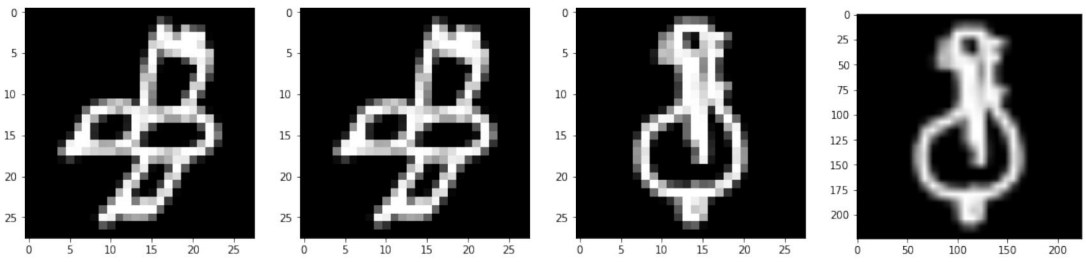
We are performing the Quick, Draw! Doodle Recognition Challenge posted by Google AI through Kaggle. The input to our models is a series of grayscale brush strokes (points) that form a single image. We use CNNs since they do well with images. Specifically, we use VGG-16, ResNet and MobileNet to classify the images into their classes.

2 Related work

Sketch recognition has been an area of research for a few years. One of the more recent papers by Sarvadevabhatla and Babu uses an ImageNet CNN and a LeNet on a freehand sketch dataset that contains 250 categories [1]. Their results indicate that the deeper ImageNet CNN performs better, with an accuracy of around 80%. Not much work has been done with deeper networks on sketch recognition, which is what our paper will focus on.

For our project, we focused on using CNNs to classify the sketches. Simonyan and Zisserman developed the first instance of a very deep network, VGG. Their network was able to achieve small errors on ImageNet with only 19 layers [2]. Residual networks, or ResNets, were also applied on ImageNet, which was the first instance of a deep network that had more than a hundred layers and outperformed previous CNNs like VGG on ImageNet classification [3]. MobileNet was originally developed for image inference on mobile phones [4]. We will use each of the aforementioned models to classify our dataset, but not without a few modifications. For a ResNet using a two-layer residual block, they found that introducing a dropout layer in between the convolutional layers actually improved accuracy and reduced overfitting [5], which we adopted for our residual networks.

3 Dataset and Features



(a) Bird (28 x 28) (b) Bird (224 x 224) (c) Cello (28 x 28) (d) Cello (224 x 224)

Figure 1: Sample images from dataset

For our dataset, we used the Quick, Draw! Doodle Recognition Challenge data set from Kaggle [6]. The data set contains the brush strokes for 50 million drawings over 347 categories (Figure 1). We trained on 100 categories because we found using all

347 categories to be too computationally intensive for the timeline of this project. Although the data comes in brush strokes, there is a bitmap representation that we converted into a 28x28 grayscale image (one channel). Since we wanted to run on larger CNNs, we needed to increase the resolution of the images so that the convolutional layers do not decrease the size too much. To do so, we used OpenCV bilinear interpolation to upsample images into 224x224 grayscale images for VGG and 128x128 grayscale images for MobileNet. We then stacked the image three times to form a 224x224x3 image (replicating across the RGB channels) for ResNet. We filtered out any images that were not recognized, since those images generally did not depict anything close to the category that it appeared in. For each category, we chose 200 training images, 50 validation images, and 50 test images.

4 Methods

For all CNNs, we used cross-entropy loss since we have multiple classes and we output the probabilities of each class (Figure 2).

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 2: Cross-entropy loss

4.1 Custom CNN

We started by using the custom CNN from our milestone to obtain a baseline result to compare further attempts against. The architecture is based on LeNet, which was used on the MNIST dataset [7]. LeNet is a five-layer CNN, but to make it slightly larger, we adopted some convolutional filter techniques of a VGG neural network [2]. We also noticed that our CNN had some variance (overfitting on the training set), so we introduced some regularization in the form of dropout layers. Our final structure is shown in Table 1. The custom CNN was run with the original 28x28 images and 100 categories. Since it is a relatively small net, the custom CNN was very quick to train so we let it train for 100 epochs.

input (28 x 28 grayscale image)
conv3-32 conv3-32
maxpool
dropout
conv3-64 conv3-64
maxpool
dropout
FC-256
dropout
FC-5 or FC-10

Table 1: Custom CNN for our data

4.2 VGG-16

We then chose to experiment with using VGG-16 which uses 16 weighted layers with many small 3x3 filters [2]. The full architecture is shown in Figure 3. The use of 3x3 filters allows for networks to be deepened while reducing parameters and introducing non-linearities. We implemented each layer individually to gain intuition on where to make changes in the network. This allowed us to freely change the input layer to accept grayscale images and change the output softmax layers to the amount of categories tested. In our implementation, we chose to use stochastic gradient descent as our optimizer. VGG-16 turned out to be the slowest network to train, so there was less hyperparameter tuning than the other networks.

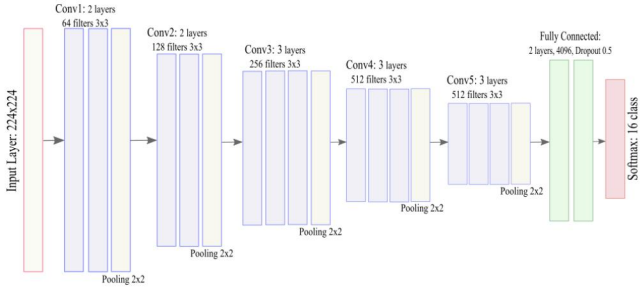


Figure 3: Architecture for VGG-16 network. Figure from Das et al. [8]

4.3 Residual Networks

We hoped that a complex network like a residual network could classify the images better using deeper connections. Residual networks were the first large, deep networks used for classification [3]. Although deeper models are harder to optimize, the ResNet should ideally perform as well as a shallower net due to its residual connections. We started off with transfer learning, using a ResNet-50 pre-trained on ImageNet [10]. ResNet-50 consists of 50 3-layer deep residual blocks (Figure 5). By using transfer learning, we did not have to train the entire ResNet-50 from scratch, which would have taken longer. However, we could not get good results from the pre-trained weights, since our images were grayscale and the pre-trained weights were for color images. Thus, we decided to try to train it from scratch. Our first model was a ResNet-50v2, which uses a new Residual Unit proposed by He et al [9]. This new unit allows deeper networks to generalize better by pre-activating the weight layers (Figure 4).

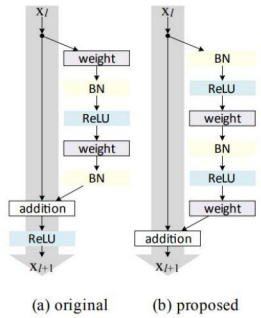


Figure 4: (a) Conventional residual unit (b) v2 residual unit with pre-activation [9]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 5: Architectures for various sizes of residual networks. In our project, we used the 34-layer (ResNet-34) and 50-layer (ResNet-50) networks. Figure from Kaiming et al. [3]

Since ResNet-50v2 tended to overfit, we decided to try some smaller residual networks. ResNet-34 is a smaller residual network that also utilizes the v2 residual blocks but has less layers of the blocks (Figure 5). We trained it from scratch since there was no pre-trained version [11].

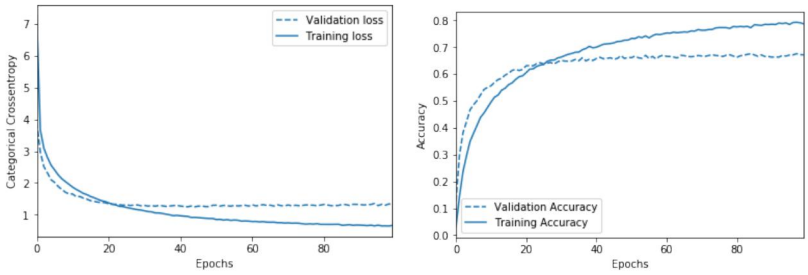
4.4 MobileNet

We ran a MobileNet model with a softmax classification layer and 128x128 grayscale images as the input. We also used Adam optimizer with a learning rate of 0.02 to train. MobileNet is a very computationally efficient algorithm (~36 seconds per epoch as compared to 200 seconds for VGG-16 and about 80 seconds for ResNet50) meant to run on mobile platforms [4]. To achieve efficiency, MobileNet uses depth-wise convolution which applies a single filter per input channel followed by a 1x1 convolution. Since MobileNet used BatchNorm and ReLU in between convolutional layers and has no dropout, we ran into an overfitting issue during training.

5 Experiments/Results/Discussion

5.1 Custom CNN

For the baseline we created our own small custom CNN. The results are shown in Figure 6. The network was able to achieve a training accuracy of 78.7%, validation accuracy of 67.1%, and a test accuracy of 67.5%. The CNN overfit by a small amount, especially starting around epoch 20, but by utilizing early stopping, the variance can be minimized. The custom CNN is a rather small network, especially in comparison with the following networks, and was able to achieve a relatively good result. This indicates that shallower networks may perform better, due to many images being similar to each other.



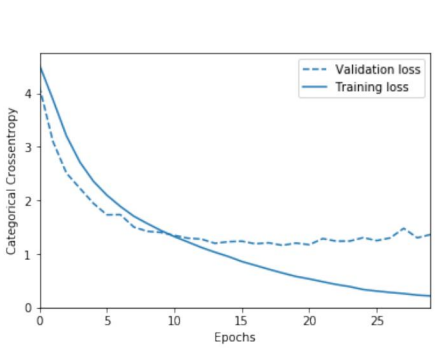
(a) Custom CNN loss (b) Custom CNN accuracy

Figure 6: Custom CNN performance over epochs

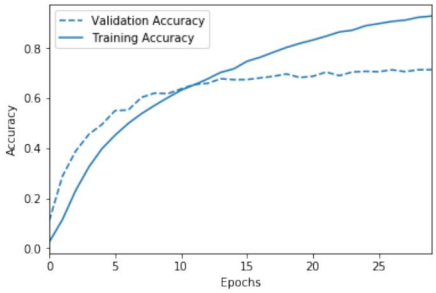
5.2 VGG-16

We fine-tuned hyperparameters for our VGG-16 network, though not to the extent of the other networks due to the amount of time it takes to train the network. We switched between using an Adams optimizer vs. stochastic gradient descent and different learning rates (0.1 to 0.001). To obtain our final results (92.8% training, 71.4% validation, 71.8% test accuracy), we used stochastic gradient descent with a learning rate of 0.001, a decay of 1×10^{-6}, and Nesterov momentum of 0.9. The final

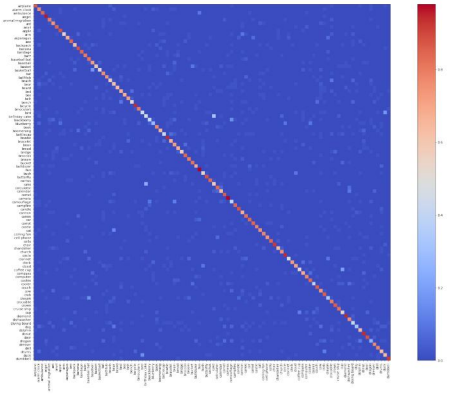
loss curves and accuracy training over time are shown in Figures 7a and 7b. As seen in the figure, the validation loss saturates around the 10th epoch and any improvements in accuracy after were small. Training accuracy continued to increase at that point, indicating that the model started to overfit against the training set, rather than learning generalized features. To prevent some of the overfitting, we added regularization in the form of dropout in the last few fully-connected layers. The confusion matrix of the network's performance is shown in Figure 7c. The network misclassified complex objects such as camouflage and dragon frequently, indicating that the network learned enough to distinguish between similar objects but not enough to learn complex ones, unlike ResNets.



(a) VGG-16 loss



(b) VGG-16 accuracy



(c) VGG-16 confusion matrix

Figure 7: VGG-16 results

5.3 Residual Networks

We tuned various hyperparameters for our residual networks, and tried different types of residual networks. For each network, we used an Adam optimizer. Our metrics were categorical cross-entropy loss and accuracy. Although we directly used ResNet-34 and ResNet-50v2 as is from the Keras contributions library, for or ResNet-50 transfer learning network, we added on a global average pooling layer, dropout layer with 0.7, and a dense layer with 100 nodes. We also tuned various hyperparameters, but were limited since we used networks from a library. For learning rate, we tried a range of 0.01 to 0.0001 and discovered that 0.001 worked best in all cases. We also tried a rate decay range of 0 to 1×10^{-8} and a dropout range of 0 to 0.5. The set of hyperparameters that gave the best validation accuracy and loss for each type of network was chosen (Table 2).

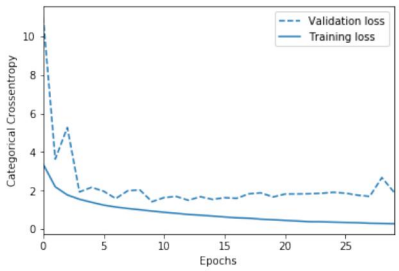
Net	Learning Rate	Rate Decay	Dropout	Transfer Learning	Epochs
ResNet-34	0.001	None	0.1	No	30
ResNet-50	0.001	None	None	Yes	15
ResNet-50v2	0.001	5×10^{-7}	0.3	No	15

Table 2: Table of hyperparameters for each residual network

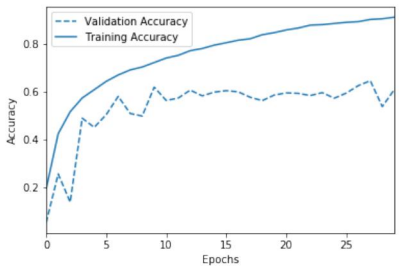
Our ResNet-34 ended up performing the best of the three (Table 3). As we can see from the plots of cross-entropy loss and accuracy, both of the ResNet-50 networks overfit the training data. ResNet-34 also overfit the dataset, but not as much (Figure 8). It obtained 62.3% accuracy. Looking at the confusion matrix for ResNet-34 (see resnets.ipynb in our repository due to space constraints), we can see that most of the misclassified objects had similar shapes. For example, baseball was often times misclassified as clock, compass, and cookie. We also had some categories that were very similar and often got misclassified, such as cake misclassified as birthday cake. Overall, ResNets were unable to capture objects that had similar shapes. This is due to our upsampled images being too blurry, which causes some of the detailed features of our images to blur. Since ResNet is such a large network, the final output will be fairly small, and may not accurately represent the blurred features of our images.

5.4 MobileNet

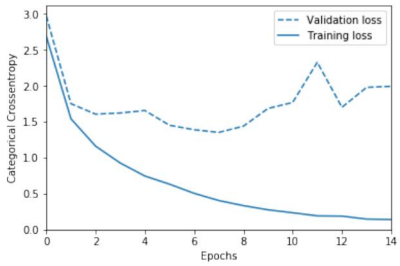
Our final MobileNet had a train accuracy of 99.95%, validation accuracy of 67.42% and a test accuracy of 67.2% (Figure 9). We see that the validation loss and accuracy are very noisy due to the large batch size. Overfitting could have been fixed by adding more training images but we were limited by memory restrictions. With MobileNet we had a faster training time but lower accuracy. Due to depth wise separable filters, we are only filtering input channels individually instead of performing a normal convolution which also combines filters to form new features. Since these images were grayscale and blurry, the ability to recognize more features from a normal convolution would have given us better results (VGG-16). In addition, MobileNet only has 2 hyperparameters that can be tuned: depth multiplier which resizes the image at each layer and a width multiplier which changes the size of each layer. With small image sizes and limited number of hyperparameters to tune, we were unable to further improve.



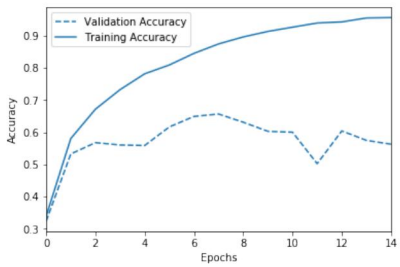
(a) ResNet-34 loss



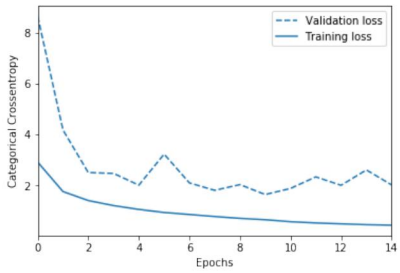
(b) ResNet-34 accuracy



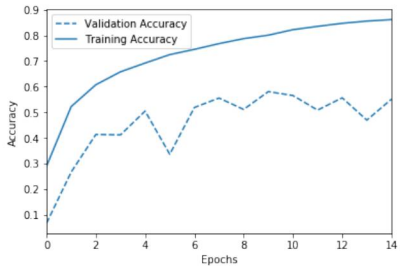
(c) ResNet-50 loss



(d) ResNet-50 accuracy

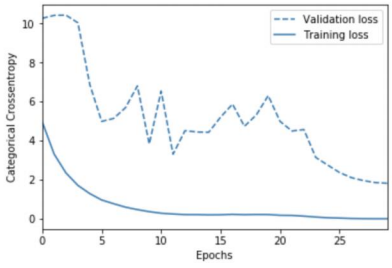


(e) ResNet-50v2 loss

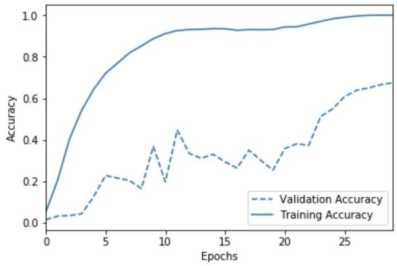


(f) ResNet-50v2 accuracy

Figure 8: ResNet losses and accuracies



(a) MobileNet loss



(b) MobileNet accuracy

Figure 9: MobileNet performance over epochs

5.5 Overall

Our results are summarized in Table 3. In general, shallower networks seemed to perform better than deeper networks. Other than the custom CNN network, each of the networks trained vastly overfit the training dataset. VGG-16 ended up with the best test performance.

Model	Training	Validation	Test
VGG-16	92.8%	71.4%	71.8%
Custom CNN	78.7%	67.1%	67.5%
MobileNet	99.5%	67.4%	67.2%
ResNet-34	90.1%	64.5%	62.3%
ResNet-50	95.6%	56.3%	55.4%
ResNet-50v2	85.2%	57.3%	56.2%

Table 3: Summary of Model Accuracies

6 Conclusion/Future Work

In conclusion, we found that our networks generally had high variance, where the training accuracies would reach over 85% while test accuracies peaked at only 71%. We believe that the networks that we trained were too deep, as increasing the depth using ResNets only furthered the overfitting problem. VGG-16 ended up with the best test performance at 71.8%. Our custom CNN used as a baseline had the least variance. With more time, we would explore shallower networks that have been proven to be good at identifying simpler objects. Our dataset also did not include every data point supplied with each image, such as the country that they originated from and brush stroke timing. To lower our variance problem, we will need more computing resources to try training on more images, since we only used 200 training images from 100 categories. We quickly ran out of memory when trying to train on all the images from all of the categories, and had to limit our data. Finally, to help the performance of our networks, we could try additional preprocessing techniques by generating better representations of our images through image generation. As the images were upsampled using bilinear interpolation, we noticed that detailed features were getting blurred. With better representations of our image, we can have more unique features identified by our networks.

7 Contributions

Sophia Chen preprocessed the data by scraping the Excel files and the Google Quick, Draw! bucket to create the images, and split the data into train/dev/test. She also trained and tuned the different residual networks. John Yu trained and tuned the VGG-16 network, contributed his personal GPU for running our milestone, and setup the GPU instances on AWS. Arsh Buch trained and tuned MobileNet and figured out how to use OpenCV to upsample our images. Everyone contributed equally to the report and the poster.

Our Github repository can be found at https://github.com/schen5050/CS230-Project.git

References

[1] Ravi Kiran Sarvadevabhatla and R. Venkatesh Babu. "Freehand Sketch Recognition Using Deep Features". In: *CoRR* abs/1502.00254 (2015). arXiv: 1502.00254. URL: http://arxiv.org/abs/1502.00254.

[2] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[4] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[5] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146. URL: http://arxiv.org/abs/1605.07146.

[6] Google AI. *Quick, Draw! Doodle Recognition Challenge*. Dec. 2018. URL: https://www.kaggle.com/c/quickdraw-doodle-recognition.

[7] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[8] Arindam Das, Saikat Roy, and Ujjwal Bhattacharya. "Document Image Classification with Intra-Domain Transfer Learning and Stacked Generalization of Deep Convolutional Neural Networks". In: *CoRR* abs/1801.09321 (2018). arXiv: 1801.09321. URL: http://arxiv.org/abs/1801.09321.

[9] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[10] François Chollet et al. *Keras*. https://keras.io. 2015.

[11] *Keras Community Contributions*. GitHub, https://github.com/keras-team/keras-contrib. Various contributors.