**Emotion Recognition with Deep Convolutional Neural Networks**

## Orkun Temiz

Middle East Technical University

[orkun.temiz@metu.edu.tr](mailto:orkun.temiz@metu.edu.tr)

# Abstract

We trained a moderate, deep convolutional neural network to classify the 3589 test samples over 28709 training samples in the custom CNN network we constructed. FER-2013 dataset has 7 different classes. On the test data, we achieved accuracy rate around 65% which is slightly greater than accuracy rate obtained by popular VGG19 and simplified version of Inception which is constructed by us. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 7-way softmax. We used GPU kernel of the Google-Colab in order make the operations faster. To reduce overfitting in the both fully-connected and convolution layers we employed dropout and batch normalization techniques. In the Kaggle competition our model could be at top 5, with a best entry of 71%

# Introduction

Facial expression recognition is an important part of human emotion recognition, which is widely used in human-computer interaction, pattern recognition, image understanding, machine vision and other fields. There are more than 10 thousand kinds of expressions, and different people have different ways to express their emotions. In 1971, Paul Ekman [23] the famous psychologist in American proposed that the facial expressions of people from different cultures are much in common, the expression of the six basic emotions of happiness, anger, sadness, disgust, surprise and fear are very similar in many cultures. In addition to those our dataset includes the neutral emotion to the set, actually it means other in this context.

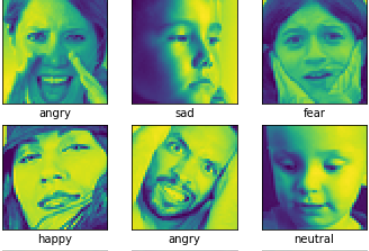
Early facial expression recognition is mainly use the common methods in face recognition to classify and recognize facial expressions. Usually, SVM, LBP and Gabor are used to classify and recognize facial expressions according to the features extract from Haar, Adaboost and neural network. However, with the development of the complex neural networks those are replaced with models such that VGG16, Inception V3 etc. In this paper we challenge the idea that very deep and complex network is necessary to do simple Emotion recognition challenge obtained from FER-2013 dataset. We tried models with different architectures and transfer learning with pre-trained model VGG19.

We tested pretrained model also in the real life images to see whether or not it is works or our model just overfitted the data collection method of the given dataset.

This paper implements an effective facial expression recognition model based on CNN+FC under the TensorFlow platform. In this paper, we use the transfer learning technique to retrain the VGG19 model on the facial expression dataset as well as a simple version of Inception model in order to compare our model. The rest of this paper is organized as follows. Section II introduces the related dataset of facial expression recognition. In section III, we introduce the model architecture of facial expression classification model. We prove the validity of the model through experiments in section IV.

# The Dataset

FER-2013 is a dataset of over 35,877 labeled 48x48 images belonging to 7 categories. The images were collected from the Kaggle competition which named as Challenges in Representation Learning: Facial Expression Recognition Challenge. There is 32,888 training images and 3,589 test images present in the data set whose pixel values comes in a string format. Therefore, before starting to do anything we need to convert them as an array structure for further processing.

The models which are constructed by us need no data reshaping except the one that converts to string with 2,304 value to array of size 48x48. Since we don’t have channels input in the dataset the images are in gray scale. However, in the transfer learning we used VGG-16 which expect RGB images from us. In order to comply this criteria we stacked our image by 3 times to get channel value of 3. We also extract the emotions column from the dataset and convert it to matrix by (#samples,7). In the end we have features matrix having shape of (#samples, 48, 48, 1) and emotions matrix with shape (#samples,7). When we investigate the data set further we can see that orientation, zoom level, location and profile of the pictures are different from each other. In order to learn features without having bias to those variables we applied data augmentation and added augmented data to original dataset. Then in order to avoid the bias occurred in the data we shuffled the dataset. Also, Normalization is required so that all the inputs are at a comparable range and variance of the features can be controlled, thus min-max normalization is applied to the dataset.

*×*

*×*

# The Architecture

The architecture of our network is summarized in Figure 2. It contains eight learned layers — five convolutional and three fully-connected. Below, we describe the features of our network’s architecture. Sections 3.1-3.4 are sorted according to our estimation of their importance, with the most important first.

## ReLU Nonlinearity

A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU for short. We used the rectifier function for the both CNN and fully connected layers in order to provide non –linearity in the network and explain the complex relations between the layers. In our model we want to exploit some advantages of the relu function which are; Computations are also cheaper: there is no need for computing the exponential function in activations and Representational Sparsity; which means that negative inputs can output true zero values allowing the activation of hidden layers in neural networks to contain one or more true zero values and It can accelerate learning and simplify the model.

## Overall Architecture

Now we are ready to describe the overall architecture of our CNN. As depicted in Figure 2, the net contains ten layers with weights; the first six are convolutional and the remaining three are fully- connected in between the fully connected and convolution layers there is a flattening layer. The output of the last fully-connected layer is fed to a 7-way softmax which produces a distribution over the 7 class labels. Our network minimizes the mean cross entropy between the one hot encoded true label and the softmax applied output of the Neural Network graph.

Convolution layers are connected to each other sequentially and batch-normalization is used in every layer. Drop-out and pooling layers are used in every two convolution layers. Filter number is decreased every two layer in order to decrease the depth wise complexity as well as decreasing height and width complexity of the tensors by max-pooling operations. The neurons in the fully- connected layers are connected to all neurons in the previous layer and Batch-normalization and drop-out layers are followed those fully connected layers. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

The first convolutional layer filters the 48x48x1 input image with 64 kernels of size 3x3x1 with a stride of 1 pixels.

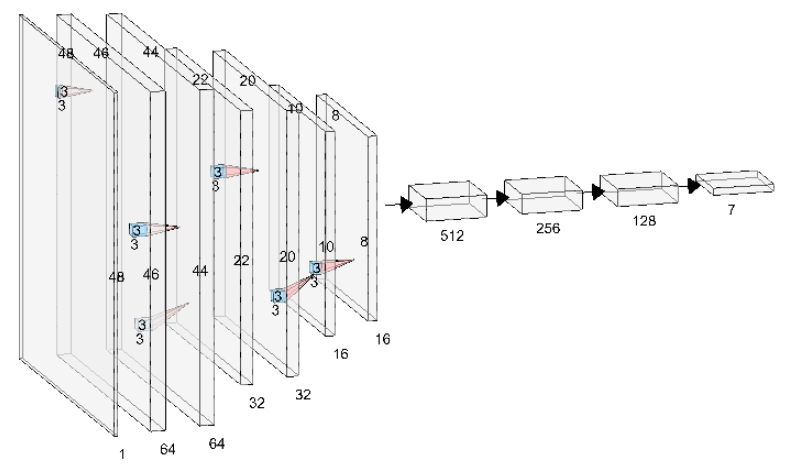


Figure2: An illustration of the architecture of our CNN The network’s input is 2,304 -dimensional, and the number of neurons in the network’s remaining layers is given by 135,424–123,904–12,800–1,600–1,024 – 512–256–128-7.



The second convolutional layer takes as input the (batch-normalized) output of the first convolutional layer and filters it with 64 kernels of size 3. Third convolution layer takes the output (batch-normalized, max-pooled and drop-out applied) of second convolution layer and filters it with 32 kernels of size 3. The architecture goes as this monotonic way about 6 convolution layers. (Two same sized and numbered convolution layer continues with pooling and drop-out layers, then decrease the filter number to half and continue, use batch normalization in every layer). Flattened layer has 256 neurons, in order to extract some features from it first convolution layer chosen with 512 neuron (it acts as a deconvolution layer in someway). This little change is increased the accuracy rate about 3% on the validation data. The fully-connected layers have 512, 256 and 128 neurons respectively in order to decrease the model structure in a controlled way.

*× ×*

# Reducing Overfitting

Our neural network architecture has 368,864 parameters. The orientation, zoom level, location and profile of the pictures are different from each other. Therefore, overfit to the training set may occur in our model. In order to prevent it we take some precautions such as data augmentation, shuffling the input data, batch normalization and usage of drop-out.

## Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformation. We employ four distinct forms of data augmentation, both of which allow transformed images to be produced from the original images.

Firstly, we used zooming, since the face of a person can be captured nearby or far away. In order to learn features independent from the distance to the face zooming is applied. However, since we investigated the dataset, all the images are focused to the face and there is a little variation to the distance of the face to the camera. Therefore, we used 10% and 25% zoom only. To do this first we take the images as tensors and centrally scale the images with the given zoom percent then we crop and resize the images. Then convert image tensors to numpy array again.

Secondly, we used translation, since the face of the person can be anywhere in the picture in order to prevent location bias of the features we shifted the given pictures 15% to the all directions. Again since the images are focused to the face generally, in order not to lose too much features we chose the shifting percent as 15. In order to do this we used used tf.image.extract\_glimpse which returns a set of windows called glimpses extracted at location offsets from the input tensor. If the windows only partially overlaps the inputs, the non-overlapping areas will be filled with random noise.

Thirdly, we used flipping images in other words taking the mirror of the image. Since person in the picture can look right or left. In order to not the overfit the profile of the picture we take the mirror of the image and added to the training set. We used the built-in function of tf.image.flip\_left\_right(X).

Fourthly, since the person’s face can be tilted to some direction in the picture and we want our features to robust to that situation, we rotated our images left and right by 30 degrees. Since the persons head most probably tilted as 30 degrees we choose this angle for augmenting our images. We used built in function of tf.contrib.image.rotate for this case.

## Dropout

Dropout, consists of setting to zero the output of each hidden neuron with probability 0.25. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in back- propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. The selection process of the drop-out probability should be cross validated. After the experiments we conducted through the model development process we have chosen the drop-out probability as 0.25 and in 30 epoch the validation accuracy is 2% more comparing to drop-out probability of 0.5. This phenomena can be explained by the model structure. Since our model is not very complex and we have only 368,864 parameters to be optimized (AlexNet has 60 million parameters) dropping out 50% of them results to huge loss of information. Therefore we limit the drop-out probability with 25%.

Drop-out is used in every 2 layers in convolution layers and every layer in FC layers. Since in convolution layers number of parameters are small comparing to FC layers, this type of architecture is chosen.

## Batch Normalization

We normalize the input layer by adjusting and scaling the activations. For example, when we have features from 0 to 100 and some from 200 to 255. Therefore we should normalize them and bring them into a same 0-1 scale to both explain the variance in the model and speed up learning.

Like in input layers Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift) in the hidden layers. We can use higher learning rates because batch normalization makes sure that there’s no activation that’s gone really high or really low since we use Relu and Relu is a linear function on the right side of 0 we can benefit from that. Therefore, things that previously couldn’t get to train, it will start to train.

It reduces overfitting because it has a slight regularization effects. Similar to dropout, it adds some noise to each hidden layer’s activations. Therefore, if we use batch normalization, we can use less dropout, which is a good thing because we are not going to lose a lot of information.

# 5 Details of learning

We trained our models using AdamOptimizer with a batch size of 256 examples with a learning rate 0.001, beta1=0.9 and beta2=0.999. Since AdamOptimizer gives faster convergence then SGD and gives comparably well results SGD with annealing and momentum. We trained our model 35 epochs therefore we passed whole training set about 35 times. Despite our model is not very complex in terms of variables due to data augmentation we have a lot of training samples and this increases the training time considerably. To be more certain we have 28709 training samples in the beginning and after the data augmentation we have 287090 training samples, we increases sample size to 10 fold, consequently this increases the training time per epoch.

*− ·*

In learning the epoch size is a hyper parameter too and large epoch size in training may result to overfitting in the training data. In order to prevent this we calculated both training and validation accuracy in every iteration and find the value which validation accuracy is no longer increase. In this particular model we chosen the 35 as the epoch size.

We initialized the weights in each layer from a Xavier Initialization which proved to be the best practice for weight initialization in most papers. We initialized the neuron biases for every convolution layer, as well as in the fully-connected hidden layers, with again Xavier Initialization. This initialization accelerates the early stages of learning by providing the ReLUs with positive

# Results

Our results on ILSVRC-2010 are summarized in Table 1. Our network achieves top-1 and top-5 test set error rates of **37.5%** and **17.0%**5. The best performance achieved during the ILSVRC- 2010 competition was 47.1% and 28.2% with an approach that averages the predictions produced from six sparse-coding models trained on different features [2], and since then the best pub- lished results are 45.7% and 25.7% with an approach that averages the predictions of two classi- fiers trained on Fisher Vectors (FVs) computed from two types of densely-sampled features [24].

We also entered our model in the ILSVRC-2012 com- petition and report our results in Table 2. Since the ILSVRC-2012 test set labels are not publicly available, we cannot report test error rates for all the models that we tried. In the remainder of this paragraph, we use

|  |  |  |
| --- | --- | --- |
| **Model** | **Top-1** | **Top-5** |
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |

validation and test error rates interchangeably because in our experience they do not differ by more than 0.1% (see Table 2). The CNN described in this paper achieves a top-5 error rate of 18.2%. Averaging the predictions

Table 1: Comparison of results on ILSVRC-

2010 test set. In *italics* are best results achieved by others.

of five similar CNNs gives an error rate of 16.4%. Training one CNN, with an extra sixth con- volutional layer over the last pooling layer, to classify the entire ImageNet Fall 2011 release (15M images, 22K categories), and then “fine-tuning” it on ILSVRC-2012 gives an error rate of 16.6%. Averaging the predictions of two CNNs that were pre-trained on the entire Fall 2011 re- lease with the aforementioned five CNNs gives an error rate of **15.3%**. The second-best con- test entry achieved an error rate of 26.2% with an approach that averages the predictions of sev- eral classifiers trained on FVs computed from different types of densely-sampled features [7].

Finally, we also report our error rates on the Fall 2009 version of ImageNet with 10,184 categories and 8.9 million images. On this dataset we follow the convention in the literature of using half of the images for training and half for testing. Since there is no es-

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Top-1 (val)** | **Top-5 (val)** | **Top-5 (test)** |
| *SIFT + FVs [7]* | — | — | *26.2%* |
| 1 CNN | 40.7% | 18.2% | — |
| 5 CNNs | 38.1% | 16.4% | **16.4%** |
| 1 CNN\* | 39.0% | 16.6% | — |
| 7 CNNs\* | 36.7% | 15.4% | **15.3%** |

tablished test set, our split neces- sarily differs from the splits used by previous authors, but this does not affect the results appreciably. Our top-1 and top-5 error rates on this dataset are **67.4%** and

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk\* were “pre-trained” to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

**40.9%**, attained by the net described above but with an additional, sixth convolutional layer over the last pooling layer. The best published results on this dataset are 78.1% and 60.9% [19].

## 6.1 Qualitative Evaluations

Figure 3 shows the convolutional kernels learned by the network’s two data-connected layers. The network has learned a variety of frequency- and orientation-selective kernels, as well as various col- ored blobs. Notice the specialization exhibited by the two GPUs, a result of the restricted connec- tivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).

5The error rates without averaging predictions over ten patches as described in Section 4.1 are 39.0% and 18.3%.

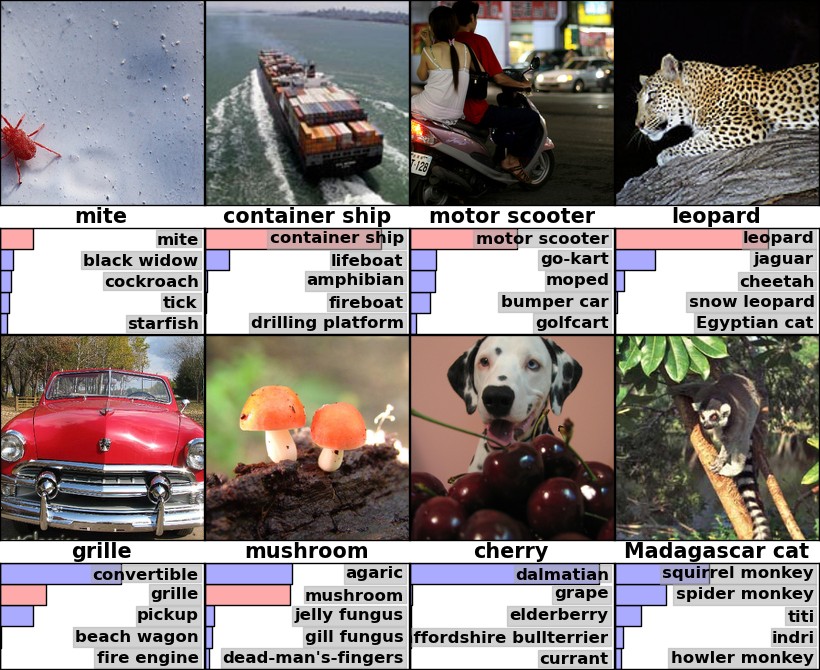


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable. For example, only other types of cat are considered plausible labels for the leopard. In some cases (grille, cherry) there is genuine ambiguity about the intended focus of the photograph.

Another way to probe the network’s visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer. If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Figure 4 shows five images from the test set and the six images from the training set that are most similar to each of them according to this measure. Notice that at the pixel level, the retrieved training images are generally not close in L2 to the query images in the first column. For example, the retrieved dogs and elephants appear in a variety of poses. We present the results for many more test images in the supplementary material.

Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vec- tors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto- encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar.

# Discussion

Our results show that a large, deep convolutional neural network is capable of achieving record- breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network’s performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. Ultimately we would like to use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.

## References

* 1. R.M. Bell and Y. Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.
  2. A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge 2010. www.image- net.org/challenges. 2010.
  3. L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
  4. D. Cires¸an, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification.

*Arxiv preprint arXiv:1202.2745*, 2012.

* 1. D.C. Cires¸an, U. Meier, J. Masci, L.M. Gambardella, and J. Schmidhuber. High-performance neural networks for visual object classification. *Arxiv preprint arXiv:1102.0183*, 2011.
  2. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
  3. J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei. *ILSVRC-2012*, 2012. URL

[http://www.image-net.org/challenges/LSVRC/2012/.](http://www.image-net.org/challenges/LSVRC/2012/)

* 1. L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understand- ing*, 106(1):59–70, 2007.
  2. G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, Cali- fornia Institute of Technology, 2007. URL [http://authors.library.caltech.edu/7694.](http://authors.library.caltech.edu/7694)
  3. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural net- works by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
  4. K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.
  5. A. Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto, 2009.
  6. A. Krizhevsky. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 2010.
  7. A. Krizhevsky and G.E. Hinton. Using very deep autoencoders for content-based image retrieval. In

*ESANN*, 2011.

* 1. Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, et al. Hand- written digit recognition with a back-propagation network. In *Advances in neural information processing systems*, 1990.
  2. Y. LeCun, F.J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97. IEEE, 2004.
  3. Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.
  4. H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng. Convolutional deep belief networks for scalable unsuper- vised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.
  5. T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric Learning for Large Scale Image Classifi- cation: Generalizing to New Classes at Near-Zero Cost. In *ECCV - European Conference on Computer Vision*, Florence, Italy, October 2012.
  6. V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th International Conference on Machine Learning*, 2010.
  7. N. Pinto, D.D. Cox, and J.J. DiCarlo. Why is real-world visual object recognition hard? *PLoS computa- tional biology*, 4(1):e27, 2008.
  8. N. Pinto, D. Doukhan, J.J. DiCarlo, and D.D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11):e1000579, 2009.
  9. B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173, 2008.
  10. J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.
  11. P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, volume 2, pages 958–962, 2003.
  12. S.C. Turaga, J.F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H.S. Seung. Con- volutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 22(2):511–538, 2010.