Pre-Training CNNs Using Convolutional Autoencoders

Maximilian Kohlbrenner
TU Berlin
m.kohlbrenner@campus.tu-berlin.de

Russell Hofmann
TU Berlin
r.hofmann@campus.tu-berlin.de

Sabbir Ahmmed
TU Berlin
ahmmed@campus.tu-berlin.de

Youssef Kashef
TU Berlin
kashefy@ni.tu-berlin.de

Abstract

Despite convolutional neural networks being the state of the art in almost all computer vision tasks, their training remains a difficult task. Unsupervised representation learning using a convolutional autoencoder can be used to initialize network weights and has been shown to improve test accuracy after training. We reproduce previous results using this approach and successfully apply it to the difficult *Extended Cohn-Kanade* dataset for which labels are extremely sparse but additional unlabeled data is available for unsupervised use.

1 Introduction

A lot of progress has been made in the field of artificial neural networks in recent years and as a result most computer vision tasks today are best solved using this approach. However, the training of deep neural networks still remains a difficult problem and results are highly dependent on the model initialization (local minima). During a classification task, a Convolutional Neural Network (CNN) first learns a new data representation using its convolution layers as feature extractors and then uses several fully-connected layers for decision-making. While the representation after the convolutional layers is usually optizimed for classification, some learned features might be more general and also useful outside of this specific task. Instead of directly optimizing for a good class prediction, one can therefore start by focussing on the intermediate goal of learning a good data representation before beginning to work on the classification problem. One possible way to do this is to use a Convolutional Auto-Encoder (CAE), an unsupervised method that allows to train the convolutional layers independently from a classification task in order to learn a new data representation. The weights learned in this first step can then be used as a starting point to initialize a neural network's convolution layers. This becomes of special interest when only very few labeled examples are available since additional, unlabeled datapoints can be used for the representation learning task. [10] showed that pre-training a neural network using autoencoder weights can improve the classification accuracy consistently by a small margin.[1] We reproduce these results on the common *MNIST* [7] and *CIFAR-10* [6] benchmarks and finally demonstrate that the approach can be used very effectively on the *Extended Cohn-Kanade* [4, 8] dataset where label information is not only extemely sparse but also ambiguous.

[1] In their paper, pre-training boosted the classification accuracy by up to 3.2% (on *CIFAR-10* restricted to 1k training datapoints.)

2 Related Work

Erhan et al. [2] showed that unsupervised pre-training adds robustness to a deep architecture and pre-trained networks consistently give better generalization. The probability of finding poor apparent local minima increases with the increasing depth of an architecture that is not pre-trained. They observed that pre-trained networks learn qualitatively different features compared to networks without pre-training and more importantly pre-training not only gives a good initial marginal distribution but also captures more intricate dependencies between parameters. Another interesting point they asserted is that unsupervised pre-training exhibits some properties of a regularizer and classical regularization techniques (e.g L1/L2) do not achieve the same performance as unsupervised pre-training. A point of caution, however, is the fact that the pre-trained models obtain worse training errors, but better generalization performance when the layers are big enough in deep architectures.

Masci et al. [10] introduced the CAE, an unsupervised method for hierarchical feature extraction, which learns biologically plausible filters. Masci et. al demonstrated that a CNN can be initialized by a CAE stack. They experimented with 20 7x7 filters of 4 CAEs of the same topology trained as follows on the *MNIST* dataset: (i) the first one on original digits, (ii) the second one on noisy inputs with 50% binomial noise added, (iii) the third one with an additional max-pooling layer of size 2x2, (iv) the fourth one on noisy (30% binomial noise) inputs with a max-pooling layer of size 2x2. When they compared the results, it turned out that interesting and biologically plausible filters emerge only when the CAE is trained with a max-pooling layer. When the CAE is trained without any additional constraints, it learns trivial solutions such as having only one weight "on" (identity function) and when the CAE is trained with additional noise, it becomes more localized. They repeated the above experiments on the *CIFAR-10* dataset and found striking impact of a max-pooling layer whereas adding noise had almost no visual effect except on the weight magnitudes, indicating that max-pooling is essential. They also demonstrated that pre-trained CNNs consistently outperform randomly initialized CNN.

Makhzani et al. [9] explored combining the benefits of convolutional architectures and autoencoders for learning deep representations in an unsupervised manner. They proposed an unsupervised learning algorithm for training CAE with a very strong sparsity constraint on the activity within each feature map. To achieve that they trained a CAE layer by layer on *MNIST*, *CIFAR-10* and *NORB* datasets, where in each layer sparsity was achieved using a winner-take-all activation function within each feature map. The sparse convolutional autoencoder was based on a CAE with a ReLU activation function.

Khorrami et al. [5] achieved state-of-the-art performance on two face expression recognition benchmarks: the extended Cohn-Kanade (CK+) dataset and the Toronto Face Dataset (TFD) by training a zero-bias CNN on facial expression data and by introducing an approach to decipher which portions of the face influence the CNN's predictions.

Hamester et al. [3] proposed a model for face expression recognition based on a variant of the Multi-channel Convolutional Neural Network (MCCNN) earlier introduced by [1] which does not depend on any hand-crafted or task specific feature extraction but exploits unsupervised learning, using a multiple CNNs based 2-channel architecture, where the CNNs first process information independently and then all streams merge in a fully connected layer, which is used for classification. The weights of the channel are trained in an unsupervised fashion as a convolutional autoencoder (CAE). They evaluated the model on *JAFFE* dataset and found that the model is a very viable method for facial expression recognition. They easily reached the state-of-the-art recognition rates with minimal effort and achieved an average of 95.8% accuracy which was almost 3% and 8% higher than the other state-of-the-art methods (Sobel-based filters and McFIS method) used by [12].

Datasets In previous work [10], the authors applied the approach of pre-training by a convolutional autoencoder to two standard image classification datasets, MNIST and CIFAR-10. Since one part of our project consists of reproducing these results, we will shortly introduce both datasets here. MNIST [7] is the standard dataset used for image classification. It consists of scanned handwritten digits, which are 28 by 28 pixel grayscale images. As they are digits, the goal is to classify the correct one of 10 classes. The dataset consists of 55000 training, 5000 validation, and 10000 test images. The CIFAR dataset [6] is made out of natural scenes, which are 32x32 colorized images. The relatively low resolution, while still made of real images and the colors lead to a more difficult classification

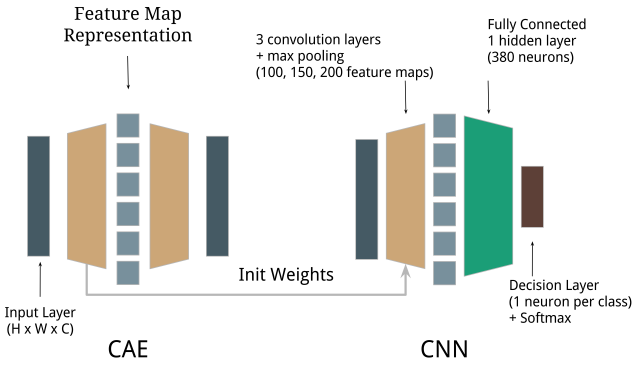


Figure 1: Experimental Setup

challenge. The CIFAR-10 variant, which we used, is built as a classification dataset with 10 specified labels (cars, cats,...). It consists of 45000 training, 5000 validation, and 10000 test images.

3 Methods

In this paper, we evaluate the influence of CNN pre-training using CAE on the network's out-of-sample accuracy. To achieve this, we train, in a first step, a convolutional autoencoder on a chosen dataset and then, in a second step, use its convolution layer weights to initialize the convolution layers of a CNN. After training, we compare the CNN's test set accuracy to a reference network that was trained under the same conditions with randomly initialized convolution weights.

A visualization of the weight transfer can be seen on the schematic in figure 1.

While we restricted the training set size for some of the CNN training experiments, the CAEs were always trained on all available training data since in a lot of settings, additional unlabeled data is easily available while labeled data might be extremely rare and costly.

3.1 Autoencoder

Architecture The CAE first uses several convolutions and pooling layers to transform the input to a high dimensional feature map representation and then reconstructs the input using strided transposed convolutions. In our experiments, we use the autoencoder architecture described in [10]. It's encoding part consists of 3 times a convolution layer followed by 2x2 non-overlapping max-pooling and a layer of scaled-tanh activations (see section 3.1). More detailed layer information is given in table 1. All convolutional layers perform a full, non-strided convolution operation, their output feature maps have the same height and width as their input.

Layer Type	input	convolution	max-pool	convolution	max-pool	convolution
Filter Size	none	5x5	2x2	5x5	2x2	3x3
Channels	1(gray)|3(rgb)	100	100	150	150	200
Activation	none	scaled-tanh	none	scaled-tanh	none	scaled-tanh
Size	(H, W, C)	$(H, W, 100)$	$(\frac{H}{2}, \frac{W}{2}, 100)$	$(\frac{H}{2}, \frac{W}{2}, 150)$	$(\frac{H}{4}, \frac{W}{4}, 150)$	$(\frac{H}{4}, \frac{W}{4}, 200)$

Table 1: CAE/CNN: shared encoding layers. Channels refers to the amount of feature maps in the given layer

For the reconstruction, we use one strided transposed convolution layer per convolution / max-pool layer in the encoding layers. The weights of the transposed convolutions are the same as learned in the convolution layers and the 2x2 strides invert the downsampling effect of the max-pooling layers.

Activation Function Out of 3 tested activation functions (sigmoid, scaled tanh and relu), we use the scaled tanh for our main experiments similar to [10]. We will talk about our ReLU experiments in

(4.1.1). By scaled tanh we mean the tanh function rescaled and shifted to the [0,1] output range. This corresponds to the function

$$scaledtanh(x) = \frac{1}{2}tanh(x) + \frac{1}{2}$$

which has a generally sigmoidal shape but a stronger gradient around $x = 0$. In our experiments, switching the activation function from sigmoid to scaled tanh seemed to give similarly good results after less training iterations.

Regularization While autoencoders are typically used for dimensionality reduction, the feature map representation of the convolutional autoencoders we are using is of a much higher dimensionality than the input images. While this feature representation seems well-suited in a CNN, the overcomplete representation becomes problematic in an autoencoder since it gives the autoencoder the possibility to simply learn the identity function. Without any further constraints, a full convolutional layer could easily learn a simple point filter such as $k = \begin{smallmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{smallmatrix}$ (for grayscale input) that copies the input onto a feature map. While this would later simplify a perfect reconstruction of the input, the CAE does not find any more suitable representation for our data. To prevent this problem, some kind of regularization is needed. Several methods can be found in the literature, most of them involving some kind of sparsity constraint for the representation layer [9]. In [10] the authors use max-pooling as an elegant way to enforce the learning of plausible filters without the need for any further regularization. Since the CNN architecture we are going to use later will contain pooling layers anyways, we stick with this technique and choose not to add any noise to our inputs either since this does not seem not to help the emergence of more natural filters in their experiments.

Training While the authors of [10] use a layer-wise training for CAE, we achieve good results by training the whole auto-encoder at once. For the *MNIST* and *CIFAR-10* datasets, the autoencoders are trained with simple gradient descent using as learning rate of $lr = 0.5$, for the *CK+* dataset, we use the adagrad optimizer with the same initial learning rate lr.[2]

3.2 CNN Pre-Training

CNN architecture The CNN uses the same architecture as the CAE encoding for it's convolution layers (feature extraction) and then uses two fully-connected layers (sizes 384 and 10) for decision making, the first having a scaled tanh activation, the second followed by a softmax layer for probabilistic output.

CNN Initialization and Training The convolutional layer weights (filters) and biases of the pre-trained networks were initialized with the values obtained from the convolutional autoencoders. The reference CNN's convolution weights were initialized with mean $\mu = 0$ and standard deviation $\sigma = 0.2$, the biases with the constant value $b = 0.0001$.

The fully-connected layers were initialized randomly for all networks.[3] This means that for all experiments, some parts of the networks are randomly initialized, all weights for the reference networks and the fully-connected weights for the pre-trained networks.

For each dataset, we split the available training data into training and evaluation set and use the evaluation set to track the training progress and adjust hyperparameters. We keep track of both cross-entropy error and accuracy on the evaluation set and stop training when these values converge. Afterwards, we evaluate the networks accuracy once on the held-out test set and use the obtained value as a result. We let every experimental setup run 10 times, average the results and conduct a t-test for significance. The test set stays the same over all runs.

For the larger datasets *MNIST* and *CIFAR-10*, we conduct several experiments restricting the amount of available training data inspired by [10]. The autoencoder used for pre-training is always trained on the whole dataset.

[2] CAE convolution filters initialized using a truncated normal distribution ($\mu = 0.001$, $\sigma = 0.05$), all bias values initialized with the constant 0.001

[3] CNN fully-connected layer weights initialized with a truncated normal distribution ($\mu = 0$, $\sigma = 0.001$), biases with the constant value 0.0001.

Figure 2: CK+ frames from neutral face to emotional face (©Jeffrey Cohn)

Since we are working with small amounts of training images for complex datasets, we are aware that problems with overfitting might arise. In order to prevent these effects, we use dropout regularization in the dense layers [4], monitor the training progress with a seperate validation set and measure the generalization performance on a held-out test set.

3.3 Extended Cohn-Kanade Dataset

Altogether the Extended Cohn-Kanade (CK+) dataset [4, 8] has 123 subjects and 593 emotion sequences, which are 10-60 frames of short video sequences, from a neutral face to the target emotion, as can be seen in figure 2. CK+ is the most interesting dataset in our project, as unlike the other two datasets, no pre-training experiments have been done on CK+, as far as we know.

As the CK+ dataset is not readily prepared for running those kinds of experiments, we have to put some work into this ourselves. We decided to not take the full images, but use the provided facial landmarks, to put the faces into a bounding box and scale them down by factor 5 to a size of 68x65 pixels. In order to train the autoencoder we use all of the frames as unlabelled input, for the CNN we use the last three frames of all sequences, labelled with the target emotion. We split the dataset into training, validation and test set by subject, so that we end up with 696 training, 87 validation and 198 test images for the CNN. Before training we randomly shuffle the images and used the same split for all runs.

Another specialty for the CK+ dataset is, that when we use all frames for the CAE, this includes frames which are neutral or even ambiguous. As this possibly broadens the available information for learning the autoencoder, and is also useful for practical use, we could possibly use a lot of unlabelled images for pre-training. In this case the pretraining makes especially sense, because we can learn features, we could not learn when training only a CNN with labelled images.

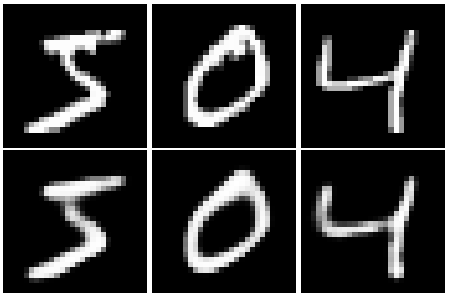
3.4 Statistical Significance

In order to validate possible resulting accuracy differences, we also want to check their significance. Therefore we run a standard t-test to measure if the accuracies from the randomly initialized CNN are significantly different from the pre-trained one. This way we calculate the probability (p-value) that the improvements could occur, when the pre-trained network was not better than the randomly initialized one (Null-hypothesis).

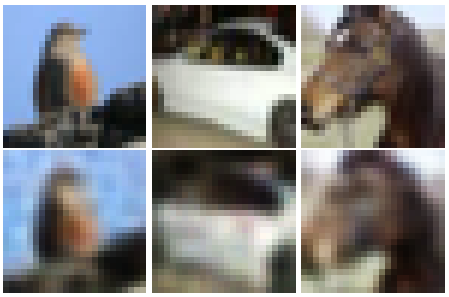
4 Experimental Results

We conduct experiments on the three introduced datasets (MNIST, CIFAR-10 and CK+), which vary in their image size, number of images and colour information.

[4] using a dropout rate of 0.5 during training time



(a) *MNIST* example images (top) and CAE reconstructions (bottom)



(b) *CIFAR-10* example images(top) and CAE reconstructions(bottom)

Figure 4: CK+ input image (top) and reconstruction (bottom) (©Jeffrey Cohn)

4.1 Autoencoder Reconstruction

When training a utilizable CAE, the most obvious goal is to get good reconstructions of the input images. Even though this does not always necessarily mean that we learned a good representation, it is still our main optimization criteria here. In the next section we will then check, if we actually learned a useful representation by applying it to a classification network.

After 300k training iterations on a dataset, the CAE reconstructs the input images sucesfully, examples can be seen in figure 3a, 3b and 4. While the reconstructions for the simple *MNIST* dataset are almost perfect, the reconstructed images on the *CIFAR-10* and *CK+* datasets seem like blurred versions of the input images, however the main features are still visible and on *CK+*, the emotions are clearly differentiable. All autoencoders have the same architecture as described in 3.1 and use the scaled-tanh as activation function. The learned first layer filters show a similar structure to the ones shown in [10][5].

4.1.1 Rectified Linear Units

When switching to a ReLU activation function [11] for the CAE, the reconstructions become almost identical to the input image, but the learned first layer filters seem rather random compared to the ones obtained in previous experiments. Since first pre-training results do not show a significant change between pre-trained and randomly initialized ReLU networks, we do not examine this setting more in detail. Example reconstructions and a random selection of first layer filters are shown in figure 5.

[5]Example filters can be seen on *Github* `https://github.com/gangchill/nip-convnet/tree/master/experiments/paper_reference_experiments/paper_reference_cae/cifar`

Table 2: Statistical Significance of Pretraining Improvements

Dataset	Size	Average Accuracy Improvement	Significance (p-value)
MNIST	1k	+ 0.37 %	< 5e-4
	10k	+ 0.39 %	< 5e-4
	50k	+ 0.32 %	< 1e-2
CIFAR-10	1k	- 0.20 %	> 0.35
	10k	+ 4.25 %	< 1e-9
	50k	+ 2.85 %	< 1e-6
CK+	1k	+ 4.5 %	< 1e-7

4.2 Pretraining and Classification

As described above, to validate the pre-training advantage we train randomly initialized networks and networks which contain the pre-trained feature maps.

We let all networks train till we observe convergence, and then use this empircally obtained number of iterations for all following trainings. The length of training is at least 30 epochs for all datasets, so 5k iterations for the 1k datasets, 10k iterations for the 10k datasets and 20k iterations for the full datasets of *MNIST* and *CIFAR-10*. *CK+* is trained for 2k iterations, this corresponds to more than 350 epochs. During this we check for overfitting, but did not notice a big trend for it. So apparently our regularization mostly works fine.

After the final training we measure the classification accuracy on the separate testset for both the randomly-initialized and the pre-trained CNN, so that we can measure generalization and compare the results.

As can be seen in figures 6 and 7, accuracy values for the pre-trained networks are higher for almost all datasets and configurations. The only exception is the *CIFAR-10* dataset with a training set of 1000 images. When looking at results of the t-test in table 2, there is a significant difference for almost all dataset configurations, the only outlier being again *CIFAR-10* 1k.

Pre-training consistently outperforms random initialization on almost all dataset configurations. Since *CIFAR-10* with its natural images seems the most complex classification task, we assume that in this scenario overfitting might be more of a problem. We suppose that with additional regularization, pre-training could as well be benificial in this context.

In addition to the significant improvement in out-of-sample accuracy, we also observe a striking difference between the filters after convergence depending on the network initialization. The pre-trained network's first-layer filters seem to preserve a lot of the structure learned in the autoencoder, a sign that the learned representation seems to be well suited for the classification task. While the randomly-initialized filters change more during training, their final appearence never shows as clearly interpretable low-level structures as can be seen in an example on the *CIFAR-10* dataset in figure 8.

4.2.1 Interpretation

For both *MNIST* and *CIFAR-10* it can be observed, that the accuracy on a pre-trained CNN is higher than with a randomly initialized CNN, when we use the full or 10k datasets. For the 1k subsets of the *CIFAR-10* dataset, no significant improvement seems to occur. This could be due to overfitting, as we train a complex network on very little data. For the *CK+* dataset we get the biggest improvement

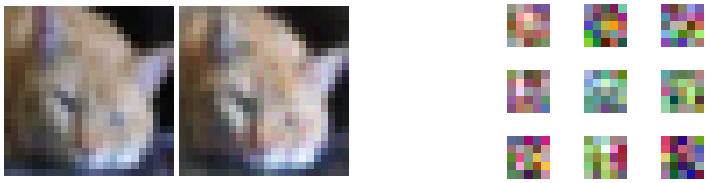


Figure 5: **ReLU CAE** example input + reconstruction (left), selection of first layer filters (right)

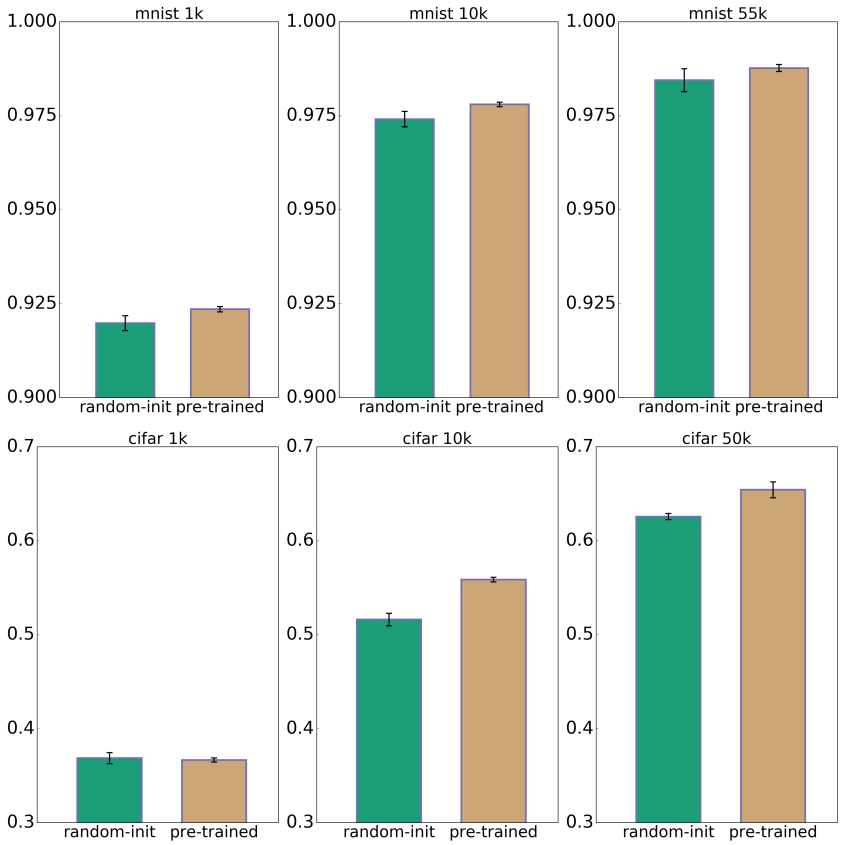


Figure 6: Pre-Training Results *MNIST*(top) and *CIIFAR-10*(bottom): Test set accuracy after training on different subsets of the training data.

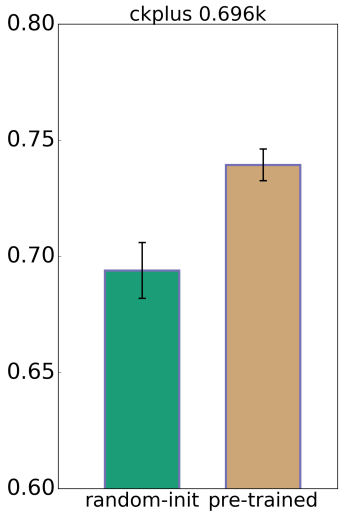


Figure 7: *CK+*: test set accuracy comparison for 696 training images (stddev in black)

Figure 8: ***CIFAR-10* CNN** selection of first layer filters. pre-trained (left), random-init (right)

through pretraining. Even though the standard deviation is quite high, we get a significant jump in the accuracy. This could be due to the fact, that for *CK+* we use all, including neutral and ambiguous, frames (5876 images) to train the CAE, so that the pre-trained filters have more material to learn features from. This leads to the conclusion, that pretraining is especially applicable, when there are much more unlabelled images, but only a few that can be used for learning classifications.

5 Conclusion

We reproduced the results of [10] showing that pre-training a CNN using the weights obtained from a convolutional autoencoder consistently boosts the classifiers accuracy by a small margin.

In our experiments with convolutional autoencoders, we came to the conclusion that the use of the mean-squared error is crucial for training success and hypothesize that when using ReLU functions, the use of max-pooling is not sufficient for the emergence of natural filters that would be helpful for pre-training a neural network.

On the *CK+* dataset we are able to improve the average accuracy of our networks from 70.4% to 73.6% clearly demonstrating the use of this technique in a domain where the amount of labeled training data is very limited.

References

[1] Pablo Barros et al. “A Multichannel Convolutional Neural Network for Hand Posture Recognition”. In: (2014), pp. 403–410.

[2] Dumitru Erhan et al. “Why Does Unsupervised Pre-training Help Deep Learning?” In: (2010).

[3] Dennis Hamester, Pablo Barros, and Stefan Wermter. “Face expression recognition with a 2-channel convolutional neural network”. In: (2015), pp. 1787–1794.

[4] Takeo Kanade, Jeffrey F Cohn, and Yingli Tian. “Comprehensive database for facial expression analysis”. In: *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*. IEEE. 2000, pp. 46–53.

[5] Pooya Khorrami, Tom Le Paine, and Thomas S. Huang. “Do Deep Neural Networks Learn Facial Action Units When Doing Expression Recognition?” In: (2017).

[6] Alex Krizhevsky and Geoffrey Hinton. “Learning multiple layers of features from tiny images”. In: (2009).

[7] Yann LeCun and Corinna Cortes. *The MNIST database of handwritten digits*. 1998.

[8] Patrick Lucey et al. “The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE. 2010, pp. 94–101.

[9] Alireza Makhzani and Brendan Frey. “A Winner-Take-All Method for Training Sparse Convolutional Autoencoders”. In: (2014).

[10] Jonathan Masci et al. “Stacked convolutional auto-encoders for hierarchical feature extraction”. In: *Artificial Neural Networks and Machine Learning–ICANN 2011* (2011), pp. 52–59.

[11] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[12] K. Subramanian, S. Suresh, and R. Babu Venkatesh. “Meta-Cognitive Neuro-Fuzzy Inference System for human emotion recognition”. In: (2012), pp. 1–7.