Ridge-I assessment project: Supervised and unsupervised training

Vegard B. Sørdal

February 11, 2020

Abstract

We use a stacked fully convolutional autoencoder to pretrain a CNN classifier network on a modified Cifar-10 dataset, where 50% of the images for three classes are removed. By implementing majority upsampling, minority downsampling and cost-function class-weighting, we are able to get accuracies comparable to that of the full dataset baseline. We did however not observe any significant benefit of pre-training.

1 Introduction

1.1 Autoencoder basics

The goal of this project is to perform unsupervised pretraining with an autoencoder, and then reuse the encoding block for a supervised classification task on the Cifar10 dataset. By initializing the classifier network with the weights and biases obtained from the pretraining, the hope is that the classifier will attain a higher accuracy, as observed in [3]

A schematic of the network architecture is shown in Fig. 1. An autoencoder is trained by having the output layer reconstruct the input layer, where the reconstruction error is usually given by the average mean square error between the input and output neurons. The central layer in a symmetric autoencoder determines the size of the latent dimension. If the latent dimension is larger than the input layer, the autoencoder is overcomplete; if not properly regularized network can learn to copy the input data without learning generalized features. If it is smaller than the input dimension, the autoencoder is undercomplete, and has to encode the most valuable low variance information in the latent dimension in order to reproduce the input.

1.2 Dataset

For this task we will use the Cifar-10 dataset [2], consisting of 60000 images with 32x32 pixel resolution in RGB, and with 10 different classes. An example image of every class is show in Fig. 2.

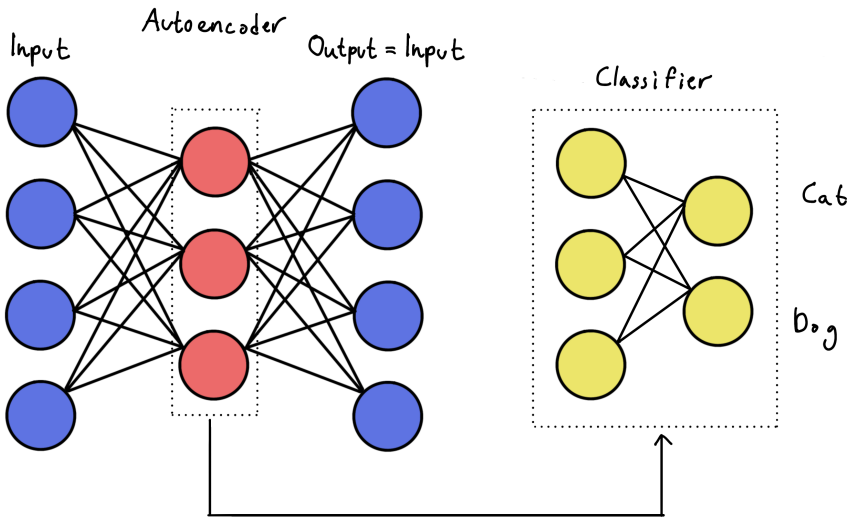

Figure 1: Schematic of the autoencoder/classifier network architecture. The output layer is trained to reconstruct the input layer. If the number of neurons in the encoding layer (red) is smaller than the input layer, the network can only store the most valuable information for reconstruction in this layer. We can then reuse this information to initialize the parameters of a classifier network with similar structure, which will hopefully give better results than a network with random initialization.

Figure 2: Example images for every class in the Cifar10 dataset.

An additional restriction to this project, is that we can only use 50% of the images for three of the classes (bird, deer and truck). Since every class in the dataset has 6000 images, that leaves us with only 3000 images for those classes.

When only using 50% of the data for those three categories the dataset becomes unbalanced. An unbalanced dataset tends to make classifier networks biased towards the categories with higher number of datapoints. It is not difficult to imagine why this happens by considering a two-class example (e.g. cats and dogs); If we have 100 cats and 50 dogs, the network will get a 67% accuracy by always classifying as cats. The unbalanced dataset would have an impact on the precision and recall of the bird, deer and truck classes. Nevertheless, since there are 7 other categories with full dataset that the network still have to distinguish, we can imagine that we won't gain much in terms of precision/recall for those full classes by balancing the dataset.

With an unbalanced dataset like this, there are several strategies we can employ:

1. **Use 3000 images for all categories:**
   When reducing the full dataset it will obviously be more difficult for the network to learn general features, but the advantage is that the images have high variance.

2. **Use data augmentation to increase the 3000 images categories to 6000:**
   Augmenting the data will artificially increase the amount of training data. However, by using geometric transformations and color augmentations, the augmented data tends to be highly correlated with the originals. By using GANs or VAE to create augmented data, these correlations can be greatly reduced.

3. **Weighting the loss function:**
   We can weight the classes in the loss function, such that the loss on minority classes is penalized heavier than others, making the network biased to not make mistakes on those classes.

There are advantages and disadvantages to all of these methods, so we'll have to try a few of them.

2 Network architecture

After some experimentation, the network architecture we ended up with is summarized in Table 1. It consists of alternating convolutions and maxpools in the encoding block, and transpose convolutions in the decoding block. The weights of the transpose convolutions is tied to the weights of the respective convolutions in the encoding block. This is an effective regularization method, which also

halves the number of parameters in the network and thus decreases computation time. In addition, we use apply Gaussian noise with a standard deviation of 0.2 to the input images before sending them to the network. For the optimizer we use Adam (adaptive moment estimation), using the default learning rate of 0.001.

	Filters	Kernel	Stride	Weight sharing
Layer 1: Conv2D	128	(5,5)	(1,1)	Layer 8
Layer 2: MaxPool	-	(2,2)	(2,2)	-
Layer 3: Conv2D	64	(5,5)	(1,1)	Layer 7
Layer 4: MaxPool	-	(2,2)	(2,2)	-
Layer 5: Conv2D	32	(3,3)	(1,1)	Layer 6
Layer 6: Conv2D.T	64	(5,5)	(2,2)	Layer 5
Layer 7: Conv2D.T	128	(5,5)	(2,2)	Layer 3
Layer 8: Conv2D.T	3	(5,5)	(2,2)	Layer 1

Table 1: Architecture of the Convolutional Autoencoder (CAE). A ReLu activation function is used for all layers (except MaxPool), and the transpose layers in the decoding block is tied to their respective convolutions in the encoding block.

2.1 Filter size

The size of the filters determines the receptive field of the convolutions. A larger filter captures more global features of the image, while a smaller filter captures more finer local structure. Both large scale and small scale features can be important for classification tasks, so it is not easy to say a priori which filters one should use. The training images is low resolution 32x32 images, and there are samples where the classes we're trying to detect fills the whole image (e.g. truck example in Fig. 2), as well as those that consist of smaller centered features (e.g. deer example in Fig. 2).

Two sizes that have been shown to be effective at image classification is (3,3) and (5,5). A (1,1) filter would be a fully connected layer, and (7,7) tends to coarse-grain too much local information. If we have deep networks, (3,3) filters tends to learn more complex features, but since the depth of our network is not that deep we can expect that the difference in our results when using (3,3) or (5,5) will be small.

2.2 Regularization

Adding noise to the input images when training a network tends to reduce its capability to straight up copying every training instance to the output layer, since the input images will be a little different for every batch (different instances of

the noise is applied for every batch). The best practice for color images is to use Gaussian noise, so we'll use that for all the networks.

Another common regularization technique is Dropout, which randomly removes a faction of the neurons (and all connected weights and biases) of a layer during training. This helps to reduce over-fitting by preventing single neurons from being "over-worked". That is, we do not want the information passing through a layer to be only going through neurons with consistently high activation. If that is the case, it is an indication that the network has found some complex high-order approximation to the input data, or in other words; the network is over-fitted. By randomly dropping out neurons, we make sure weight updates is passed backwards to low-activation neurons as well when those with high activation is dropped. This technique reduces over-fitting and helps the network generalize better to unseen data, so we'll implement it in our classification networks.

2.3 Data augmentation

By performing random geometric transformations on training images, we can increase the variability and/or size of our training set. Typically data augmentation is used to transform batches of training images in-place, such that there are many variations of the original images throughout the training epochs. Since we'll have three underrepresented classes, we can perform data augmentation on only those three classes to increase their training size to that of the other classes. A powerful and simple augmentation method is to perform an horizontal flip. Given that the images are not symmetrical about the central vertical axis (they tend not to be, by looking at the examples in Fig. 2), these new images present valuable information about the classes we want to identify.

3 Experimental results

For all subsequent analysis, we perform sample-wise standardization of the training,validation and test images, for both the autoencoder and classification networks. This is because we don't want the network to be influence of the brightness of the images when performing classification; a frog in dark environments should be as easily recognized as a frog in the light.

3.1 Visualizing filters and feature maps

We found no significant difference in the classification results between using 5x5 and 3x3 filters, but the 5x5 filters were more visually interpretable as many of the 3x3 ones appeared rather random. In Fig. 3 we show 100 of the filters from the first constitutional layer. Every filter is min-max normalized, so the color of the image indicates which color in the input the filter reacts strongest

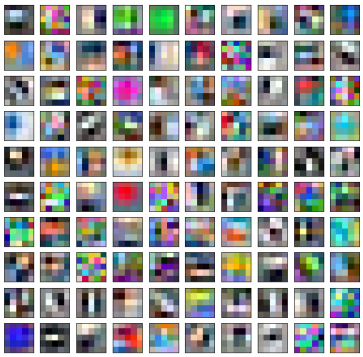

Figure 3: Here we show 100 of the filters obtained from the first constitutional layer when using a 5x5 kernel size.

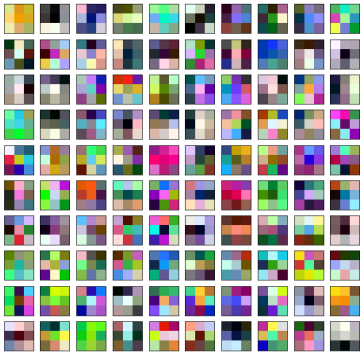

Figure 4: Here we show 100 of the filters obtained from the first constitutional layer when using a 3x3 kernel size.

to. We see a good amount of black-white filters with straight lines, which we can be interpreted as performing general edge detection since they react equally to all the input color channels. In comparison, the 3x3 filters shown in Fig. 4 seems much more random, although we see some structure in many of them. To be clear, just because a filter is visually appealing to us, that does not mean they are more valuable to the network; a deep neural network is a non-linear universal approximator and can find complex mappings between the inputs and outputs.

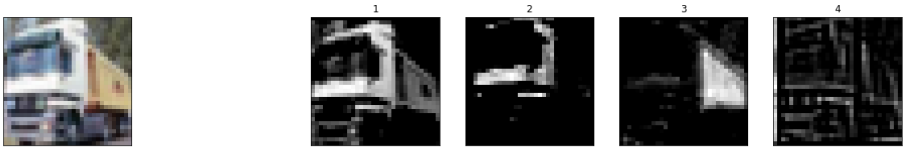

Figure 5: Example feature maps from the first convolutional layer in the autoencoder network.

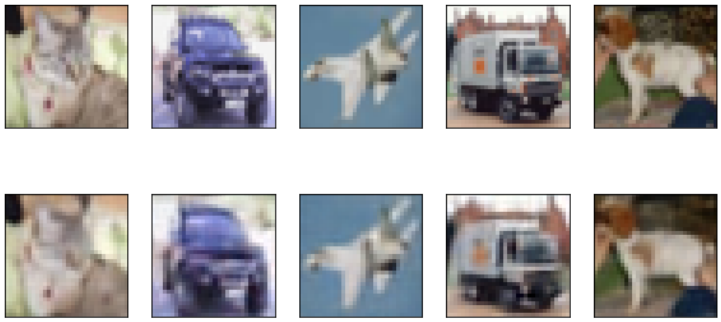

Figure 6: Examples of Cifar10 images (top row) compared with their autoencoder reconstruction (bottom row).

Some selected examples of the feature maps obtained by applying the previous filters is shown in Fig. 5. The first filter (1) detects the white colored parts of the original image (shown to the left). The two filters (2) and (3) is activated by the chassis and the cargo box, respectively. Finally, the last filter shown (4) seems to be a general edge detector, which highlights all the borders between the colors in the original image. Although the main goal of this exercise is to perform classification, we also plot examples of the image reconstruction achieved by the autoencoder in Fig. 6.

3.2 Classification

For the classifier networks we reuse the first five layers for the classification network (i.e. the encoder block), and add a fully connected dense layer of 256 neurons with ReLU activation, as well as a final layer of 10 neurons with the softmax activation function for classification. We also add dropout layers before each of the dense layers for regularization. When we are training the pretrained

classifier we copy the weights from the autoencoder, while for the randomly initialized classifier we use the same network architecture but without copying weights from the autoencoder.

As a baseline to compare our results to, we first train the autoencoder and classifier networks on the full Cifar-10 dataset (all 60 000 images). We then remove 50% of the images for the three classes specified (bird, deer and truck) and run the following experiments:

1. **Majority downsampling:**
   Remove images from the majority classes as well, such that we have 30 000 images in a balanced dataset.

2. **Minority upsampling**:
   Use data augmentation to upsample the minority classes in the training set. We create new images for the minority classes by including horizontally flipped versions of the ones we already have. The testing set is not augmented; instead we remove 50% of all classes to balance it.

3. **Class weights:**
   Keep the dataset unbalanced, and use higher weights for the minority classes in the cost function.

In all cases, we set aside 5% of the data for validation and 10% for testing. Data augmentation was used to sample-wise standardize the images for all networks, and for the classifiers we also included 15 degrees random rotations, 10% vertical and horizontal shifts, as well as horizontal flips for all networks except for the majority up-sampling method (since we already have upsampled the minority classes using horizontal flip). We trained all the networks for 20 epochs using a batch size of 128 for the classifiers and 64 for the autoencoder, which was enough for the loss to converge. We repeated the experiment 10 times for every method, and the average accuracy on the test set is shown in Table 2.

	Full set	**Min. up**	**Maj. down**	**Class weights**
# train imgs	47.5k	47.5k (/w 7.5k aug.)	23.75k	40.37k
Pretrained	76.98±0.63%	76.41±0.51%	73.37 ± 0.81%	74.67 ± 0.43%
Rand. init	76.72±0.42%	76.17±0.91%	73.73 ± 0.43%	74.18 ± 0.36%

Table 2: Table summarizing the average test-accuracy of all the methods.

All methods employed resulted in high test accuracy, comparable to the full data-set baseline. The minority up-sampling method consistently out-preformed the majority downsampling method by about $\sim 3\%$, and the class weighting by about $\sim 2\%$, so we can consider that as the best of the methods we employed. We also calculated the precision and recall of all the classes for every method, and although the results were not homogeneous for all the classes, we did not

observe any significant variations between the methods. That is; even when performing the upsampling or class weighting method, the precision and accuracy were comparable to that of the full data-set baseline, even for the minority classes.

4 Conclusions

We have used a stacked fully convolutional autoencoder for pretraining of a classifier network on a modified Cifar-10 dataset where 50% of the images is removed for three of the ten classes. To account for the unbalanced dataset, we employed three different methods: upsampling of the minority classes, downsampling of the majority classes, and weighting the cost function. In neither of the three methods did we observe a significant improvement by pretraining the network. In [3] they observed a small but significant improvement by pretraining, however the network they employed used a scaled tanh function for activation of the autoencoder. In [1] the results from [3] is reproduced with similar outcome, however they mention that by using ReLu activation the pretrained network did not outperform the randomly initialized one.

The key point to take away from this project is essentially an appreciation of the power of data augmentation. Any possible benefit of pretraining the classifier is washed out by the power of augmentation. When augmentation was turned off, we observed substantial overfitting to the training set. In theory, pretraining would be more beneficial for deeper neural networks than that what we've employed, but due to computational limits we had to stick with a rather shallow one. Another common usecase where pretraining could be beneficial, is when we have a large amount of data where only a small subset is labeled. We can then pretrain an autoencoder on the full data-set, and train the classifier on the labeled subset.

References

[1] Maximilian Kohlbrenner, Russell Hofmann, Sabbir Ahmmed, and Youssef Kashef. Pre-training cnns using convolutional autoencoders. 2017.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[3] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.