Ridge-i Technical Assignment

Tiago Oliveira

November 22, 2019

1 Introduction

Autoencoders are a special case of neural networks. An autoencoder is trained to copy its input to its output. Therefore, an this type of modelis viewed as consisting of two parts: an encoder($h = f(x)$) and a decoder ($g(f(x)) = x$). Usually, autoencoders are restricted so as to be only able to approximately copy the input in order to make them learn useful features from data.

Regarding the size of h, one can distinguish two types of autoencoders: undercomplete, when h has a smaller dimension than x; and overcomplete when h has a higher dimension than x. Herein, an overcomplete autoencoder is used.

Several works point out the usefulness of autoencoders in multiple computer vision tasks, such as anomaly detection, image denoising, or even image classification. The encoded layer of an autoencoder provides a high level representation of the data in its feature maps, which can be useful in a classification task. To test this, a Convolutional Autoencoder (CAE) is trained on the CIFAR10 dataset, the decoder is disposed and the output of the encoder is used as the input of a Convolutional Neural Network (CNN) for image classification.

2 Task Description

The task includes the design of a network that combines supervised and unsupervised architectures in one model to achieve a classification task. The model must start with an autoencoder which is then connected through its hidden layer to another network as shown in Figure 1. The autoencoder takes as input an image at node 1 and reconstructs it at its output at node 3. It creates valuable features at its hidden layers at node 2. It is hypothesized that if node 2 is used as input for the CNN (node 4), then the classification can be improved.

3 System Description

The models in this project were trained in a system with the following specifications:

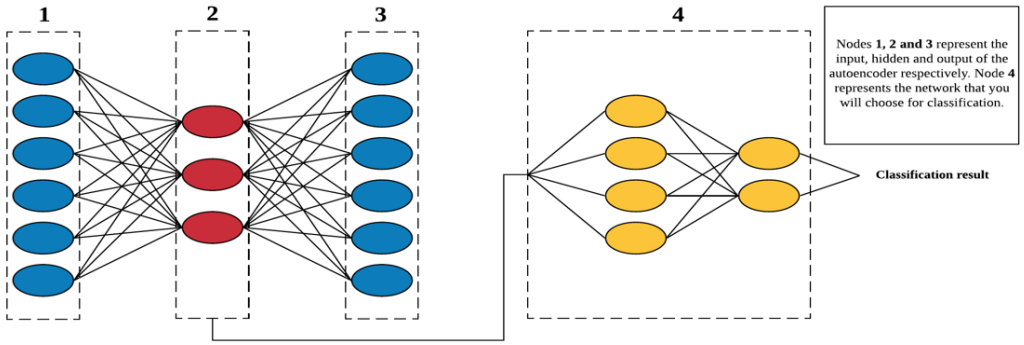


Figure 1: Abstract diagram of the semi-supervised model for the classification of CIFAR10 images.

- **GPU:** 1xTesla K80, having 2496 CUDA cores, compute 3.7, 12GB(11.439GB Usable) GDDR5 VRAM;
- **CPU:** 1xsingle core hyper threaded (1 core, 2 threads) Xeon Processors @2.3Ghz (No Turbo Boost), 46MB Cache;
- **RAM:** 12.5 GB available;
- **Disk:** 310 GB available.

In terms of python modules, the following were used in this project:

- Keras 2.2.5;
- Tensorflow 1.15.0;
- Scikit-learn 0.21.3;
- NumPy 1.17.4;
- Matplotlib 3.1.0;
- Json 2.0.9;
- Pandas 0.25.3.

4 Dataset Description

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains 1000 randomly-selected images from each class. The

Class	Training Set	Validation Set	Test set
airplane	3910	1090	1000
automobile	3891	1109	1000
bird	1944	556	10000
cat	3874	1126	1000
deer	1981	519	10000
dog	3913	1087	1000
frog	3894	1106	1000
horse	3889	1111	1000
ship	3905	1095	1000
truck	1949	551	1000
Total	33150	9350	10000

Table 1: Number of samples per class in training, validation and test sets.

training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain 5000 images from each class.

Training and test batches can be easily downloaded with the *load()* function from the *cifar10* module of Keras. This function returns the five training batches consisting of 50000 images in one dataset and a test batch consisting of 10000 images in another dataset.

5 Pre-processing

After loading the CIFAR-10 data, 50% of images from classes bird, deer, and truck were randomly removed from the training batches. These batches were then split into a training set and a validation set, both with approximately the same class proportions as the training batches after removal, i.e., bird, deer, and truck each have approximately half as many images as each of the remaining classes. The percentage of images in the training batches dataset used for validation was set at 22% so that the validation set would have approximately the same number of samples in each class as the test set (except in bird, deer, and truck).

The following step was the normalisation of pixel values to the [0,1] interval by diving each image pixel in the training, validation and test sets by the maximum pixel value found in the training set. Typically, this maximum pixel value is 255. This was also the case herein. The number of samples in each class and the total number of samples in the training, validation and test sets are described in Table 1.

6 Performance Metrics

The fact that the model used for this multiclass classification task is an ensemble of two different models, a supervised model and an unsupervised model, means that their respective training and validation performances were assessed with different performance metrics. The CAE was assessed with Mean Squared Error Loss (MSE_{loss}) as we are only interested in monitoring the pixel-wise difference between the input and the output. The loss function used in the classifier was Categorical Cross-entropy Loss (CCE_{loss}), typical of a multiclass classification tasks. The fitness of the classifier in was assessed with accuracy, precision, recall and f1-score. These performance metrics are formally defined in the ensuing sections.

6.1 Mean Squared Error Loss

MSE_{loss} is defined as:

$$MSE_{loss}(y, \hat{y}) = \frac{1}{m}\frac{1}{p}\sum_{j=1}^{m}\sum_{i=1}^{p}(\hat{y}_i - y_i)^2, \tag{1}$$

where m is the number of image samples, p is the number of pixel predictions per sample, y is the ground truth for the prediction and $\hat{y}$ is the model prediction.

6.2 Categorical Cross-entropy Loss

CCE_{loss} is defined as:

$$CCE_{loss}(y, \hat{y}) = \sum_{j=1}^{m}\sum_{i=0}^{n}(y_{ij} * log(\hat{y}_{ij})), \tag{2}$$

where m is the number of predictions, n is the number of different classes, y is the ground truth for the prediction and $\hat{y}$ is the model prediction.

6.3 Accuracy, Precision, Recall, and F1-score

In a multiclass classification problem such as this one, the accuracy of a model is defined as follows:

$$Accuracy = \frac{1}{m}\sum_{c=1}^{n} TP_c, \tag{3}$$

where m is the total number of samples in the dataset, n is the total number of classes, and TP_c is the number of true positives with repect to class c.

Precision for a single class is defined as:

$$Precision_c = \frac{TP_c}{TP_c + FP_c}, \tag{4}$$

where c is a class, TP_c is the number of true positives with respect to class c, and FP_c is the number of false positives with respect to class c.

Recall for a single class is defined as:

$$Recall_c = \frac{TP_c}{TP_c + FN_c}, \tag{5}$$

where c is a class, TP_c is the number of true positives for class c, and FN_c is the number of false negatives for class c.

Finally, the f1-score combines precision and recall as follows:

$$F1 - score_c = 2 * \frac{Precision_c * Recall_c}{Precision_c + Recall_c} \tag{6}$$

where c is a class, $Precision_c$ is the precision of the model for class c, and $Recall_c$ is the recall of the model for class c.

7 Modelling

7.1 Autoencoder

The type of autoencoder chosen for this project was a fully convolutional autoencoder. This type of CAE does not contain dense any layers. The CAE first uses convolutions and pooling layers to transform the input into a high dimensional feature map representation and then reconstructs the input using convolutions and up-scaling (in common configurations). The advantage of CAEs over fully connected autoencoders (AEs) resides in the fact that the former consider 2D relations whilst the latter do not.

Figure 2 shows the architecture of the CAE used in this project. The encoder has 3 convolution blocks, each one with two convolution layers. At each convolution block the number of filters is doubled. Every convolution layer is a 3x3 filter with padding and defined with a ReLU activation.The convolution layer is then followed by a batch normalisation layer (which is not represented in Figure 2 for simplicity) in order to stabilise and accelerate the learning process. The first and second convolution blocks end with with a max-pooling layer that halves the size of feature maps in order to localise the object and to reduce the computational cost. As shown in 2, the CAE is asymmetric. This design was also an attempt to reduce computational cost with smaller feature maps on the decoder side. The decoder consists of 3 convolution blocks, each with two convolution layers with ReLU activations and, in blocks 2 and 3, an up-sampling layer at the end. The decoder ends with a convolution layer which produces an image with the same dimensions as the input 32x32x3. Similarly to the encoder, after each convololution layer there is a batch normalisation layer. The outut layer uses a Sigmoid activation function.. The whole model is compiled with a MSE_{loss} loss function.

Various optimisers were tested to train the CAE. The optimiser responsible for the best convergence within 50 epochs (the amount of epochs set for training the autoencoder) was the Adam optimisers with a learning rate of 0.001. The amount of epochs for training the

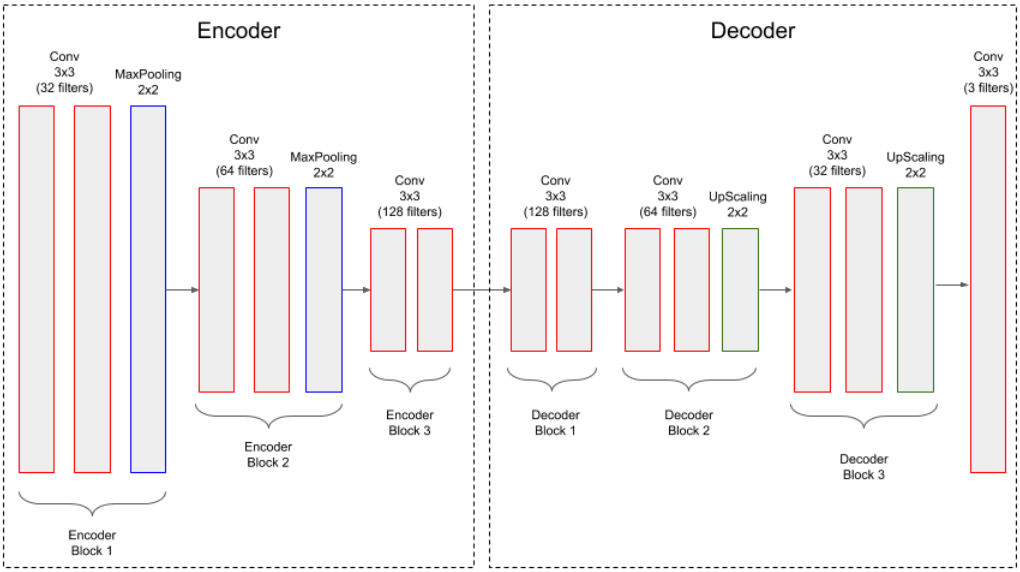


Figure 2: Asymmetric CAE with encoding size 8x8x128.

CAE were limited due to the size of the network leading to a long training time per epoch. Additionally, it was observed that large batch sizes were harmful to convergence, so a small batch size of 64 was used.

In order to increase the robustness of the autoencoder, prevent it from simply copying the input to the output and force it to learn meaningful features, three possibilities were considered:

- Dropout: add a dropout layer at the end of each convolution block of the encoder;
- Gaussian noise : add a gaussian noise layer after the input, or after each convolution layer of the encoder, or both;
- Activity regulariser: define an l1 activity regulariser at each convolution layer of the encoder in order to create sparse activations.

The results of exploring these three forms of regularisation are described in Section 8. No regularisation was implemented in the decoder so as to not compromise its ability to reconstruct the images and recover from prior regularisation in the encoder.

7.2 Classifier

The architecture of the classifier is depicted in figure 3. The first components mirror the encoder of Figure 2 and are followed by a convolution layer with 128 filters, ReLU activation,

and padding. A max-pooling layer is then applied to reduce feature map size and computational cost during training. The resulting feature maps are then flattened and submitted to two dense layers with ReLU activations before the output layer. The output is produced by 10 nodes, each standing for one of the classes in the classification task. This layer has a Softmax activation function. The model uses CCE_{loss} as loss function.

In order to leverage the encoding produced within the autoencoder and feed it to the remaining layers in the model for classification, the layers in the components that mirror the encoder are set to non-trainable and have the pre-trained weights of the encoder loaded onto them. Then, the rest of network is trained in order to produce classification predictions. Again, a small batch-size of 64 and Adam optimiser with learning rate of 0.001 were used. In a first stage, the classifier was only trained during 20 epochs in order to check the effects differently trained autoencoders would have on it. These results are disclosed in Section 8.

After the best form of autoencoder regularisation was established. The weights of the encoder portion of the selected autoencoder were used to build a classifier to be trained during 100 epochs. The layers corresponding to the encoder were kept as non-trainable.

In order to prevent the classifier from overfitting, reduce validation loss and improve accuracy while using the pre-trained weights of the encoder, the following regularisation techniques were considered:

- Dropout: add dropout layers with dropout value of 0.2 to the portion of the classifier that is not part of the encoder, i.e, after the convolution layer and dense layers, except for the penultimate and last layers;
- Real-time data augmentation: the types of implemented augmentations were width shift of 0.1, height shift of 0.1, 40 degree rotation, zoom range of 0.1, and horizontal flip;
- Class weights: include class weights according to respective number of samples in the computation of the loss function to compensate for class imbalance.

8 Results and discussion

The following autoencoders were trained during 50 epochs:

- baseline autoencoder with no regularisation;
- with dropout layers with 0.2 dropout value after each convolution block;
- with dropout layers with 0.4 dropout value after each convolution block;
- with a gaussian noise layer with standard deviation of 0.1 added after the input;
- with gaussian noise hidden layers with standard deviation of 0.1 added after each hidden convolution layer of the encoder;

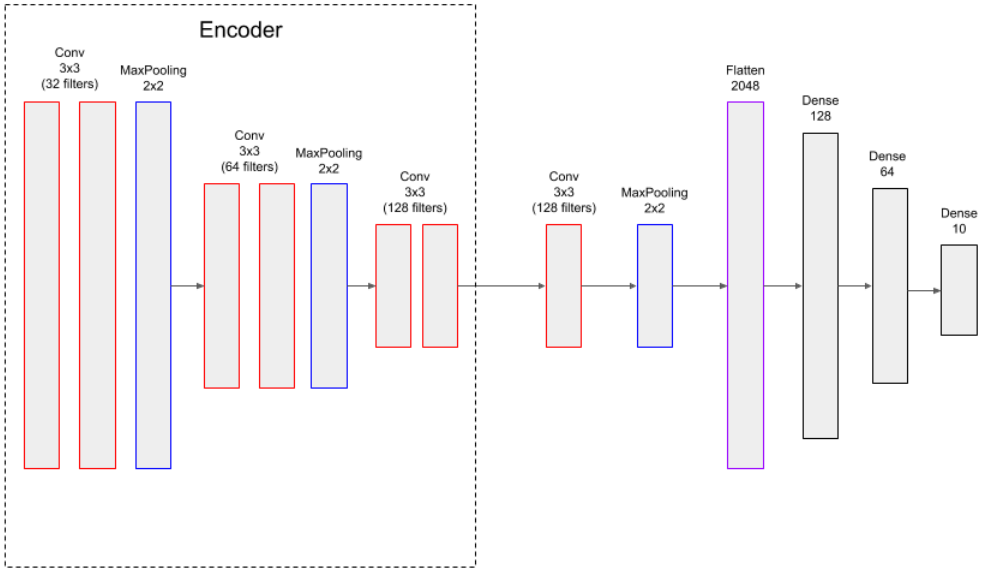


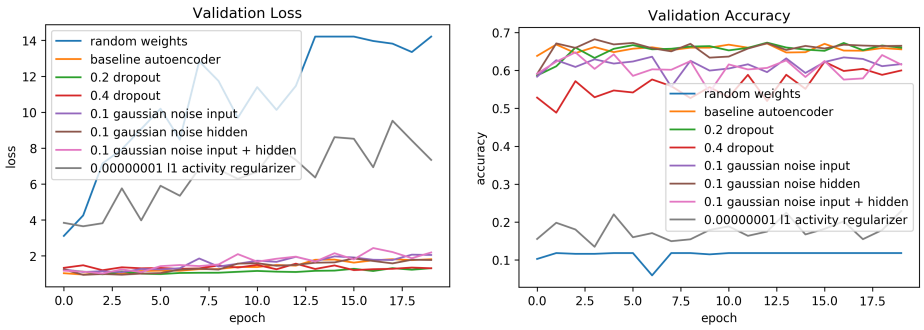
Figure 3: Classifier architecture.

- with gaussian noise layers with dtandard deviationod 0.1 added after the input and after each hidden convolution layer;
- with l1 activity regulariser with $\lambda = 1e^{-8}$ at each convolution layer of the encoder.

Besides training classifiers with pre-trained weights of the encoders of the above-mentioned autoencoders, a classifier with randomly generated weights was trained from scratch with all layers set to trainable. For comparison their respective validation losses and accuracies are shown in Figure 4

It is possible to observe in Figure 4a that the classifiers using encoder pre-trained weights present significantly lower validation losses than the classifier trained from scratch initialised with random weights within 20 epochs. It is also true that these models show higher validation accuracy values in Figure 4b. This corroborates the hypothesis that autoencoders improve classification. Amongst the models using pre-trained encoder weights, the model with 0.2 dropout presents a relatively lower loss while having a high accuracy when compared to the remaining models. Based on this observation, the classifier with 0.2 dropout was selected to move forward as the new baseline classifier.

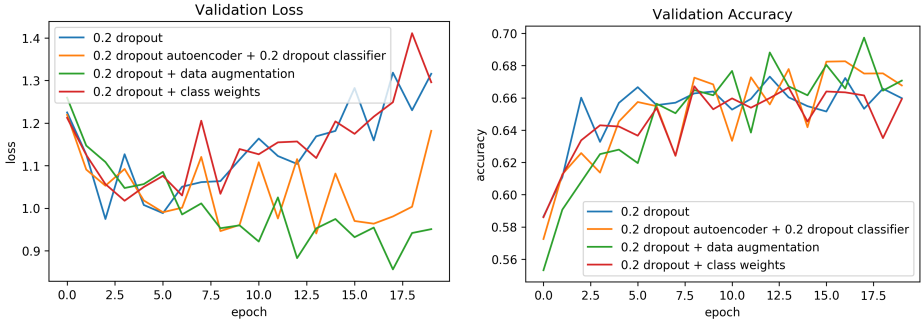
With the new baseline classifier, the methods to prevent overfitting mentioned in Section 7.2 were applied, yielding the results shown in Figure 5. The classifier using real-time data augmentation seems to be the only model that is converging, as its loss looks to be on the way of decreasing even more. Additionally, this was also the classifier that reached the highest validation accuracy. Surprisingly, the classifier with weighted classes showed a



(a) Validation loss. (b) Validation accuracy.

Figure 4: Validation metrics for classifiers using weights from different autoencoders.

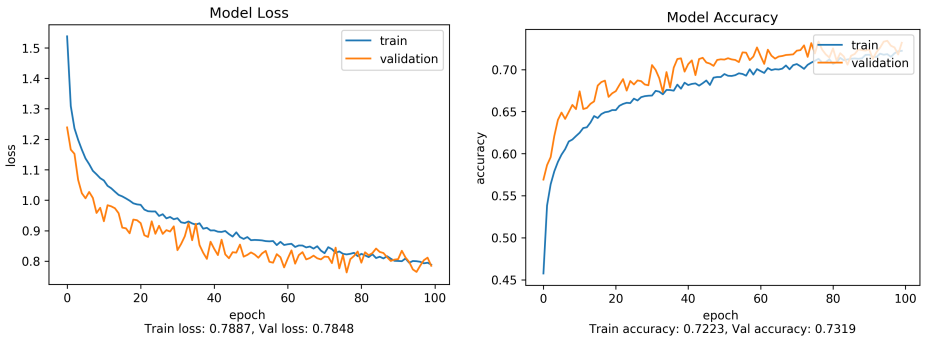
significant loss increase, which was contrary to expectation.



(a) Validation loss. (b) Validation accuracy.

Figure 5: Validation metrics for different classifiers using weights from an autoencoder with a dropout value of 0.2.

Based on the results of Figure 5, the classifier using real-time data augmentation was chosen as the optimal classifier to be trained for 100 epochs. Training and validation metrics for this part of the experiment are shown in Figure 6. It is possible to observe that the model is not overfitting and could have had converged further if trained with more epochs. The results show that training and validation losses are converging together. According to the classification reports of Tables 2 and 3 the validation and test f1-scores are very similar, which suggests a very close performance in validation and testing. The problem of imbalanced classes still persists and, unsurprisingly, the precision, recall and f1-score are significantly lower for classes bird, deer, and truck. Additionally, these measures for the cat class, which was not supposed to be unbalanced, are also low. This may be due to the slightly lower number of cat images in the training set that resulted from the random splits.



(a) Training and validation loss. (b) Training and validation accuracy.

Figure 6: Training and Validation metrics for classifier with real-time data augmentation and using weights from an autoencoder with a dropout value of 0.2.

Class	**Precision**	**Recall**	**F1-score**
airplane	0.82	0.81	0.82
automobile	0.91	0.80	0.85
bird	0.55	0.56	0.56
cat	0.59	0.52	0.56
deer	0.59	0.61	0.60
dog	0.67	0.63	0.65
frog	0.63	0.91	0.74
horse	0.82	0.79	0.81
ship	0.86	0.86	0.86
truck	0.81	0.64	0.71
Accuracy			0.73
Macro Avg	0.73	0.71	0.71
Weighted Avg	0.74	0.73	0.73

Table 2: Classification report of validation data for the classifier with real-time data augmentation and using weights from an autoencoder with a dropout value of 0.2.

9 Conclusions

With this work it was possible to verify the hypothesis that autoencoders can improve classification results, particulary in the early experiments described in this document. However, a more thorough comparison between the final model and a similar model without pre-trained weights, trained from scratch, was warranted but, unfortunately, not executed. Additionally, a stronger effort in order to achieve higher performance in classes bird, deer, and truck was required. Techniques such as on-batch data balancing could have been useful in this situation.

Class	Precision	Recall	F1-score
airplane	0.74	0.81	0.78
automobile	0.86	0.80	0.83
bird	0.69	0.54	0.60
cat	0.56	0.52	0.54
deer	0.72	0.61	0.66
dog	0.65	0.67	0.66
frog	0.56	0.92	0.69
horse	0.76	0.78	0.77
ship	0.85	0.84	0.84
truck	0.89	0.65	0.75
Accuracy			0.71
Macro Avg	0.73	0.71	0.71
Weighted Avg	0.73	0.71	0.71

Table 3: Classification report of test data for the classifier with real-time data augmentation and using weights from an autoencoder with a dropout value of 0.2.

Regarding the hyperparameters suggested for the autoencoder, we have the following:

- Batch size: 64;
- Epochs: 50;
- Optimiser: Adam;
- Learning rate: 0.001;
- Regularisation: dropout layer with 0.2 dropout value after each convolution block ;

As for the classifier, the suggested hyperparameters are:

- Batch size: 64;
- Epochs: 100;
- Optimiser: Adam;
- Learning rate: 0.001;
- Data augmentation: real-time data augmentation with width shift of 0.1, height shift of 0.1, 40 degree rotation, zoom range of 0.1, and horizontal flip;

One way to further improve the selected model would have been to set all layers to trainable and use the encoder weights and the newly learnt weights of layers near the output as initial weights to re-train the whole classifier.