# Neural Networks - Working with SVMs

Lazaros Gogos
3877
lazarosg@csd.auth.gr

December 20, 2023

Aristotle University of Thessaloniki

# Contents

# 1 Introduction

We are working with the CIFAR-10 dataset. The framework I chose to go with is *keras*. The aim of this paper is to showcase different approaches to utilizing SVMs in image classification tasks. The architectures we're going to explore are fitting an SVM on raw data, data after PCA, but also fitting SVMs on the internal representations of Deep Neural Networks.

Both *RBF* and *linear* kernels are tested to see which one results in better performance.

Also, we test to see whether a neural network can be trained if hinge loss is used as a loss function.
Below, in figure 1-1, we can see 10 random images taken straight from the dataset, and their respective labels.


Figure 1-1: Ten random images from the CIFAR-10 database of images

This report goes hand-in-hand with the python notebook accompanied by it, and its corresponding chapters.

Also, the models needed to be pre-loaded are also located in the same folder. All that would be needed to run the notebook is to change the `prefix` variable in the `Initial Code` section.

# 2 Hinge loss sequential model

## 2.1 Would a CNN work, if the loss function was hinge loss?

Before working with SVMs, it's worth seeing whether training a Convolutional Neural Network with a hinge loss function would work in classifying the images, in order to simulate what an SVM would behave like.

After training a convolutional model, we can see its architecture and its behavior during 50 epochs of training.

```
Model: "cnn_hinged"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 3x3__32_filters (Conv2D)     (None, 30, 30, 32)        896
 3x3__32_filters__2 (Conv2D) (None, 28, 28, 32)        9248
 2x2 (MaxPooling2D)           (None, 14, 14, 32)        0
 0.25 (Dropout)               (None, 14, 14, 32)        0
 3x3__32_filters__3 (Conv2D) (None, 12, 12, 32)        9248
 3x3__32_filters__4 (Conv2D) (None, 10, 10, 32)        9248
 2x2__2 (MaxPooling2D)        (None, 5, 5, 32)          0
 0.30 (Dropout)               (None, 5, 5, 32)          0
 flatten (Flatten)            (None, 800)               0
 64neurons (Dense)            (None, 64)                51264
 0.50 (Dropout)               (None, 64)                0
 10neurons (Dense)            (None, 10)                650
=================================================================
```

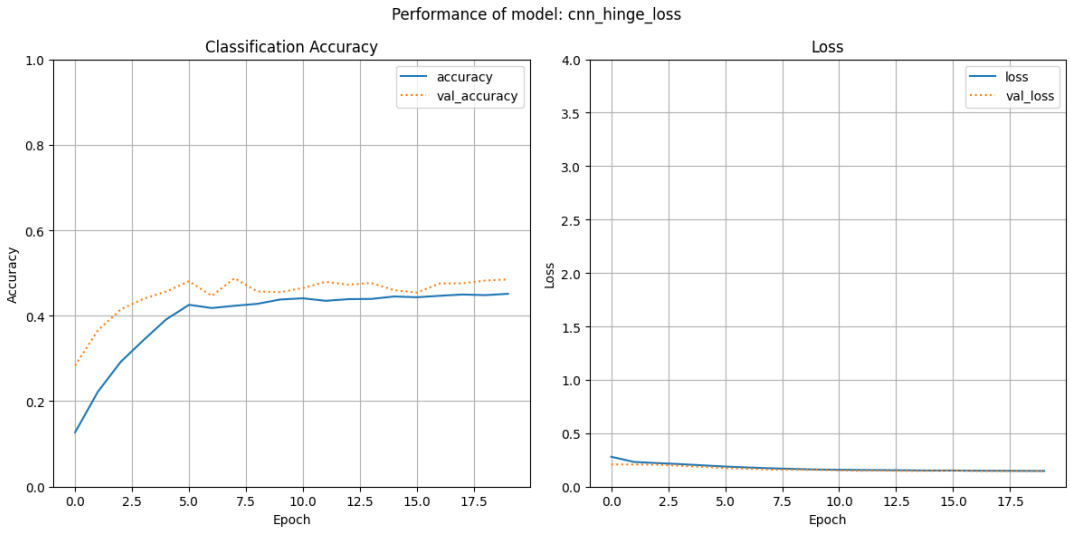


Figure 2-2: Plots of accuracy and loss over 50 training epochs

## 2.2 Observations

The classifier reaches its limits and cannot be trained any more, as seen in figure 2-2. Its peak accuracy is 48.51% and its loss is 0.14 after 20 epochs of training.

# 3 Support Vector Classifier - SVC

## 3.1 What we already have

We have a convolutional model which is already trained on our dataset. The model's architecture looks like this.

```
Model: "Surpassed128CNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 3x3__32_filters (Conv2D)    (None, 30, 30, 32)        896
 3x3__32_filters__2 (Conv2D) (None, 28, 28, 32)        9248
 2x2 (MaxPooling2D)          (None, 14, 14, 32)        0
 0.25 (Dropout)              (None, 14, 14, 32)        0
 3x3__32_filters__3 (Conv2D) (None, 12, 12, 32)        9248
 3x3__32_filters__4 (Conv2D) (None, 10, 10, 32)        9248
 2x2__2 (MaxPooling2D)       (None, 5, 5, 32)          0
 0.30 (Dropout)              (None, 5, 5, 32)          0
 flatten (Flatten)           (None, 800)               0
 128neurons (Dense)          (None, 128)               102528
 0.50 (Dropout)              (None, 128)               0
 10neurons (Dense)           (None, 10)                1290
=================================================================
```

Let's take that model and use its second-to-last layer to get the internal representation the model has of each image. How do we do that? We ask for the model to predict the output based on that image, and keep the outputs in a variable.

## 3.2 Plan and execution

Afterwards, we train a Support Vector Machine using these representations, in order to classify the images.
Let's get to the details.

In order to make the training time faster, we sample randomly $\frac{1}{5}$th of our data, resulting in 10,000 images. Then, a grid search is performed to find a good enough combination of $C$, $\gamma$.

## 3.3 Implementing an SVC on top of a CNN

Our initial search for a good combination is in the range C=$[2^{-7}, 2^{-5}, ..., 2^{13}]$ and $\gamma$=$[2^{-15}, 2^{-13}, ..., 2^{3}]$. The heatmap found in figure 3-3 is produced, where the values closer to white mean a better accuracy score, whereas the more black the color, the more the accuracy is close to 0. This grid search took 1 and a half hours to complete.

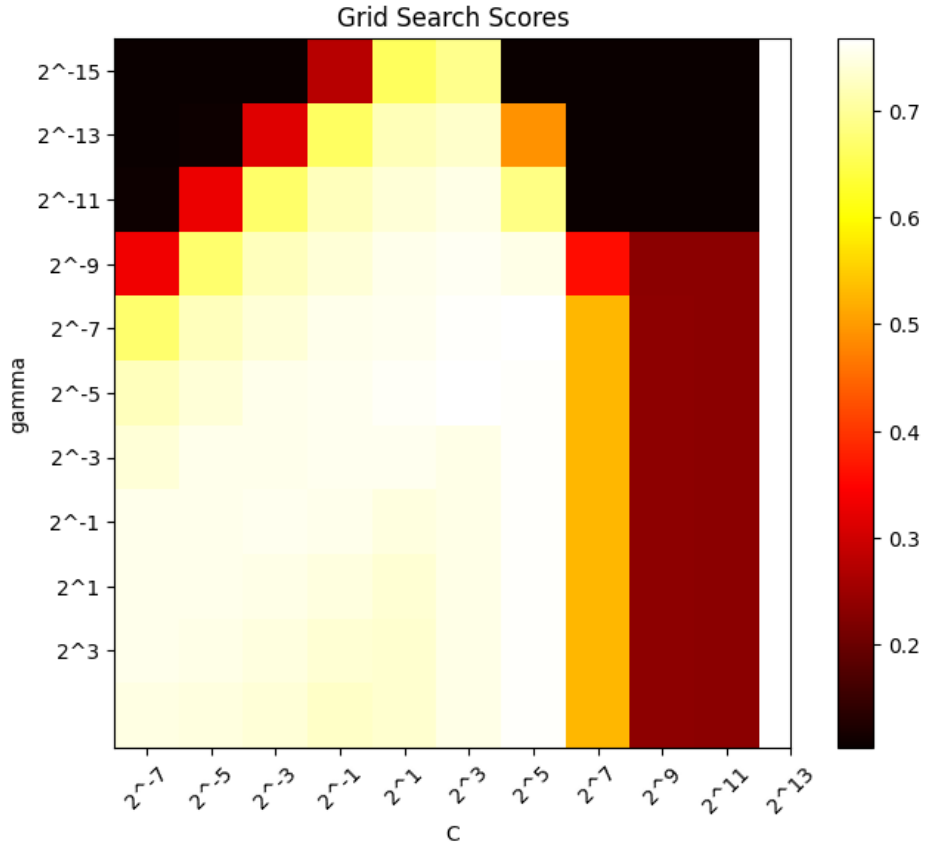


Figure 3-3: A heatmap, representing the accuracies over combinations of C, $\gamma$

We can see that the whiter values exist for C=32, but as for the value of $\gamma$, it cannot be determined which one is the best.

So, let's perform a narrower grid search, and see what combination is the best one. The values that will be tested are the following: C=$[2^{-1}, 2^{-0.75}, ..., 2^{2.75}]$ and $\gamma$=$[2^{-5}, 2^{-4.75}, ..., 2^{-1.25}]$. This grid search took 3 hours to complete. Results found in figure 3-4.

Having found this combination, we can train a Support Vector Classifier for our whole dataset, which takes about 5 minutes. Both the one-versus-one and the one-versus-all approaches give the same results. This has to do with the implementation of SVMs in scikit-learn. After 5 minutes of training, both the OVR and the OVO strategy yield the same accuracy, **73.58%**.

### 3.3.1 A note on OVR and OVO being the same

The reason why One-Versus-Rest and One-Versus-All accuracies are the same is that, under the hood, `sklearn.svm.SVC` always performs OVO and then the best classifier is voted. That is the why the OVR and OVO results are identical. More info can be found here.

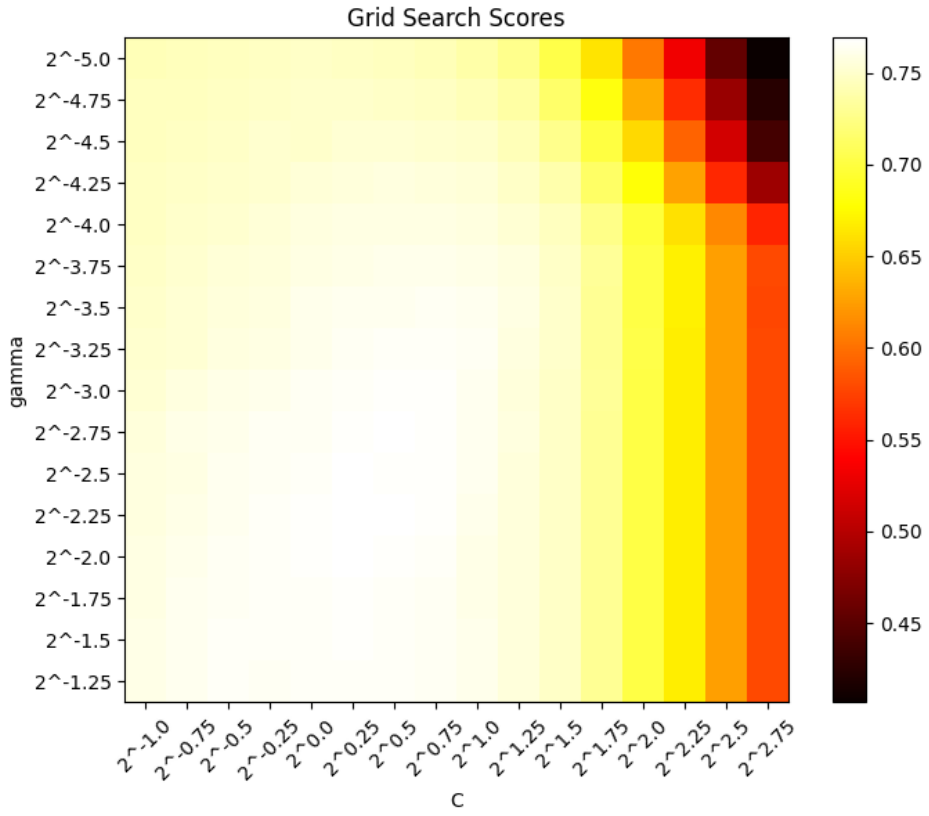


Figure 3-4: Finer grid search

# 4 Linear SVC

## 4.1 The approach

Let's try a different approach, by utilizing a linear kernel instead of an RBF one. This time, instead of using `sklearn.svm.SVC` we will use `sklearn.svm.LinearSVC`. Why would we do that?

The reason is simple. A linear SVC converges faster when the number of samples is a lot bigger than the amount of features. In our case, the number of features is 3072, whereas the actual amount of samples is 50,000. This guarantee in performance holds only if we solve the primal problem (n_samples > n_features).

## 4.2 Strategy and implementation

Again, the procedure is the same. We go for a grid search based on a smaller sample of our data (10K images), find a good-enough `C` value, do a finer grid search, and find the best `C` there is.

Initially, we search for values of `C` in $[2^{-12}, 2^{-11}, ..., 2^{11}]$. After only 3 minutes of training, the following scores are observed, found in figure 4-5.

After establishing the best value of C is somewhere close to $2^{-4}$, as seen in figure 4-5, we perform a narrower grid search to find the best value of `C`, in the range $[2^{-4}, 2^{-3.75}, ..., 2^{3.75}]$.

Even though it is not easily seen here, the resulting scores are the following:
$[0.7381, 0.738, 0.7381, 0.738, 0.7385, 0.7392, 0.7397, 0.7397, 0.7401, 0.7393, 0.7393, 0.7387, 0.7393, 0.7389, 0.7389, 0.7389]$

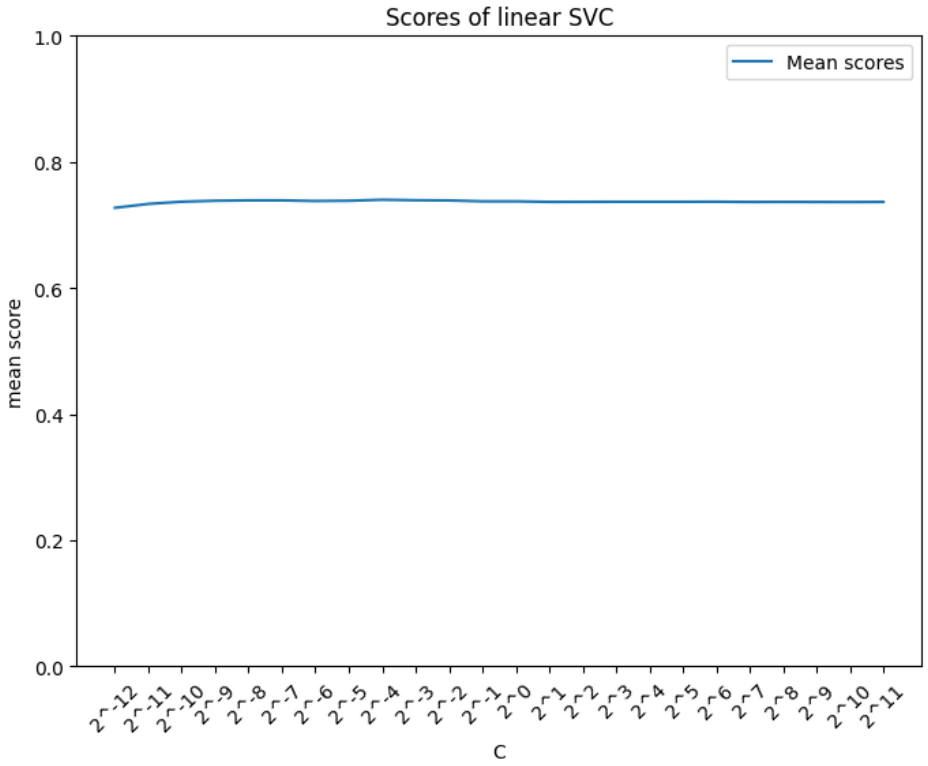


Figure 4-5: Scores of linear SVC based on different values of C

The best possible score is indeed for C=$2^{-4}$, as seen in figure 4-6. This was found after **only 1 minute** of search the grid. Comparing this to a grid search with an RBF kernel, which needed 1.5 hours plus another 3 hours for the finer grid search to complete, the difference is immeasurable, considering the difference in accuracy is just **2%**.

The final accuracy of the Linear Support Vector Classifier is 71.57%. Taking into account the massive drop in training time, the loss of 2% compared to RBF kernel parameter search, it is obvious why it would be the preferred option by many.

## 4.3 Bonus findings

What if we only had 1K samples? In that case, having less samples than features, we could train a Linear SVC, but instead of solving the problem in its primal form, we can solve it in its dual form, and test the classifier.
After performing a grid search in $[2^{-8}, 2^{-7.75}, ..., 2^{-5.25}]$, the best value of C is $2^{-6}$, and a train accuracy of 73%, after only 4 seconds of training.

Testing this estimator for all the test data, it reaches an accuracy of 69.39%.

So, a 5% drop in accuracy compared to the RBF kernel SVC, but the time needed to search for good parameters dropped from 5 hours to merely seconds.

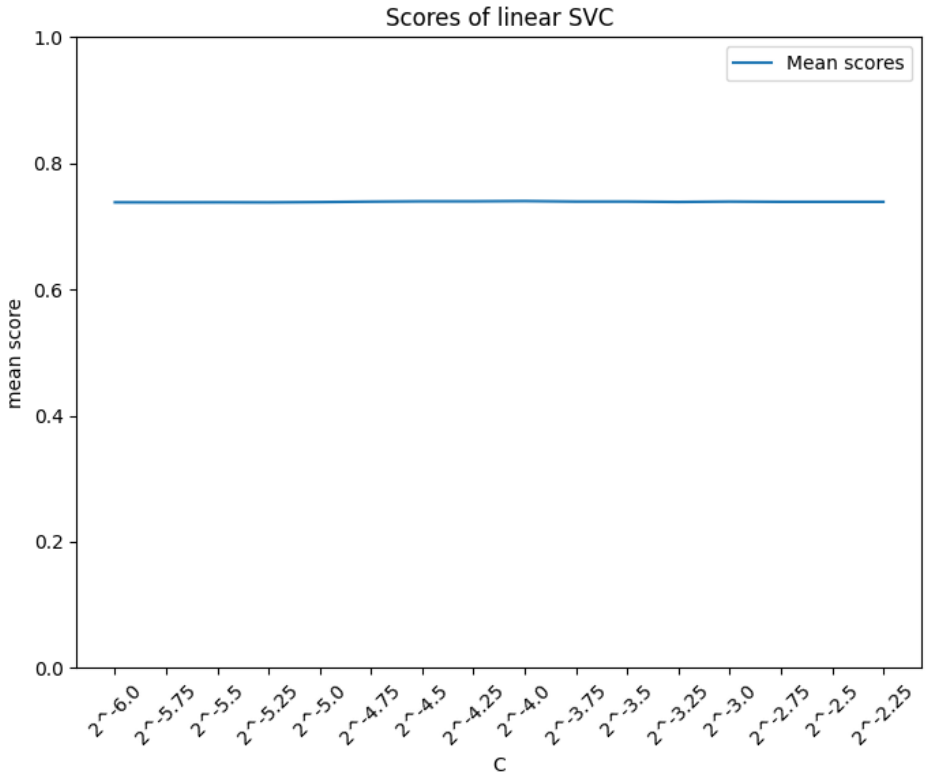


Figure 4-6: Narrow grid search scores of linear SVC based on different values of C

# 5 Combining an AutoEncoder with an SVC

## 5.1 SVM in the middle of an AutoEncoder

Another strategy that can be tested is whether an SVM would work well with the internal representations of images taken from an AutoEncoder.

Our first task is to train an AutoEncoder with our images. The model's summary we're going to train is the following.

```
Model: "ConvolutionalAutoEncoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 e64 (Conv2D)                (None, 32, 32, 64)        1792
 max_pooling2d (MaxPooling2  (None, 16, 16, 64)        0
 D)
 e128 (Conv2D)               (None, 16, 16, 128)       73856
 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)
 flatten (Flatten)           (None, 8192)              0
 middle (Dense)              (None, 256)               2097408
 dense (Dense)               (None, 8192)              2105344
 reshaper (Reshape)          (None, 8, 8, 128)         0
 d128 (Conv2DTranspose)      (None, 16, 16, 128)       147584
 d64 (Conv2DTranspose)       (None, 32, 32, 64)        73792
```

```
OUT (Conv2D)                 (None, 32, 32, 3)         1731
=================================================================
```

## 5.2 AutoEncoder training

After training this model with our whole dataset of images for 25 epochs, the accuracy and loss plots are found in figure 5-7. The model's peak accuracy is 77% in reconstructing the images. The loss function used is binary cross-entropy, and the optimizer is Adam.

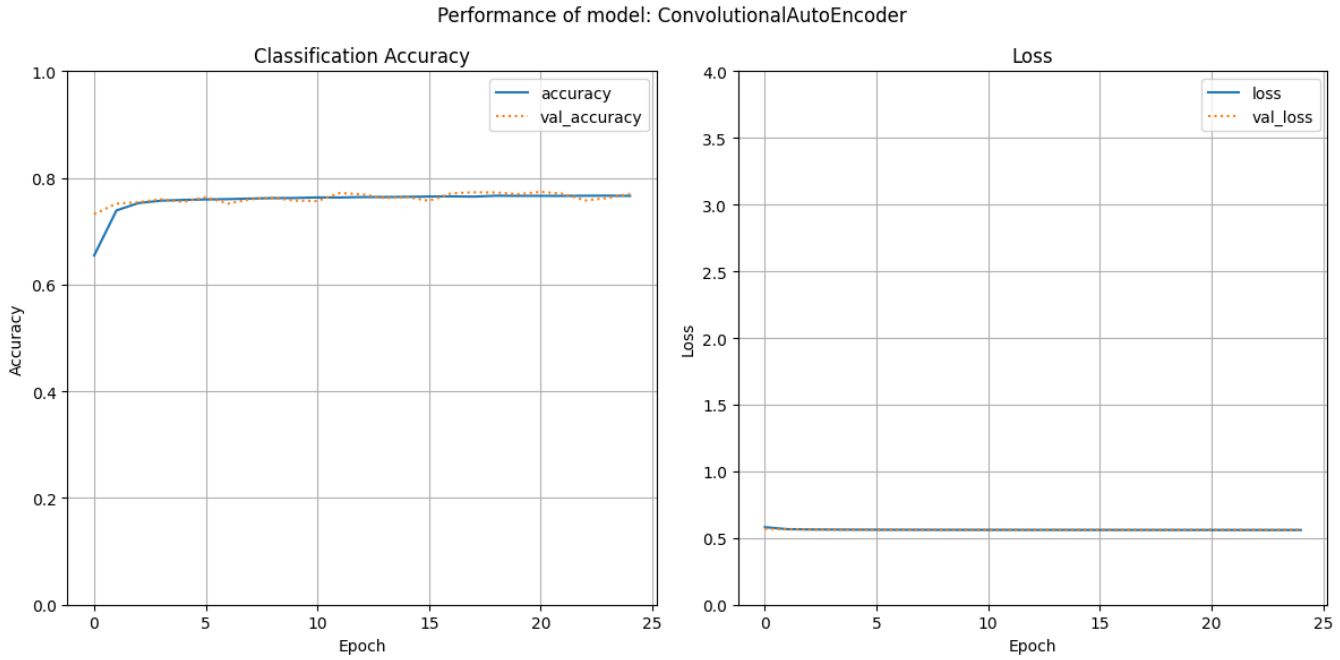


Figure 5-7: Accuracy and loss plot over the epochs of training the AutoEncoder

In figure 5-8 below, an example can be found of how well the model reconstructs the images.

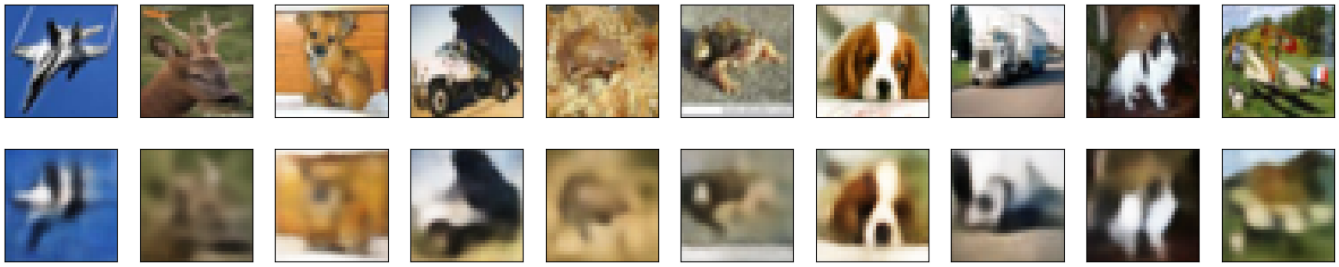

Figure 5-8: The original images and their reconstructions

## 5.3 Training the linear SVC

Let's perform a grid search in the range $[2^{-15}, 2^{-13}, ..., 2^{13}]$ to approximate the best C value. The scores that have been observed are in figure 5-9. Their numerical values are the following: $[0.3624, 0.3862, 0.3892, 0.3901, 0.3931, 0.3926, 0.391, 0.3902, 0.3906, 0.3905, 0.3905, 0.3905, 0.3905, 0.3903, 0.3904]$. We use a LinearSVC, solving the problem in its primal

form, since n_samples >n_features.

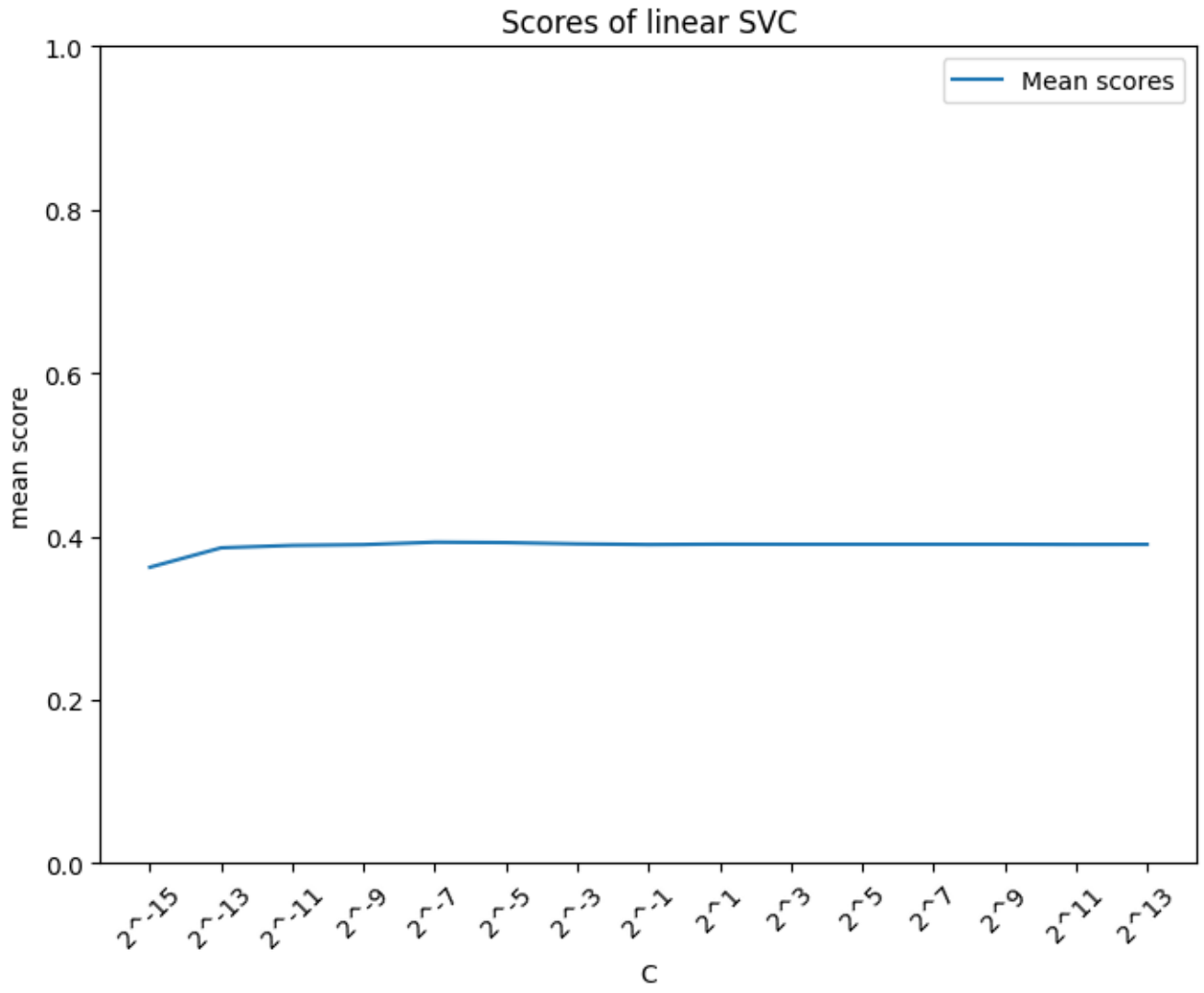


Figure 5-9: Wide grid search of C values for an SVM on the middle layer of an AutoEncoder

The best score is 39.31% found when C=$2^{-7}$. Performing a narrower grid search in $[2^{-9}, 2^{-8.75}, ..., 2^{-5.25}]$, we find that the best score is 39.38% for C is $2^{-7.75}$. The results of this narrower grid search are seen in figure 5-10.

## 5.4 Conclusions

From the above we conclude that training an SVM based on the internal representations of 256 neurons of an AutoEncoder does not yield good classification results. Other architectures have to be explored (such as using an RBF kernel, for example), in order to find one that would work with an SVM.

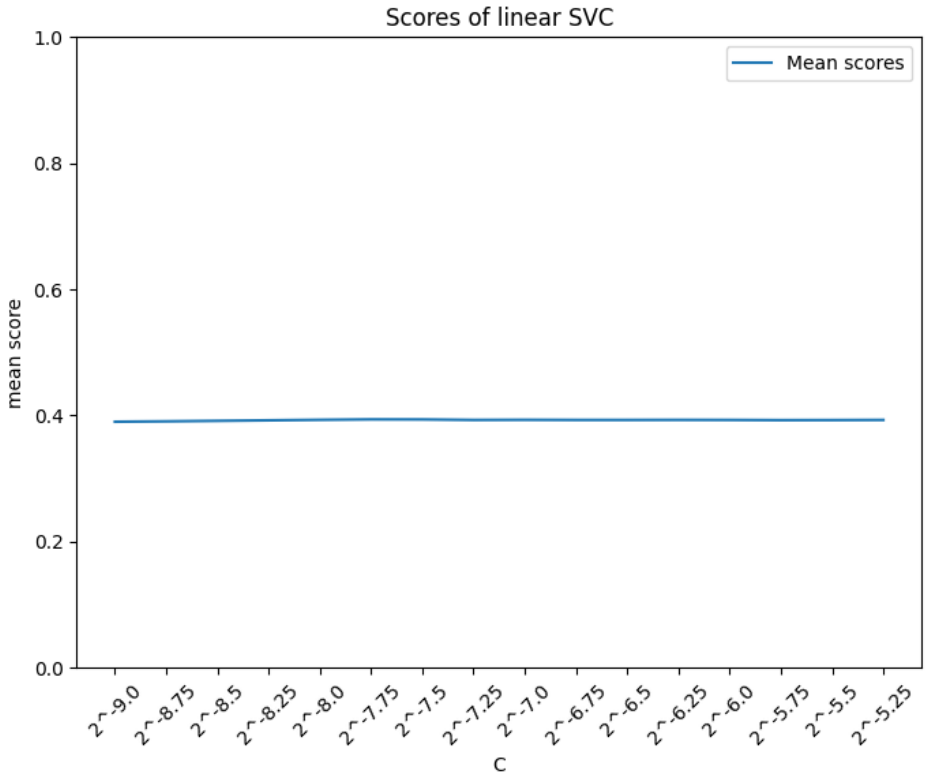


Figure 5-10: Narrow grid search of C values for an SVM on the middle layer of an AutoEncoder

# 6 Classifying raw images with an SVM

## 6.1 The technique

Another technique that can be explored is using SVMs on raw image data, instead of internal representations of images that neural networks can produce.

## 6.2 Implementation

After flattening our image data from shape (32,32,3) to shape (3072), we can try to find a good C, $\gamma$ combination to train an SVC with an RBF-kernel. The heatmap in figure 6-11 plots the scores, after performing a grid search in the range $[2^{-15}, 2^{-13}, ..., 2^{11}]$ for both C and $\gamma$.

Important note
To train this SVC, since it will be RBF-kernel based, we take only 1000 images randomly selected, to reduce the grid search time, because it is very computationally expensive.

The best accuracy is 36.3% for C=2, $\gamma$=$2^{-7}$, which is not that good. Nevertheless, we can try a finer search around the values of C,$\gamma$ to see if it can be improved. This finer grid search will happen in the range $[2^0, 2^{0.25}, ..., 2^{3.75}]$ for C and in $[2^{-9}, 2^{-8.75}, ..., 2^{-5.25}]$ for $\gamma$. The results are seen in figure 6-12. Indeed, an accuracy of 36.9% is achieved.

Consequently, we train the estimator with the whole dataset. The accuracy it reaches is an impressive 57.6%, given the fact that it was trained on just 1000 images. What's worth noting,

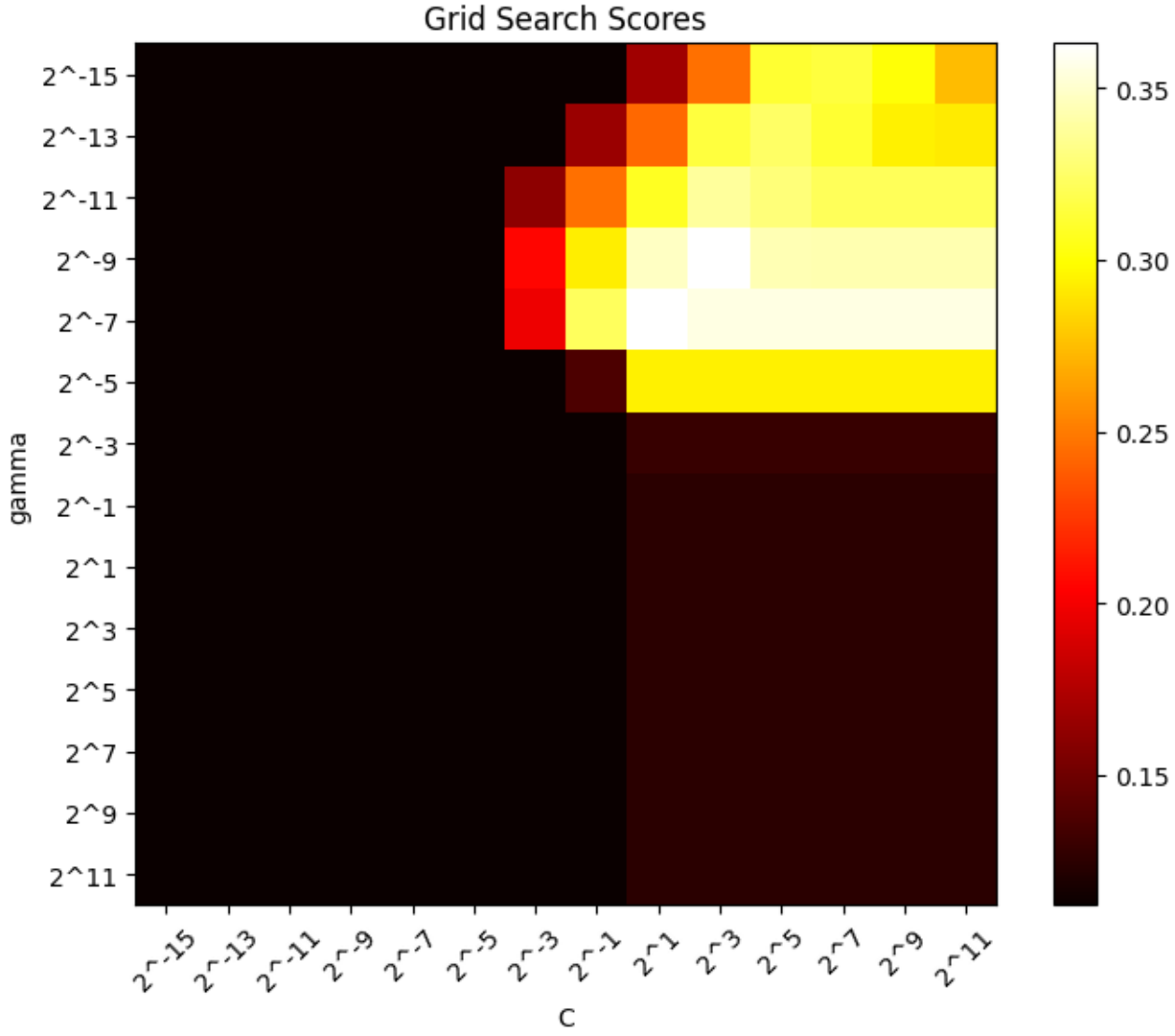


Figure 6-11: SVC with RBF kernel, C,γ search heatmap for raw image data

though, is that it took 2 and a half hours to train the estimator and measure its accuracy, meaning it would take a lot of computational power and time to perform a grid search on a bigger sample of the dataset, to find actual good hyper-parameters.

### 6.3 Results

The best possible accuracy is 57.6%, which is quite good for an image classifier given the circumstances. This has been greatly affected by the fact that the hyper-parameter search (C,γ), was done based only on 1,000 images and not a bigger sample due to lack of resources.

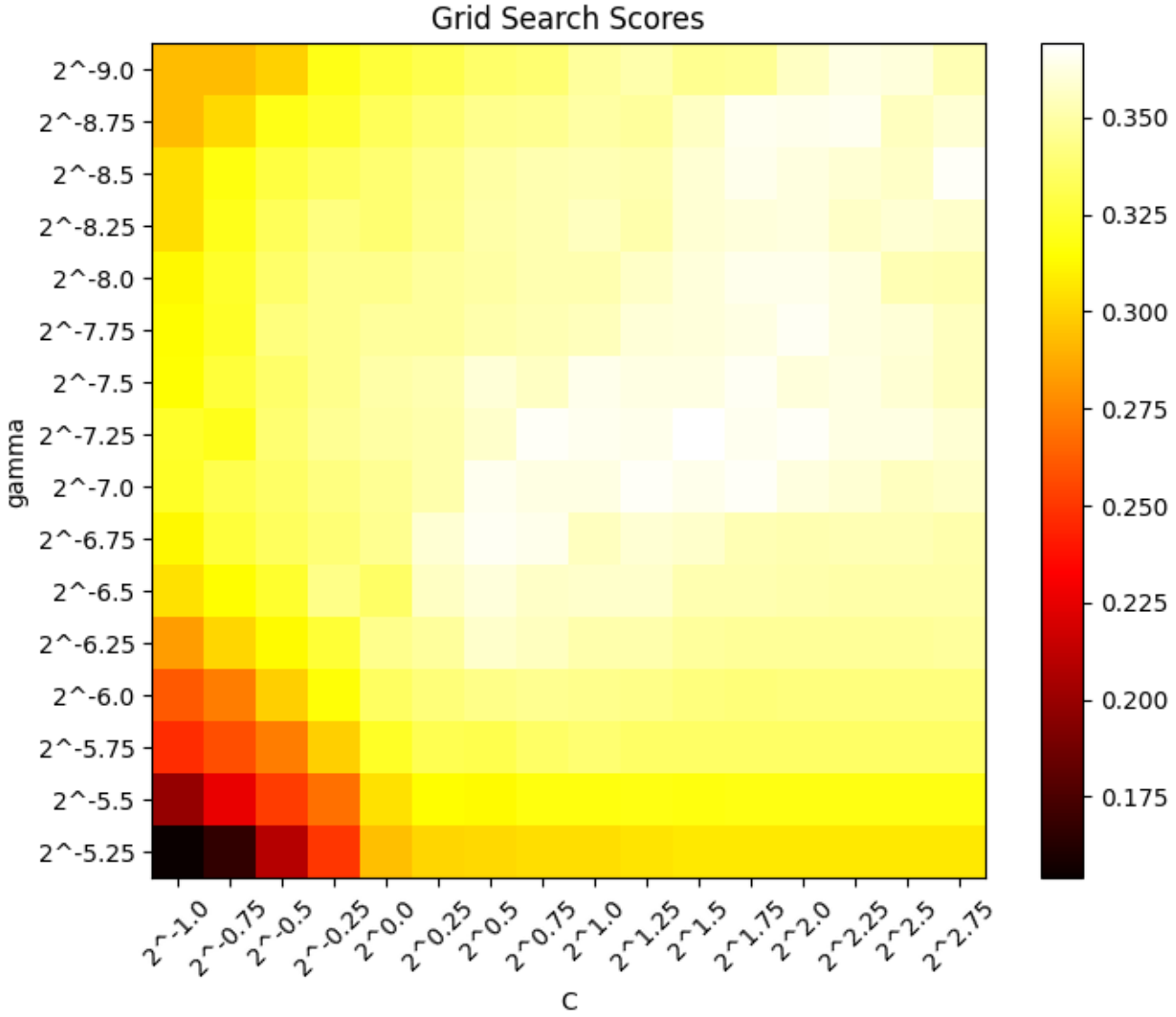


Figure 6-12: SVC with RBF kernel, C,$\gamma$ search heatmap for raw image data

# 7 Using PCA before classifying with an SVM

## 7.1 The plan

What if we tried to fit a Support Vector Machine on the CIFAR-10 images after performing PCA on them?

First of all, we must reduce the number of dimensions. Our goal is to keep at least 90% of the initial information. Hence, the number of components needed is automatically calculated by `sklearn.decomposition.PCA` to be 99. This time, we'll try to fit a LinearSVC.

## 7.2 Searching for a good C

We'll look for Cs in $[2^{-15}, 2^{-13}, ..., 2^{13}]$. The best C is found to be at $2^{-11}$, with an accuracy score of 39.06%, although marginally better than the other accuracies, as seen in figure 7-13. The scores' values are the following: [0.38388, 0.38762, 0.38926, 0.39028, 0.39062, 0.39024, 0.39042, 0.39022, 0.39022, 0.39014, 0.39006, 0.39012, 0.39014, 0.39008,

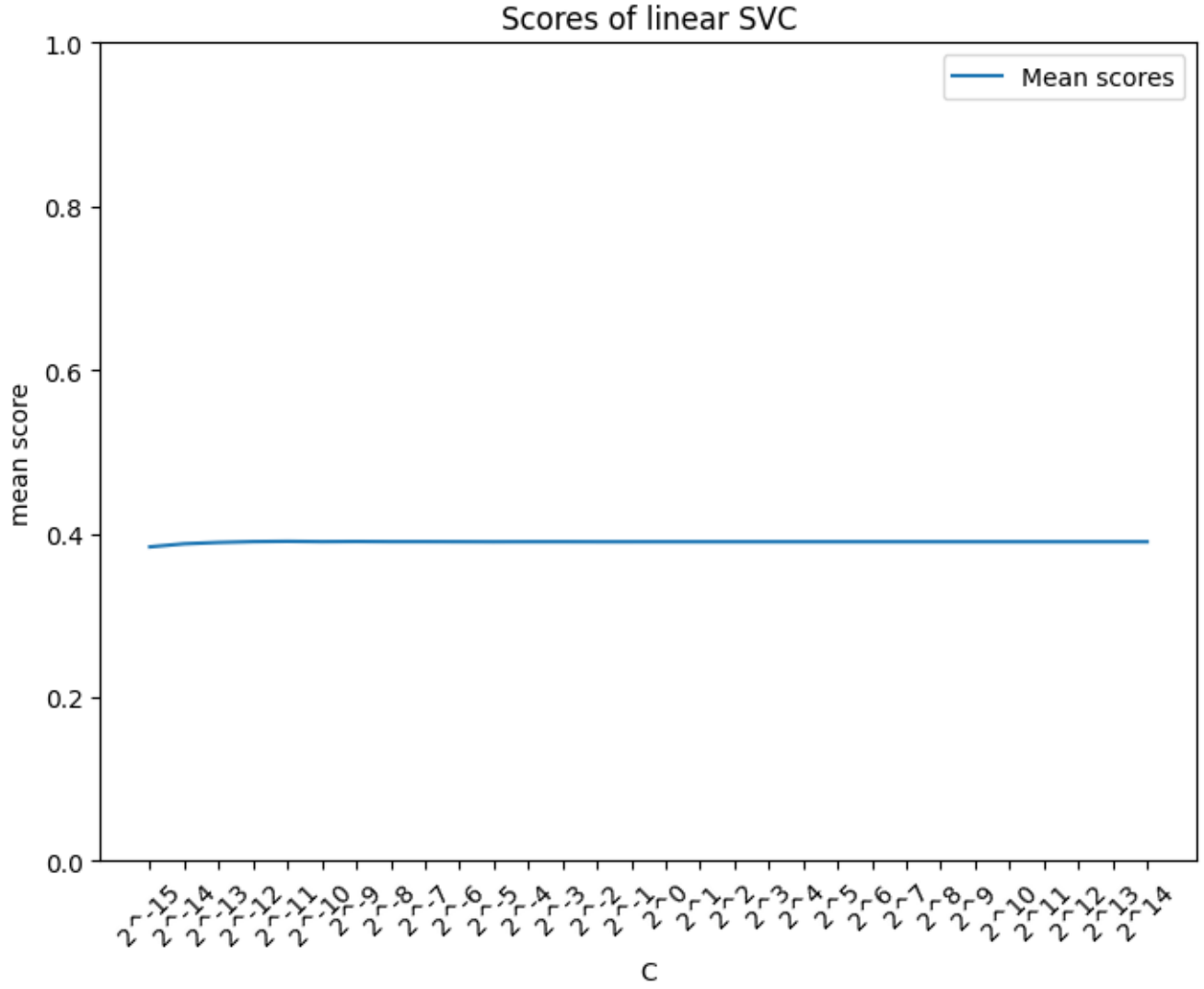


Figure 7-13: Searching for a good value of C for a Linear SVC to be applied on images after performing PCA

$0.39008, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901, 0.3901]$

After a finer grid search for C in $[2^{-12}, 2^{-11.75}, ..., 2^{-10.25}]$, it is ascertained that the best accuracy is 39% when the value of C is $2^{-11}$.

## 7.3 Findings

We train our estimator for the whole dataset and the resulting accuracy is 40.98% No estimator was found to be adequate enough in classifying the images, since the accuracy is not nearly close to something like 75 or 80%.

# 8 Conclusion

## 8.1 What we've seen so far

Many approaches were explored in this report. We tried fitting SVMs on the internal representations of a Convolutional Neural Network, using either a linear kernel one or an RBF kernel Suport Vector Classifier.

We checked whether it's possible to simulate the functionality of an SVM with a neural network, by using hinge loss instead of crossentropy, which resulted in an OK 45% accuracy and leaves plenty of room for improvement.

Also, we tried fitting an SVM with the internal representations of a simple AutoEncoder, to see how well it is able to classify the CIFAR-10 images.

Afterwards, we tried fitting an SVM on raw image data, as well as data that have gone through Principal Component Analysis.

The best results were retrieved by using a Support Vector Classifier with an RBF kernel on the representations of images of a Convolutional Neural Network, taken from its second-to-last layer.

## 8.2 Comparison to other approaches

SVMs, when used correctly, are massively better than using a K-Nearest-Neighbors or a NearestCentroid classifier in classifying images. The best accuracy of a KNN classifier is about 35%, NearestCentroid's best accuracy is 27%, whereas a well-configured SVM can reach up to 74% accuracy. On top of that, it can be optimized to get even better results, given more computational power and resources to work with.