Universitat Pompeu Fabra Barcelona

SVHN Classification with CNN

Deep Learning

Lida Calsamiglia, Patricia Castelijns, Pau Cobacho

u172787, u172949, u161616

June 8, 2023

1 Introduction

In this lab we implement Convolutional Neural Networks that learn to classify RGB images of street view house digits labeled from 1 to 10, from the SVHN dataset.

2 Training and Testing of different variants of the CNN

In this section we want to obtain the maximum test accuracy possible in the SVHN dataset by training and testing different variants of the CNN provided in the examples. This CNN has the following properties:

Table 1: Provided CNN properties

layer	filters	kernel size	padding
layer 1	16	5x5	2
layer 2	32	3x3	1

With this specification, the CNN trained on the SVHN dataset with an Adam optimizer and a learning rate of 0.001, gives us an accuracy of 85.37%. We believe that this is a good start taking into account that this model has 26346 parameters, but we need to improve it. Hence our purpose now is to improve this architecture's result by experimenting with the amount of hidden layers, learning rates and optimizers. For this, we defined three new architectures (2) named CNN 2, CNN3 and CNN4 and trained them (3).

We decided to first train the default architecture differently, that is, with different optimizer methods and a higher learning rate. This allowed us to see that the default configuration (Adam optimizer and a learning rate of 0.001) is the one performing the best with respect to the other cases as they reach a lower accuracy. We also observe here that the learning rate affects differently each optimizer; the SGD improves a lot when going from a 0.001 to a 0.005 learning rate whereas the Adam optimizer performs always better with the smaller value, the higher one is always giving a slightly lower accuracy.

Following, we improved the default architecture, CNN1 by adding a larger number of filters in each layer. CNN2 accuracy results show that increasing the number of filters positively affects the performance of an architecture as we obtained a higher percentage for each different training case.

With these findings we were eager to observe the effect of adding one more hidden layer; CNN3

Table 2: Different CNNs implemented

CNN variant	layers	# filters	kernel size	padding
CNN 1	layer 1	16	5x5	2
	layer 2	32	3x3	1
CNN 2	layer 1	48	5x5	2
	layer 2	64	3x3	1
CNN 3	layer 1	48	5x5	2
	layer 2	64	5x5	2
	layer 3	128	3x3	2
CNN 4	layer 1	48	5x5	2
	layer 2	64	3x3	2
	layer 3	128	3x3	2
	layer 4	256	3x3	2

Table 3: Accuracy for different variants of CNNs

CNN variant	**Optimizer**	**Learning Rate**	**Accuracy (%)**
CNN1	Adam	0.001	85.37%
	Adam	0.005	80.76 %
	SGD w/ momentum	0.001	20.07%
	SGD w/ momentum	0.005	80.77%
CNN2	Adam	0.001	85.95%
	Adam	0.005	83.40%
	SGD w/ momentum	0.001	24.07%
	SGD w/ momentum	0.005	82.10%
CNN3	Adam	0.001	89.18%
	Adam	0.005	86.97%
	SGD w/ momentum	0.001	19.59%
	SGD w/ momentum	0.005	84.41%
CNN4	Adam	0.001	90.68%

is similar to CNN2 but with one added hidden layer of 128 filters. This, as we expected, turned out to improve the accuracy again as a new highest accuracy was obtained. Taking all this into account, we finally defined CNN4 by adding one more hidden layer to CNN3 with 256 filters and trained it following the Adam optimizer and a learning rate of 0.001 as this proved to be the best choice in terms of accuracy. This last architecture gave us the best result and allowed us to observe the fact that, by increasing the number of layers and filters, a better performance can be achieved.

In conclusion, the results obtained lead us to observe that adding more layers to the CNN and having more filters allows us to extract more features and therefore obtain a better performance but we are not entirely satisfied since a 90% accuracy is not enough and we think that it could be improved by following a different architecture prototype than the one proposed here in the first place.

3 Building our own efficient CNN architecture for SVHN

In this exercise we had to build our own efficient CNN architecture with a maximum of 150K parameters. We first implemented the MobileNet V2 architecture because it enables a richer set of computations while keeping the amount of parameters relatively small, compared with normal convolutions. To do so, we implemented a class defining the Depthwise-Separable Convolution block, which includes expansion, depth-wise and pointwise convolutions (each with filters of 1x1, 3x3 and 1x1, respectively) and a forward pass function inside this block. Batch normalization and dropout were added during training to improve the training speed and prevent overfitting. Apart from this model, we also tested other approaches to check if we could gain more accuracy.

Note that the learning rate = 0.1 and optimizer = SGD with momentum, were the same for all implementations. See below the table that visualizes the parameters of the implemented models:

Table 4: MobileNet1 (V2) (modifyable parameters)

Block	**#in channels**	**#out channels**	**kernel size**	**s**	**p**	**exp**
conv_1	3	32	3	2	1	-
dewiseSepConvBlock_1	32	32	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_2	32	64	e=1, d=3, p=1	2	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_3	64	64	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_4	64	128	e=1, d=3, p=1	2	p$_e = p_p = 0, p_d = 1$	3

Table 5: MobileNet2 (V2) (modifyable parameters)

Block	**#in channels**	**#out channels**	**kernel size**	**s**	**p**	**exp**
conv_1	3	16	3	2	1	-
dewiseSepConvBlock_1	16	16	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_2	16	32	e=1, d=3, p=1	2	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_3	32	32	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_4	32	64	e=1, d=3, p=1	2	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_5	64	64	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3

Table 6: MobileNet3 (V2) (modifyable parameters)

Block	**#in channels**	**#out channels**	**kernel size**	**s**	**p**	**exp**
conv_1	3	32	3	2	1	-
dewiseSepConvBlock_1	32	32	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_2	32	128	e=1, d=3, p=1	2	p$_e = p_p = 0, p_d = 1$	3
dewiseSepConvBlock_3	128	128	e=1, d=3, p=1	1	p$_e = p_p = 0, p_d = 1$	3

Table 7: ResNet (modifyable parameters)

Block	**#in channels**	**#out channels**	**kernel size**	**s**	**p**
conv_11	3	32	3	1	1
conv_12	32	32	3	1	1
conv_21	32	64	3	1	1
conv_22	64	64	3	1	1

See below a table with the accuracy for each of the neural networks:

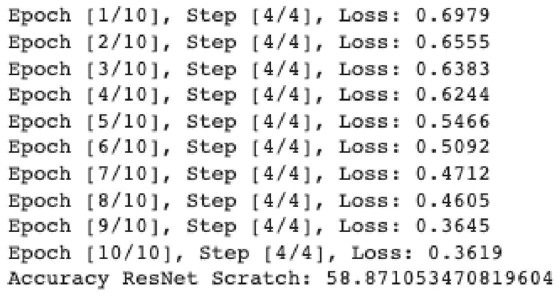
Table 8: Accuracy of the different Models

CNN Variant	**# parameters**	**Accuracy (%)**
MobileNet1 V2	$\sim$ 87K	92.56
MobileNet2 V2	$\sim$ 51K	91.18
MobileNet3 V2	$\sim$ 130K	91.57
ResNet	$\sim$ 106K	86.17

We can observe that the four models gave us a pretty high accuracy, with the MobileNet V2 architecture having the highest accuracies. In particular the MobileNet1 V2, with 92.56% of accuracy. We can also note how a large number of parameters does not always improve the performance of the model, this could be because overfitting has not been sufficiently prevented and the larger the number of the parameters, the more specific features of a particular training data the model learns.

4 Fine-tuning

In this exercise we want to train the previous model so that it classifies between 0s and 9s, i.e. a binary classifier. However, the new data set containing 0s and 9s has only 200 samples and training the network like this does not give a good accuracy as the model has not seen enough information to accurately differentiate between the two digits. We tested the accuracy for this scenario and, as expected, it is not good:

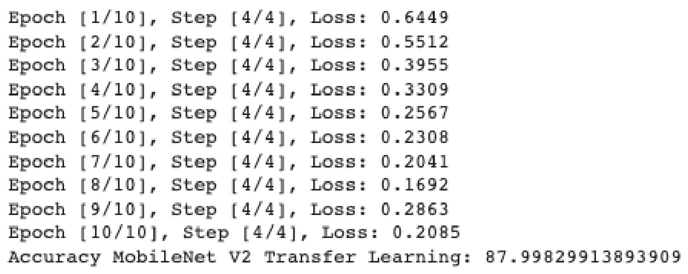


```
Epoch [1/10], Step [4/4], Loss: 0.6979
Epoch [2/10], Step [4/4], Loss: 0.6555
Epoch [3/10], Step [4/4], Loss: 0.6383
Epoch [4/10], Step [4/4], Loss: 0.6244
Epoch [5/10], Step [4/4], Loss: 0.5466
Epoch [6/10], Step [4/4], Loss: 0.5092
Epoch [7/10], Step [4/4], Loss: 0.4712
Epoch [8/10], Step [4/4], Loss: 0.4605
Epoch [9/10], Step [4/4], Loss: 0.3645
Epoch [10/10], Step [4/4], Loss: 0.3619
Accuracy ResNet Scratch: 58.871053470819604
```

Figure 1: Accuracy obtained by training the network directly with the 0s and 9s dataset

Therefore, we are meant to use transfer learning to solve this: the idea is to train the whole model with a bigger data set containing numbers from 1 to 8 to then fine tune the last fully connected layer with the 0s and 9s data set. The reasoning behind this is for the architecture to first have enough data to be able to discriminate properly between numbers (even though targeting numbers from 1 to 8, different to our binary classification goal) and then to fine tune the last fully connected layer, forcing it to use the model's learnings to classify only between 0s and 9s.

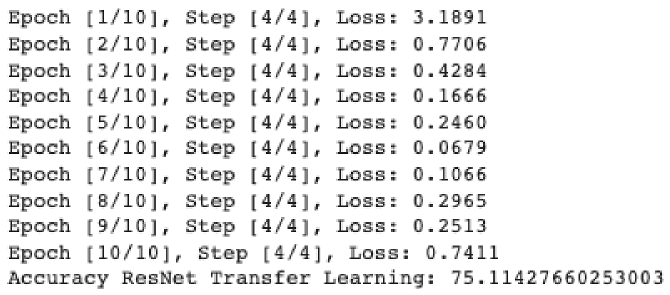
So, once the model is trained with the bigger data set (numbers from 1 to 8), we fine tune the last fully connected layer to obtain the following result:



```
Epoch [1/10], Step [4/4], Loss: 0.6449
Epoch [2/10], Step [4/4], Loss: 0.5512
Epoch [3/10], Step [4/4], Loss: 0.3955
Epoch [4/10], Step [4/4], Loss: 0.3309
Epoch [5/10], Step [4/4], Loss: 0.2567
Epoch [6/10], Step [4/4], Loss: 0.2308
Epoch [7/10], Step [4/4], Loss: 0.2041
Epoch [8/10], Step [4/4], Loss: 0.1692
Epoch [9/10], Step [4/4], Loss: 0.2863
Epoch [10/10], Step [4/4], Loss: 0.2085
Accuracy MobileNet V2 Transfer Learning: 87.99829913893909
```

Figure 2: Accuracy after fine tuning the fully connected layer

With this we can see that the fine tuning has worked as the accuracy has improved a lot. Now, if we fine tune not only the last fully connected layer but the whole architecture with the 0s and 9s data set we see the following results:



```
Epoch [1/10], Step [4/4], Loss: 3.1891
Epoch [2/10], Step [4/4], Loss: 0.7706
Epoch [3/10], Step [4/4], Loss: 0.4284
Epoch [4/10], Step [4/4], Loss: 0.1666
Epoch [5/10], Step [4/4], Loss: 0.2460
Epoch [6/10], Step [4/4], Loss: 0.0679
Epoch [7/10], Step [4/4], Loss: 0.1066
Epoch [8/10], Step [4/4], Loss: 0.2965
Epoch [9/10], Step [4/4], Loss: 0.2513
Epoch [10/10], Step [4/4], Loss: 0.7411
Accuracy ResNet Transfer Learning: 75.11427660253003
```

Figure 3: Accuracy after fine tuning all the parameters of the model

The accuracy here has lowered because we believe that by fine tuning the entire model and updating more parameters, lots of the learning that was obtained from the other bigger data set (that produces a better number detector) has now vanished by focusing the inner model's neurons with a much smaller data set that produces a worse number detector.