Developing a CNN from scratch

Group members: Jose Manuel López, Alex Martín, Marcos V. Conde

INDEX

Statement of the problem

We want to use deep learning techniques for image classification.

Given a dataset, how can we a CNN model that optimizes the tradeoff ***model-complexity vs vs model performance*** **(accuracy)** ? Are always literature/SOTA architectures the best option?

To find this answer we run manual **Architecture Search** (AS) experiments, and try +15 different CNN configurations using well-known techniques like: Convolutional Blocks, Batch Normalization, Dropout, etc.

We use a common experimentation setup in order to compare fairly all the models: same validation split and validation metrics, same data-loader and augmentations, same number of epochs and optimizer etc.

At the end of this experiment, we analyze the **ablation study** results and take conclusions.

Base model

All the tested models have the same configuration in common:

- N Convolutional Layers/Blocks → Feature extractors
- Global Average Pooling (GAP) layer → produces the feature vector (aka embedding).
- M dense layers → conforms the classification head.

Fig: Dissecting AS into three steps: 1) exploring the search space, where operations (e.g. pooling, convolution etc) live; 2) have a good detective strategy to pick the right bits and pieces for a specific tasks; 3) have an evaluator system for any created architecture.
[Image from https://towardsdatascience.com/neural-architecture-search-a-model-creation-company-602c6ba4f576]

Fixed parameters

After starting to test different parameters for the CNN we fix some of them in order to focus more on the other ones:

- The **activation functions** used were always ReLu and Softmax for the last dense layer.
- The **optimizer** used was Adam with the default parameters.
- The models were trained with learning rate decay depending on the validation accuracy during 100 Epochs.

To set up our initial model we used 64 filters per convolutional layer, a kernel size of 5 and 1 dense layer with 1024 neurons before the last layer.

	Train Accuracy	Test Accuracy	Num. Params	Ratio
Initial model	0,614	0,656	79624	$7{,}71\cdot10^{-07}$

Variable parameters: Number of layers

The first variable tested was the **optimal number of convolutional layer** and dense layer to be used:

Nº of conv layers	Train Accuracy	Test Accuracy	Num. Params	Ratio
2	0,742	0,774	182088	$4{,}08 \cdot 10^{-07}$
4	0,860	0,850	387016	$2{,}22 \cdot 10^{-07}$
8	0,867	0,843	796872	$1{,}07 \cdot 10^{-07}$

Nº of dense layers	Train Accuracy	Test Accuracy	Num. Params	Ratio
1	0,897	0,850	387016	$2{,}22 \cdot 10^{-07}$
2	0,897	0,854	1436616	$6{,}17 \cdot 10^{-08}$
4	0,898	0,869	3535816	$2{,}54 \cdot 10^{-08}$

Variable parameters: Number of filters

After setting the number of convolutional layers and dense layer to 4 and 3 respectively, we started to seek the optimal parameters for the number of filters in each convolutional layer:

Nº of filters	Train Accuracy	Test Accuracy	Num. Params	Ratio
32	0,862	0,844	1170920	$7{,}36 \cdot 10^{-08}$
64	0,897	0,854	1436616	$6{,}25 \cdot 10^{-08}$
128	0,883	0,846	2428808	$3{,}64 \cdot 10^{-08}$
256	0,870	0,862	6256392	$1{,}39 \cdot 10^{-08}$

Regularization

After the previous testing, the best model was like this:

- 4 convolutional layers with 256 filters
- 2 dense layers with 1024 neurons
- Kernel size 5

The next step was to **introduce regularization techniques** into the model. We tested the following methods one by one and combining them:

- **MaxPooling:** Reduce spatial dimension → compact features maps
- **Batch Normalization:** train models faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. Work by Ioffe and Szegedy et al.
- **Dropout:** randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.

Regularization

Regularization method	Train Accuracy	Test Accuracy	Num. Params	Ratio
MaxPooling	0,920	0,864	6256392	$1{,}47 \cdot 10^{-08}$
Dropout (0.2)	0,886	0,861	6256392	$1{,}42 \cdot 10^{-08}$
BatchNorm	0,987	0,879	6268680	$1{,}57 \cdot 10^{-08}$
MaxPooling + Dropout (0.2)	0,927	0,867	6256392	$1{,}48 \cdot 10^{-08}$
MaxPooling + BatchNorm	0,970	0,887	6268680	$1{,}55 \cdot 10^{-08}$
MaxPooling + BatchNorm + Dropout (0.2)	0,960	0,890	6268680	$1{,}53 \cdot 10^{-08}$

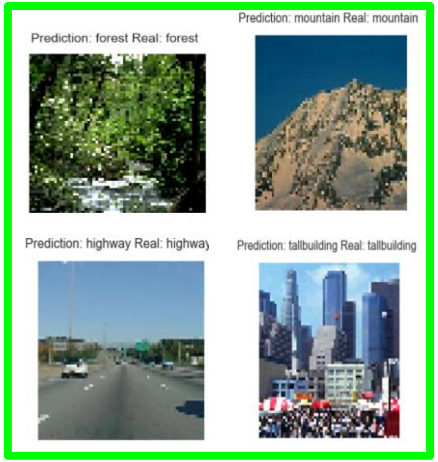
Variable parameters: Kernel size

Kernel size ablation studies:

Kernel size	Train Accuracy	Test Accuracy	Num. Params	Ratio
3	0,900	0,844	3098376	$2{,}90 \cdot 10^{-08}$
5	0,886	0,861	6256392	$1{,}42 \cdot 10^{-08}$
7	0,830	0,829	10993416	$7{,}55 \cdot 10^{-09}$
9	0,758	0,763	17309448	$4{,}38 \cdot 10^{-09}$

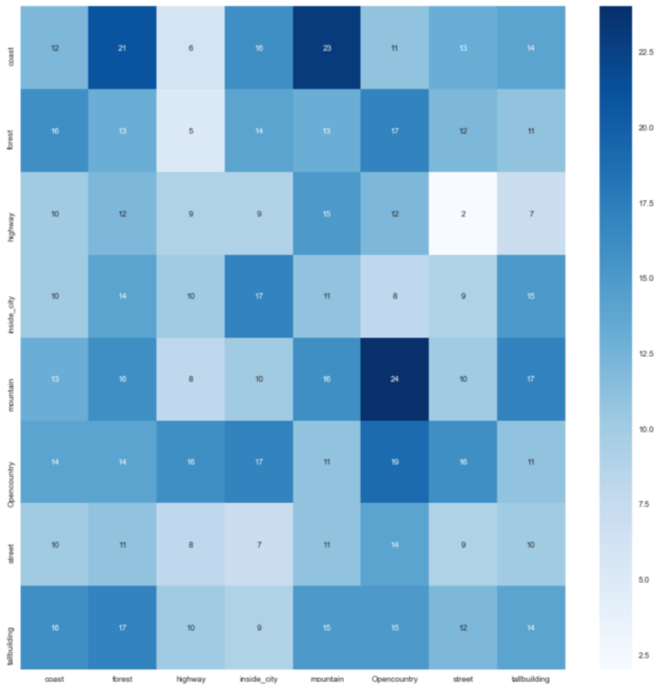
Examples of prediction

- Our model has a fantastic **performance**. However, there are some scenarios that can lead to a misclassification.
- **Street, tallbulding** and **inside_city** are often confused because they all relate to urban landscapes.
- **Street** and **highway** because they both rely on roads.
- **Forest** and the rest of classes if there are high presence of vegetation



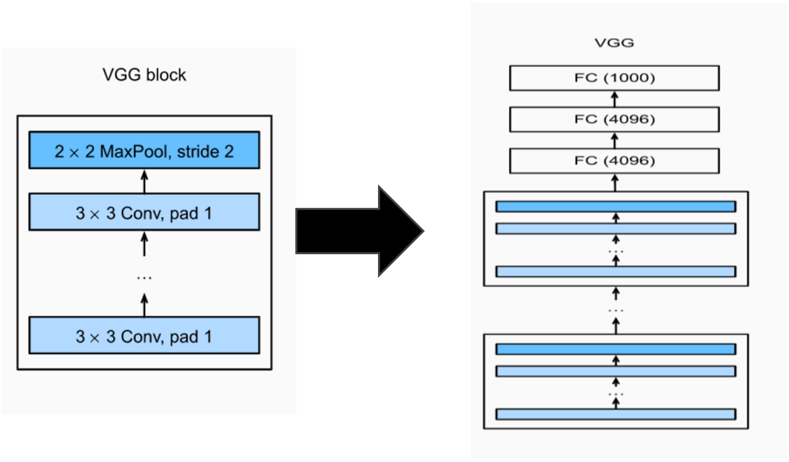
Examples of prediction

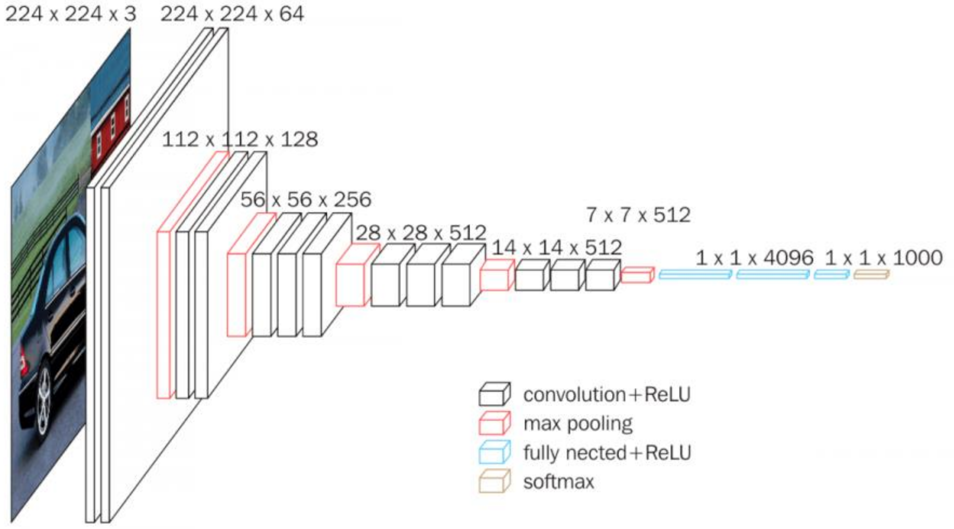
We can check more deeply those kind of errors by looking at the **confusion matrix**.



Literature model

We compare the obtained model with well-known architectures in the literature like VGG by Karen Simonyan and Andrew Zisserman, 2014.





Our model vs Literature model

We compare the obtained model with well-known architectures in the literature, like VGG by Karen Simonyan and Andrew Zisserman, 2014.

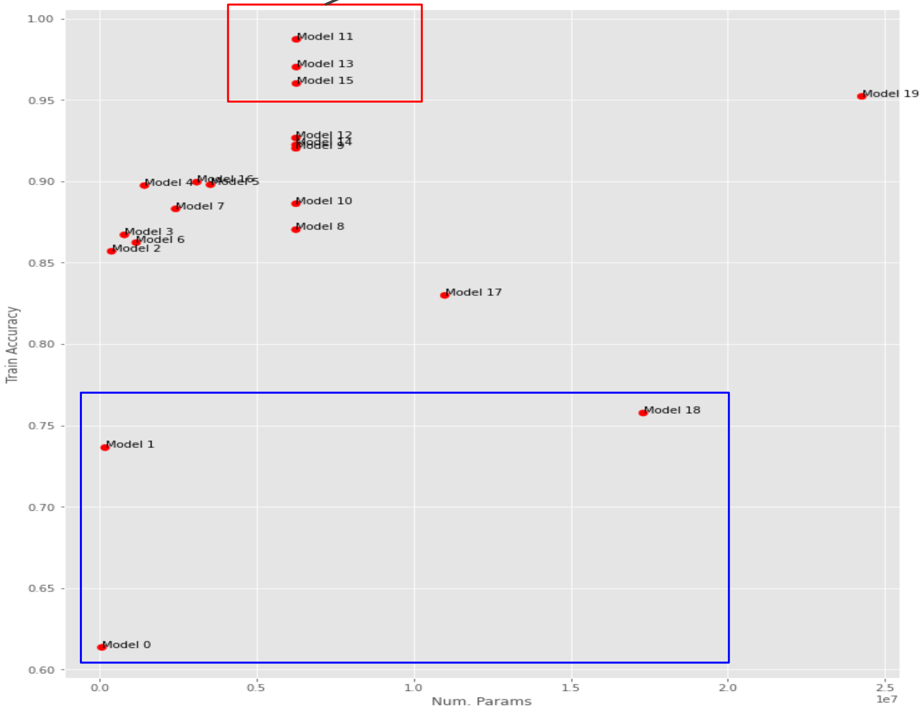
Our custom model achieves competitive results in comparison with the well-known VGG19.

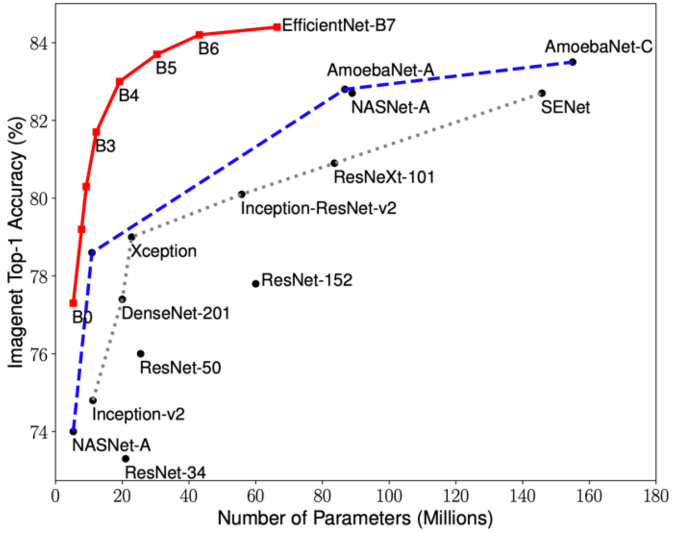
We think the gap between training-validation is due to the difference in the model complexity, and the Dropout that we used. Our model does not overfit training and therefore can generalize decently.

Model	Train Accuracy	Test Accuracy	Train Loss	Test Loss	Num, Params	Ratio
VGG	0,952	**0,879**	0,141	0,460	**24 M**	$3{,}92 \cdot 10^{-09}$
Ours	0,886	**0,861**	0,303	0,437	**6 M**	$1{,}42 \cdot 10^{-08}$

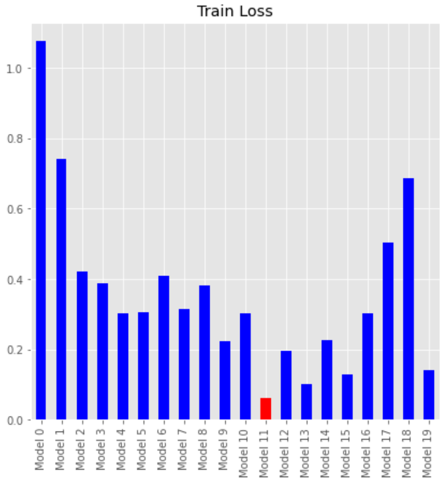
Summary

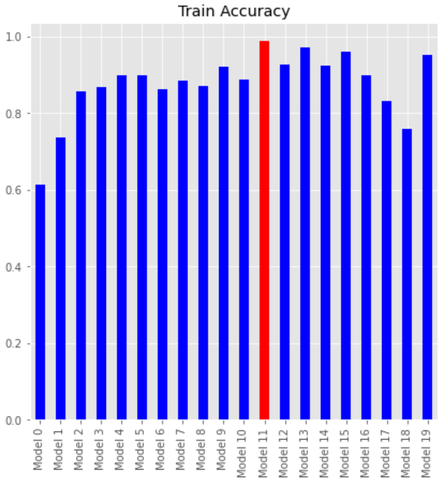
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with BatchNorm	Model 11	0.987241	0.878563	0.061328	0.440767	6268680	1.574878e-08
Conv 4 filters 256 kernel size 5 dense 2 neurons with MaxPooling and Dropout	Model 12	0.926635	0.867410	0.196407	0.408125	6256392	1.481101e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with MaxPooling and BatchNorm	Model 13	0.970229	0.887237	0.102440	0.398179	6268680	1.547740e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with MaxPooling and Dropout	Model 14	0.922382	0.864932	0.226596	0.437865	6256392	1.474303e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with BatchNorm, MaxPooling and Dropout	Model 15	0.960128	0.889715	0.127559	0.357219	6268680	1.531626e-08

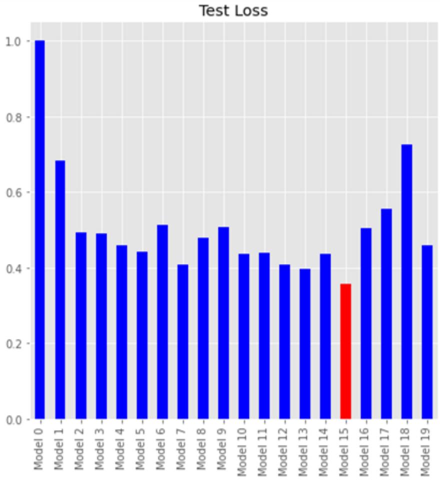


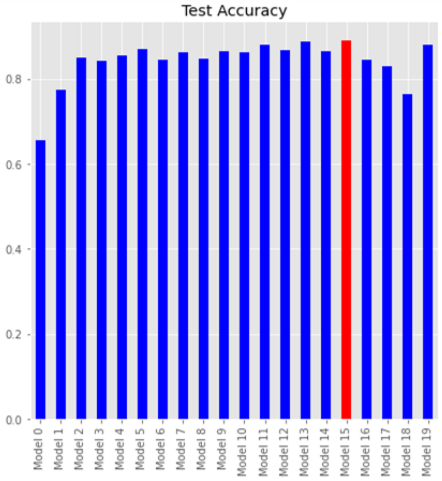


Summary


Train Loss
1.0
0.8
0.6
0.4
0.2
0.0
Model 0
Model 1
Model 2
Model 3
Model 4
Model 5
Model 6
Model 7
Model 8
Model 9
Model 10
Model 11
Model 12
Model 13
Model 14
Model 15
Model 16
Model 17
Model 18
Model 19


Train Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Model 0
Model 1
Model 2
Model 3
Model 4
Model 5
Model 6
Model 7
Model 8
Model 9
Model 10
Model 11
Model 12
Model 13
Model 14
Model 15
Model 16
Model 17
Model 18
Model 19


Test Loss
1.0
0.8
0.6
0.4
0.2
0.0
Model 0
Model 1
Model 2
Model 3
Model 4
Model 5
Model 6
Model 7
Model 8
Model 9
Model 10
Model 11
Model 12
Model 13
Model 14
Model 15
Model 16
Model 17
Model 18
Model 19


Test Accuracy
0.8
0.6
0.4
0.2
0.0
Model 0
Model 1
Model 2
Model 3
Model 4
Model 5
Model 6
Model 7
Model 8
Model 9
Model 10
Model 11
Model 12
Model 13
Model 14
Model 15
Model 16
Model 17
Model 18
Model 19

Summary

	Model	Train Accuracy	Test Accuracy	Train Loss	Test Loss	Num. Params	Ratio
1 Conv 64 filters 5 kernel size 1 dense 1024 neurons	Model 0	0.613503	0.655514	1.075764	1.001122	79624	7.705007e-07
2 Conv 64 filters 5 kernel size 1 dense 1024 neurons	Model 1	0.736310	0.774473	0.741201	0.683470	182088	4.043707e-07
4 Conv 64 filters 5 kernel size 1 dense 1024 neurons	Model 2	0.856991	0.850062	0.419730	0.494306	387016	2.214355e-07
8 Conv 64 filters 5 kernel size 1 dense 1024 neurons	Model 3	0.867092	0.842627	0.386379	0.490809	796872	1.088119e-07
4 Conv 64 filters 5 kernel size 2 dense 1024 neurons	Model 4	0.897395	0.853779	0.303307	0.460579	1436616	6.246589e-08
4 Conv 64 filters 5 kernel size 4 dense 1024 neurons	Model 5	0.897927	0.868649	0.305947	0.442636	3535816	2.539517e-08
4 Conv 32 filters 5 kernel size 2 dense 1024 neurons	Model 6	0.862307	0.843866	0.408355	0.513012	1170920	7.364357e-08
4 Conv 128 filters 5 kernel size 2 dense 1024 neurons	Model 7	0.883041	0.862454	0.315263	0.407398	2428808	3.635697e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons	Model 8	0.870282	0.846344	0.382219	0.477707	6256392	1.391028e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with MaxPooling	Model 9	0.920255	0.863693	0.223434	0.507280	6256392	1.470904e-08
4 Conv 256 filters 5 kernel size 2 dense 1024 neurons with 0.2 Dropout on FC	Model 10	0.886231	0.861214	0.303123	0.437434	6256392	1.416520e-08

Fig. Our log for the ablation studies.

Conclusions

- We have explored the fundamental blocks of CNNs and found an optimal setup emprically using <u>manual</u> Neural Architecture Search.
- We used different techniques from the most influential computer vision literature "Deep Residual Learning for Image Recognition" by *He at al.*
- A proper **experimentation setup is key** for tracking results, save/log meaningful information and take decisions about the modelling.
- Our ablation study shows that the models with: **deep feature representations** (256-filters) and **strong classification heads** (1024-space MLPs), **plus**, Batch Normalization and Dropout for training stability, achieved the most competitive performance in the **tradeoff model complexity vs accuracy.**
- Depending on the dataset and task (in this case simple classification) shallow and "compact" models, like the ones we have developed, can achieve the same results or even outperform SOTA architectures.