Introduction to Machine Learning for Computer Vision

José Manuel López Camuñas, Marcos V. Conde, Alex Martin Martinez

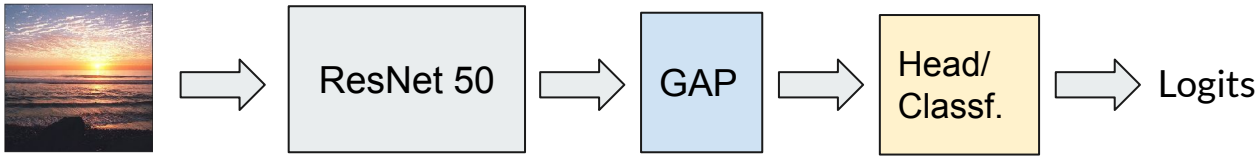
Index

Optimizing an existing trained architecture

```
# ResNet Base Model
base_model = ResNet50(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    weights='imagenet', include_top=False)
```

Our pre-trained model, ResNet50, was fine-tuned on our dataset. As it was previously trained with the Imagenet dataset (1000 classes), we have to modify the last layer or head of the network.

We freeze the other layers of the network, to make sure we only train the head classifier.

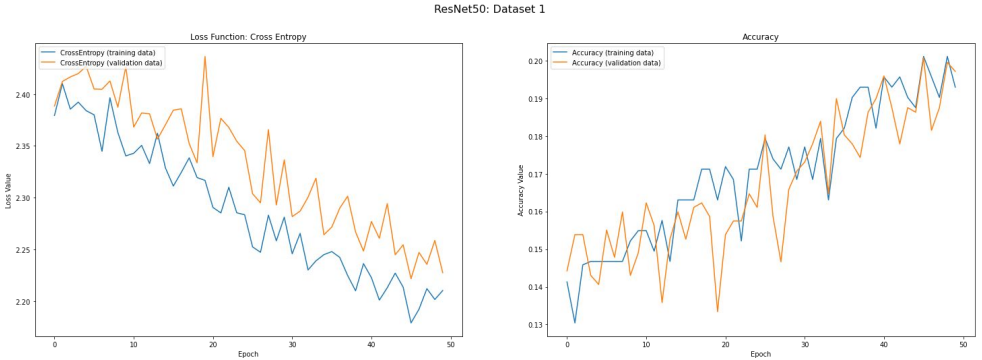



```
avg_pool (GlobalAveragePooling2 (None, 2048)         0           conv5_block3_out[0][0]
predictions (Dense)             (None, 1000)         2049000     avg_pool[0][0]
==================================================================================================
Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120
```

Optimizing an existing trained architecture

With the weight of the neural network froze, we first trained the last layer with the default hyperparameters that were given in the example code to see how well it worked and the result was not satisfying after 50 epochs as the validation accuracy didn't reach the 20%.

Parameters used:
Batch size = 32
Optimizer = Adadelta
Learning rate=0.001
Epochs = 50



Optimizing an existing trained architecture

After that, we started to search for the right **hyperparameters** with a random search. The hyperparameters to be tested will be batch size, learning rate, optimizer and momentum if needed. The range of the parameters from which the random values will be generated is the following:

- **Batch size** from 10 to 100.
- **Learning rate** from 0.0001 to 0.3
- **Optimizer Momentum** from 0 to 0.9
- **Optimizers** used will be SGD, RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam.

The epochs trained will not be included in this to have time to then train all the models.

The maximum number of epochs is set to 50 with an **early-stopping** if the validation accuracy doesn't improve during 10 epochs.

1.a Hyperparameter search

These are the models tested with **random** hyperparameters:

	Batch size	Optimizer	Learning rate	Momentum	Epochs
Model 1	76	Adadelta	0.120	None	50
Model 2	49	RMSprop	0.035	0.10	20
Model 3	61	RMSprop	0.035	0.10	24
Model 4	97	RMSprop	0.182	0.50	21
Model 5	29	SGD	0.044	0.13	30
Model 6	67	SGD	0.088	0.26	18

1.a Hyperparameter search

These are the best epoch results of the models:

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Accuracy	0.9753	1	1	0.9406	1	0.988
Validation accuracy	0.9026	0.9219	0.9092	0.9148	0.9328	0.9342
Loss	0.2084	5.98·10^-5	3.74·10^-6	3.848	0.0082	0.0361
Validation loss	0.3486	1.0268	1.1813	8.7942	0.2042	0.208

1.a Hyperparameter search

As we can see, model 5 and 6 that use **SGD have the best results** with the validation data. So we will perform another random search only using SGD and focusing on the values around the ones that were used to train the model 5 and 6. Now the range of values will be:

- **Batch size** from 30 to 70.
- **Learning rate** from 0.04 to 0.1
- **Optimizers Momentum** from 0. 1 to 0.3

1.a Hyperparameter search

These are the models tested with random hyperparameters, using as baseline previous experiment:

	Batch size	Optimizer	Learning rate	Momentum	Epochs
Model 1	43	SGD	0.066	0.18	36
Model 2	42	SGD	0.097	0.29	23
Model 3	48	SGD	0.085	0.24	31
Model 4	63	SGD	0.080	0.23	32
Model 5	62	SGD	0.045	0.11	37

1.a Hyperparameter search

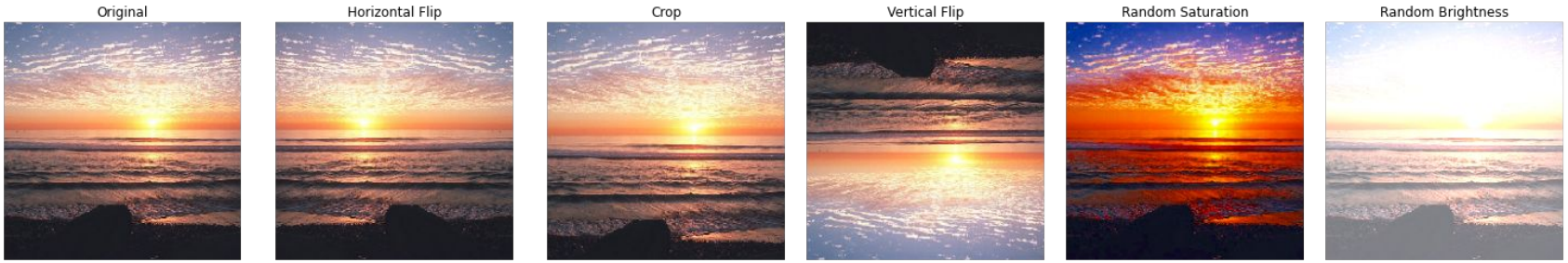
These are the best epoch results of the models:

	Model 1	Model 2	Model 3	Model 4	Model 5
Accuracy	1	1	1	1	1
Validation accuracy	0.9311	0.9272	0.9246	0.9343	0.9310
Loss	0.0065	0.0050	0.0054	0.0102	0.0210
Validation loss	0.2349	0.2648	0.2636	0.1983	0.2028

We observe that the models this time don't present a lot of variance between them. We will use model 4 as it is the one having the highest validation accuracy and the lowest validation loss.

1.b Data augmentation

Now with the previously found hyperparameters, we will also apply data augmentation to our dataset to see if the model can get any further improvement, and also we will reduce the learning rate with the epochs. The transformation applied to the image to have a bigger dataset will be **width shift, height shift, zooming and horizontal flip**. Below we show many of the augmentations that we have tried:



We checked many operations from: https://www.tensorflow.org/tutorials/images/data_augmentation

1.b Data augmentation

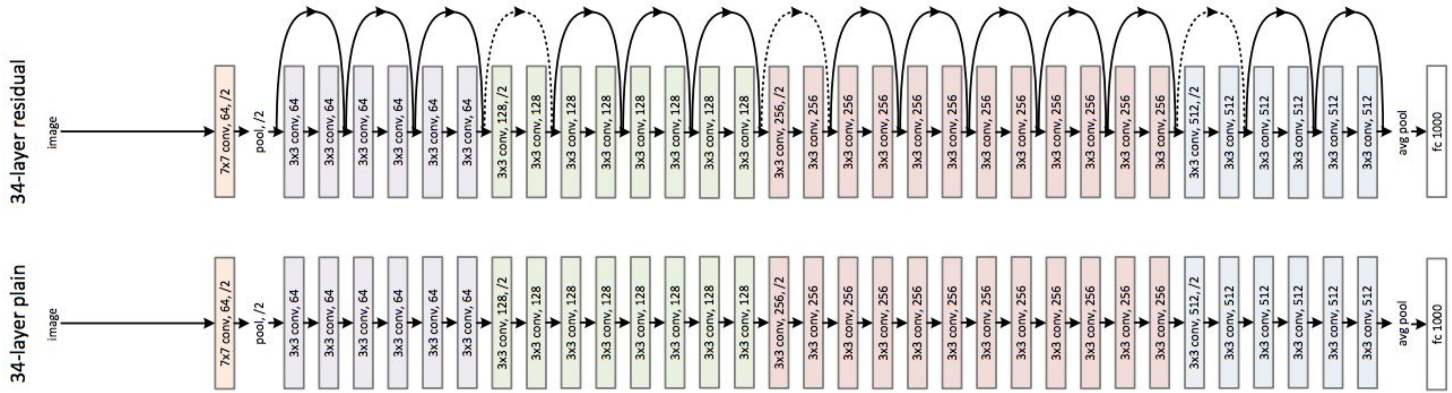
Data Augmentation slightly improved our loss/validation loss but decreased our validation accuracy.

The difference between applying or not data augmentation is so small that we don't really think that helps our model in this particular dataset.

	Model 4	Model 4 (DataAug)
Accuracy	1	1
Validation accuracy	0.9311	0.9296
Loss	0.0065	0.0026
Validation loss	0.2349	0.2024

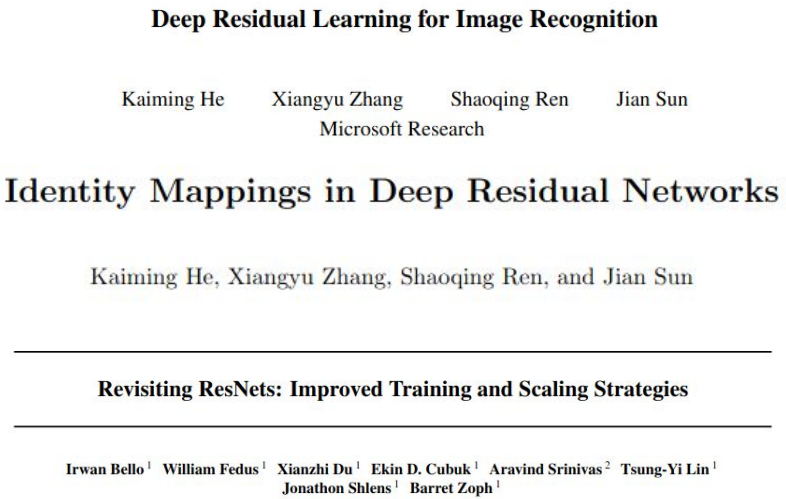
2. Modify the architecture

Improving ResNet is not an easy task. We opt to not increase the number of parameters trying architectures like ResNet-101 or Resnet-152. Below we can see classical ResNet structure:



2. Modify the architecture

We take ideas from these papers to modify ResNet:



Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research

Identity Mappings in Deep Residual Networks

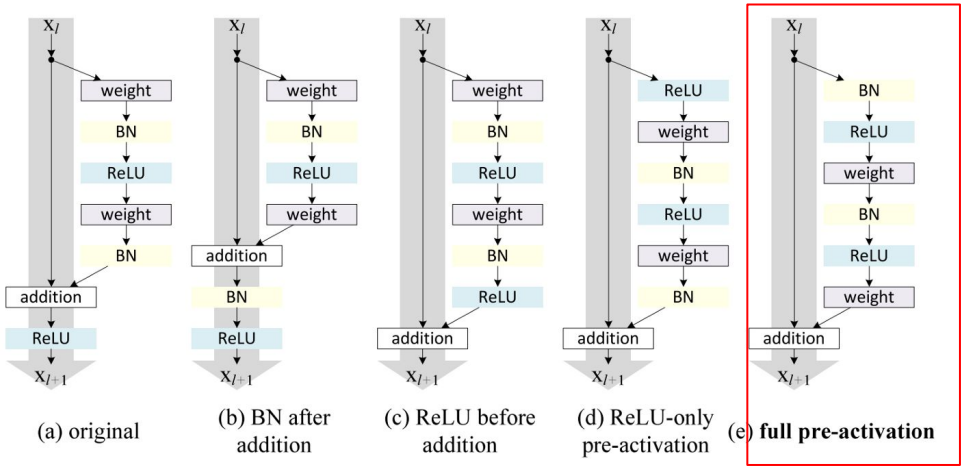
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

Revisiting ResNets: Improved Training and Scaling Strategies

Irwan Bello [1] William Fedus [1] Xianzhi Du [1] Ekin D. Cubuk [1] Aravind Srinivas [2] Tsung-Yi Lin [1]
Jonathon Shlens [1] Barret Zoph [1]

2. Modify the architecture

We can find the possible residual block variations, we use the highlighted.

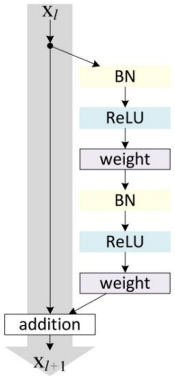


Identity Mappings in Deep Residual Networks

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

2. Modify the architecture

The proposed modified architecture is ResNet34 (more compact than ResNet50) and using the modified blocks instead of the original ones. We follow the paper definition of ResNet34:

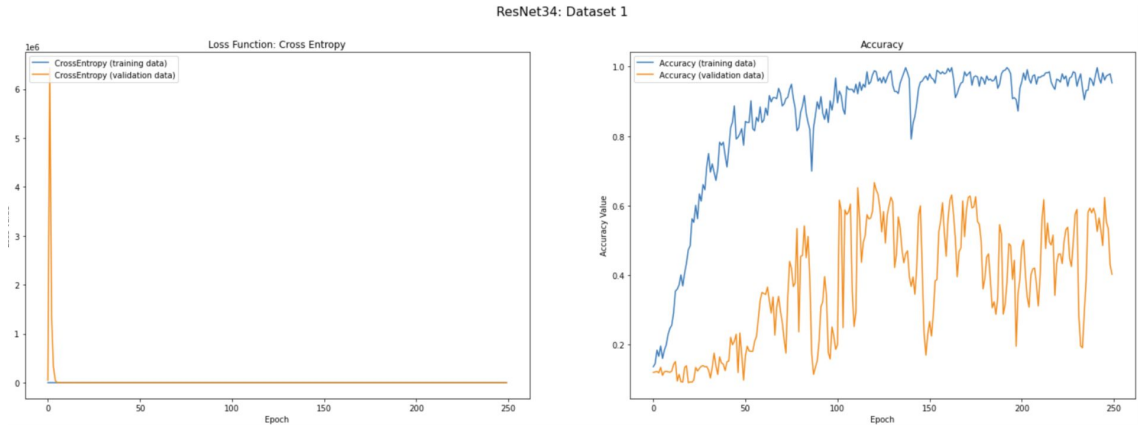


e) **full pre-activation**

layer name	34-layer	50-layer	101-layer
conv1		7×7,64, stride 2	
		3×3 max pool, stride 2	
conv2_x	$\begin{bmatrix} 3 \times 3,64 \\ 3 \times 3,64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,64 \\ 3 \times 3,64 \\ 1 \times 1,256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,64 \\ 3 \times 3,64 \\ 1 \times 1,256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3,128 \\ 3 \times 3,128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1,128 \\ 3 \times 3,128 \\ 1 \times 1,512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1,128 \\ 3 \times 3,128 \\ 1 \times 1,512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3,256 \\ 3 \times 3,256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1,256 \\ 3 \times 3,256 \\ 1 \times 1,1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1,256 \\ 3 \times 3,256 \\ 1 \times 1,1024 \end{bmatrix} \times 23$
conv5_x	$\begin{bmatrix} 3 \times 3,512 \\ 3 \times 3,512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,512 \\ 3 \times 3,512 \\ 1 \times 1,2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,512 \\ 3 \times 3,512 \\ 1 \times 1,2048 \end{bmatrix} \times 3$
		average pool,2048-d fc	

2. Modify the architecture

When training the ResNet34 with the previous parameters for 100 epochs and without the data augmentation, the model overfit and the validation accuracy remained lower than the accuracy during all the training.



2. Modify the architecture: Results

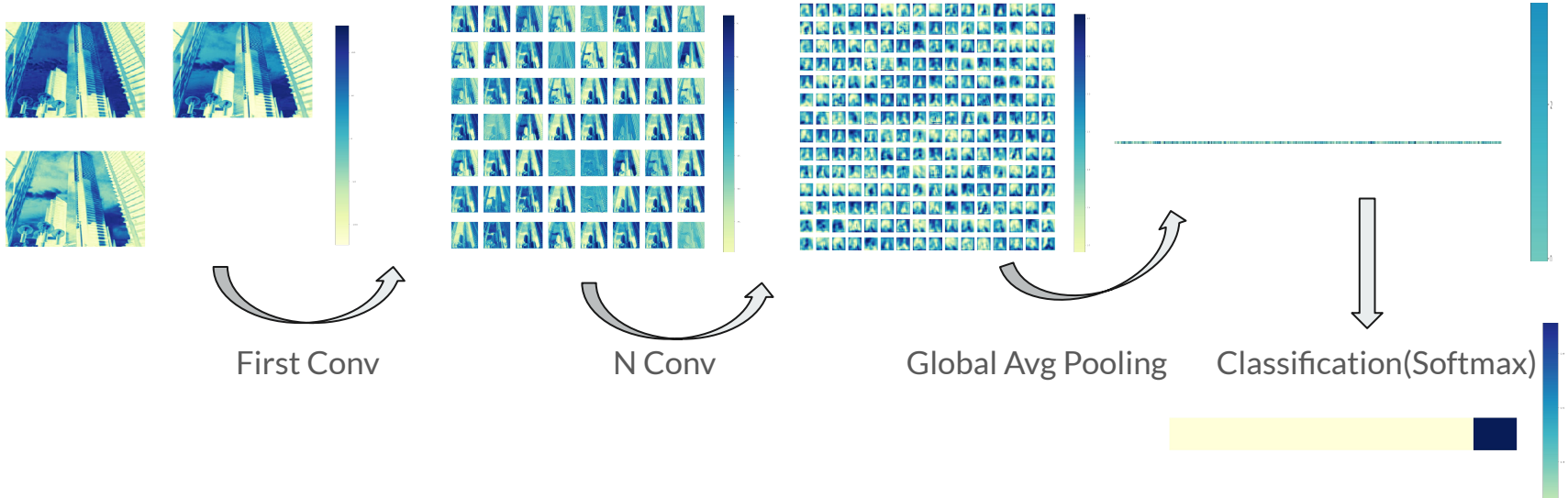
In comparison with the Imagenet pre-trained ResNet50, our model that uses the ResNet34 and is trained with the given dataset and now using data augmentation (with the early stop and learning rate decay as previously defined) achieves:

	Accuracy	Validation accuracy	Loss	Validation loss
ResNet34	1.0000	0.5892	0.0095	2.2347
ResNet34(DataAug)	0.9937	0.6329	0.0626	1.6717

Pre-trained ResNet50 is better than a fully new trained net with such a small train dataset but we obtain some decent results. Here we can see significant gains after using Data Augmentation.

3. Feature maps

We can observe feature maps through different layers of our ResNet32 for this image (full feature maps available on notebook).



4. Conclusions

- Data augmentation as a regularization technique is a good way to improve a model's performance and its generalization capabilities, but in our case the improvement is not significative with the already trained ResNet50.
- The most important hyperparameters are the ones related with the optimization: the optimizer and learning rate.
- Increment the complexity of the head classifier does not necessarily improve performance, while fine-tuning or training from scratch does have more effect.
- The main modifications and improvements on ResNet50 (including the ones tried in this work) are related with the feature representation and its propagation cross layers.