NATIONAL CHIAO TUNG UNIVERSITY

Institute of Computer Science and Engineering

Selected Topics in Visual Recognition using Deep Learning

Homework 1

Author: Alessandro Saviolo

Student Number: 0845086

17th October 2019

Summary

1. Introduction

The purpose of this document is to present and analyze the methodology used to solve the Kaggle challenge [7] for Homework 1. The methodology consists of 4 steps: hyperparameters tuning, data preprocessing, building the model architecture and fitting the model with the training data, computing predictions on the test data.

At first, the dataset is augmented and balanced, in order to prevent overfitting and provide some bias towards the minority classes while training the model. The data is splitted in training and validation, in order to obtain an estimate of the final accuracy of the model. Then, the model architecture is fixed and it is fitted with the augmentated training data, by monitoring the validation accuracy.

Once the model is built and fitted, the best weight computed during the training phase are restored and the final model is used for predicting the test data.

The code of the project has been uploaded to Github at the repository [5].

2. Methodology

2.1 Data Preprocessing

The dataset is a collection of 3859 images, where 2819 images are used for training and 1040 for testing. The images belong to 13 different classes, namely bedroom, coast, forest, highway, insidecity, kitchen, livingroom, mountain, office, opencountry, street, suburb, tallbuilding.

To improve the ability of the model to generalize, I used data augmentation to expand the size of the dataset. In particular, I augmented the data by applying the following transformations:

- rescale = 1./255
- rotation_range = 20
- zoom_range = 0.2
- shear_range = 0.2
- horizontal_flip = $True$

Since the dataset is slightly imbalanced (i.e., the classes are not represented equally, as presented in Figure 2.1), I provided some bias towards the minority classes while training the model. This helps in improving the performance of the model while classifying the different classes.

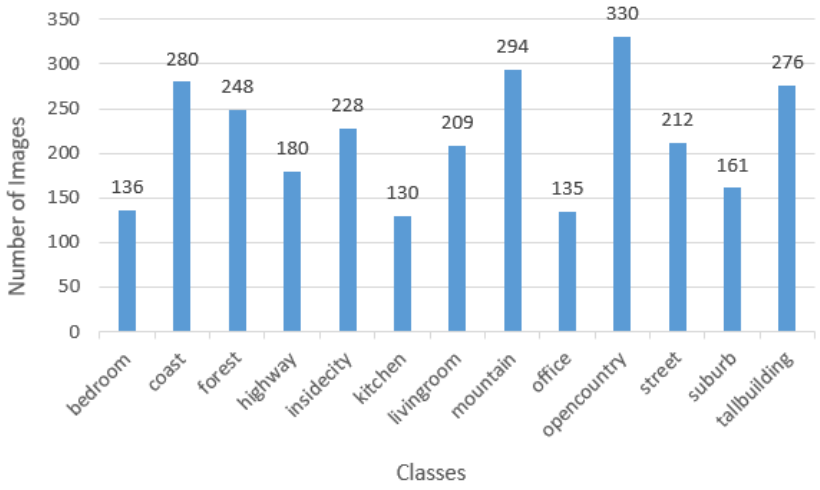

Figure 2.1: Distribution of the images based on the class.

2.2 Tuning Hyperparameters

By resizing images to small resolutions the model loses important information, on the other hand if the images have their original resolution than the model captures too many informations (e.g., training image noise) and tends to overfit. Following these considerations and by examining the content of the images, I decided to resize all the images to the size 100×100.

I found that a batch size of 64 offers the best trade-off between model training time and batch volatility.

I fixed the number of epochs to 100 and I used the EarlyStopping callback.

2.3 Model

Since the dataset is rather small (Section 2.1) and has some similarities with ImageNet [6], I based the core of my model on VGG16 [3] (trained on ImageNet and without the top layers) by using the *transfer learning* technique. On top of the VGG16 model I added a new block of layers consisting of a Global Average Pooling layer and some Dense layers with regularization in the form of dropout. There are 13 output classes, so the final output layer has 13 nodes. The model architecture is presented in Table 2.1.

I used transfer learning to boost the model accuracy and to reduce the training time. The choice of VGG16 among all the pre-trained models was mainly based on the idea that the bigger models (in MB) are slower but more accurate (see Figure 3 for the characteristics of the state-of-the-art models).

I chose to use the Global Average Pooling layer instead of the Flatten because, even if the validation accuracy of the solutions returned by the model using the two different layers is similar, the model with Global Average Pooling layer seems to overfit less the dataset (as suggested in [4]).

To prevent overfitting, I also chose to only fine-tune the last convolutional block rather than the entire network, since the entire network would have a very large entropic capacity and thus a strong tendency to overfit. The features learned by low-level convolutional blocks are more general, less abstract than those found higher-up, so it is sensible to keep the first few blocks fixed (more general features) and only fine-tune the last one (more specialized features).

To compile the model, I found that Adam optimizer [1] (using a callback for decreasing the learning rate when the training process is stuck in a local minimum) gives better results than SGD.

To fit the model, I used the following three callbacks:

- EarlyStopping, which stops the training of the model if the validation accuracy consistently gets worse after a certain number of epochs
- ReduceLROnPlateau, which reduces the learning rate 10x at a time when it detects model performance is no longer improving between epochs
- ModelCheckpoint, which saves the weights of the best model computer so far

Layer (type)	Output Shape	Number of Parameters
Conv2D	(None, 150, 150, 64)	1792
Conv2D	(None, 150, 150, 64)	36928
MaxPooling2D	(None, 75, 75, 64)	0
Conv2D	(None, 75, 75, 128)	73856
Conv2D	(None, 75, 75, 128)	147584
MaxPooling2D	(None, 37, 37, 128)	0
Conv2D	(None, 37, 37, 256)	295168
Conv2D	(None, 37, 37, 256)	590080
Conv2D	(None, 37, 37, 256)	590080
MaxPooling2D	(None, 18, 18, 256)	0
Conv2D	(None, 18, 18, 512)	1180160
Conv2D	(None, 18, 18, 512)	2359808
Conv2D	(None, 18, 18, 512)	2359808
MaxPooling2D	(None, 9, 9, 512)	0
Conv2D	(None, 9, 9, 512)	2359808
Conv2D	(None, 9, 9, 512)	2359808
Conv2D	(None, 9, 9, 512)	2359808
MaxPooling2D	(None, 4, 4, 512)	0
Flatten	(None, 8192)	0
Dense	(None, 256)	2097408
Dropout	(None, 256)	0
Dense	(None, 13)	3341

Table 2.1: Final model architecture. Total parameters: 16815437 (trainable: 9180173, non-trainable: 7635264).

3. Summary

In this document, I have presented and described the methodology used to solve the Kaggle challenge for Homework 1. Each step of the methodology has been discussed and motivated.

Due to the size of the dataset, the model easily tends to overfit. To prevent this phenomenon, different techniques have been applied to the model (e.g., data augmentation, class balance, transfer learning). To fine-tune the model and to deal with overfitting and underfitting problems, I made use of learning curves. The accuracy and loss curves of the final model (discussed in this document) are represented, resp., in Figure 3.1 and Figure 3.2.

By building and training the model as described in this document, the test accuracy obtained in the Kaggle challenge is 95.384.

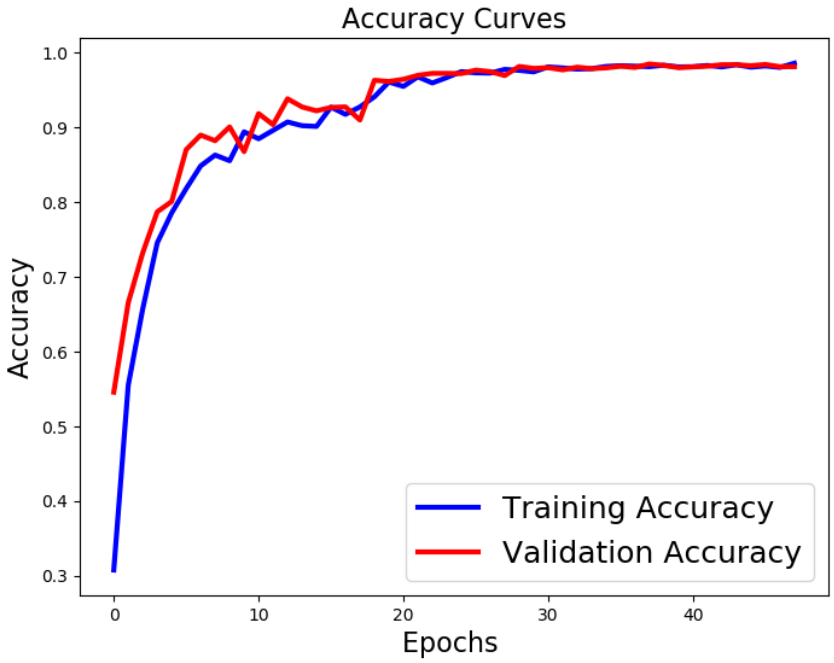

Figure 3.1: Accuracy plot produced by the final model. The model has been trained for 48 epochs with batch size 64 and images of size 100×100. The training of the model has been stopped by the EarlyStopping callback. The validation and the test accuracy are very similar, which means that the model is capable of learning the task efficiently. Note that without using the EarlyStopping callback the model would have probably overfitted the training data and the validation accuracy would have decreased.

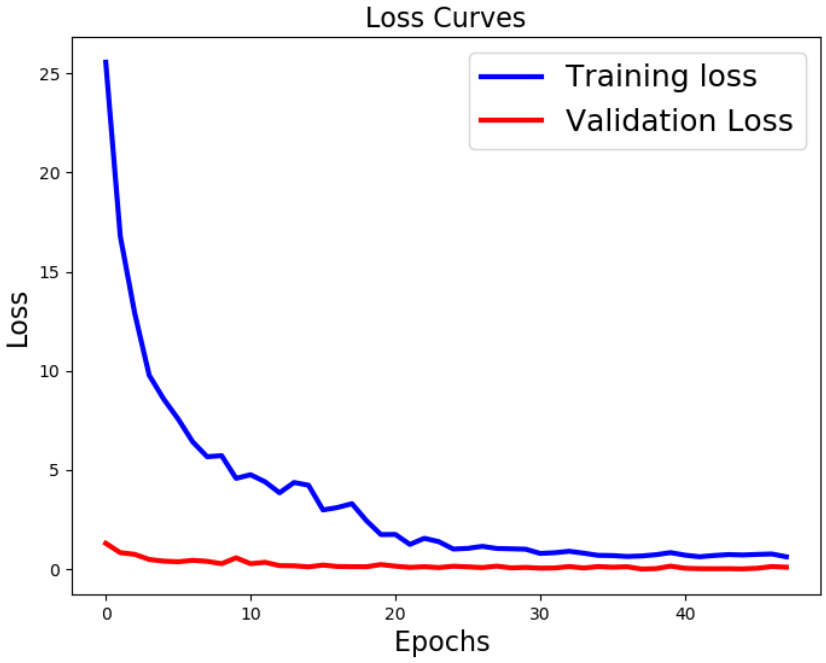

Figure 3.2: Loss plot produced by the final model. The model has been trained for 48 epochs with batch size 64 and images of size 100×100. The training of the model has been stopped by the EarlyStopping callback.

APPENDIX

A.1 GitHub Repository

The code of the project is available on GitHub, in the repository [5].

A.2 Keras Models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Figure 3: Keras Models.

Bibliography

[1] Kingma D., and Ba J. *Adam: A Method for Stochastic Optimization.* Conference paper at ICLR, 2015.

[2] Menegola A., Fornaciali M., Pires R., Bittencourt F., Avila S., and Valle E. *Knowledge Transfer for Melanoma Screening with Deep Learning.* RECOD Lab, DCA, FEEC, University of Campinas (Unicamp), Brazil.

[3] Simonyan K., and Zisserman A. *Very deep convolutional networks for large-scale image recognition.* Conference paper at ICLR 2015, Visual Geometry Group, University of Oxford.

[4] Wu Y., and Lee T. *Reducing Model Complexity for DNN Based Large-Scale Audio Classification.* IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, 2018, pp. 331-335.

Sitography

[5] Github Repository, last access 17 Oct. 2019, [Online].
https://github.com/AlessandroSaviolo/CS_IOC5008_0845086_HW1

[6] ImageNet (2016), last access 13 Oct. 2019, [Online].
http://www.image-net.org

[7] Kaggle (2010), last access 17 Oct. 2019, [Online].
https://www.kaggle.com/

[8] Keras Challenge (2015), last access 16 Oct. 2019, [Online].
https://www.kaggle.com/c/cs-ioc5008-hw1

[9] Keras Blog (2016), last access 16 Oct. 2019, [Online].
https://blog.keras.io/