Homework 2 - Machine Learning: Weather Classification

Edoardo Piroli - 1711234

December 15, 2019

Contents

1 Introduction

1.1 Assignment

The assignment of the homework was to build an image classifier capable of assigning pictures to one of the following classes: {*RAINY*, *HAZE*, *SUNNY*, *SNOWY*}; based on how the weather was like at the moment of the picture's capture. In particular, the request was to address the former problem in 2 different ways:

1. Applying transfer learning and fine-tuning on a pre-trained model;
2. Defining and training a CNN from scratch for this particular task.

1.2 Dataset

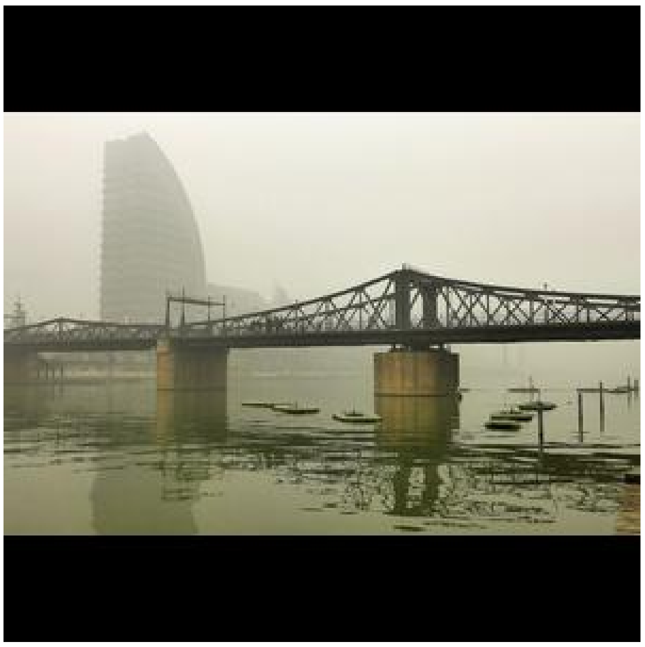
The provided datasets[1] are separated in several different archives and folders. I have chosen to use *MWI-Dataset-1.1_2000* for training, *MWI-Dataset-1.2.4* for testing and *MWI-Dataset-1.2.5* for validation. All these datasets are perfectly balanced, in particular the training dataset is composed by 2000 pictures, 500 per class, while the validation and testing datasets are composed by 400 pictures(100 per class) each.

Chosen resolution The pictures in the datasets don't share a common resolution, this is why I had to resize them in order to be able to train and test.

In particular, I have chosen to use 299x299 as the pictures' resolution; I have tried other resolutions but this seemed to outperform the others(as expected since the chosen pre-trained model was originally trained on that resolution). I have tried 2 different approaches to resize the pictures:

1. **Padded**: Preserving the original aspect ratio, I have resized the longest side to 299, keeping constant the aspect ratio; i.e. proportionately resizing also the other side; and pasted the so-obtained picture on a black 299x299 square; basically adding some padding to the edges. As an example of this resizing process here is one sample picture:

(a) Original picture with resolution 900x601


(b) Resized picture with resolution 299x299

2. **Non-Padded**: I've used the keras.preprocessing.image.ImageDataGenerator flow_from_directory method to resize the pictures.

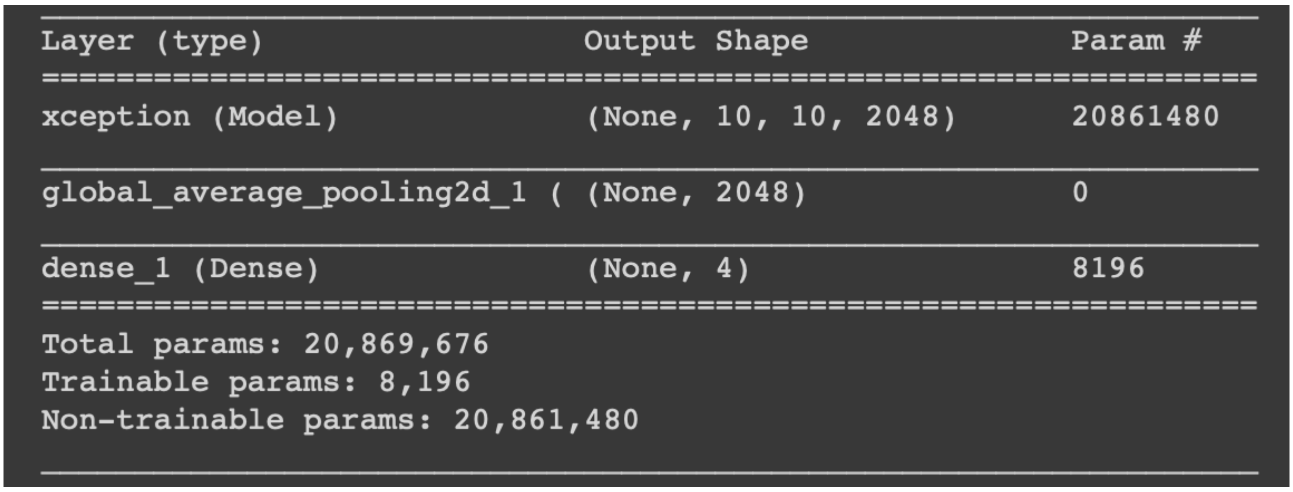
Although, differently from what I expected, the second, more classical, method of resizing performed better.

[1]available here: https://drive.google.com/drive/folders/1UzH28Q8xki8_DMYdDgHxi40-CJ800Kaq

2 Pre-trained Model

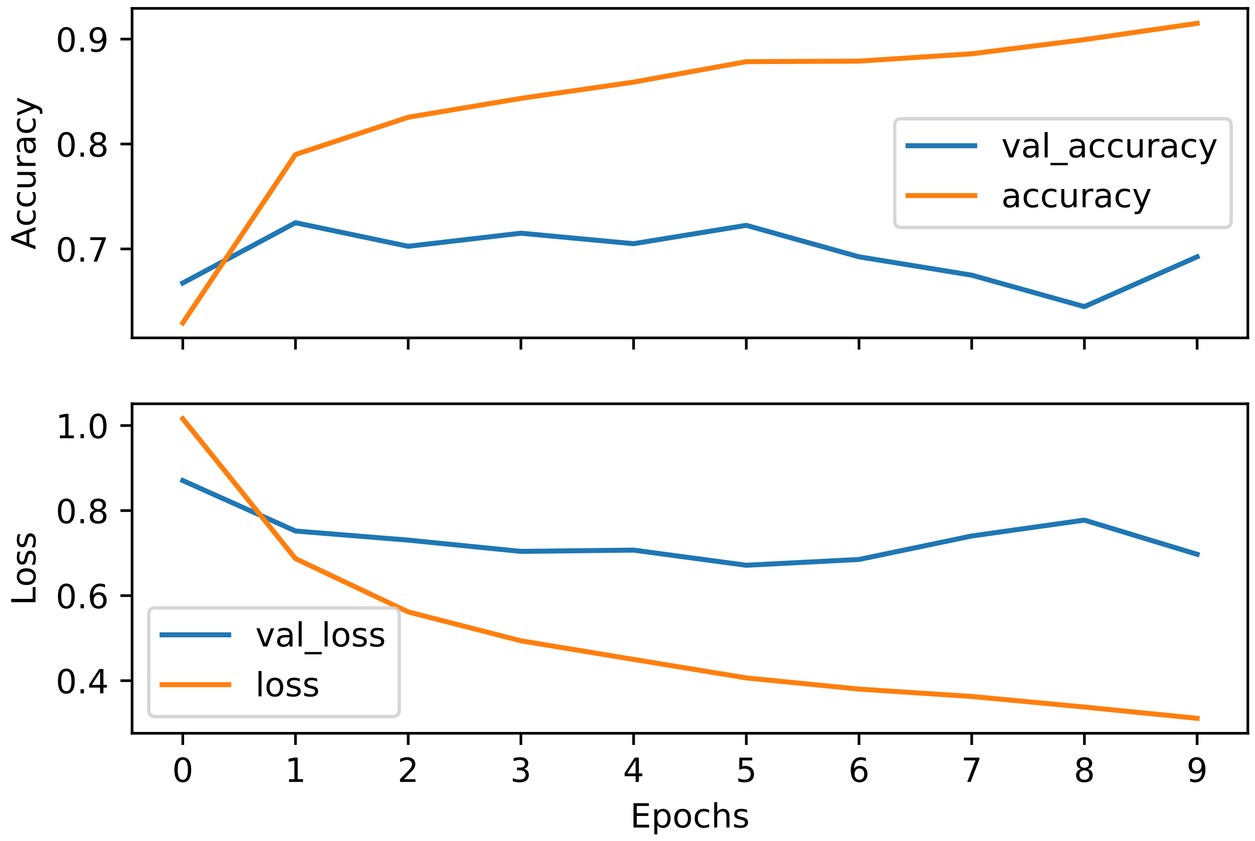
After looking at the pre-trained models available here, I have chosen to use the Xception model, characterised by a good ratio between size and accuracy.

The first pre-trained model that I tried is the following one, note that the xception model is completely frozen, i.e. its layers are not going to be modified during the backtracking phase:

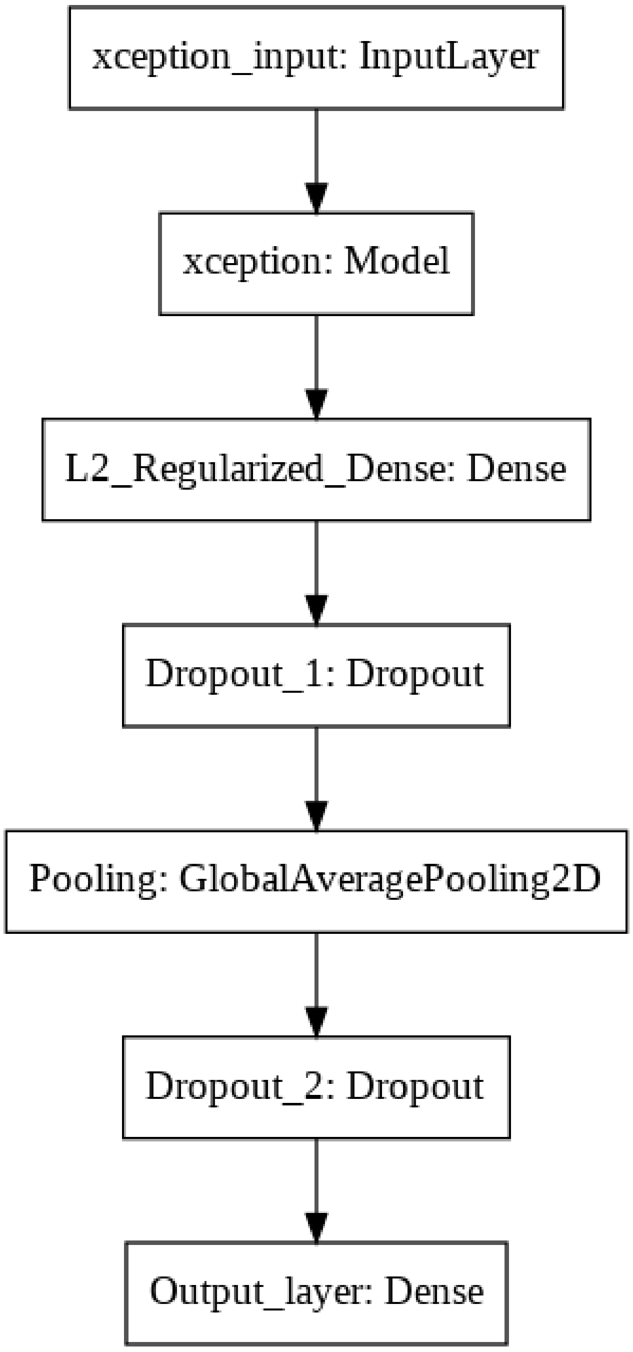


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, 10, 10, 2048)      20861480
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 8196
=================================================================
Total params: 20,869,676
Trainable params: 8,196
Non-trainable params: 20,861,480
_________________________________________________________________
```

Due to the small number of samples in the training dataset, this model performed poorly, clearly overfitting in less than 10 epochs:



In order to alleviate this problem I have decided to try to add some dropout and regularization terms in the model, the final model which I obtained is the following one:


xception_input: InputLayer
xception: Model
L2_Regularized_Dense: Dense
Dropout_1: Dropout
Pooling: GlobalAveragePooling2D
Dropout_2: Dropout
Output_layer: Dense

2.1 Hyperparameters

The hyperparameters of the pre-trained model are:

- **Dense size**: The size of the regularized dense layer;
- **Regularization term**: The value for the kernel regularizer and the bias one;
- **Dropout rate**: Shared among the 2 dropout layers;
- **Activation Functions**: I have used 'softmax' for the output layer and 'relu' for the regularized dense layer;

- **Optimization Function**: I have just used RMSProp;
- **Learning rate**: The value used by RMSProp;
- **Rho**: Other value used by RMSProp which I have set to 0.9, as suggested in the keras documentation;
- **Epochs**: The number of epochs to train the models;
- **Batch size**: I have used 50.

2.2 GridSearch

In order to try to optimize as much as possible the model I wanted to tune the hyperparameters performing a GridSearch, which in the end I carried out only partially due to time and computational power limitations.

In particular, I wanted to test the following values:

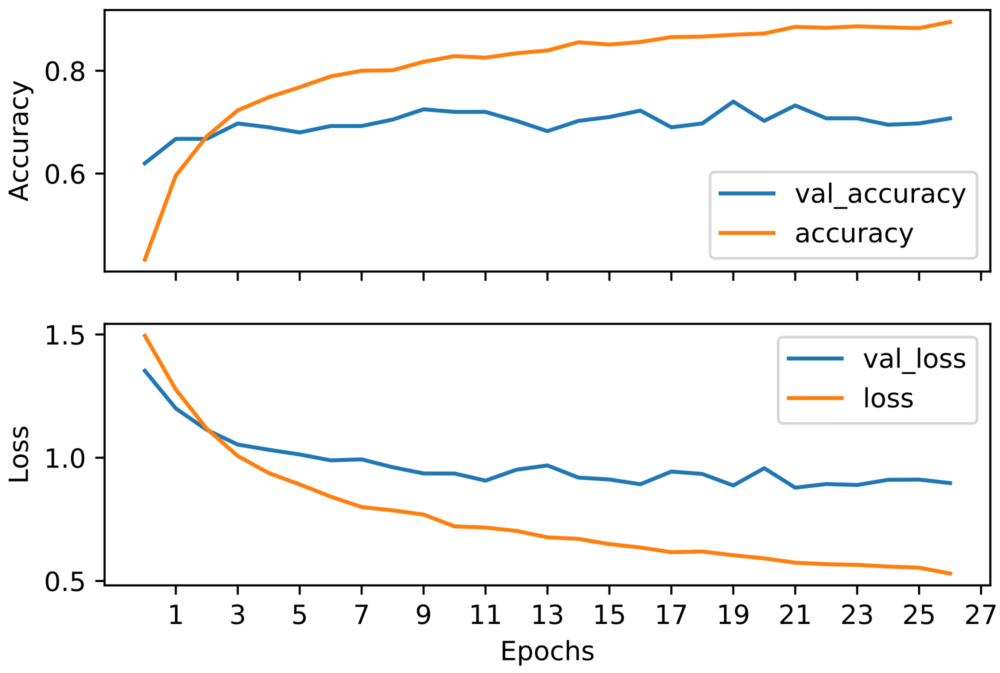
- **Dense size**: {64, 128};
- **Regularization term**: {0.001, 0.01};
- **Dropout rate**: {0.3, 0.5};
- **Learning rate**: {0.0001, 0.001};
- **Epochs**: I have used early stopping on the validation loss with patience=3.

Pre-trained model best hyperparameters: As expected, due to the great presence of overfitting, a higher value of the dropout rate and a smaller learning rate performed better.

Dense size	Regularization term	Dropout rate	Learning rate
128	0.001	0.5	0.0001

2.3 Training tuned pre-trained model

These are the plots of the training of the tuned pre-trained model:



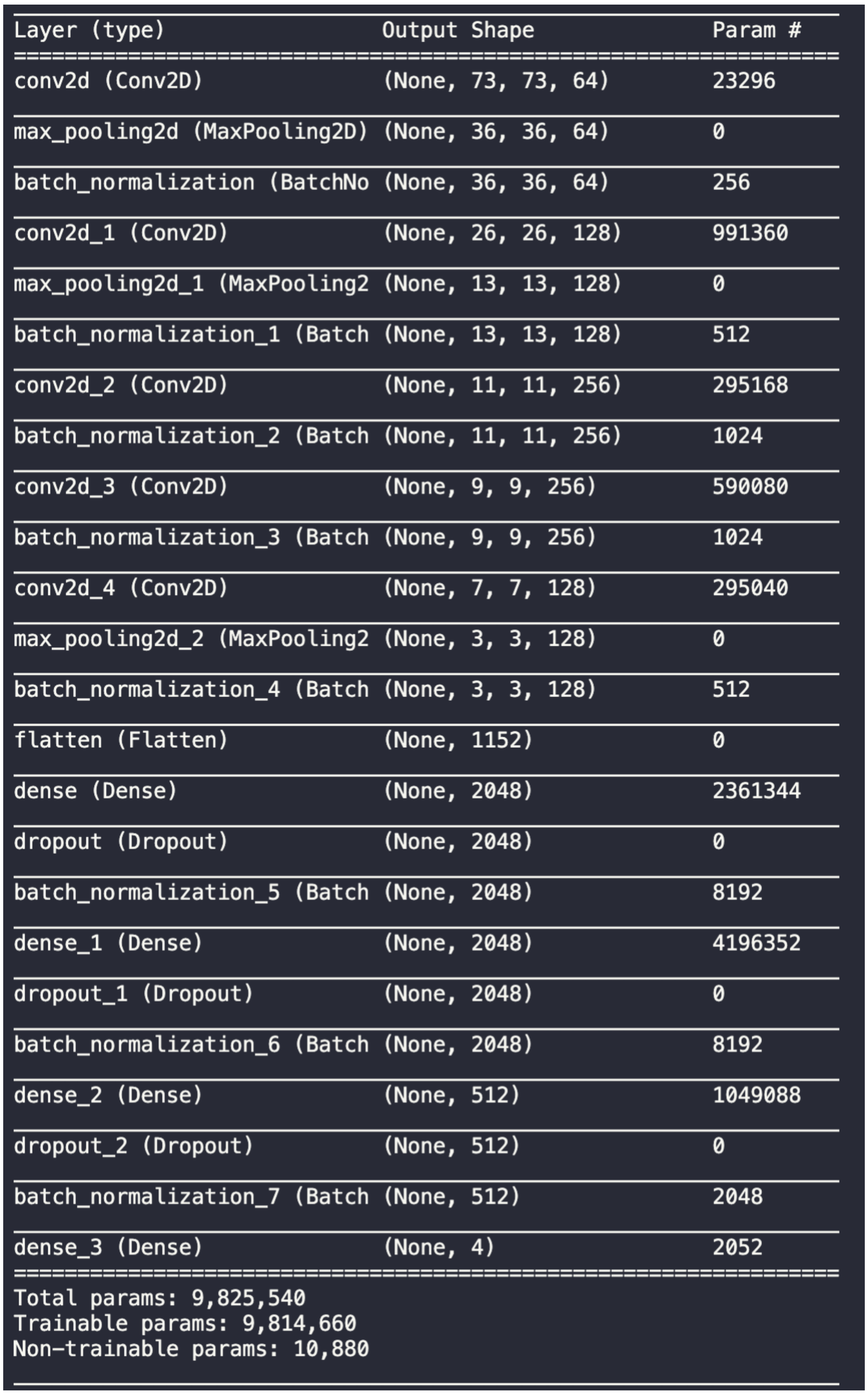
We can see that this model still overfits a lot but on a larger number of epochs w.r.t. the first model.

Results of the final model:

	Accuracy	Loss
Training	0.886	0.574
Validation	0.733	0.878
Testing	0.715	0.983

3 New Model

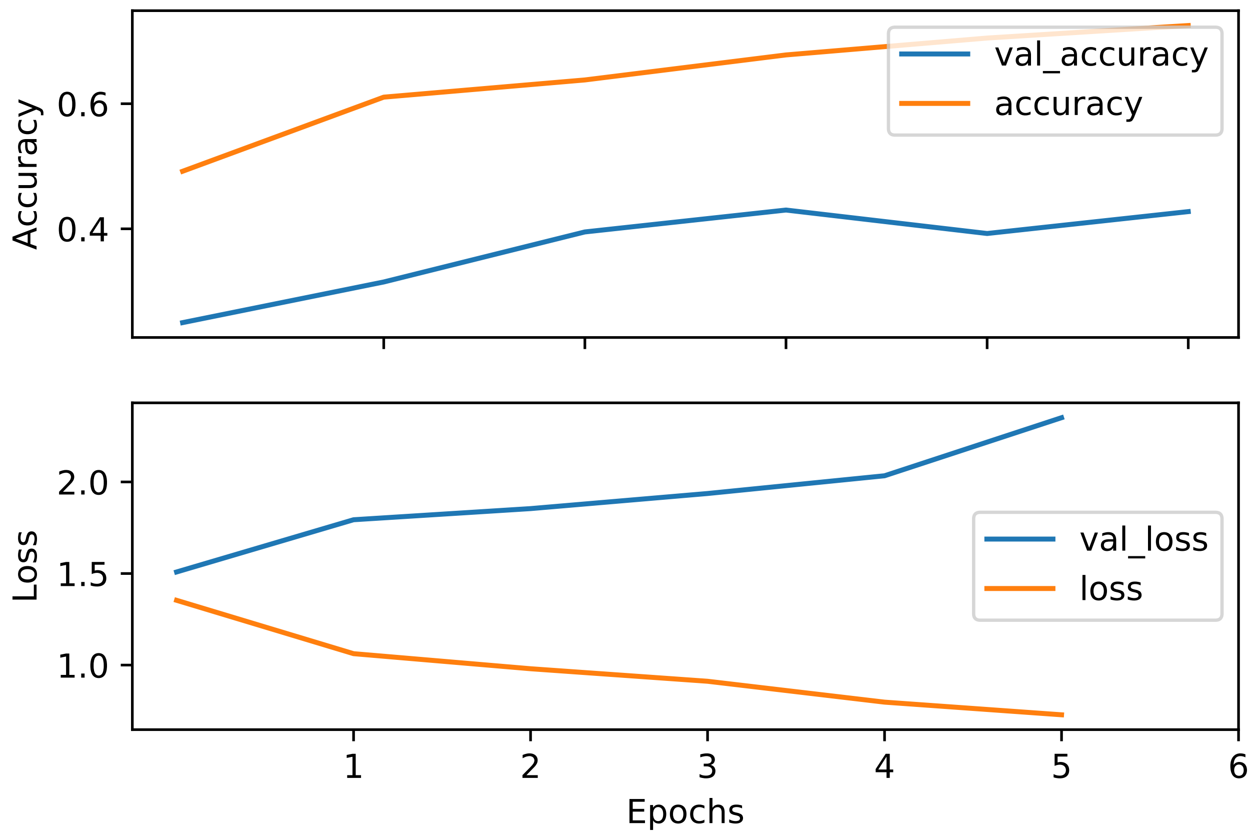
I have chosen to implement the AlexNet model, as follows:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 73, 73, 64)        23296
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 36, 36, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 36, 36, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 128)       991360
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 256)       295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 256)         590080
_________________________________________________________________
batch_normalization_3 (Batch (None, 9, 9, 256)         1024
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 128)         295040
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 2048)              2361344
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 2048)              8192
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              4196352
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 2048)              8192
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 2052
=================================================================
Total params: 9,825,540
Trainable params: 9,814,660
Non-trainable params: 10,880
_________________________________________________________________
```

3.1 Training

For this model, due to time limits, I have not run any hyperparameters tuning and this is one of the reasons why it performed extremely badly as we can see in the following plots:



Results of the final model:

	Accuracy	Loss
Training	0.492	1.354
Validation	0.250	1.508
Testing	0.433	2.267

4 Results

4.1 Comparison

In the end, the pre-trained model, largely, outperformed the new one, this is mainly due to the following reasons:

1. I have spent more time and effort on the former model, and did not even manage to tune the hyperparameters of the latter one;
2. The pre-trained model has been trained for many epochs on a huge dataset of images and it can generate a well-refined set of features from each picture, this can represent a huge advantage when dealing with a niche task, like this one, with very few training samples.

Obviously, I have submitted the pre-trained model.

4.2 Proposed Improvements

The results achieved can be largely improved, some ideas to do so are the following:

- Test other pre-trained models;
- Improve the architecture of the model;
- Train on more samples or perform data augmentation;
- Test other hyperparameters' values or perform RandomSearch;
- Test other resolutions.