Weather image classification

Homework 2 - Machine Learning
Engineering in Computer Science
"La Sapienza" University of Rome

Costa Marco 1691388

15 December 2019

Contents

1 Introduction

The purpose of this homework is to solve a *multi-class* image classification problem on the ***Multi-class Weather Image Dataset***. The problem must be solved using two modes:

1. Define a CNN and train it from scratch;
2. Apply transfer learning and fine tuning from a pre-trained model.

After that, we have to evaluate the two models in a proper way. The metrics used for the comparison of the models are the accuracy and the loss function (*Categorical Crossentropy*). Finally we have to perform a prediction on the ***WeatherBlindTestSet*** and create a *.csv* file in which there is the predicted label for each image. Moreover, we have to provide a personal dataset made with our photos.

1.1 Dataset

The ***Multi-class Weather Image Dataset*** contains images grouped in four classes: **HAZE**, **RAINY**, **SNOWY**, **SUNNY**. It is very balanced: the whole dataset has 1000 images for each class. It is organized in a file system structure with four folders corresponding to the four classes *HAZE*, *RAINY*, *SNOWY*, *SUNNY* and containing the corresponding images. In addition, there is the ***SMART-I weather test set*** that can be used as test set. It contains only the three classes *RAINY*, *SNOWY*, *SUNNY* and it is very unbalanced. Images in the datasets have different shapes and resolutions. It is necessary to reshape these images to make image dimensions consistent with the input layer of the network. Finally there is the ***WeatherBlindTestSet*** on which we have to perform the prediction with the best model.

1.2 Hardware

Since we use tensorflow and keras libraries, the performance (training time, prediction, etc.) depends on the hardware, and in particular on the GPU. We used the *Nvidia GeForce GTX 950M*, having 640 *CUDA cores*, 4Gb of *DDR3 Memory* and 1000 MHz of clock.

2 Define a CNN and train it from scratch

Since defining a CNN is very difficult, I decided to use a known network, namely AlexNet, to have a comparison. All the nets were trained in 100 epochs, with the same image input dimension (118×224), all the images were loaded in color mode "*RGB*" with batch size 32 and data augmentation. My nets are named *KucoNet v.x*.

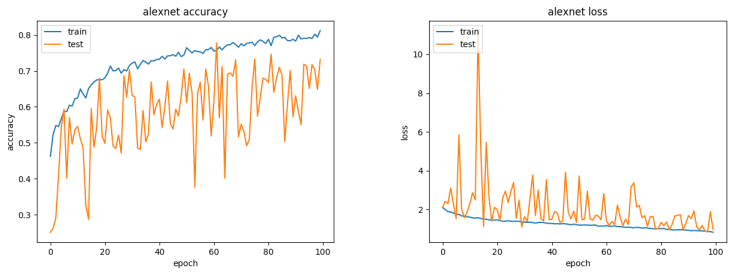
2.1 AlexNet

Since *AlexNet* is a well-known CNN, its definition isn't shown.
Parameters:

- Total params: 28,083,756
- Trainable params: 28,062,620
- Non-trainable params: 21,136

Results:

- Final epoch performances:
  - Accuracy: 0.8122
  - Loss: 0.8266
- Test set performances
  - Accuracy : 0.7320
  - Loss: 0.9699
- Training time: $\approx$ 1 hour and 5 minutes



(a) Accuracy (b) Loss

Figure 1: AlexNet performaces plots

2.2 KucoNet v.1

The first CNN is defined as follow:

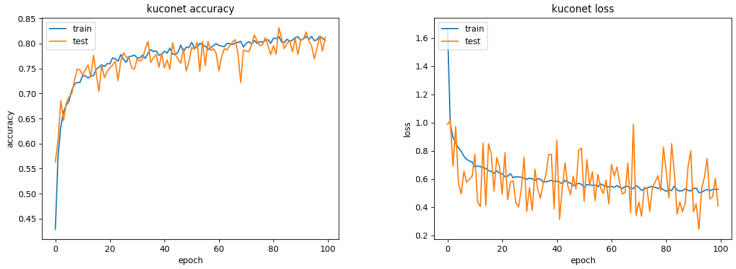
Layer (type)	**Parameters**
Convolutional	32, kernel 5×5
Max Pooling	pool 2×2
Convolutional	64, kernel 3×3
Max Pooling	pool 2×2
Flatten	
Fully Connected	1024 neurons
Fully Connected	4 neurons

The first convolutional layer produces 32 feature maps using a 5×5 kernel followed by max-pooling operation. The second convolutional layer produces 64 feature maps using a 3×3 kernel followed by a max-pooling operation. After that, feature maps are sent to the dense layer, which includes 1024 neurons. Finally, the last layer is a dense layer composed by 4 neurons that produces the output. The activation function is *relu* for all the layers, except the last one that uses *softmax*. The model is compiled with the *Adam* optimizer and the *categorical_ crossentropy* loss function.
Parameters:

- Total params: 95,577,540
- Trainable params: 95,577,540
- Non-trainable params: 0

Results:

- Final epoch performances:
  - Accuracy: 0.8066
  - Loss: 0.5284
- Test set performances
  - Accuracy : 0.8125
  - Loss: 0.4069
- Training time: $\approx$ 1 hour and 33 minutes



(a) Accuracy (b) Loss

Figure 2: MyNet v.1 performaces plots

2.3 KucoNet v.2

The second CNN is defined as follow:

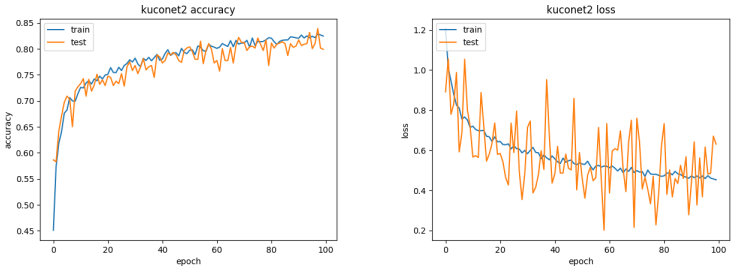
Layer (type)	**Parameters**
Convolutional	56, kernel 3×3
Max Pooling	pool 2×2
Convolutional	32, kernel 3×3
Max Pooling	pool 2×2
Flatten	
Fully Connected	64 neurons
Fully Connected	4 neurons

The first convolutional layer produces 56 feature maps using a 3×3 kernel followed by max-pooling operation. The second convolutional layer produces 32 feature maps using a 3×3 kernel followed by a max-pooling operation. After that, feature maps are sent to the dense layer, which includes 64 neurons. Finally, the last layer is a dense layer composed by 4 neurons that produces the output. The activation function is *relu* for all the layers, except the last one that uses *softmax*. The model is compiled with the *Adam* optimizer and the *categorical_ crossentropy* loss function.
Parameters:

- Total params: 3,114,628
- Trainable params: 3,114,628
- Non-trainable params: 0

Results:

- Final epoch performances:
  - Accuracy: 0.8247
  - Loss: 0.4534
- Test set performances
  - Accuracy : 0.7993
  - Loss: 0.6309
- Training time: $\approx$ 56 minutes



(a) Accuracy (b) Loss

Figure 3: MyNet v.2 performaces plots

2.4 KucoNet v.3

The third CNN is defined as follow:

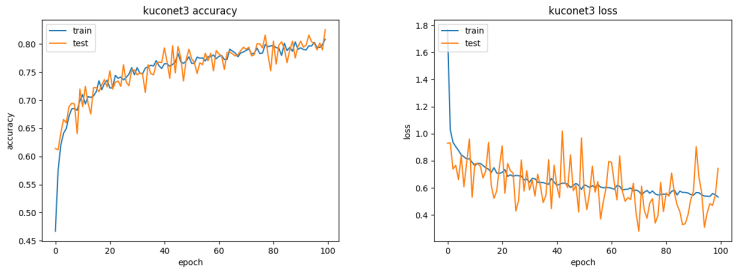
Layer (type)	**Parameters**
Convolutional	32, kernel 3×3
Max Pooling	pool 2×2
Convolutional	64, kernel 2×2
Max Pooling	pool 2×2
Flatten	
Fully Connected	256 neurons
Fully Connected	4 neurons

The first convolutional layer produces 32 feature maps using a 3×3 kernel followed by max-pooling operation. The second convolutional layer produces 64 feature maps using a 2×2 kernel followed by a max-pooling operation. After that, feature maps are sent to the dense layer, which includes 256 neurons. Finally, the last layer is a dense layer composed by 4 neurons that produces the output. The activation function is *relu* for all the layers, except the last one that uses *softmax*. The model is compiled with the *Adam* optimizer and the *categorical_ crossentropy* loss function.
Parameters:

- Total params: 27,076,804
- Trainable params: 27,076,804
- Non-trainable params: 0

Results:

- Final epoch performances:
  - Accuracy: 0.8087
  - Loss: 0.5324
- Test set performances
  - Accuracy : 0.8257
  - Loss: 0.7442
- Training time: $\approx$ 54,5 minutes



(a) Accuracy (b) Loss

Figure 4: MyNet v.3 performaces plots

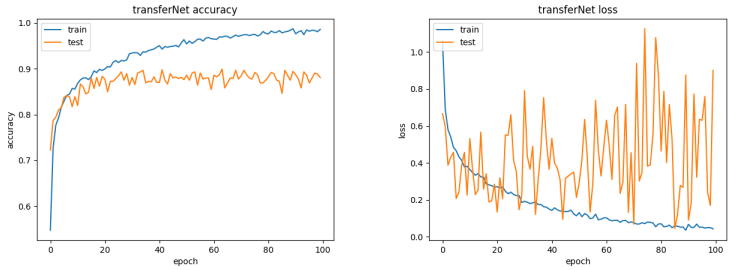
3 Transfer Learning with fine tuning

For this task I used a ***VGG16*** net pre-trained on *imagenet*. I loaded the pre-trained model without the last layer (*output*) and I added some new layers: a Global Average Pooling layer, a Fully Connected layer composed by 100 neurons and a Fully Connected layer of 4 neurons. Also in this case the activation function is *relu* for middle layersand *softmax* for the output layer. The model is compiled with the *Adam* optimizer and the *categorical_ crossentropy* loss function.
Parameters:

- Total params: 14,766,392
- Trainable params: 13,030,904
- Non-trainable params: 1,735,488

Results:

- Final epoch performances:
  - Accuracy: 0.9859
  - Loss: 0.0433
- Test set performances
  - Accuracy : 0.8810
  - Loss: 0.9003
- Training time: $\approx$ 2 hours and 15 minutes



(a) Accuracy (b) Loss

Figure 5: TransferNet performaces plots

4 Testing on other datasets

All the nets were tested on *SMART-I Weather Dataset* and my personal dataset named *Kuco Dataset*. Finally I used the *TransferNet* model for prediction.

4.1 SMART-I Weather Dataset

Testing the nets on this dataset I had the following results:

Net	**Performances**
AlexNet	Accuracy: 0.331797
	Loss: 2.828256
KucoNet v.1	Accuracy: 0.478934
	Loss: 1.728153
KucoNet v.2	Accuracy: 0.499671
	Loss: 1.944088
KucoNet v.3	Accuracy: 0.455563
	Loss: 3.439912
TransferNet	Accuracy: 0.598420
	Loss: 2.567181

It's possible to see how all the nets perform terribly on this dataset.

4.2 Kuco Dataset

This dataset is composed by some personal photos. I tried to create a balanced dataset (5 images for each class), but unfortunately I had only one rainy photo. I tested all the networks on this dataset and got the following results:

Net	Performances
AlexNet	Accuracy: 0.437500
	Loss: 1.581957
KucoNet v.1	Accuracy: 0.25
	Loss: 1.615923
KucoNet v.2	Accuracy: 0.375
	Loss: 2.081731
KucoNet v.3	Accuracy: 0.5625
	Loss: 1.726111
TransferNet	Accuracy: 0.5
	Loss: 4.648948

Surprisingly, *KucoNet v.3* performed better than *TransferNet*, and *AlexNet* better than *KucoNet v.1* and *KucoNet v.2*

4.3 WeatherBlindTestSet prediction

I used the *TransferNet* model to perform the prediction on *WeatherBlindTestSet Dataset*. In attachment is provided the output *.csv* file. 321 images were classified as *SUNNY*, 502 as *SNOWY*, 259 as *HAZE* and 418 as *RAINY*.

5 Conclusions

Let's summarize the results in a compact way:

Net	Accuracy	Loss	Training time
AlexNet	0.7320	0.9699	$\approx$ 1h:5m
KucoNet v.1	0.8066	0.5284	$\approx$ 1h:33m
KucoNet v.2	0.7993	0.4534	$\approx$ 56m
KucoNet v.3	0.8257	0.7442	$\approx$ 54,5m
TransferNet	0.8810	0.9003	$\approx$ 2h:15m

It's easy to see that all *KucoNets* perform almost the same way (each of them performs better than *AlexNet* but worst than *TransferNet*). The *TransferNet* model has the best accuracy, but the loss is higher than the other nets. Moreover, it required much more time for training. From the graphs it is also possible to highlight how loss is very unstable in time for all

the nets. *KucoNet v.3* performs better than the other two, having the lower training time.

After all these considerations it is possible to conclude that solve an image classification problem based on the weather condition is a very difficult task.