Deep Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Convolutional Neural Network – Weather Dataset

Group Members - 23

António Fonseca number: r20181154

João Carvalho, number: r20181122

Tiago Sousa: r20181077

Diogo Pereira: m20210657

Pedro Joel: m20180040

April, 2022

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1 INTRODUCTION

Our group was asked to develop a Deep Learning project where the topics covered in the course would be applied. The chosen problem concerns the identification of the weather on the images provided. Therefore, the objective is to develop a model that predicts to which class each image belongs.

2 TASK DEFINITION

This goal of the project is to develop a model that identifies the weather from the images presented to it, whether it is cloudy, rainy, shine or sunrise. The information retrieved from the data can then be transformed into knowledge by optimizing a smart home application. For instance, the application would use the information given by our model to set the correct lightning in a house, open or close the motorized blinds and adjust the luminosity of the outdoor security cameras' app. Furthermore, we can determine whether or not a garden needs to be watered by using the past information weather.

To conclude, our solution trains a model to identify the weather in the image. The real-world problem that this system tries to solve is optimizing smart home applications by recording frames from outside cameras and analysing the weather in each image.

3 DATA PRE-PROCESSING

The first step before starting to build the model is to pre-process the data. The dataset is composed of 1125 images and by exploring the dataset, we verified that the data is labelled into four categories: cloudy, rain, shine and sunrise. Then, we checked whether or not the dataset was balanced. The class with more images is sunrise (31.7%) and the one with fewer images is rain (19.1%).

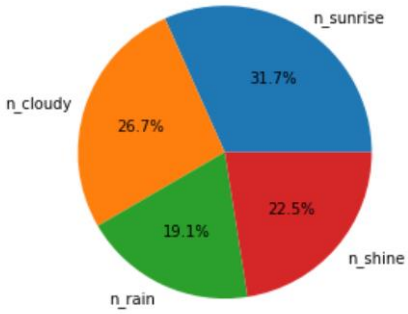


Figure 1 - Composition of the original dataset

In the original dataset the images were not split into folders by class and, therefore, we had to perform that task (the code has to be inside the folder). In order to implement the CNN models, we also had to divide our dataset into train, validation, and test sets. 70% of the data was allocated for the training set and 15% each to the validation and test sets.

4 APPROACH

After analysing the dataset and concluding the pre-processing part we started creating several models. Since images have high dimensionality and Convolutional Neural Network models are effective in reducing the number of parameters without losing the quality of their models, we decided to use CNNs.

The models were developed to range in complexity and efficiency in order to analyse the trade-off between accuracy and running time. Since our problem has four classes it is a Multi-class Classification and, therefore, all the models created have these parameters in common: activation function = *softmax* and loss function = *categorical_crossentropy*.

We used Data augmentation to import all the data but performed some adjustments on the training set in order to maximize the predictive power of our CNN by making sure that the test and validation data are not too similar with the training data.

To make our models efficient we used some callbacks: *EarlyStopping* which acts during the training of the model and stops the training when the performance of the model is not increasing during a pre-defined number of epochs; *ReduceLROnPlateau* which also acts during the training of the model and reduces the learning rate once the performance stops increasing, similar to *EarlyStopping*.

Finally, for evaluating the model we decided to plot the *accuracy* and the loss per epoch and created a confusing matrix and a classification report where in addition to *accuracy* we also analysed *precision*, *recall*, *f1-score* and *support*.

5 MODELS

For the creation of our models, we based ourselves in the various CNN architecture that already exist, such as *LeNet*, *AlexNet*, *VGGNet*, *GoogLeNet*, between many others. However, these architectures were too complex and with too many layers (therefore taking too much time running). Having this in mind, we decided to keep our architecture as simple as possible, while still maintaining the typical layers that exist in the CNNs mentioned above (as you can see in the image below).

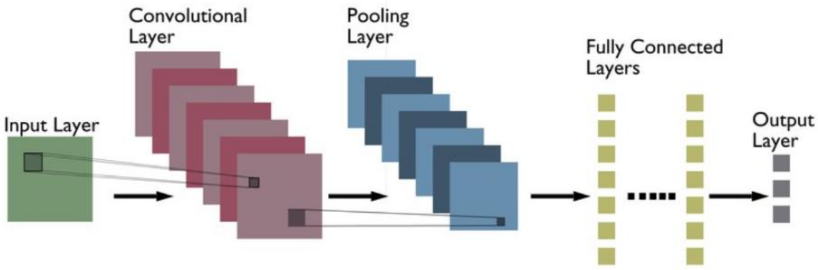


Figure 2 - Typical CNN architecture

Having this in mind, we built 4 different models to decide on which CNN architecture our data would work the best. The following table is a simple representation of each one of them, where you can see the number of layers and the number of blocks used (blocks as in sequences of *Conv2D* and *MaxPool* together). Note that all these architectures do not have more than 10 layers each. We opt for doing so because, even though a more complex network might yield better results, we saw that networks too complex would take really a lot more time to run.

Model 1	Model 2	Model 3	Model 4
Conv2D	Conv2D	Conv2D	Conv2D
MaxPool	MaxPool	Conv2D	MaxPool
Conv2D	Conv2D	MaxPool	Conv2D
MaxPool	MaxPool	Conv2D	MaxPool
GlobalMax	Conv2D	Conv2D	GlobalAverage
Dense (weight regularizer)	MaxPool	MaxPool	Dense (weight regularizer)
Dropout (0.2)	GlobalAverage	GlobaMax	Dropout (0.4)
Dense (weight regularizer)	Dense (weight regularizer)	Dense (weight regularizer)	Dense (weight regularizer)
Dropout (0.2)	Dropout (0.3)	Dense (softmax)	Dropout (0.4)
Dense (softmax)	Dense (softmax)		Dense (softmax)

Table 1 - CNN architectures

- **Convolution Layers**

  - Input shape: only the first convolutional layers receive this parameter as it is the only one which receives the original image; the other layers´ input is the output of the preceding layer. We decided to only use 100*100 input dimensions as this is a parameter that might increase too much the computational effort of the system and decrease its efficiency (we experimented 250*250 input shape, but the training would take hours to run and for a minimal or inexistent better performance).

  - Filters: a filter is the simple feature which is going to be identified on the input and the number of filters is expected to grow along the CNN so that simple features can be turned into more complex and more discriminative features. We tried different numbers of filters for the different models (32, 64, 128,etc.) but the number always grows as we go deeper in the system, as it is expected to give the better results.

  - Kernel size: It represents the shape of the filters that were just mentioned. We mostly decided to use a 3*3 filter as it is the most commonly used but other shapes can also produce good results (5*5 filter e.g.).~

- Padding: It is an optional parameter which we decided to include in order to keep the edges of the image as relevant to the model as the other parts of the image.

➢ Pooling Layers

The pooling layers are meant to reduce the dimensionality of the input and keep only its most important features by sliding a window on the input map. We decided to try *MaxPooling* (which takes the highest element inside the window) and *AveragePooling* (which keeps the average of all the elements in the window). We mostly tried 2*2 windows as well as 3*3 windows

➢ Fully Connected Layers

The fully connected layers will produce the final output of the system, which is the probability of an image belonging to a certain class. More than one fully connected layer can exist: we tried with 1 and 2 layers as we don´t want to make the CNN inefficient by adding too many layers. The final layer has 4 units because we are dealing with a categorical problem including 4 different classes. The activation function is 'softmax' which ensures that the 4 probabilities sum up to 1

➢ Reduce Overfit Layers

- **Dropout:** It is a technique used to mitigate overfitting. It will randomly drop out (set to zero) a number of output features of the layer during the training process in order to introduce some randomness to the network and thus prevent it from learning very specific patterns from the training data (which would reduce its capacity of predicting new images). It is advised that the dropout rate - fraction of features that are ignored – are between 0.2 and 0.5, so we used values in this interval in our models.

- **Weight regularization:** The idea behind weight regularization is that simpler models are less likely to overfit when compared to more complex ones. We can achieve this by putting constraints on the complexity of the network we are building, forcing its weights to take only small values. There are two main ways to do this: L1 regularization (adds absolute value of magnitude of coefficient as penalty to the loss function, this way shrinking the parameters towards zero), and L2 regularization (adds squared magnitude as a penalty term to the loss function). After applying both of them, separately and together, we established that using L2 regularization would be more than enough to create models more resistant to overfitting.

6 EVALUATION

We evaluate the models by observing their validation accuracy and loss values along the epochs. We set the maximum of 100 epochs, but the callback *EarlyStoppping* as mentioned earlier, and for that reason, the total number of epochs is different for every model. We chose the patience parameter as 10 because we believe that is fair to consider that, if a model does not improve in 10 straight epochs, it is better to stop the process before it starts overfitting.

The following images show the performances of the 4 models in terms of accuracy and loss.

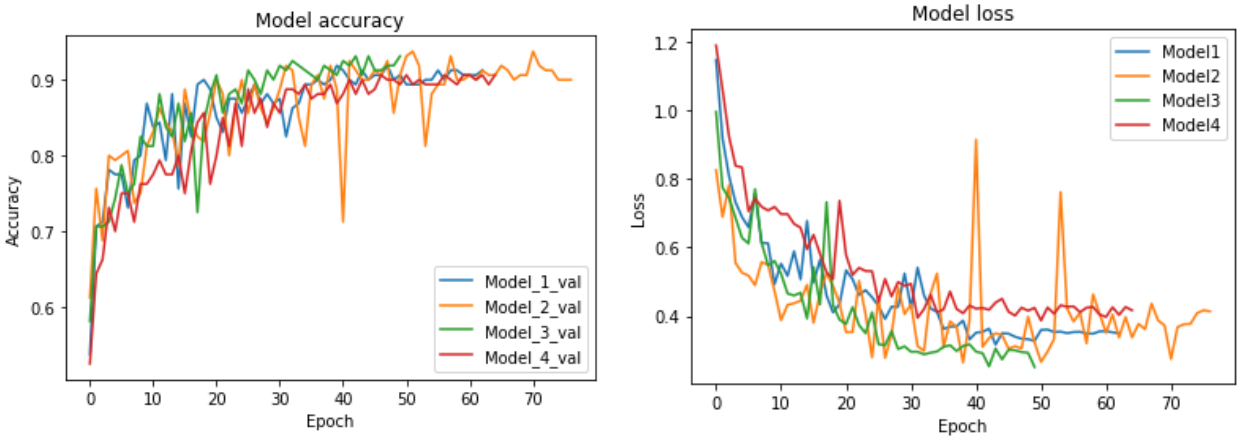


Figure 3 and 4 - Model accuracy and Model Loss

The models 1 and 3 seem to produce the best results as the accuracy and loss are increasing and decreasing, respectively, at a steady and constant pace. The model 4 is outperformed by the other models as well as the model 2, which is very inconstant. We will evaluate models 1 and 3 more closely using the classification report which is displayed below.

Model 1

	precision	recall	f1-score	support
0	0.97	0.87	0.92	45
1	0.89	1.00	0.94	33
2	0.97	0.97	0.97	39
3	0.98	1.00	0.99	55
accuracy			0.96	172
macro avg	0.96	0.96	0.96	172
weighted avg	0.96	0.96	0.96	172

Model 3

	precision	recall	f1-score	support
0	0.97	0.87	0.92	45
1	0.89	1.00	0.94	33
2	0.95	0.97	0.96	39
3	1.00	1.00	1.00	55
accuracy			0.96	172
macro avg	0.95	0.96	0.96	172
weighted avg	0.96	0.96	0.96	172

Figure 5 and 6 – Classification Report

Note that the labels stand for the 4 classes that we are trying to predict:

0: cloudy; 1: rainy; 2: shine; 3: sunrise

Both models have very similar performances with an overall accuracy of 0.96. However, we acknowledge that our data is not perfectly balanced, and, for that reason, we should consider also other measures more suitable for unbalanced data like the f1-score. Both models have more difficulties predicting cloudy and rainy images as their f1-scores on these labels are the lowest. This is expected since these images can be more similar to each other, than, for example, a *cloudy* and a *sunrise* image.

The main difference between the models is that model 3 is perfect at identifying *sunrise* images since both the precision and recall are 1 on this label, whilst model 1 is more levelled between the classes. However, both models are very similar and produce very good results.

Finally, model 1 got 0.967 accuracy on training data and 0.912 on validation data while model 3 got 0.951 on training data and 0.931 on validation data. With this, we can conclude that model 1 is a little more probable of having overfitting than model 3 (since there is a larger difference of performance between training and validation data). For that reason, the model 3 and its architecture is the one which we will focus on and execute a parameter tuning in order maximize its performance.

7 BEST MODEL

Once we decided on the best architecture for the model, there were some parameters that could be optimized in order to maximize the performance. Firstly, we decided to evaluate the performance of the CNN with different Batch Size and a different optimizer. The results are displayed in the table below.

Number of Epochs *	Batch Size	Optimizer	Train Accuracy	Validation Accuracy	Test Accuracy
100	32	adam	0.9619	0.9688	0.93
		rmsprop	0.9735	0.95	0.94
	16	adam	**0.9864**	**0.9812**	**0.96**
		rmsprop	0.978	0.944	0.95

* Note that, because we are using EarlyStopping as a callback, the network may stop before reaching the 100 epochs

Table 2 - Model 3 Tuning: Batch Size and Optimizer

As we can easily conclude from the results, the best batch size is 16, and the best optimizer is Adam: these parameters choices outperformed the others on both the training and validation data as well as on the test data, obtaining values very close to one which is indicative of a very good predicting power.

However, all the 4 models present a little of overfitting and, for that reason, we decided to try different parameters design specifically to control overfit: the Dropout rate and the Weight Regularization. The results are displayed below.

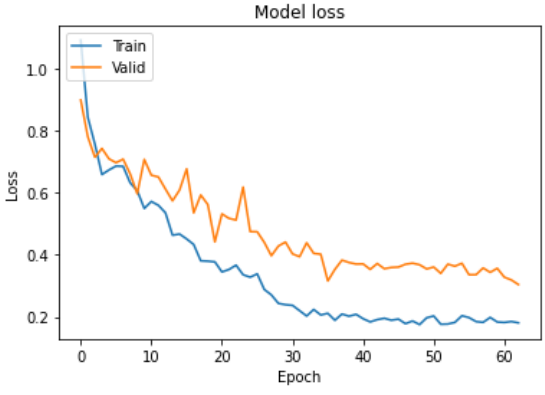
Number of Epochs *	Regularizer	Dropout	Train Accuracy	Validation Accuracy	Test Accuracy
100	0.001	0.2	0.9714	0.9438	0.94
		0.4	**0.9864**	**0.9812**	**0.96**
		0.5	0.9597	0.95	0.92
	0.1	0.2	0.9875	0.9625	0.94
		0.4	0.9547	0.95	0.91
		0.5	0.949	0.95	0.94

* Note that, because we are using EarlyStopping as a callback, the network may stop before reaching the 100 epochs

Table 3 - Model 3 Tuning: Regularizer and Dropout

Examining the results, we conclude that the best parameters to control the overfit is a Dropout Rate of 0.4 and a Weight Regularizer of 0.001 as this parameters choice outperforms all the other options both in training and testing data. Hence, the final model is concluded and the training and validation accuracy and loss results are displayed below.

Final Model
Conv2D
Conv2D
MaxPool
Conv2D
Conv2D
MaxPool
GlobaMax
Dense (weight regularizer)
Dropout (0.4)
Dense (softmax)



8 LIMITATIONS

As for the limitations on the project we consider that the lack of data in the dataset was the main difficulty that we had to overcome. A trade-off between allocating more data for the training set in order to produce a better model and have a substantial amount of data for the testing set was considered and analysed. The final split of the data was 70% train, 15% validation and 15% test. Another limitation was the running time of some architectures which made it very hard and time consuming for us to develop and improve them.

9 CONCLUSION

In this project we were asked to develop a Deep Learning project where the topics covered in the course would be applied. Initially, we explored and pre-processed the data with the objective of creating folders for each class and split the data in training, validation, and test sets.

Afterwards, we decided to create four different models that were based in various CNNs architectures that already existed. None of the models built exceeded more than ten layers since although these networks would possibly produce better results, they were not efficient due to the time they took running.

We conducted several experiences to design the best CNN regarding both performance as well as computational effort and running time. We consider that the final solution is very suitable for the task we proposed ourselves to solve and, therefore, a success.

10 REFERENCES

Callbacks in neural networks

Available at: https://towardsdatascience.com/callbacks-in-neural-networks-b0b006df7626

Retrieved on 10/04/2022

Different Types of CNN Architectures

Available at: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

Retrieved on 12/04/2022

Regularization in Deep Learning – L1, L2, and Dropout

Available at: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036

Retrieved on 12/04/2022