Jupyter Notebooks for Simplifying Transfer Learning

Manuel García-Domínguez(✉), César Domínguez, Jónathan Heras, Eloy Mata, and Vico Pascual

Department of Mathematics and Computer Science,
University of La Rioja, Logroño, Spain
{manuel.garcia,cesar.dominguez,jonathan.heras,eloy.mata,
vico.pascual}@unirioja.es

Abstract. Nowadays, the use of transfer learning, a deep learning technique, is growing to solve imaging problems in several contexts such as biomedicine where the amount of images is limited. However, applying transfer learning might be challenging for users without experience due to the complexity of the deep learning frameworks. To facilitate the task of creating and using transfer learning models, we developed FrImCla, a framework for creating image classification models. In this paper, we have developed a set of Jupyter notebooks that use FrImCla to facilitate the task of creating and using image classification models for users without knowledge in deep learning frameworks and without any special purpose hardware.

Keywords: Deep learning · Transfer learning · Image classification

1 Introduction

Transfer learning is a deep learning method that has been successfully applied in computer vision to solve several problems [3,4,6,10,11]. This technique consists in re-using a neural network model that has been trained in a source task in a new target task, and it can be applied in different ways [12]. Namely, given a pre-trained model, we can distinguish three approaches for applying transfer learning: (1) use the architecture and the weights of the pre-trained model as the starting point to train a new model; (2) freeze the weights of some layers of the model and train the rest of the layers; and, (3) use the output of the pre-trained network as "off-the-shelf" features to train a completely new classifier for the target task. The first two approaches are data-demanding and require a lot of computational power; on the contrary, the last approach can be employed with small datasets and does not require special purpose hardware (like GPUs). In this work, we focus on the last approach.

Partially supported by Ministerio de Industria, Economía y Competitividad, project MTM2017-88804-P; Agencia de Desarrollo Económico de La Rioja, project 2017-I-IDD-00018; and FPI Grant of the Comunidad Autónoma de La Rioja.

Using pre-trained networks as feature extractors presents several challenges for non-expert users. First of all, there are several source models (for instance, DenseNet [8], GoogleNet [13], or Resnet 50 [7]) that can be employed for feature extraction, and their use differ from one to another. Moreover, those feature extractors must be combined with different machine learning algorithms to search for the best combination. Finally, once a model is trained following this approach, the developer must provide an interface for the model that can be employed by the final users.

To solve these problems, we developed FrImCla. This framework takes advantage of transfer learning features and helps users to create and train classification models. However, the interaction with the FrImCla library is provided by means of the command line interface. To simplify the interaction with FrImCla, we have created a set of Jupyter notebooks that allows users without knowledge in deep learning to employ this technology, see Sect. 2. With this approach users do not have to write a single line of code to create classifications models, as shown in Sect. 3. The paper ends with a conclusions and future work section.

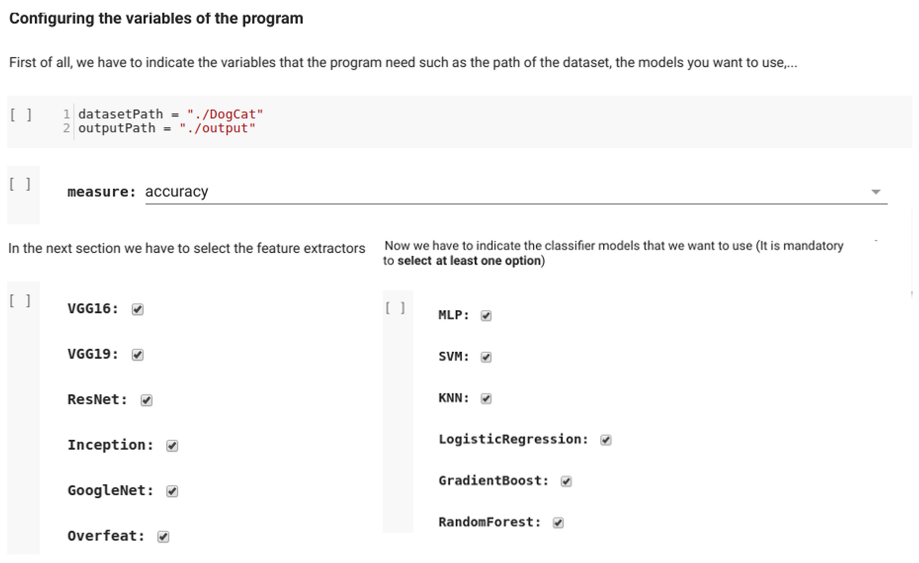
2 Jupyter Notebooks for FrImCla

FrImCla [5] is an open source Python library for image classification. This framework provides algorithms to help the user in each step of the construction of machine learning models for image classification using both traditional and deep learning features. The workflow of FrImCla can be summarized in five different steps. First of all, the user selects the input that it is composed by the path of a dataset of images and some configuration parameters that include the feature extractor and classification algorithms. Then, FrImCla goes through the dataset collecting the features of the images using the feature extractor methods given by the user. For each feature extractor, FrImCla generates a dataset of features. From those dataset of features, FrImCla trains the classification algorithms. Another parameter is the list of machine learning algorithms that FrImCla uses to train the classifiers. Finally, a statistical analysis is employed to select the best combination of features and machine learning algorithm. The best combination is used to create a model for further use.

Even if FrImCla users only need to provide the path of the dataset of images and some configuration options, some users might find difficult to use this framework. This complexity is due to the fact that the configuration parameters must be provided by means of a JSON file and the framework is invoked from the command line. For this kind of users, we have developed Jupyter notebooks.

A Jupyter Notebook [9] is an interactive environment that allows users to develop and interact with Python code dynamically. It allows developers to integrate fragments of text, graphics or images along with the code to document it. We have developed a set of Jupyter notebooks that explain to non-expert users how to create and use their own classification models by applying transfer learning thanks to FrImCla. Even more, FrImCla can be used without any installation on the user computer. We have made accessible the Jupyter notebooks through Google colaboratory [2], a tool that allows users to construct models using FrImCla in the cloud in any computer with access to the Internet.

In the notebooks, there is a detailed explanation of the process of execution of FrImCla. In particular, the user only has to fix four parameters (see Fig. 1): the path of the dataset of images, the feature extractors (the user can select among eight deep feature extractors based on transfer learning and seven traditional computer vision methods), the supervised algorithms (among others, Neural Networks and Random Forest [1]) and the measure (currently, there are available metrics such as accuracy or AUROC). Then, the notebook connects with FrImCla and explores the different combinations of feature extractors and classification algorithms to obtain the best model for the given dataset. This process is carried out without any user interaction as shown in Fig. 2.


Configuring the variables of the program

First of all, we have to indicate the variables that the program need such as the path of the dataset, the models you want to use,...

```
datasetPath = "./DogCat"
outputPath = "./output"
```

measure: accuracy

In the next section we have to select the feature extractors

VGG16: ✓
VGG19: ✓
ResNet: ✓
Inception: ✓
GoogleNet: ✓
Overfeat: ✓

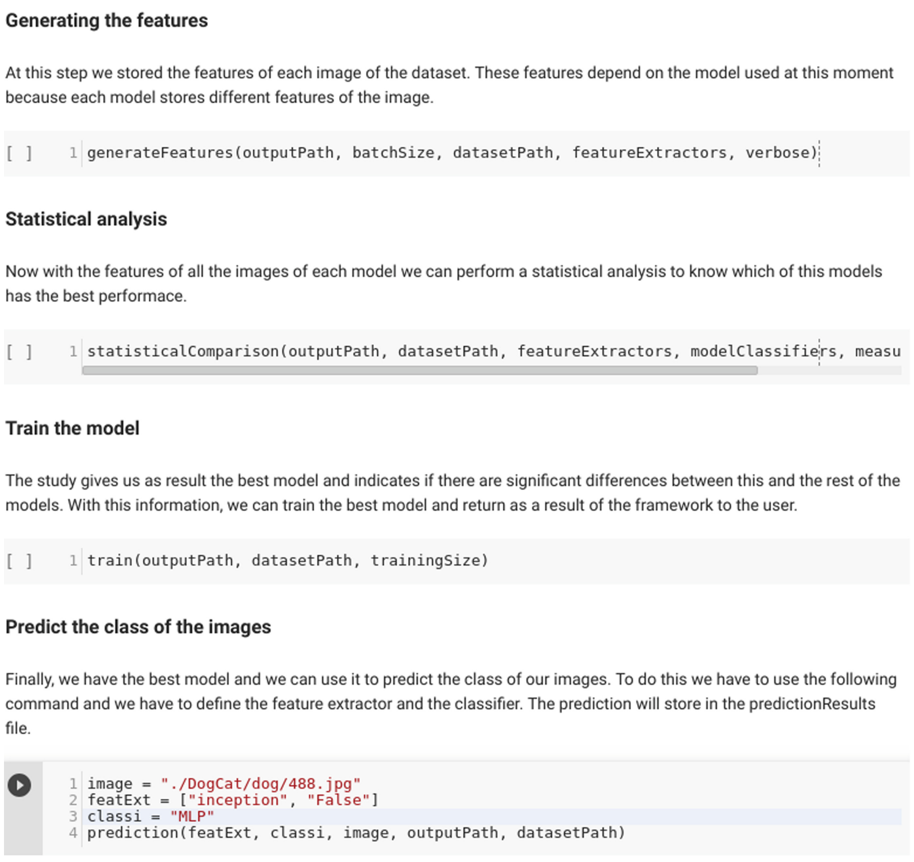
Now we have to indicate the classifier models that we want to use (It is mandatory to **select at least one option**)

MLP: ✓
SVM: ✓
KNN: ✓
LogisticRegression: ✓
GradientBoost: ✓
RandomForest: ✓

Fig. 1. Example of the Jupyter notebook showing the configuration options.

3 Case Study

In order to show the feasibility of constructing models using FrImCla, we have used a dataset of dogs and cats that contains 600 images, 300 images for dogs and 300 images for cats for training the models. For testing we have used a dataset of 100 images to know the performance of the framework. Using FrImCla, we have performed a thorough study combining all the available feature extractors and machine learning models. Namely, we employed as featured extractors: VGG16,



Generating the features

At this step we stored the features of each image of the dataset. These features depend on the model used at this moment because each model stores different features of the image.

```
[ ]  generateFeatures(outputPath, batchSize, datasetPath, featureExtractors, verbose)
```

Statistical analysis

Now with the features of all the images of each model we can perform a statistical analysis to know which of this models has the best performace.

```
[ ]  statisticalComparison(outputPath, datasetPath, featureExtractors, modelClassifiers, measu
```

Train the model

The study gives us as result the best model and indicates if there are significant differences between this and the rest of the models. With this information, we can train the best model and return as a result of the framework to the user.

```
[ ]  train(outputPath, datasetPath, trainingSize)
```

Predict the class of the images

Finally, we have the best model and we can use it to predict the class of our images. To do this we have to use the following command and we have to define the feature extractor and the classifier. The prediction will store in the predictionResults file.

```
image = "./DogCat/dog/488.jpg"
featExt = ["inception", "False"]
classi = "MLP"
prediction(featExt, classi, image, outputPath, datasetPath)
```

Fig. 2. Example of the Jupyter notebook showing the rest of the steps of the process. The user only needs to press the run button.

VGG19, Resnet, Inception, GoogleNet, Overfeat, Xception, DenseNet, LAB444, LAB888, HSV444 and HSV888 (LABXYZ and HSVXYZ are respectively LAB and HSV histograms using X,Y,Z bins per chanel); and as machine learning algorithms: SVM, KNN, Multilayer perceptron (MLP), Gradient Boost (GB), Logistic Regression (LR) and Random Forest (RF). The results are presented in Table 1 and Fig. 3. In order to compare all the possible combinations, we analyse the statistical study performed by FrImCla. Using transfer learning features we can easily obtain an accuracy higher than 85% but only with a few of them we can achieve an accuracy over 95%, see Fig. 3. On the contrary, models trained using traditional features are far from the 80% accuracy. This shows the importance of trying different models.

As a result of the analysis, we see that the best combination of feature extractor and machine learning algorithm is *DenseNet* with *MLP* (although similar results are obtained with other methods) with an accuracy of 99.0%. The result with the test set is a 96.0% accuracy.

Table 1. Mean (and standard deviation) of the different studied models for the dogs and cats dataset. The best result for each model in *italics*, the best result in **bold** face. $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$; $>$: there are significant differences; $\simeq$: there are not significant differences.

Network	KNN	LR	MLP	RF	SVM	GB	Test (ANOVA or Friedman)	After post-hoc procedure
VGG16	89.5 (0.04)	95.8 (2.0)	95.3 (3.0)	*96.7 (2.2)*	96.0 (2.1)	95.5 (2.5)	5.84^{***}	RF $\simeq$SVM, LR, MLP, GB; RF$>$ KNN
VGG19	92.2 (3.8)	95.2 (3.3)	95.5 (3.2)	*96.0 (2.5)*	96.0 (2.9)	95.2 (2.2)	2.37	RF$\simeq$ SVM, LR, GB, KNN, MLP
Xception	90.8 (5.5)	98.8 (1.3)	*98.8 (1.1)*	98.5 (1.2)	97.0 (2.6)	98.1 (2.1)	5.81^{***}	MLP $\simeq$ LR, GB, RF, SVM; MLP $>$KNN
Resnet 50	87.5 (2.4)	95.5 (1.8)	95.8 (1.7)	95.0 (3.1)	94.7 (2.4)	*95.7 (3.0)*	7.89^{***}	GB $\simeq$ MLP, LR, RF, SVM; RF $>$ KNN
Overfeat	70.2 (7.5)	92.8 (3.0)	93.0 (3.9)	91.7 (4.2)	*93.3 (1.8)*	91.0 (2.4)	8.36^{***}	SVM $\simeq$ MLP, LR, RF, GB; SVM $>$ KNN
Inception v3	80.1 (8.1)	*91.9 (3.8)*	67.7 (11.9)	87.8 (4.1)	91.9 (5.1)	85.8 (4.0)	22.59^{***}	LR $\simeq$ SVM, RF; LR $>$ GB, KNN, MLP
DenseNet	92.3 (4.3)	98.7 (1.0)	***99.0 (1.3)***	96.7 (2.2)	98.5 (2.2)	96.7 (2.9)	5.64^{***}	MLP $\simeq$GB, RF, SVM, LR; MLP$>$ KNN
GoogleNet	96.2 (1.4)	96.4 (1.4)	*96.8 (1.7)*	95.0 (3.7)	96.5 (1.6)	94.8 (3.1)	1.11	MLP$\simeq$ RF, SVM, LR, GB, KNN
LAB888	54.5(4.5)	*55.5 (9.5)*	55.5 (6.8)	56.5 (6.3)	54.2 (6.6)	56.7 (7.6)	0.19	GB $\simeq$ LR, RF, SVM, KNN, MLP
LAB444	*54.6 (8.9)*	54.2 (6.2)	52.5 (6.2)	53.2 (8.0)	53.8 (6.4)	54.3 (5.3)	0.12	KNN $\simeq$ SVM, GB, LR, RF, MLP
HSV888	55.2 (3.7)	56.7 (8.7)	50.5 (7.5)	*59.3 (6.5)*	57.3 (8.7)	61.8 (5.3)	2.8^{*}	GB $\simeq$ RF, LR, SVM, KNN; GB $>$ MLP
HSV444	54.0 (7.8)	56.7 (5.3)	55.3 (4.8)	57.7 (6.9)	56.8 (4.8)	*58.7 (7.5)*	0.62	GB $\simeq$ RF, KNN, SVM, LR, MLP

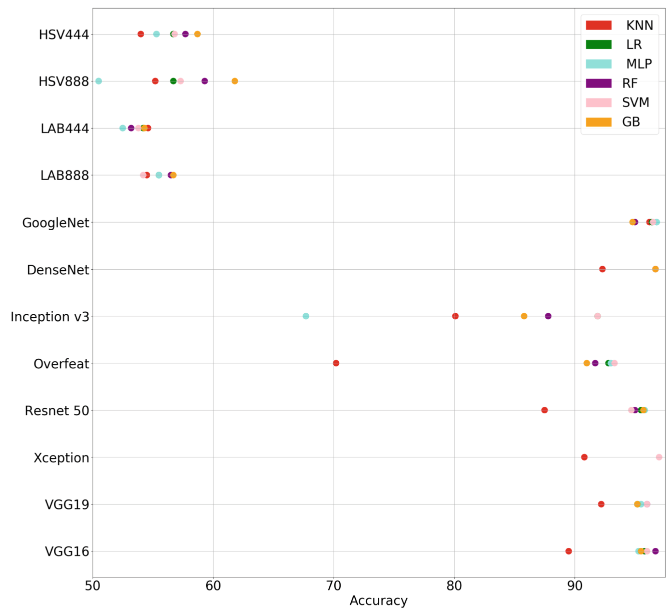


Fig. 3. Scatter plot diagram of the models constructed for the cats and dogs dataset.

4 Conclusions and Further Work

To conclude, Jupyter notebooks can be a really good tool to bring difficult technologies like transfer learning closer to non-expert users. This tool allows developers to combine code with it explanation. In this way, non-expert users can understand the process of the code that they are going to execute without write a single line of code. We have combined Jupyter notebooks with FrImCla, a tool that facilitates the task of creating and using transfer learning models. Using FrImCla, we can create accurate models for classification without any knowledge in machine learning and without any special purpose hardware.

Availability

– Project name: FrImCla notebooks.
– Project home page: https://github.com/ManuGar/FrImCla.
– Operating system(s): Platform independent.
– Programming language: Python.
– Other requirements: None.
– License: MIT.

References

1. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
2. Carneiro, T., Da Nóbrega, R.V.M., Nepomuceno, T., Bian, G., De Albuquerque, V.H.C., Filho, P.P.R.: Performance analysis of google colaboratory as a tool for accelerating deep learning applications. IEEE Access **6**, 61677–61685 (2018). https://doi.org/10.1109/ACCESS.2018.2874767
3. Christodoulidis, S., et al.: Multisource transfer learning with convolutional neural networks for lung pattern analysis. IEEE J. Biomed. Health Inf. **21**(1), 76–84 (2017)
4. Domínguez, C., Heras, J., Mata, E., Pascual, V.: DecoFungi: a web application for automatic characterisation of dye decolorisation in fungal strains. BMC Bioinform. **19**(1), 66 (2018)
5. García-Domínguez, M., et al.: FrImCla: A Framework for Image Classification using Traditional and Transfer Learning Techniques. Preprint (2019). https://github.com/ManuGar/FrImCla
6. Ghafoorian, M., et al.: Transfer learning for domain adaptation in MRI: application in brain lesion segmentation. In: Descoteaux, M., Maier-Hein, L., Franz, A., Jannin, P., Collins, D.L., Duchesne, S. (eds.) MICCAI 2017. LNCS, vol. 10435, pp. 516–524. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66179-7_59
7. He, K., et al.: Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), pp. 770–778. IEEE Computer Society, IEEE (2016)
8. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), pp. 2261–2269 (2017)
9. Kluyver, T., et al.: Jupyter notebooks – a publishing format for reproducible computational workflows. In: Proceedings of the 20th International Conference on Electronic Publishing, pp. 87–90. IOS Press (2016)
10. Menegola, A., Fornaciali, M., Pires, R., Bittencourt, F.V., Avila, S., Valle, E.: Knowledge transfer for melanoma screening with deep learning. In: 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017), pp. 297–300, April 2017. https://doi.org/10.1109/ISBI.2017.7950523
11. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
12. Razavian, A.S., et al.: CNN features off-the-shelf: an astounding baseline for recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2014), Columbus, Ohio, USA, pp. 512–519. IEEE Computer Society, IEEE (2014)
13. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), pp. 1–9. IEEE Computer Society, IEEE (2015)