Using Convolutional Neural Networks for Galaxy Class Prediction.

Alex Beltran and Unai Carbajo

Abstract

Galaxy classification is a task that has been bothering researchers since the early 1926s. Almost all current systems of galaxy classification are continuations and expansions of the initial scheme proposed by the American astronomer Edwin Hubble in 1926. In Hubble's scheme, which is based on the optical appearance of galaxy images , galaxies are divided into three general classes: ellipticals, spirals, and irregulars. Hubble subdivided these three classes into finer groups. Due to the increasing interest on galactic matters from researches, the so called "last frontier" has been the focus of machine learning for a while. From automating vehicles to work outside our atmosphere to complex planet classification.

Tasks like planet/galaxy classification end up being really relevant due to how many there are, as the known space and universe is infinite and constantly expanding. As more and more galaxies are eventually found, the need to have experts look at every single one of them becomes too much of a burden, and that's where automated galaxy classification comes in.

This work will be our take on one of the kaggle challenges that was celebrated 7 years ago: The Galaxy Zoo prediction problem. Our intent with this work will be to try to give a good intuition of what galaxy zoo and galaxy classification are all about, while also presenting a model of our own to solve the challenge.

Contents

1 Description of the main Task.

In this project, we have been given the task of designing a convolutional neural network that outputs the probability that a given galaxy image belongs to one of the possible categories. This is a supervised classification problem. The data-set we are going to use was used for one of the Kaggle challenges.

Overall the objectives are:

- Preprocess the images.
- Design the network architecture and train it.
- Validate the network.

Not all images in the training set have to be necessarily used for the project. The team can decide whether to use all or a subset of them. Around 10000 should be sufficient for a good classification.

2 The Galaxy classification problem.

2.1 Galaxy-zoo.

Galaxy-zoo is an online crowd-sourcing project where random selected users have been asked to, given an image of a galaxy in colour, "describe" its morphology.

In order to do that, since it cannot be expected for random individuals to be experts on galactic matters, the users were given a set of questions that would serve as "guidelines" for the aforementioned descriptions. This questions could range from describing more general features like "How rounded is the galaxy?", to in depth more specific questions: "Does it have a central bulge?".

Since the system for the descriptions was based on questions, this could be easily seen as a decision tree.

2.2 The galaxy-zoo decision tree.

The galaxy-zoo decision tree is a simple in depth decision tree that allowed the randomly selected individuals to do an accurate prediction of the probabilities of X galaxy being of class C. The decision tree consists of 11 questions that divide the tree, which ends up with 37 possible classes for every single galaxy.

The decision tree (and thereby its questions) has been designed to encompass all points in the traditional Hubble-tuning-fork as well as a range of more irregular mythologies. The Hubble-tuning-fork (The Hubble tuning fork - classification of galaxies) is a basic classification scheme of galaxies designed by the American astronomer Edwin Hubble in 1926. Even thought, as can be seen below, the idea is pretty simple, it still holds as a solid reference for galaxy classification.

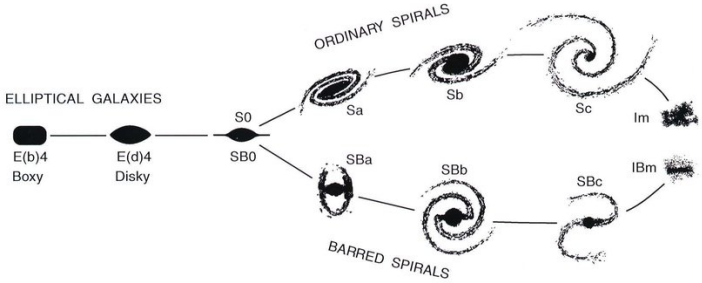

Figure 1: The Hubble fork.

2.2.1 Decision tree: the questions.

Now we can see how the galaxy-zoo decision tree is a more ”in depth” version of the Hubble-tuning-fork. For starters, as mentioned above, the galaxy-zoo decision tree consists of the following 11 questions:

- Q1. Is the object a smooth galaxy, a galaxy with features/disk or a star?
- Q2. Is it edge-on?
- Q3. Is there a bar?
- Q4. Is there a spiral pattern?
- Q5. How prominent is the central bulge?
- Q6. Is there anything ”odd” about the galaxy?
- Q7. How round is the smooth galaxy?
- Q8. What is the odd feature?
- Q9. What shape is the bulge in the edge-on galaxy?
- Q10. How tightly wound are the spiral arms?
- Q11. How many spiral arms are there?

Every question does not necessarily have the same number of answers as the rest. For example, Q2 (Is it edge-on?) has two possible responses, while some others like Q11 (How many spiral arms are there?) has up to 6 responses. This is due to the binary nature of some questions (yes/no) while some others the answer might be between several options (more than 2) or it needs to be chosen between a range of numbers (like Q11).

The original version given to the participants of the galaxy-zoo project can be seen below:

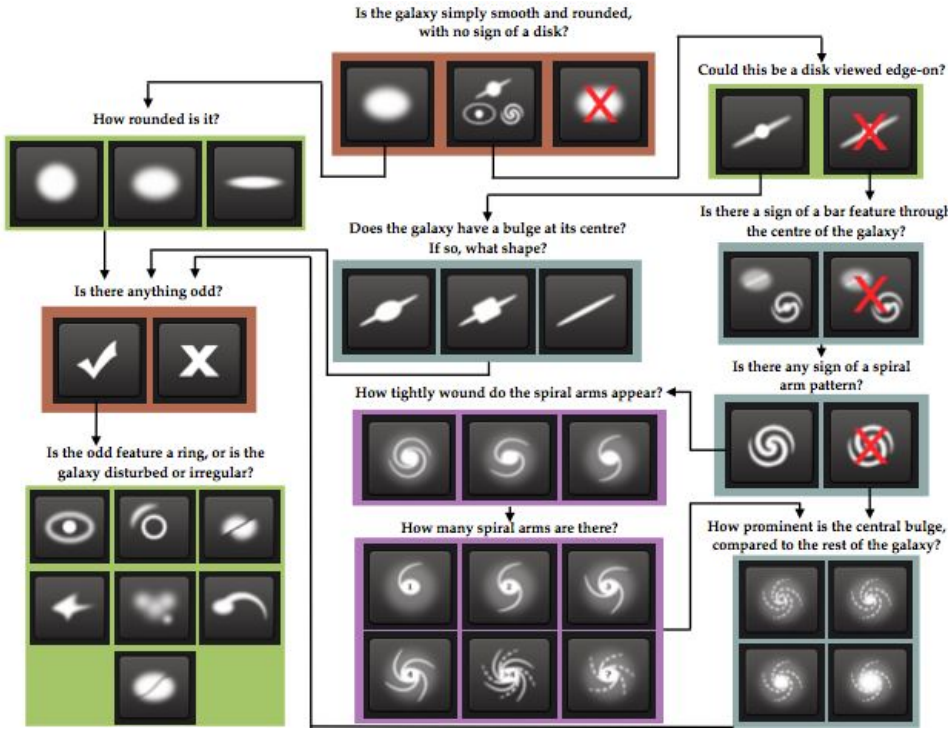

Figure 2: Simplified version of the Galaxy-Zoo decision tree given to the participants.

2.2.2 Decision tree: the answers.

Lets see how the probabilities of the tree work now. For starters, at each node or question, the total initial probability of a classification will sum to 1.0. In order to compute the values that can be seen in the solution files, the following steps are taken:

- For the first set of responses (smooth, features/disk, star/artifact), the values in each class are the likelihood of the galaxy falling in a given class C. These values will always sum to 1.0.
- For the rest of following questions, first the probabilities are computed (will still sum to 1.0) and then they will be multiplied by a value P, where P is the value that led to that new set of answers. Due to this multiplication, this new probabilities will NOT sum to 1.0.

The reason for this following step multiplication is that in order for a solution to be good, it needs to emphasize that it must get the high-level, large-scale morphology categories correct. A side effect of this is that, alongside what was mentioned before, a good solution will have a good accuracy on the further down the decision tree.

Since this weighting can be seen as a little confusing when can use the following example given by the own Galaxy-Zoo team. Given a galaxy G that had the following user descriptions:

- 80% of the users identified it as smooth.
- 5% of the users identified it as having features / disks.
- 5% of the users identified it as a star / artifact.

The class of G would be computed as follows:

$$ClassSmooth = 0.80$$
$$ClassFeatureDisk = 0.15$$
$$ClassStarArtifact = 0.05$$

Lets go more in depth. Given the 80% of users that classified the galaxy as "smooth", they also recorded responses for the galaxy's relative roundness.

- 50% of the users identified it as completely round.
- 25% of the users identified it as in-between round.
- The rest of the users identified it as cigar-shaped.

The new class probabilities of G would be computed as:

$$ClassCRound = 0.80 * 0.50 = 0.40$$
$$ClassInBewteen = 0.80 * 0.25 = 0.20$$
$$ClassCigar = 0.80 * 0.25 = 0.20$$

As we can see, the probabilities do no longer sum to 1.0. Mention that because of the structure of the decision tree, each individual participant answered only a subset of the questions for each classification (since it followed the tree structure). When many participants have classified the same image, their answers are aggregated into a set of weighted vote fractions for the entire decision tree. These vote fractions are used to estimate confidence levels for each answer, and are indicative of the difficulty users experienced in classifying the image.

3 Description of our approach

3.1 Research

At first we had problems trying to understand the main problem of the task, whether the problem was a classification problem or not. In order to have a quick idea of how could we approach it, we decided to search for some solutions made for the original competition. We checked many of them, but the one that drew our attention was the one made by the competition's winner. This process is widely explained in the section 3.4.

But apart from being the winner's approach, it was really interesting the explanation made by S. Dieleman in his personal blog[1]. Adding to the step explanation, which is impeccably wrote, there's a comment section, where the author replied to all the questions about the "why"s and the "what"s that the other competitors had about the decisions made by him at the time of approaching the problem in certain way or another. We encourage everyone to check all the explanations.

3.2 Data loading and preprocessing.

We decided to load a total of 12000 images, all of them of the same size (424x424). Leaving a total of 10800 (90%) images for training, and the remaining 1200 for test (10%), since the validation set has also been used as test.

Thinking about the results' use, we chose to use the pandas framework to load the .csv data into easy to control and manage data-sets. A data-set is a column ordered table containing all of our data.

Initially, we don't crop or reshape any image early, like other works would do. Instead of that, we decided a easier and more dynamic way of approaching this.

Since CNNs are already computationally expensive, and since we are working with a large size of images that already have a high disk-space requirement to save them all, we decided to use a trick, it consisting in using KERAS cropping2D layers at the start of our CNN, in such a way that the initial cropping we would apply to all the images, applies at the start to every input.

Turns out this is really efficient since KERAS functions are made to be fast and computationally cheap, while also we gain the advantage of only preprocessing the images we are gonna use, so its more dynamic. Again, when more powerful tools are available, separate preprocessing might be a better way to go, but we are restrained by our computational limits.

3.3 Data augmentation.

Data augmentation consists on several approaches that modify the training data in ways that change the input representation while keeping the label the same (rotating an image, for example). This is usually both a cheap way to add new data (increase our train data-set size) and a way to add prior knowledge to the CNN.

We also use this as a way to add some regularization to our model, since it reduces the generalization error (the network will learn to classify X class even if an image of X class is actually rotated or over/under sized).

To achieve this we create a Image-data-generator, a function that is an easy, clean and cheap way to apply this data augmentation methods to our train and test sets. The resulting data-generator is then applied to both set. This data-generator will randomly re-scale, vertically/horizontally flip the images, change the width/height and the rotation (90º rotations applied X times).

In our case the was no need of augmenting the data in the practical sense, the database was longer enough; our principal objective of this practice was to get different representations of the same galaxy, using techniques as flipping, shifting, etc, in order to archive better results, as the model could learn the same galaxy from different "perspectives".

[1] https://benanne.github.io/2014/04/05/galaxy-zoo.html

3.4 Our models.

First of all, we decided to make one CNN model from scratch, using *Keras* library. The first designs were based on the solution from *N. Khalifa et al.*[1], even though their main idea was to predict the galaxy type based on some rules, we decided to draw from it. The model they proposed was formed by one convolution layer[2] followed by the classification part (FC 24 neurons + ReLU + FC 3 neurons). Our first model was formed by a final FC of 37 neurons instead of 3, due to our objective of predicting 37 probabilities.

After many changes, we decided to follow the path of the model defined by *Dieleman et al.*[2], were they defined several convolution layers instead of one. Following this idea, we decided to make an architecture with 3 convolution layers, where every block was formed by 2 convolutional layers instead of 1, same idea as K. Simonyan's and A. Zisserman's *VGG-16* [3] net. The two convolutional layers contained in each block had the same filter number as output, being totally equal. With this kind of changes we ended up increasing the numbers due to the total filter number, but we decreased the number of parameters created with the flatten, because the size of the image by the time of applying the flatten was quite small, reducing highly the parameter number. This could imply having less capacity, but, as I mention, this is compensated at the time of applying double convolutions.

Apart from this last model, we decided to apply the *transfer learning* technique, in order to use a pre-trained model and train it briefly to fit it to our problem. We think about several models we could try, as the *VGG-16* [3] net, what was our first attempt, but ended up being too wide to be runned by the environment. This was a important problem during all the development, as we didn't have a powerful GPUs; so we decided to make all the executions in the *Google's Colaboratory* environment. This provided as the compute capacity we needed but it limited us the execution time to 12h each. Finally, after many attempts we had been able to use the well-known *ResNet-50*[4] (*K. He et al.*).

As mentioned, all the models were trained in the *Google's Colaboratory* environment, along with the final results; thus, all the libraries we used were updated to the latest versions.

3.4.1 Our Convolutional Neural Network.

The model we defined consists in few initial layers that worked as dynamic preprocessing-like layers, which cropped the initial images and applied a batch normalization; the reason of this crop stage is defined in the section 3.2.

Following this initial stage, there were defined 3 convolution groups, which consisted of 2 side by side convolutions (along with their respective ReLUs) and one down-sampling layer.

And finally there's defined a output group which included a flatten layer, a dropout layer to inhease regularization of the model (widely explained in section 3.3), and 3 side by side FC layers with 256, 128 and 37 neurons respectively.

The hyperparameters (along with loss and activation functions and optimizers) of our network were the next ones:

- **Activation function:**
  - FC 1: *ReLU*
  - FC 2: *ReLU*
  - FC 3: *Sigmoid*
- **Loss function:** *Binary cross-entropy*. It computes the cross-entropy loss between true labels and predicted labels. This kind of cross-entropy is optimized for problems whose class label is intented to be between 0 and 1 (probabilities). Is defined in the next way:

$$H_p(q) = -\frac{1}{N}\sum y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{1}$$

- **Optimizer:** *Adam* (learning rate = 0.001)

[2]Convolution layer refers to the block formed by one (or more) convolutional layer + non-linear activation (ReLU) + down-sampling (max-pooling commonly).

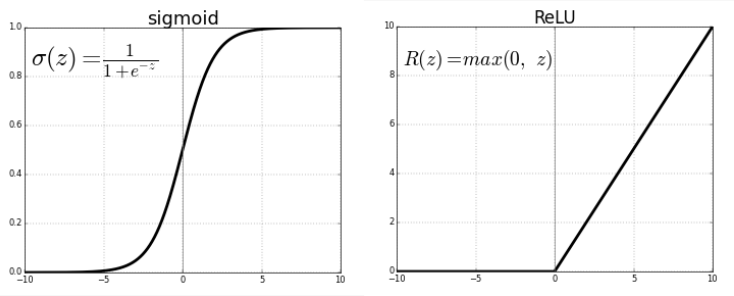

Figure 3: Sigmoid vs ReLU. Source: Medium.com

Everything sums up to the next table: We also can visualize the whole net in the following way:

Operation Group	Operation	Layer name	Number of filters	Filter size	Stride size	Padding size	Size of output image
Input group (G_0)	Input image	Input layer	-	-	-	-	212x212x3
	Cropping	Cropping layer	-	-	-	-	74x74x3
	Batch Normalization	Batch Normalization layer	-	-	-	-	74x74x3
Conv. group 1 (G_1)	Convolution (2 times)	Convolution layer	128	3x3x3	1x1	1x1	(1) 72x72x128 (2) 70x70x128
		ReLU layer	-	-	-	-	(1) 72x72x128 (2) 70x70x128
	Pooling	Max Pooling layer	1	2x2	2x2	0	35x35x128
Conv. group 2 (G_2)	Convolution (2 times)	Convolution layer	64	3x3x3	1x1	1x1	(1) 33x33x64 (2) 31x31x64
		ReLU layer	-	-	-	-	(1) 33x33x64 (2) 31x31x64
	Pooling	Max Pooling layer	1	2x2	2x2	0	15x15x64
Conv. group 3 (G_3)	Convolution (2 times)	Convolution layer	32	3x3x3	1x1	1x1	(1) 13x13x32 (2) 11x11x32
		ReLU layer	-	-	-	-	(1) 13x13x32 (2) 11x11x32
	Pooling	Max Pooling layer	1	2x2	2x2	0	15x15x64
Output group (G_4)	Flatten	Flatten layer	-	-	-	-	800
	Dropout (0.5)	Dropout layer	-	-	-	-	800
	3 FC	Dense layer	-	-	-	-	256
		Dense layer	-	-	-	-	128
		Dense layer	-	-	-	-	37

Table 1: Layer structure of the CNN.

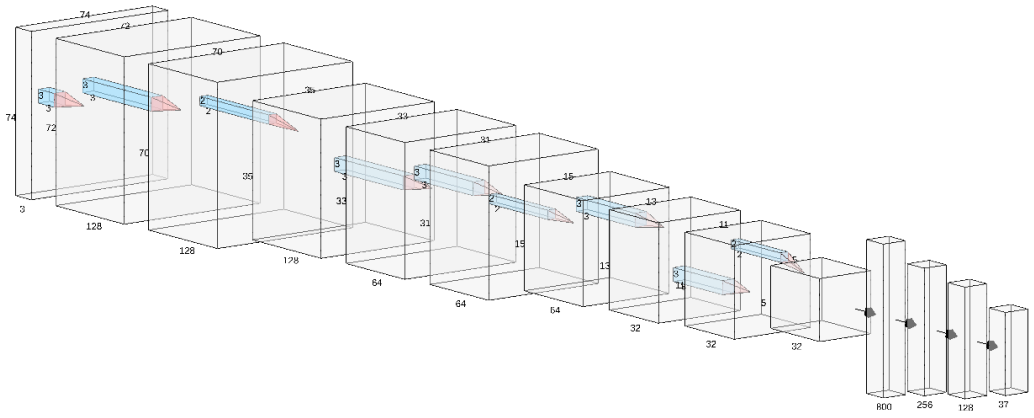

Figure 4: Layer scheme of the CNN.

3.4.2 Transfer Learning: ResNet-50.

Along with the previously defined network, we decided to apply the transfer learning technique, which aims at improving the performance of target learners on target domains by transferring the knowledge contained in different but related source domains (F. Zhuang et al) [5].

In our case the chosen pretrained networks has been the *ResNet-50* [4], a variant of the original *ResNet* model which is formed by 48 convolution layers along with 1 maxpool and 1 Average pool layer. In our case, as we choose to use keras' implementation of the *ResNet*, it was previously trained with *Image-Net's* database. This certain type of nets use a technique known as "residual learning". Which, roughly speaking, is a simple way of describing connecting the output of certain layer to the output of the new ones. For example, in the case of having 2 layers, the output of the first layer would not only be connected as input to the next layer, it will also be connected to the output of the layer 2.

These kind of units are defined in the next way:

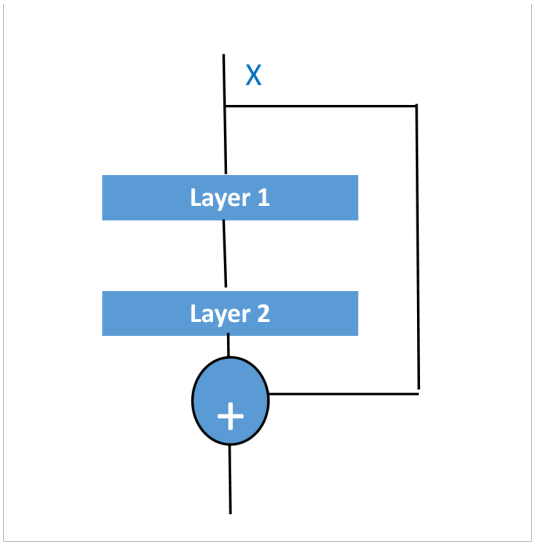

Figure 5: Structure of a residual learning unit.

This ResNet-50 model has served as backbone for our 2 second model, but as originally defined it can't be used with our data and prediction objective, thus, in order to fit the model to our data we found ourselves needing to making some changes:

- Input layer (212, 212, 3): Cropping layer (size: 69, 69)
- ResNet model
- Output layers: Flatten (18432 neurons) + FC (37 neurons)

Another decision we took was to unable some layers to be trained, in order to have a lighter model and be able to be trained in a proper way. The first 143 layers of the ResNet-50 are disabled as trainables, meaning that the parameters defined in those layers are not getting trained. The only trainable parameters are the ones defined in the last convolution layer of the ResNet and the ones defines in the "output layers". This was set in this way due to the lack of computational resources we had. So the model finally ended up with 15,658,021 trainable parameters, and 8,611,712 non-trainable parameters. With this settings we could to take advantage of the trained ResNet-50 while we fit the model to our requirements.

Operation Group	Operation	Layer name	Number of filters	Filter size	Stride size	Padding size	Size of output image
Input group (G_0)	Input image	Input layer	-	-	-	-	212x212x3
	Cropping	Cropping layer	-	-	-	-	74x74x3
ResNet group (G_1)	-	-	-	-	-	-	NonexNonexNonex2048
Output group (G_2)	Flatten	Flatten layer	-	-	-	-	18437
		Dense layer	-	-	-	-	37

Table 2: Layer structure of the CNN.

3.5 Results.

This section will analyse the overall results of the CNN in matters of loss and train/test errors. The exact results for each galaxy can be interpreted the same way we explained in section 2.2.2.

3.5.1 Results for our Convolutional Neural Network.

Overall we got really good results for our take on the Convolutional Neural Network for this galaxy classification problem. The net ended up taking 4h 36min to train with 10 epochs, getting the following values on training (at the end of the 10nth epoch):

- Final Train Loss: 0.2599
- Final Train RMSE (root mean square error): 0.1239

On our validation set (the test set) we got an slightly higher RMSE, while still being a good result:

- Final Test RMSE: 0.19709530577189951

The following figures illustrate how the train / test error and loss evolve as the number of epoch increases towards 10:

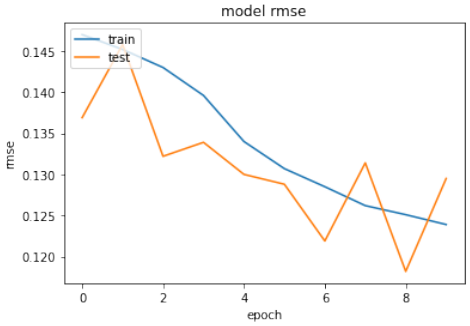

Figure 6: RMSE - Epoch

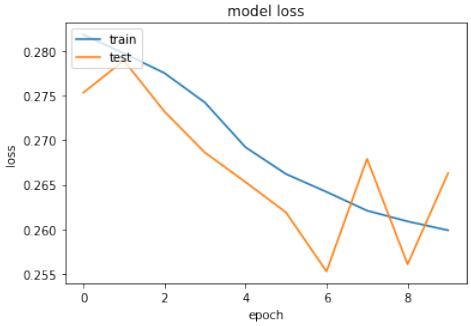

Figure 7: Loss - Epoch

3.5.2 Results for the ResNet-50.

Our application of Transfer Learning, the ResNet-50, did not end up being as good as expected. As we are gonna see soon, it didn't have absolutely bad results either, but our more "specialised" CNN ended up having better results since it was crafted for this problem, so it was to be expected. The ResNet was trained with 35 epochs, and with a bigger number of parameters (As mentioned before, 24,269,733, which of those 8,611,712 where non trainable), ended up needing a way longer time to train than our model: 9 and a half hours. With all of this into account, it ended up getting these values for loss and RMSE at the last epoch:

- Final Loss: 0.2681
- Final Train RMSE (root mean square error): 0.1342

Things start to go south as we compute the final RMSE on the test set:

- Final Test RMSE: 0.22349962399141185

If we analyse how the train / test error and loss evolve as the number of epoch increases, we can definitely spot what went wrong here:

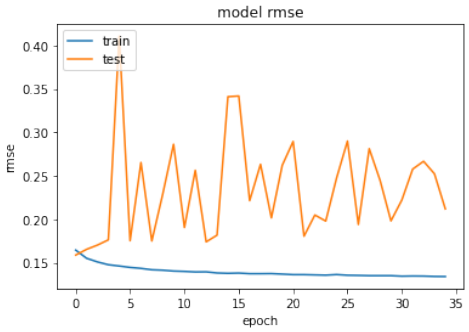

Figure 8: RMSE - Epoch

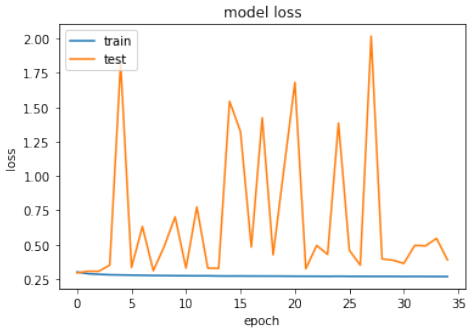

Figure 9: Loss - Epoch

As we can see, we have run into a problem of over-fitting. Over-fitting is a problem that arises when a Machine Learning model gets a potentially good result on the training set, but ends up having way worse results on the test sets. This problem can arise due to a multitude of factors, the most likely one being the fact that we might have too many parameters, that end up learning overly specific inner features of the training data. We are led to believe this since our hand crafted model has way less parameters and learns a better more generalised version of the inner distribution data, which allows it to have both good train error and test error. Although, to be fair, as the number of epochs seems to go up, this seems to stabilize, since the model sees more and more examples that helps it stabilize and generalize better. We can see this happening on figure 9 between the 30 and 35 epoch. We also theorize that we need more batch size and more preprocessing to make this problem fade away.

In addition to this, it would be a proper practical method to let the model train during more epochs, because when talking about pretrained models, you're talking about a model that (usually) has been previously trained with different data than yours, thus, in order to make the model fit the desire data, the initial learning (or fitting) state should be longer, at least until the model finally fits the data, then, the model should theoretically improve considerably regarding to the previous epochs.

4 Implementation.

All the project steps were implemented in Python. We used pandas for reading and preprocessing the data-set, and for the classification task we used TensorFlow and Keras. We illustrate how the implementation works in the Python notebook `galaxy-zoo.ipynb`. The jupyter notebook along with the generated models (ones lighter than 25MB) can be accessed on Github[3]. The data used is accesible in the original kaggle competition web [4].

5 Conclusion.

From this work we can infer a couple of thoughts / conclusions:

- First, mention again the powerful idea of of transfer learning and recycling filters on CNNs. While in this work ended up giving mediocre results due to over-fitting, its really fast and simple to implement, and makes designing CNNs more efficient overall. As discussed in the results section, the over-fitting can still be worked around, but that's out of the scope of this paper.
- Focusing on the convolutional neural networks, it was really interesting to finally be able to work with the most "well known" type of neural network, and point out how easy to interpret and program the neural net ended up being thanks to all the work other researchers invested on Keras and Tensor-flow.

[3] https://github.com/unaicarbajo/galaxy-zoo

[4] https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge

Now about galaxy classification. We ended up having some trouble initially to correctly approach and understand the problem as the galaxy-zoo database can be hard to interpret if not explained correctly, since you're not exactly predicting a class so it becomes a bit of a fuzzy concept. In some papers, its even mentioned as a "regression like" problem. Outside of that, it was really interesting to see how important good preprocessing and good CNN architecture are when processing this kinds of problems where you need to get specific features with the filters to make good predictions (Like detecting the spirals on a galaxy etc...).

If we had to mention some limitations it would be the common limitation of CNNs that is the time needed to train the parameters. Since our focus was on trying to do a good CNN that out hardware could train without inferring into absurd computational costs, this was one of our main concerns. Overall we think that we managed to get that sweet spot between quality of the prediction and training time / computational cost and we are happy with the results.

To finish, on things that could be improve, we find hard to pin point anything that would still strike our balance between quality and cost, but we think a better preprocessing could be done that would relieve more the specific features of galaxies. Not to mention that if we wanted a better version without caring about the cost, the winner of this Kaggle challenge can be taken as an example, as it was the best CNN for the problem (for now, of course). And, as mentioned in section 3.1, we encourage everyone to check S. Dieleman's resolution of the problem (and comment section), because, leaving in one side the problem, it's an impressive work and could be taken as a master-class of deep learning, where he explains the hot (and not that hot) topics of deep learning, in an easy-to-follow way, from the point of view of an experience research scientist at DeepMind.

References

[1] Nour Eldeen M. Khalifa, Mohamed Hamed N. Taha, Aboul Ella Hassanien, and I. M. Selim. Deep galaxy: Classification of galaxies based on deep convolutional neural networks, 2017.

[2] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459, Apr 2015.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[6] Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, and Daniel Thomas. Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society*, 435(4):2835–2860, 09 2013.