Francesco Blangiardi

# Beliven Report

## AI Developer Junior – Test Medium Level [1]

## 1 - Introduction

This project has been developed as an assignment for a selection process started by Beliven. The related position is that of Junior AI Developer, and therefore requires knowledge about AI, Machine Learning, data pre-processing, and the libraries needed to put these concepts into practice using Python.

The candidate was asked to choose one of three projects, each exploring a different area of AI, and the choice fell on “Test Medium Level [1]”. Such test consists in a Visual Classification Project using Tensorflow and Keras, where the candidate is required to develop a machine learning model able to classify images depicting either cats or dogs, with the goal of achieving an accuracy of at least 90%. As a part of the test, it is also required to perform an analysis of the dataset and of the results, as well as the writing of this report.

The following sections will tackle all the topics required by the assignment and will also dwell on how the candidate decided to approach the challenge.

## 2 – Code Structure

Before describing the dataset, it may be more practical to give a brief description of the developed code.

The project repository contains several files, of which the most important ones are described in the following:

- models.py: this file contains the models used within the project, all enclosed within a keras.Model subclass.
- data_analysis.py: a file containing a single function, performing all required operations to analyze the dataset, and producing all the related figures shown in this report.
- train.py: this file contains a single function performing both training and testing of the model and produces all related plots according to the arguments passed to it.
- argument_parser.py: a file specifying all launch arguments of the application.
- main.py: the script that parses the arguments provided through the command line and that represents the access point for the data_analysis.py and train.py files.

A Google Colab notebook showing several examples of how the code can be launched can be found at this link. The output cells of the notebook are generated from the same set of experiments discussed in section 5.

# 3 – The Dataset

Beliven proposed two different datasets to be used for the project: one that can be downloaded at this link, or alternatively the Kaggle Cats and Dogs Dataset. As the Kaggle dataset is considerably larger, the candidate decided to develop the project around the smaller one, purely for hardware and time reasons. It is worth noting that this does not lead to an easier challenge, as handling large datasets has in principle the same amount of difficulty as dealing with fewer data, although in different aspects.

Therefore, the chosen dataset is comprised of a total of 3000 images, already divided into training and testing sets containing 2000 and 1000 images respectively. Moreover, images belonging to each class are already divided into different folders, making it easy to load both datasets by using Tensorflow utility functions. Each image file is named simply after the related class together with an id, therefore it is not possible to infer any further information from the file names alone. As per best practice, the rest of this analysis will be only performed on the training dataset.

The distribution of the two classes is perfectly balanced, as the number of images depicting dogs is the same as the ones depicting cats (1000 images per class). As shown by figure 1 however, the image size is not the same among all pictures.

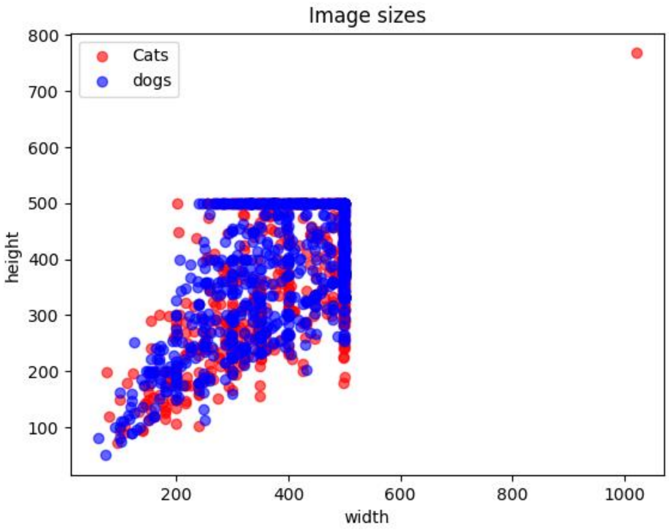

**Fig. 1:** Scatter plot showing the distribution of the image dimensions in pixels.

Outside of one outlier, most images do not have more than 500 pixels for both of their dimensions, and the size distribution seem to be independent of the class label. Figure 2 instead might help visualize the actual area distribution among the images.

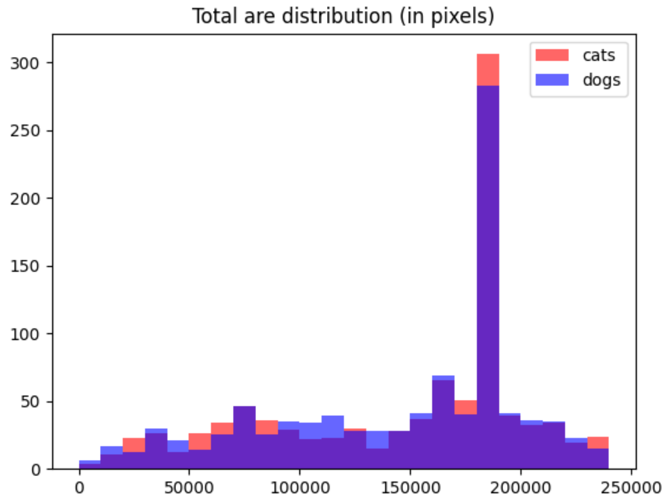

**Fig. 2:** Histogram showing the distribution of the image sizes in pixels.

As shown in the histogram, most pictures contain more than 150k pixels, with a peak at around 180k pixels (around 400x400 images). With that into account, the entire dataset has been resized into 256x256 images as a preprocessing step, in order to not upscale too aggressively while also avoiding the loss of too much information.

As for the actual images depicted, some examples are shown in figure 3.

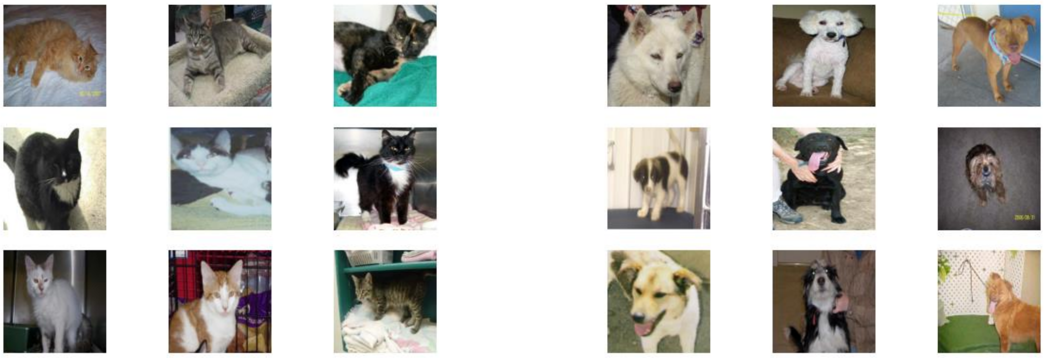

**Fig. 3:** Batches of pictures depicting images from both classes (left for cats, right for dogs) after the resizing step.

It is possible to do some observations:

- Both classes contain images of animals from different breeds, with the dog class having a notably higher diversity.
- The pictures depict different parts of the animal (body, paws etc.) at different degrees, but most of them include the face, with very diverse backgrounds for both classes.
- All depicted animals are facing the camera and are placed in a mostly "canonical" position.

This means that the model will have to generalize its predictions regardless of factors unrelated to the task (i.e. background, breeds etc.) which might prove slightly challenging. Moreover, some extra care will be required during the design of possible data augmentations, in order to not introduce additional difficulties in the task.

# 4 – Models and Training

## Models

Since the task is that of Image Classification, it is safe to assume that a CNN will probably lead to good results.

As a first model, the candidate implemented a custom CNN from scratch, with a core comprised of 3 Convolutional layers interleaved by Max Pooling layers. Attached to these layers is then a classifying head, comprised of a Flatten layer, a Fully Connected layer with ReLu activation, and finally another Fully Connected layer outputting the two class probabilities through a softmax activation.

Since such model might be too simple for the task, the candidate also decided to employ a slightly deeper model as an alternative. At first the choice fell on a model of the Resnet family, but as in this case the Tensorflow framework only provides implementations for models with 50+ layers, this idea was discarded in favor of a slimmer VGG16 model, suitably modified with a classifier head similar to that of the custom model. Choosing a Tensorflow implementation over coding one from scratch is also useful as it allows to perform transfer learning: the dataset for the task may be too small to train a model properly, but by using a pretrained version of the VGG it is possible to keep all its layers frozen and fine tune only the attached classifier head, thus overcoming the scarcity of data.

## Training and Testing

As a standard for both models, the training procedure has been set to last for 25 epochs, with a starting learning rate of 0.0001, and a batch size of 32 images. The optimizer of choice was the Adam optimizer, while the choice of a softmax activation for the output of the models led to the use of a Categorical Crossentropy loss. Both the loss and the optimizer used came from the Tensorflow implementation, just as the training loop which was done through the *fit* function of the keras.Model class

As mentioned earlier, all datasets have been resized into pictures of 256x256 pixels, and as a good practice, all pixel values for the three RGB channels have been normalized in the range [0,1].

In order to have a way of performing validation during training, a validation set has been extracted from the training set comprised of 20% of the total training data, which is reserved exclusively for monitoring the performances of the model during training. This also allows to perform some regularizations techniques for the models, such as reducing the learning rate over the epochs: this has been implemented through a callback to the *fit* function which, by monitoring the accuracy on the validation set at the end of each epoch, lowers the learning by a factor of 0.2 with a patience of 3 epochs.

Other than that, another form of regularization has been implemented in the form of data augmentation. Since the dataset is not very big in size, it is in fact possible that providing augmented data over different epochs may help the models converge without overfitting. This has been implemented through some transformation layers inside the models, with figure 4 showing an example.

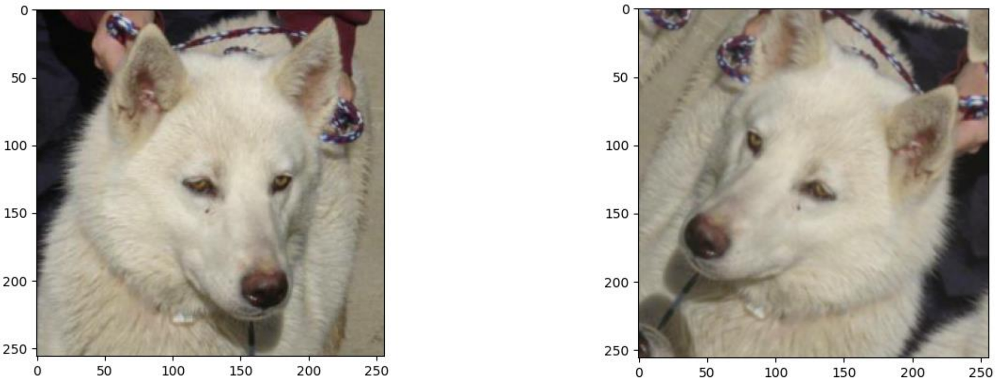

**Fig. 4:** Comparison between a regular picture (left) and its augmented version (right). Considering the observations made in section 3, the augmentation layers implemented are: a random horizontal flip; a random rotation (0.2 max factor); a random contrast transformation (0.2 max factor). These transformations are disabled at inference time.

As for the testing procedure, the candidate decided not to use the built-in function in order to have more freedom on how to aggregate the collected information. What was done instead was a simple forward of the test images through the model to obtain the predictions, which were later combined with the ground truth to compute metrics such as the accuracy, precision, recall, f1 score as well as the confusion matrices through the use of the *sklearn* library.

# 5 – Results

For the analysis of the model performances, 4 main experiments were conducted: two for each model, with and without data augmentation.

Figure 5 shows the accuracies attained during training over the epochs for both the training and validation set and for all 4 experiments.

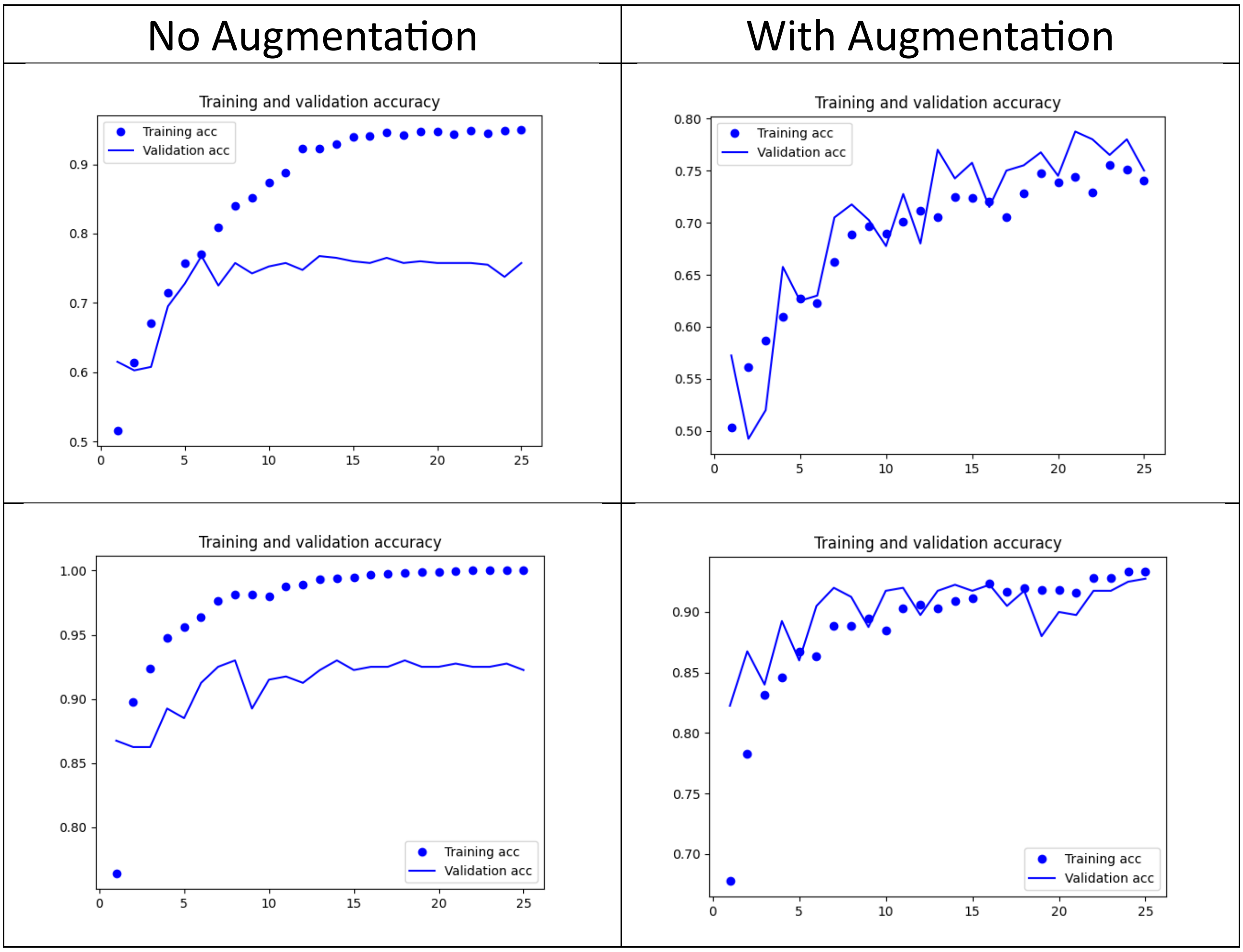

**Fig. 5:** Accuracy on Training/Validation set over all training epochs for the 4 experiments. The two top pictures are taken from the training of the custom model, while the bottom ones

It is possible to do some observations:

- The custom model is able to learn fairly well from the training set, as it reaches an accuracy above 94%. However, the validation accuracy shows that the model is not able to generalize properly from the training, as it converges at a significantly lower value indicating that overfitting has occurred.
- Regularization in the form of data augmentation seems to be slightly useful, as the validation accuracy peaked at 78% (vs 76% without augmentation). The training accuracy instead drops significantly, which is rather normal since the training set is changing constantly among epochs (unlike the validation set to which augmentation is not applied). Overfitting doesn’t seem to be a problem anymore, but since the model is still far from achieving the desired 90% accuracy, it is possible that the model is too simple for the given dataset.
- Differently from the custom model, the pretrained VGG is able to achieve very high training accuracy while also bringing the validation accuracy above the desired threshold. This means that the model is capable of learning well from the training set, and to still achieve

pretty good results when provided with data with a slightly different distribution from that of the training ones.

- Augmentation on VGG leads to a small loss in training accuracy but pulls up the validation accuracy by a few percentage points by the end of the last epoch.

The trends shown from the validation set are then reflected in the results collected from the test set, which are shown in the following table 1 and figure 6

| | Custom | Custom (aug) | VGG | VGG (aug) |
|---|---|---|---|---|
| Accuracy | 71.1% | 73.0% | 92.5% | 93.7% |
| Recall (cats – dogs) | 0.708 – 0.714 | 0.642 – 0.818 | 0.932 – 0.918 | 0.930 – 0.944 |
| Precision (cats – dogs) | 0.712 – 0.709 | 0.779 – 0.695 | 0.919 – 0.931 | 0.943 – 0.930 |
| F1 score (average) | 0.7118 | 0.7518 | 0.9244 | 0.9374 |

**Table. 1:** All metrics collected for the 4 experiments.

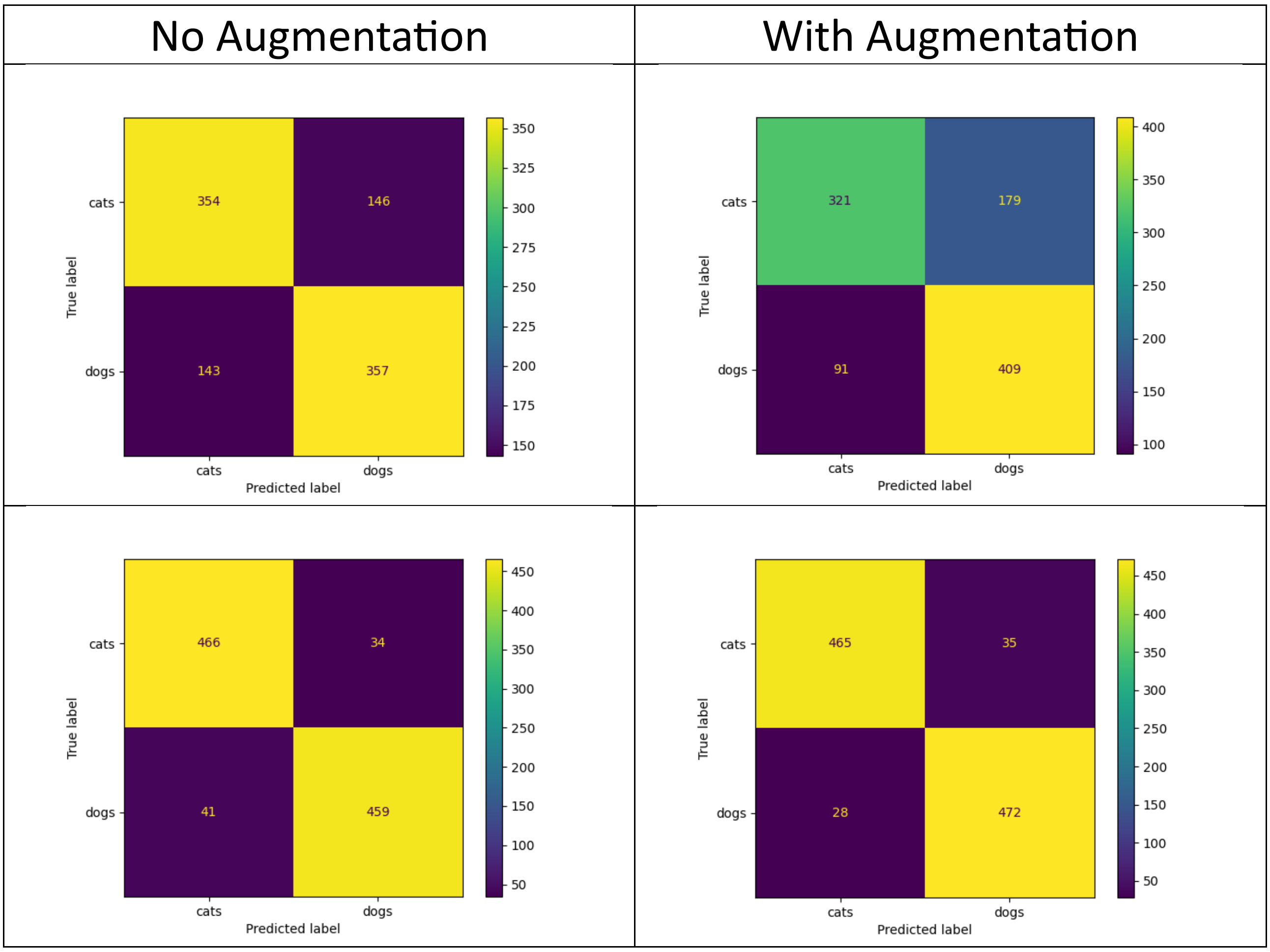

**Fig 6:** Confusion matrices built from the predictions (Custom top, VGG bottom)

Just like in validation, the model achieving the best results is the pretrained VGG with data augmentation. More in detail, both VGG models are able to achieve an accuracy on the test set above the desired 90%, and by analyzing precision and recall from both classes it is possible to see that they are generally unbiased when predicting each of the classes.

It is instead more curious to see the impact of data augmentation in the custom model: the model was indeed able to achieve a better accuracy overall through augmentation, but this came at the expense of a slightly lower recall for the cat class, meaning that the model is more likely than before to mistake cat images as dog ones. A possible explanation for this is that many cat images have been taken in indoor locations with the animals leaning in different positions instead of standing: this may have been exploited by the model without augmentation to better discriminate between the two classes, but this factor may have been slightly tampered with by the Rotation transformation applied during training for the augmented model.

# 6 – Conclusions

In conclusion, all the task required by the assignment have been performed:

- The dataset has been studied and analyzed, highlighting possible challenges for the task, and was also pre-processed and normalized according to the best practices for visual classification tasks.
- Two different models have been developed around the task using the Tensorflow framework, which consisted in a custom CNN network and a VGG16 model exploiting transfer learning.
- Optimizations have been performed in the form of regularization, consisting in both adjustments to the learning rate as well as data augmentation.
- The models were evaluated on the test set, and the results have been collected and analyzed in terms of accuracy, precision, recall and F1 score. Moreover, the goal of achieving an accuracy above 90% has been met successfully by the two best models, with the best achieving an accuracy of 93.7% on the test set.

As for some further work to improve the results, some ideas are reported in the following:

- Collect more data for the task. The dataset chosen for the task is in fact rather small in size, and although it is possible to overcome this issue through transfer learning, this still precludes the training of deeper custom models from scratch.
- Perform more data augmentation transformations. Both models achieved better results through data augmentation, but the transformations themselves were rather simple. Experimenting with more augmentation techniques would probably be useful to virtually expand the dataset and help the models generalize better during training.
- Try different architectures. VGG has been a successful model in this project, but it's possible that other models such as Resnet variants might lead to even better results.

Thank you very much for reading through this report.

Written and developed by: Francesco Blangiardi