Problem 3

Kaggle Team

The Kaggle team entry for this competition is called "Jakil" and it represents the following members:

- Akila Daniel Jeeson
- Myriam Laiymani
- Marc-André Piché
- Khalil Slimi

Introduction

The purpose of this Kaggle competition is to design, train and tune the best CNN model possible, to achieve binary classification of images of cats and dogs. The training dataset consists of 19998 64x64 RGB images divided evenly between dog and cat categories. The test dataset contains 4999 unlabeled images.

On the data side, we did a simple cleaning by not considering images not containing 3 channels for model training. This kind of exclusion is generally not useful nor feasible and can decrease the accuracy on the blind test, but since the size of the dataset is relatively small, we noticed that the grayscale images in the dataset are mainly blank images and won't help in the task of classification at hand (dogs vs cats). Furthermore, we ran some tests and found that removing the said images slightly improves the final accuracy of the model in both validation and test (Kaggle) datasets.

On the software side, we used the framework Keras with TenserFlow as its backend. This choice was motivated by its intuitive, yet powerful API, that allows quick prototyping and easy customization.

CNN architecture and model hyper-parameters

CNN architecture

To tackle this classification task, we decided to opt for a customized version of the VGG16 model. The idea was to benefit from a tried and proven CNN architecture for image classification, while finding a way to reduce the number of parameters of the original architecture, to speed up the training phase.

In order to achieve that, we decided to remove the fully connected layers of VGG16 and replace them with a global average pooling layer (we coded ourselves), that calculates a mean of of every

feature map at the end of the classic CNN layers of VGG16. This modification flattens the image representations as a vector but with a smaller length than what is the case with a fully connected layer (512 instead of 2048), with the added benefit of preserving the spatial information encoded in the image representation.

In summary, the retained CNN architecture consists of 13 convolution layers, 5 max pooling layers, 1 custom average pooling layer and 1 fully connected layer, for a total of 14,715,201 parameters to learn. The details are shown in figure 1, below.

Figure 1: Our CNN architecture.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 64)	1792
conv2d_2 (Conv2D)	(None, 64, 64, 64)	36928
max_pooling2d_1 (MaxPool2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73856
conv2d_4 (Conv2D)	(None, 32, 32, 128)	147584
max_pooling2d_2 (MaxPool2D)	(None, 16, 16, 128)	0
conv2d_5 (Conv2D)	(None, 16, 16, 256)	295168
conv2d_6 (Conv2D)	(None, 16, 16, 256)	590080
conv2d_7 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_3 (MaxPool2D)	(None, 8, 8, 256)	0
conv2d_8 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_9 (Conv2D)	(None, 8, 8, 512)	2359808
conv2d_10 (Conv2D)	(None, 8, 8, 512)	2359808
max_pooling2d_4 (MaxPool2D)	(None, 4, 4, 512)	0
conv2d_11 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_12 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_13 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_5 (MaxPool2D)	(None, 2, 2, 512)	0
custom_average_pooling2d	(None, 512)	0
dense_1 (Fully connected)	(None, 1)	513

Total params: 14,715,201
Trainable params: 14,715,201
Non-trainable params: 0

Model hyper-parameters

The hyper-parameters we considered for tuning our model were the learning rate and the batch size, since they directly influence the convergence of the stochastic gradient descent algorithm.

Grid searching for the optimal hyper-parameter values for our CNN model would be difficult since it has around 14M parameters and its training is somewhat slow. To circumvent this issue, we picked a reasonably small range of value for our hyper-parameters. We selected the range [0.001, 0.01, 0.1] for learning rate and the range [20, 40, 60] for batch size, and then we evaluated the accuracy of our model on the validation dataset, over all the combinations of values for the hyper-parameters. Our experiments showed that the optimal value for batch size was 20, along with a learning rate of 0.1.

Model training

We started our experiments by training the model over 50 epochs with a 90/10 training/validation data split and using a learning rate of 0.1 with a batch size of 20.

As seen in figure 2, the model overfits the training data around the 10th epoch. This is expected since we have a high capacity model, relative to the size and complexity of the training dataset.

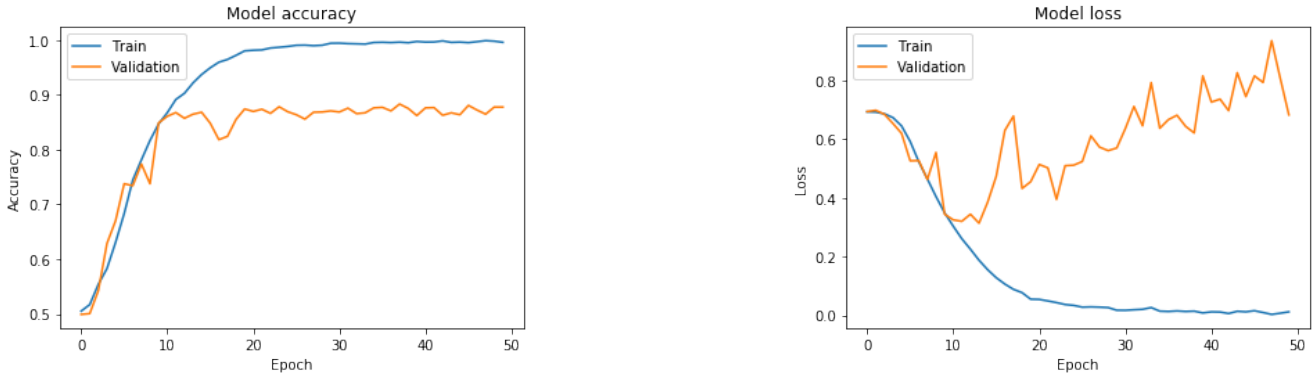


Figure 2: The CNN model overfits the training data.

In order to reduce overfitting and improve the performance on the validation dataset, we decided to use data augmentation in our next experiment.

Model training with data augmentation

We applied data augmentation in the training of the model by applying different transformations to the existing images. The transformations included translations, rotations, flips, shears etc. Obviously, the range and type of these transformation should be reasonable in the sense that the category of a transformed image should remain the same as the original image.

After using data augmentation and redoing the training phase with the same data split and hyper-parameters as in the first experiment, obtained the learning curves shown in figure 3

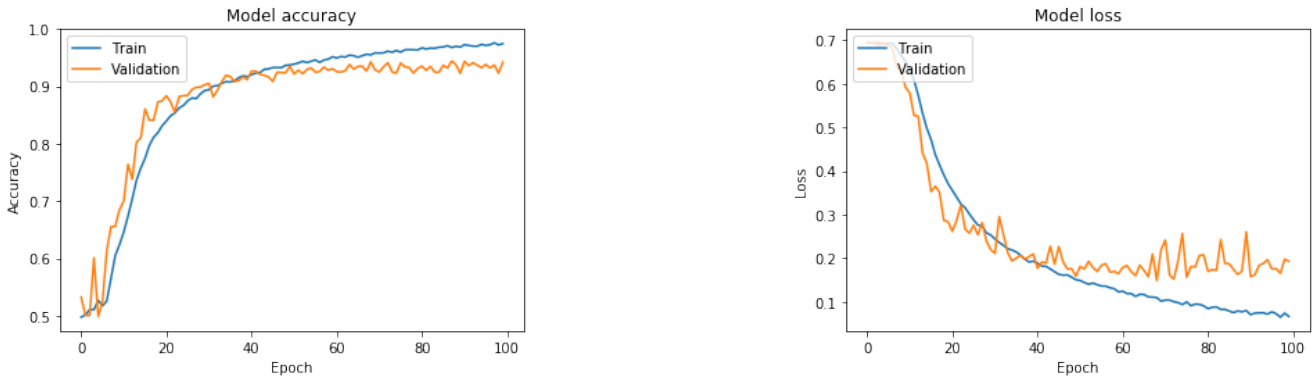


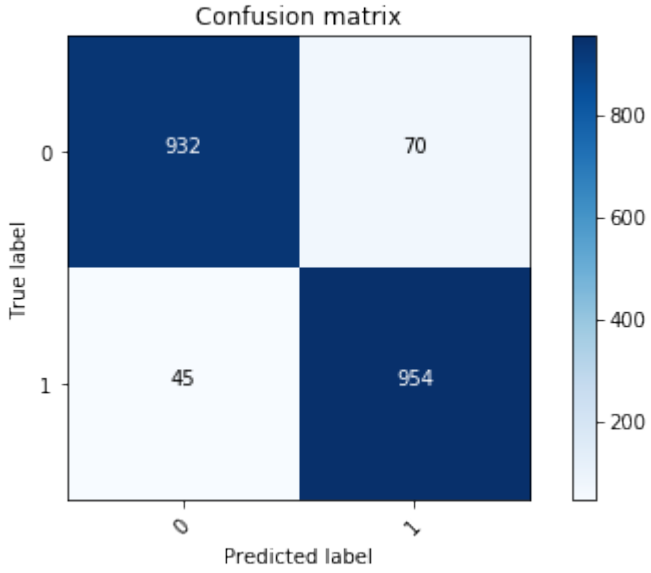
Figure 3: The CNN model training with data augmentation.

It is clear that using data augmentation prevented the overfitting of the model and increased its performance on the validation dataset.

Results

To obtain the best possible performance on the validation dataset, we trained the model over 100 epochs and saved the parameters that gives the smallest validation loss overall. Over 100 epochs, the best accuracy achieved by the model on the validation set was **94.25%**. The confusion matrix is shown in figure 4.

Figure 4: The confusion matrix on the validation dataset.



On the qualitative side of things, figure 5 shows some of the clearly misclassified images (high certainty about the wrong category), whereas figure 6 shows uncertain predictions (around 50%).

Figure 5: Examples of clearly misclassified images.

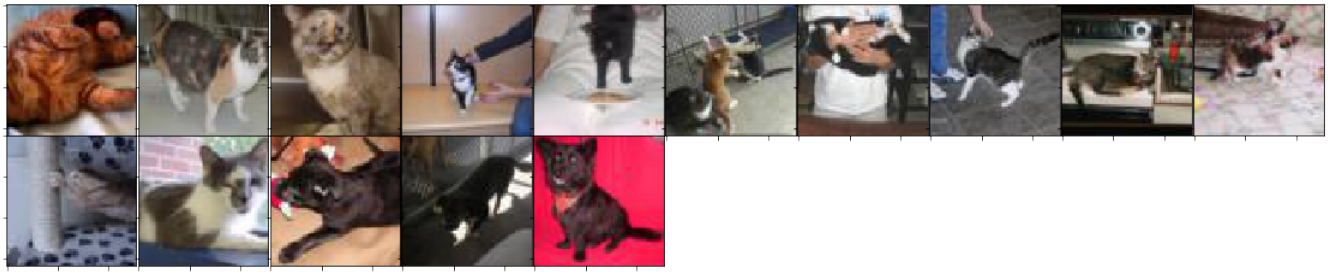
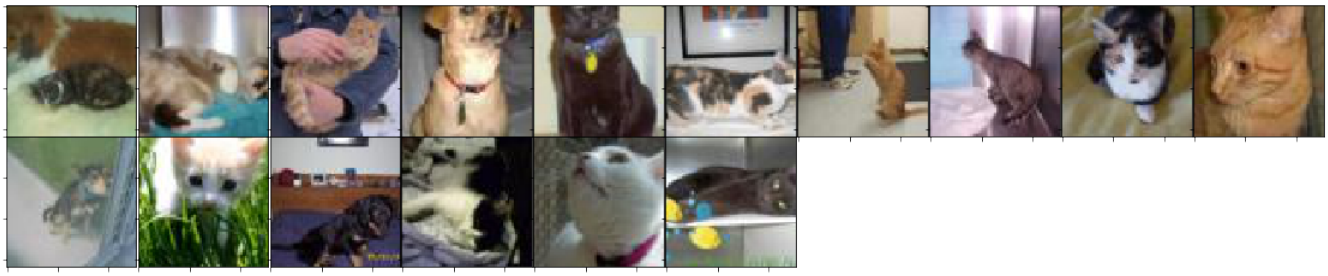


Figure 6: Examples of uncertain predictions.



Discussion

The obtained accuracy on the validation dataset is slightly better than the accuracy achieved by using the trained model using data augmentation on the Kaggle test dataset, which was **94.59%**. This result may seem a bit surprising at first since the validation error rate is a biased estimate of the true error rate and is generally smaller than the latter, but this can be explained by the variance of the model due to the rather small data size.

Although good enough, the obtained accuracy can be improved by using better or more aggressive data augmentation like random cropping of the images. This is justified by the fact that some images don't show the head of the pets or display unusual angles, while others involves the presence of unrelated features (humans, tables, beds...), all of which can lead to uncertain or wrong predictions. Another way to improve the performance of the model is to introduce more regularization techniques like batch normalization, further tune the hyper-parameters of the model to ensure a better SGD convergence, or train the model using a more sophisticated optimization method.

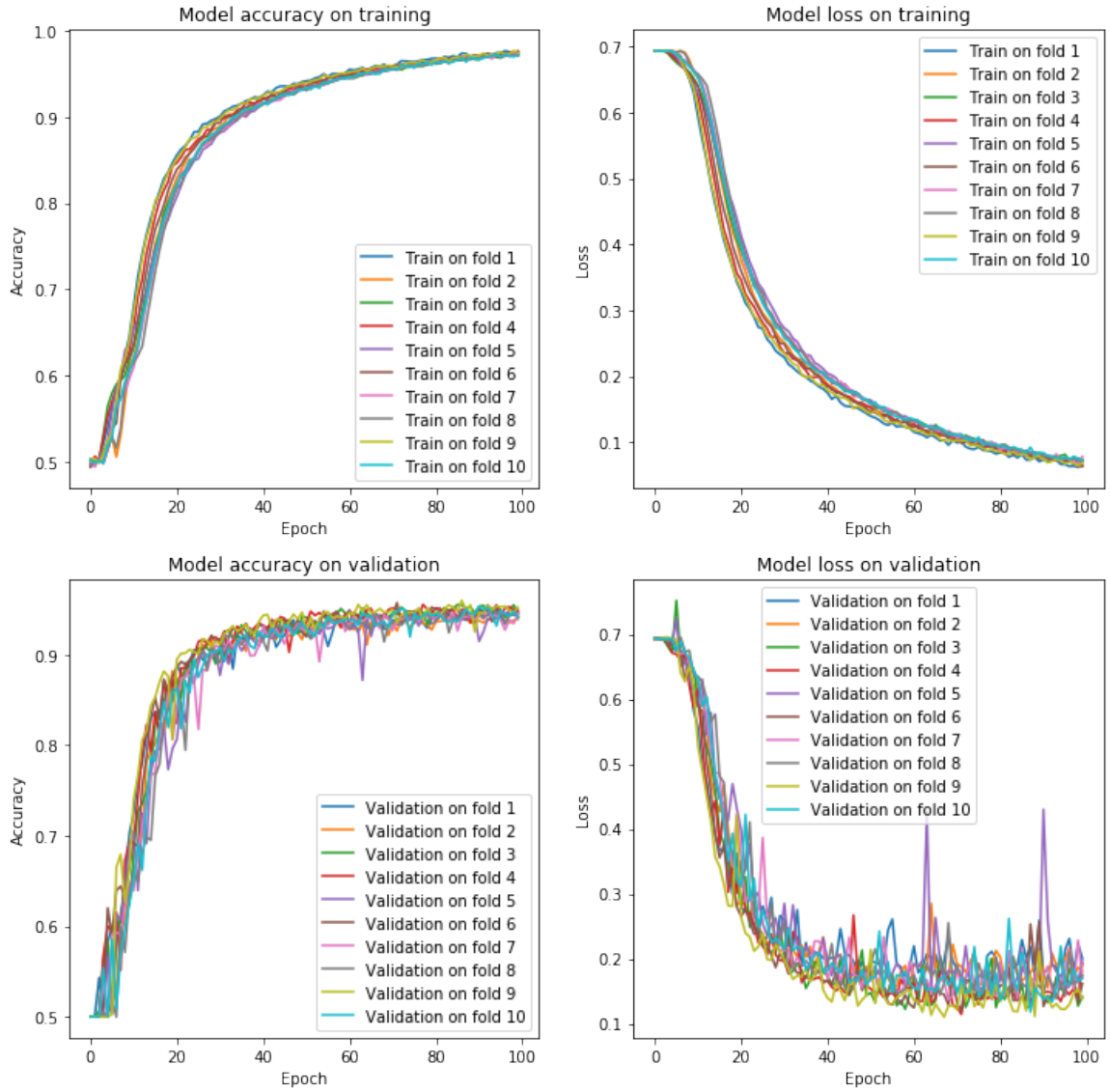
To have a more robust measure of the performance of our CNN model, we decided to evaluate it using 10-fold cross-validation. An added benefit of using this technique, is that we will end up with 10 trained models with each one using a different training/validation data split. We took advantage of this to come up with an ensemble prediction involving the 10 trained models. Indeed, since the training data size is rather small, the variance of the model can be high, and our guess was that an ensemble vote of models trained on different data splits would mitigate this and improve the true error rate on the Kaggle test dataset.

Evaluating the model with cross-validation training

We used 10-fold cross-validation method to evaluate the performance of the model over 100 epochs, using the same hyper-parameters as before (0.1 for learning rate, 20 for batch size) while saving the best weights, with regards to validation loss.

Figure 7 shows the learning curves for this training.

Figure 7: 10-fold cross-validation training.



Over the 10 splits, the mean accuracy was **94.48%** (+/- **0.38%**), which gives a more robust estimation of the performance of the model.

Ensemble prediction

After using 10-fold cross-validation, we ended up with 10 tuned models. For a given image to classify, the output of each model is a probability whether it's a cat (around 0) or a dog (around 1). Thus, it makes sense to average the outputs of the models as a way to come up with more accurate predictions.

After using this ensemble prediction, we reached **96%** on the Kaggle test dataset, which is better than the performance of any individual model in our ensemble.