Capstone project - Dog breed classifier

Riccardo Mattia Galli

Udacity data scientist nanodegree

1 Definition

In this section we outline the current project domain background, how we propose to solve it and which metrics will be used to evaluate the proposed solution.

1.1 Project overview

This project aims to solve a supervised machine learning classification task. In a supervised learning context, image classification algorithms extrapolate information from a labelled dataset in order to learn how to appropriately classify new data. A machine learning algorithm usually contains the following components:

1. A loss function, which measures how close the algorithm performance is from its objective. In a classification setting the loss function, for example, can describe the number of misclassified observations. The goal of the algorithm is indeed to adjust its parameters in order minimize this loss at each iteration.
2. A model architecture, which defines the behaviour of the algorithm, its parameters and its structure.
3. An optimizer, which determines how a given model should update its parameters to minimize the loss function and therefore learn how to perform better.[2].

In recent years, the state of the art for image classification tasks has been achieved by artificial neural networks (ANN). These models are a mathematical representation of the biological neural networks. Each node of this network is responsible to apply computations to parts of the input data. An ANN is divided in layers of neurons, where each layer applies specific operations to the received input and then passes the output as input for the next layer. There are three different layers categories:

1. Input layer, which is responsible of processing the input data.
2. Hidden layer(s), which represent the layer(s) between input and output layer. The hidden layer(s) are devoted to extracting and increase the feature space from the input one, thus generating new parameters and retrieving ulterior useful information for the classifier task.
3. Output layer, which generates the final output (scalar or probability) based on the previous layers computations.

Deep learning is a machine learning subfield which works with ANN using multiple hidden layers. in this field, convolutional neural networks (CNN) represent the state of the art in image classification tasks [4]. Classical ANN architectures, called multi-layers perceptrons (MLPs), have dense layers, where each node in one layer is fully connected with all the nodes of the next layer, as show by figure 1. CNNs, on the other hand, instead of using dense layers, analyze the input with convolutional kernels mimicking the biological visual receptive field [5]. This technique has the advantage of considering the spatial relationship in the input as information. In image classification, for example, pixels that are close probably represent the same object and this information is lost in the MLP architecture due to considering each input part independently. A second advantage is that CNNs have less parameters to train, since they do not have fully connected layers. This results in CNNs training faster than MLPs when considering the same classification task. Overall, these are some of the reasons that make CNNs the top performing model architectures in image classification task, such as on the ImageNet database. [3]. A representation of a CNN architecture applied to digits images classification is shown by figure 2.

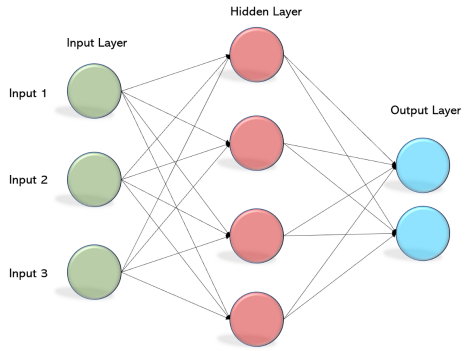

Fig. 1. Multilayer perceptron example, image taken from this Medium article.

1.2 Problem statement

The goal of the project is to build a machine learning model which can:

1. Effectively predict dog breeds from dog images.
2. When a human image is provided, predict the most similar dog breed.
3. Ignore pictures of other classes.

1.3 Metrics

The model is evaluated following accuracy. In a classification task, accuracy represent the percentage of correctly classified images out of all the classified

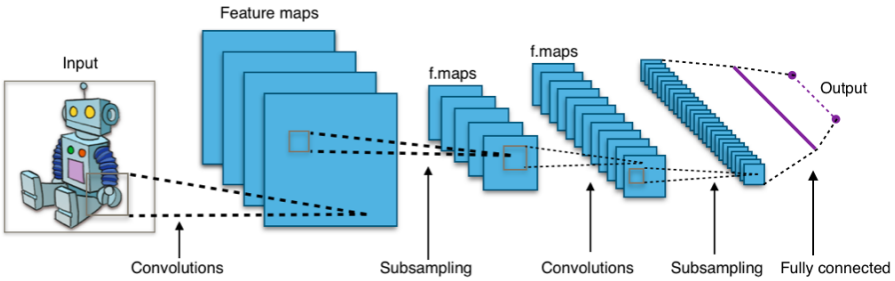

Fig. 2. Convolutional neural network example, image taken from Wikimedia.

images, as shown by equation 1.

$$\frac{\text{correctly classified}}{\text{total classified}} \tag{1}$$

In order to provide an unbiased evaluation metric, the model accuracy is computed based on the model performance over the test set. The test set contains images which have not been used nor to train or validate the model.

2 Analysis

In this section we describe the dataset and techniques used to solve the image classification task.

2.1 Data exploration

There are two main datasets, provided by Udacity, used to complete this project. Both of the datasets are a collection image files:

1. The first dataset corresponds to 13233 RGB input files of human faces images. The images are standardized and have both height and width of 250 pixels.
2. The second dataset corresponds to 8351 RGB input files of dog images. The images width and height span from 113 to 4003 and 112 to 4278 pixels respectively.

A third custom dataset as test the final algorithm. The custom dataset contains 6 images, 2 representing human faces, 2 representing dogs and 2 representing objects.

2.2 Exploratory visualization

Visualizing examples of the two main datasets can help understanding their type of images used to train and evaluate the machine learning algorithms used. Figure 3 shows three images of the human faces dataset, while three images of the dog dataset can be observed in figure 4.

Fig. 3. Images from the human faces dataset.

Fig. 4. Images from the dog dataset.

2.3 Algorithms and techniques

The project is a combination of four different machine learning models, in order:

1. Face detector, which is an OpenCV implementation of a Haar feature-based cascade classifier [6]. It detects whether there is a human face in a given picture and therefore, whether a human is present in an image.
2. Resnet-50 classifier, This is a pre-trained CNN that classifies input images into a 1000 different classes, using weights obtained from the imagenet recognition challenge [9]. 118 of the 1000 classes describe different dog breeds. The final algorithm use this model to predict whether a dog is present in the image.
3. Custom CNN, which is a CNN developed from scratch with Keras [11] in order to learn to classify different dog breeds. This model takes as input the dog dataset described in subsection 2.1. The dog dataset is divided into training, validation and test set which respectively serve to learn the model parameters, extract the best model and test the model's accuracy.
4. Transfer learning model, which is a more accurate model than the custom one in the same task. It exploits the pretrained architecture of the ResNet50 [9] to predict dog breeds.

2.4 Benchmark

Benchmarks have been established by Udacity regarding the models accuracy score, metric discussed in subsection 1.3. The custom CNN model should achieve an accuracy greater than 1% on the test set. The transfer learning model should achieve, on the test set, an accuracy of more than 60%.

3 Methodology

In this section we explain the steps performed to prepare the input data and implement the algorithms described in subsection 2.3. Keras is the main library used to implement the models and prepare the input data [11].

3.1 Data Preprocessing

Data preprocessing is applied to the input data in order to increase the quality and stability of the prediction during training. The face detector requires input images to be converted to gray scale images, as suggested by the OpenCV documentation. ResNet50 input images require preprocessing as well. Input images are resized, converted from 3D to 4D (n samples, rows, columns, channels). Image channels are then reordered from RGB to BGR and lastly normalized according to fixed mean and standard deviation vectors, as described by equation 2.

$$normalization = \Big\{ mean = [103.393, 116.779, 123.68] \tag{2}$$

The custom CNN and transfer learning model use the same data preprocessing of ResNet50 to transform validation and test set data.

3.2 Implementation

The implementation concerns the custom CNN, the transfer learning model and the final algorithm.

The custom CNN implementation takes inspiration from a simple CNN described from Udacity in the notebook. It consists of multiple convolutional layers that extract more feature maps as the network goes deeper, each paired with a maxpooling layer to reduce maps sizes. The final two layers are a general average pooling layer to flatten the input for the last dense layer with softmax activation that transforms the feature maps of the last convolutional/maxpooling layers to probabilities distributed for the 133 dog breeds outputs. As anticipated in subsection 2.1, the input images consists of 3 feature maps (RGB channels), which are passed to the hidden layers by the input layer. The 3 hidden convolutional layers outputs, respectively, 16, 32 and 64 feature maps. The network is implemented with a kernel size of 2, a striding of 1 and no padding. It follows a more detailed description of the network components:

1. 2D convolution: performs a convolution operation on given input feature maps. The kernel size, stride, padding and output feature maps define the behaviour and the number of learnable parameters for this operation.
2. ReLU: the rectified linear unit (ReLU) activaton follows the convolution operation in order to capture the non-linear features contained in the images.
3. Max pooling: performs a down sampling of the input. It works by outputting the maximum element from subpart in the input space. The behaviour of the max pooling operation is defined by the kernel size and the stride parameters. This operation serves to both diminish the number of learnable parameters, hence the training complexity and to avoid overfitting.

The output layer takes as input the output of the hidden layers and computes the following operations to transform the input data to class probabilities:

1. General average pooling: performs flattening on the feature maps generated from the previous hidden layers. It prepares the hidden layers output to be connected to the last linear layer.
2. Dense layer: fully connected layer which collects the previously generated flattened data and outputs a class probability following the softmax activation function.

The transfer learning model is an improvement over the custom CNN one. It solves the same task, dog breeds classification, but it exploits a more powerful pre-trained architecture. We selected the ResNet-50 model to perform a more advanced image classification on the dog dataset. The ResNet-50 is a convolutional neural network that has 50 hidden layers. This state of the art model has shown optimal performance on the ImagNet dataset and won, in 2015, the ILSVRC competition [9]. These properties make the ResNet-50 architecture an optimal choice to serve for transfer learning with the dog dataset. The idea is to take advantage of the pretrained hidden layers and to substitute the output layer with one with 133 output neurons, matching the dog breeds classification task. Only the weights and biases of the last linear layer are trained with the dog dataset. Figure 5 visually summarizes how transfer learning works.

The last implementation step regards the final algorithm which combines the previously described models together. The final algorithm is a combination of the previously described models. The ResNet50 dog detector and OpenCV human face detector components check whether the input picture contains a dog or a human. If this is the case, the algorithm predicts the resembling or actual dog breed with the ResNet-50 transfer model. Otherwise, if neither a human or a dog is detected, the algorithm returns an error.

3.3 Refinement

The models and algorithms implemented in this project have been refined to improve the overall accuracy and meet the benchmarks stated in subsection 2.4. Mainly:

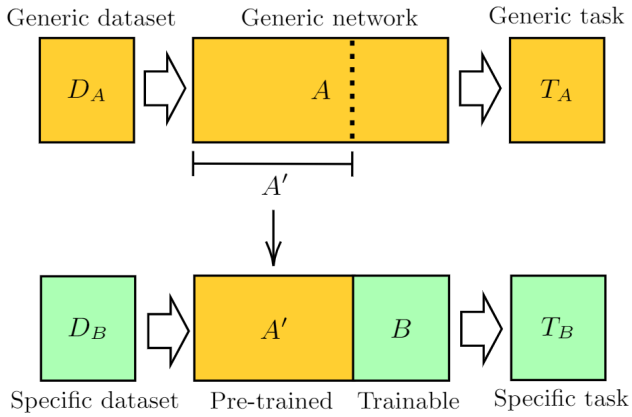

Fig. 5. Image representing how transfer learning works, from this website.

1. Batch size: Both the custom CNN and the transfer learning model are trained following mini-batch gradient descent. Due to the high volume of neural networks training set it is a common practice to optimize the model's parameters on subset of the training data. Higher batch sizes result in higher inference accuracy but also in higher computational complexity [10]. Analyzing empirical evidence, we discovered that a batch size of 20 was a good trade-off between performance and accuracy.
2. Optimizer choice: RMSprop [12] was found to be a good optimizer for this task.
3. Loss function: Categorical cross-entropy was found well describe the loss during the training of the neural network.
4. Detecting dogs before humans. The order of the detectors has been optimized to first use the model that produces less false positives. Indeed, the dog detector has a smaller misclassification rate on human images (0%) than the human detector on dog images (11%).

4 Results

In this section we evaluate the trained models and discuss the results of the algorithm.

4.1 Model evaluation and validation

Both the custom CNN and the transfer learning model have been trained on the training set. The number of epochs have been selected empirically and resulted in 20 epochs for both. At each epoch, the model was tested on the validation set to see how well the model generalizes on unseen data. The model that performed best on the validation, with its learned parameters, was saved and then tested

for accuracy on the test set. Saving the model with the best validation score ensures that the final model is the one that better generalizes on unseen data. The custom CNN scored an accuracy of 4.2% on the test set. The ResNet-50, as expected, scored an higher accuracy of 82.6% on the same test set. The results of the models are sensitive to random shuffling of the training data.

The last step was testing the final algorithm on the custom dataset, described in 2.1, in order to provide unseen input images with dogs, humans or none of the two. The results confirm that the algorithm is working as expected and that distinguishes between the three type of images following the accuracy metrics of its components. A visualization of the application output can be seen in figure 6

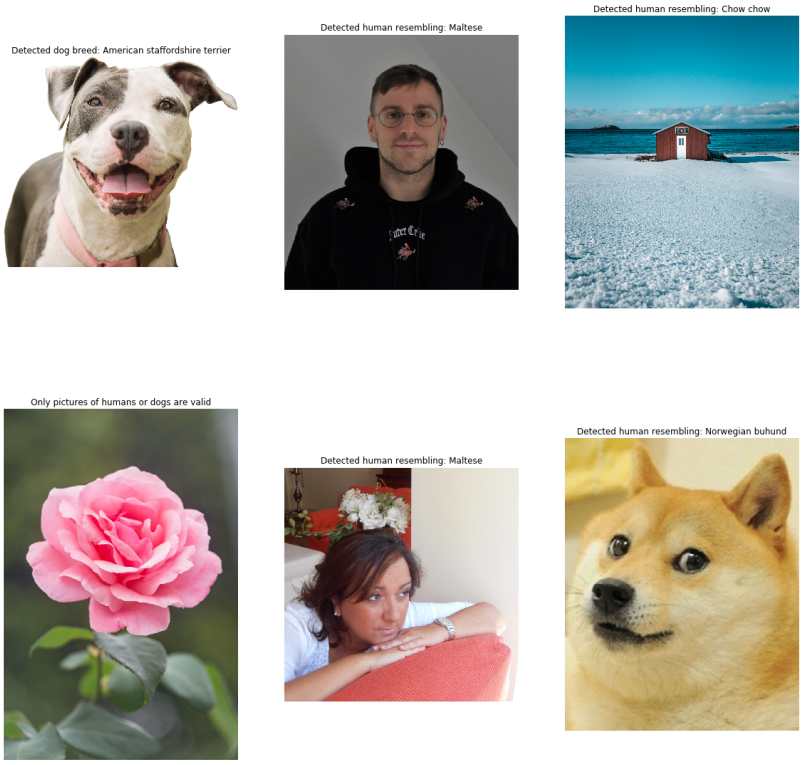

Fig. 6. Visualization of the final algorithm applied to the custom dataset.

4.2 Justification

Both the trained models have an higher accuracy of the benchmarks described in subsection 2.4.

5 Conclusion

In this section we briefly summarize the project once again and provide possible future improvements.

5.1 Reflection

In project we have analyzed and solved an image classification task. First, we introduced the problem to be solved and outlined the main background concepts like supervised learning, artificial neural networks and convolutional neural networks (see 1). We described, in section 2, the input datasets and the models used to exploit these datasets with their relative benchmarks. We then explained, in section 3, the steps to pre-process the input data and to build a CNN and a transfer learning model. Lastly, we showed, in section 4, that both the CNN and the transfer learning model obtain positive results and have an higher classification accuracy than their benchmarks.

5.2 Improvement

The implemented algorithm solves the original problem, with good accuracy in classifying dog breeds. There is, though, a margin of optimization to improve further the resulting algorithm. It follows an outline of the main points which could help in enhancing the developed algorithm.

1. Increasing the dog dataset size. Having more examples to train an algorithm helps increasing the overall model accuracy and reducing the occurrence of overfitting.
2. Testing other models for transfer learning. Using different models for transfer learning could result in a better classification algorithm for this task.
3. Testing different optimization algorithms and loss functions. This could help in selecting techniques which work better, for this setting, than the Rmspropr [12] optimizer and the categorical cross entropy loss function.
4. Increasing the batch size. As described in subsection 3.3, having a bigger batch size directly impacts the model accuracy. Training a model with a more powerful hardware could make it possible to exploit this improvement.

References

1. Caruana, R. and Niculescu-Mizil, A., 2006, June. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning (pp. 161-168).
2. Minka, T.P., 2003. A comparison of numerical optimizers for logistic regression. Unpublished draft, pp.1-18.
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

4. Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
5. Luo, W., Li, Y., Urtasun, R. and Zemel, R., 2016. Understanding the effective receptive field in deep convolutional neural networks. In Advances in neural information processing systems (pp. 4898-4906).
6. Bradski, G. and Kaehler, A., 2000. OpenCV. Dr. Dobb's journal of software tools, 3.
7. Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
8. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (pp. 8024-8035).
9. He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
10. Radiuk, P.M., 2017. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. Information Technology and Management Science, 20(1), pp.20-24.
11. Chollet, F., and others, 2015. Keras. GitHub. Retrieved from https://github.com/fchollet/keras.
12. Zaheer M., Reddi S. J., Sachan D., Kale S. and Kumar S.. 2018. Adaptive methods for nonconvex optimization. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18) (pp. 9815–9825).