Multiclass classification for dog breeds

Machine Learning project 2019-2020

Simone Antonelli
1753685

Donato Crisostomi
1754001

Contents

1 Introduction

In this project we aim to build a convolutional neural network which can recognize dog breeds. We are going to present two different approaches:

- realization of an ad-hoc *Convolutional Neural Network* from scratch;
- use of state-of-the-art pre-trained networks for *transfer learning*.

As we will see, the former approach suffers from the small dimension of the dataset, making it suitable for binary or 3-classes classification but unable to generalize to an arbitrary number of classes, while the latter succeeds to grasp the signal even when a significant number of classes is given as input.

Figure 1: Some of the breeds.

2 Data

As for every supervised learning problem, a labeled dataset is needed. We found the following dataset, provided by the Standford University, which is composed of 120 dog breeds, where each class has in average ≈ 170 images, for a total of 20580 images.
The dataset is then given as a set of folders, where the folder name is the breed and the images inside it are specimens of that breed.

2.1 Data wrangling

The following function loads the pictures and stores them in two numpy arrays, X containing pictures and y containing labels, where y_i is the label of X_i.

Listing 1: Images loading.

```
def load_pictures(dataset_path):
    X, y = [], [];
    for breed in os.listdir(dataset_path):
        breed_folder = dataset_path+breed
        for img_file in os.listdir(breed_folder):
            img_path = breed_folder + '/'+ img_file
            img = cv2.imread(img_path)
            X += img
            y += breed
    return np.asarray(X),np.asarray(y)
```

2.2 Data normalization

Images in the dataset are all different in sizes, encoded as standard RGB pictures with integer values in $[0, 255]$. To feed the model with heterogenous data, we normalize the images by dividing them by 255 so that they are all in $[0, 1]$ and by resizing them to shape $(150, 150, 3)$ where the first two values are the width and the height and the third is the number of channels.

Listing 2: Images normalization.

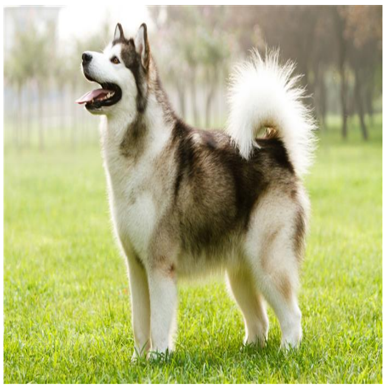
```
def normalize_imgs(imgs):
    norm_imgs = []
    for img in imgs:
        resized_img = cv2.resize(img, (150,150))
        norm_img = np.clip(resized_img/255.0, 0.0, 1.0)
        norm_imgs.append(norm_img)
    return norm_imgs
```

2.3 Dataset issues

As for most real cases scenarios, the available images are not what a computer scientist would dream of. We listed few of the reasons that made this dataset challenging.

2.3.1 Low intra-class variance

Some breeds are hard to distinguish even for an human expert, take for example Figure 2: these two breeds are incredibly similar, the easiest way for an human to tell them apart is by their tails, which may often not be visible in the image provided to the model.


(a) Malamute

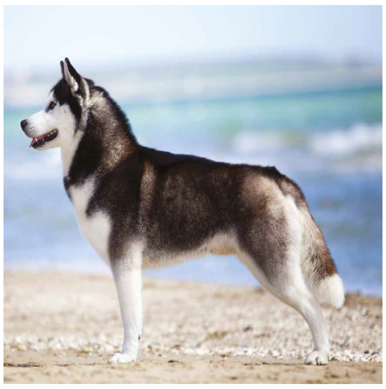
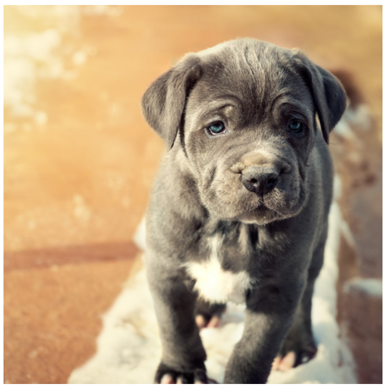

(b) Husky

Figure 2: Similarity between the Alaskan Malamute and the Siberian Husky.

2.3.2 High inter-class variance

The model should be able to recognize a breed even when provided an image of a puppy. This is intrinsecally hard to accomplish because of the huge diversity in size and traits, take for example Figure 3, the puppy can be ten times smaller than an adult, while the adult has a more pronounced wrinkled skin.


(a) Puppy

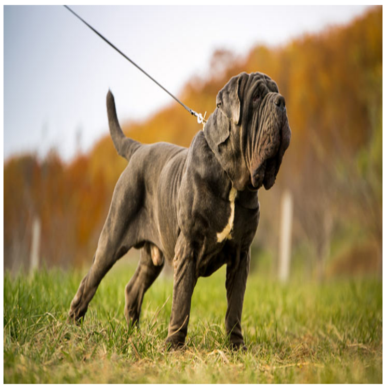

(b) Adult

Figure 3: Diversity between puppy and adult Neapolitan Mastiffs.

2.3.3 Multi-scale

Pictures are taken from varying distance, so the class to recognize will be scaled accordingly. This means that the model should be robust to scaling, instead of learning to recognize a fixed-size object.

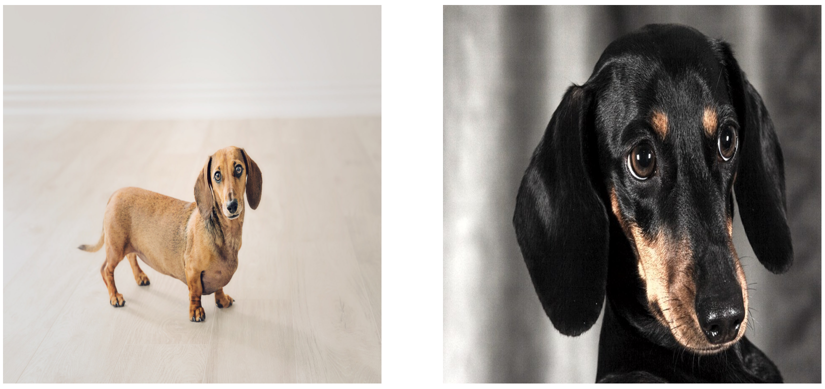

(a) Background (b) Foreground

Figure 4: Basset pictures taken at different distances.

2.3.4 Varying illumination

The accuracy of recognition is greatly affected by varying degree of illumination on the images. Particularly, the changes in direction and intensity of illumination are two major contributors to varying illumination. These depend on various factors, among these are the environment where the picture was taken, the time of the day and the season.

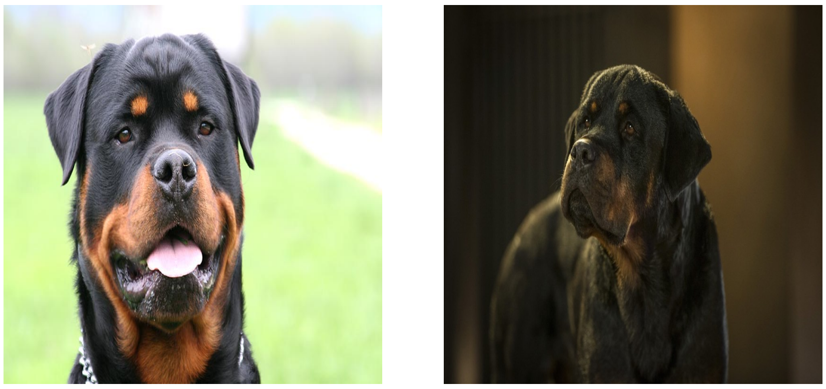

(a) Bright (b) Dark

Figure 5: Rottweiler photographed with different illumination.

2.3.5 Cluttered scenes

Last but not least, images in the dataset often represent cluttered scenes in which the dog is just one of the many objects and persons in the frame. For example, exemplars of Afghan Hound are often photographed during dog beauty contests, resulting in photos in which the owner and/or the jury are also present.

Figure 6: An Afghan Hound.

2.4 Data augmentation

To overcome some of the issues cited in the previous section, we performed data augmentation to expand the dataset with variations of the available images [1]. This is extremely important as it allows the model to learn useful invariants, making it more robust, while increasing the size of the dataset by a relevant factor.

2.4.1 Rotation

A reasonable rotation may help the model learn rotation-invariant features; for this task we avoided rotating the images in unnatural ways and opted for 45 and 315 degree rotations, which can give the effect of the dog tilting his head.

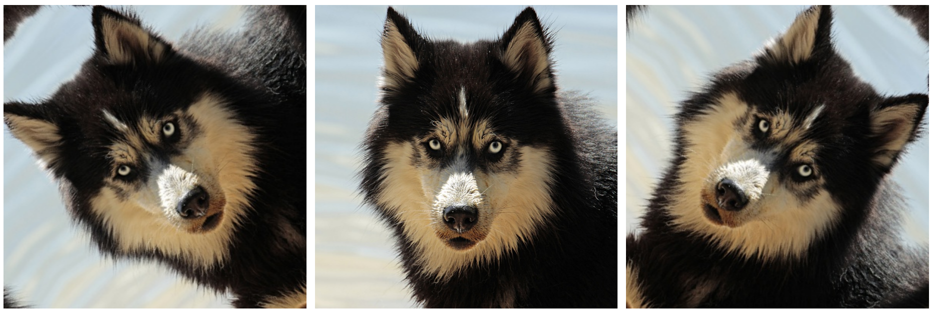

(a) 45 degrees rotation (b) original (c) 315 degrees rotation

Figure 7: Rotations of a Husky photo.

2.4.2 Translation

Translation involves moving the image along the X or Y direction (or both). This method of augmentation is very useful as most objects can be located at almost anywhere in the image. This forces the model to look everywhere.

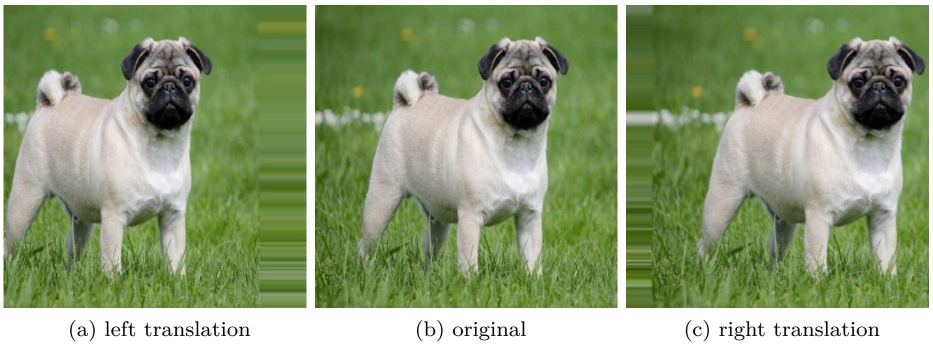

(a) left translation (b) original (c) right translation

Figure 8: Translations of a Pug photo.

2.4.3 Flip

An image can be flipped horizontally or vertically, but the latter isn't really appropriate for this type of task.

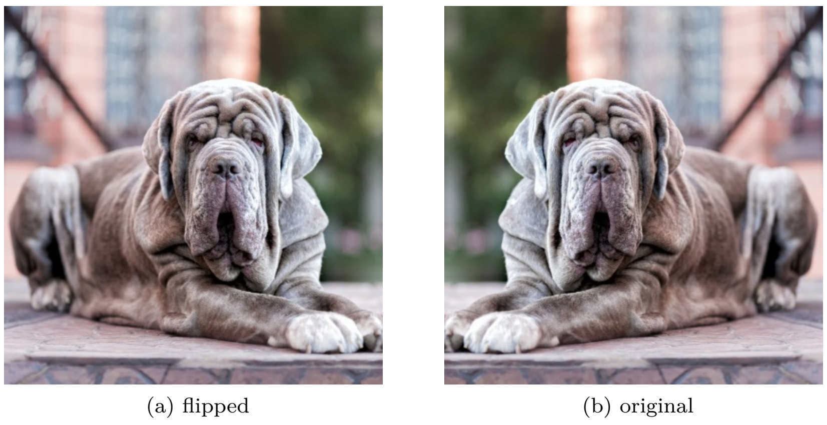
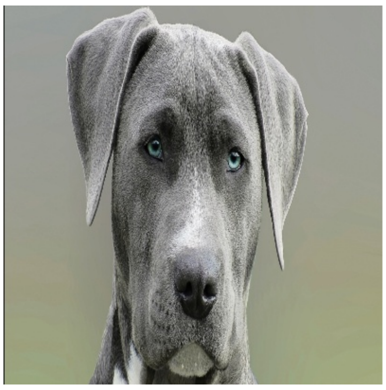

(a) flipped (b) original

Figure 9: Flipping of a Neapolitan Mastiff photo.

2.4.4 Noise

Over-fitting usually happens when the network tries to learn high frequency features (patterns that occur a lot) that may not be useful. Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features. This also means that lower frequency components are also distorted, but the neural network can learn to look past that. Adding just the right amount of noise can enhance the learning capability.


(a) original

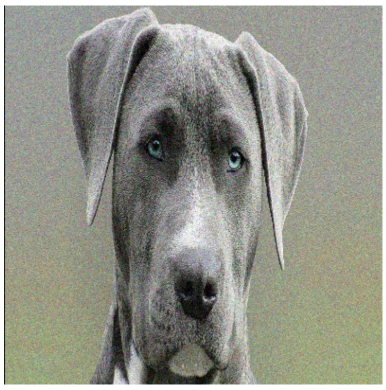

(b) added Gaussian noise

Figure 10: Addition of Gaussian noise.

3 Convolutional Neural Networks

Convolutional Neural Networks represent the state-of-the-art model for image processing, being thus the first choice we opted for.

3.1 Introduction

CNNs are a specialized kind of neural network for processing data that has a known grid-like topology. The convolution operator is a specialized kind of linear operation and CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. Among the motivations that make this model efficient are *sparse interactions*, *parameter sharing* and *equivariant representations*.

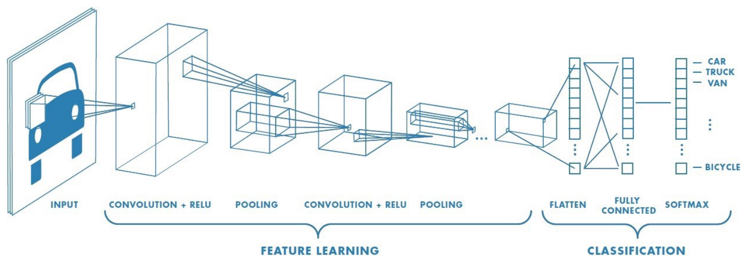


Figure 11: A typical CNN architecture.

A typical CNN is composed of an arbitrary number of convolutional layers, each usually followed by a pooling layer. In the convolutional layers, a digital filter (a small 2D weight mask, also called *kernel*) is applied to the input, sliding it over the different positions. For each position, an output value is generated by replacing the source pixel with a weighted sum of itself and its nearby pixels.
Filters encode specific aspects of the input data: a first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on.
Pooling layers are used to aggregate data in input in order to produce lower-dimensional feature maps, reducing the computational complexity of the network and preventing the model from overfitting. The idea is to reduce the number of unnecessary details, keeping the invariants.
The last layers are instead fully connected and use the representation provided by the previous layers to effectively do the classification.

3.2 Implementation

To implement a CNN, a large number of hyperparameters must be chosen. Among the most important ones are the size of the kernels, the width of each layer in terms of kernels, the depth of the model and the kind of pooling to apply.
As for most deep networks, $ReLU$ has been used as activaction function for the internal layers to avoid the vanishing gradient problem.

$$Relu(z) = max(0, z).$$

Since this is a multi-class classification problem, the *softmax* function has been used for nodes n_k in the final layer

$$y_k = f(net_k) = \frac{e^{net_k}}{\sum_j e^{net_j}}.$$

Finally, the *categorical cross-entropy* has been used as loss function.

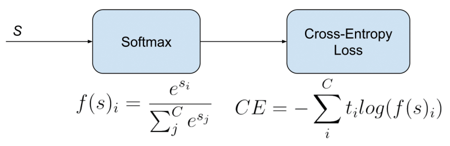
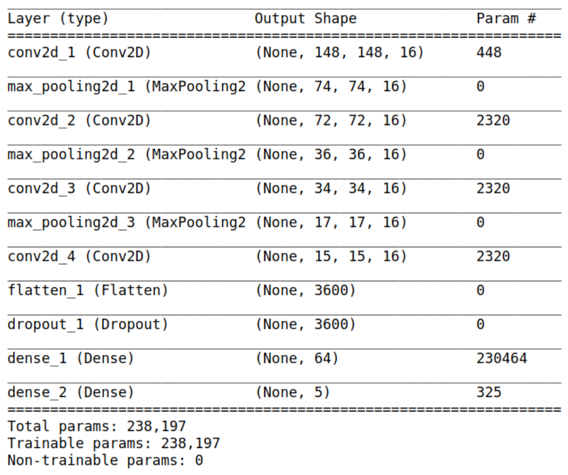

s
Softmax
Cross-Entropy Loss
$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$
$CE = -\sum_i^C t_i log(f(s)_i)$

Figure 12: Categorical crossentropy.

The chosen architecture can be seen in Figure 13.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 16)        2320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 16)        2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 16)        2320
_________________________________________________________________
flatten_1 (Flatten)          (None, 3600)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                230464
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 325
=================================================================
Total params: 238,197
Trainable params: 238,197
Non-trainable params: 0
```

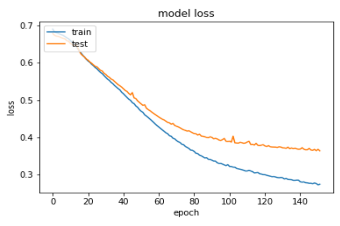
Figure 13: Summary of the CNN.

The chosen parameters are fruit of a search in which various combinations have been tested. In particular, the model seemed very sensible to overfitting; this isn't too surprising given the limited size of the dataset. For this reason, various regularization techniques have been tested. [4]

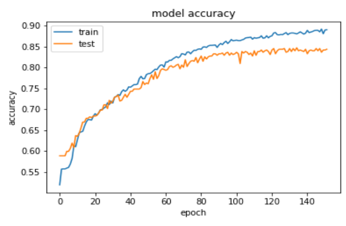
3.2.1 Early stopping

When training large models with sufficient representational capacity to overfit the task, the training error is often observed to decrease steadily over time, while the validation error at a certain point begins to rise again. This means we can obtain a model with hopefully better generalization error by returning to the parameter setting at the point in time with the lowest validation set error. This is achieved by storing a copy of the model parameters every time the error on the validation set improves, and returning these parameters when

the algorithm terminates, instead of the latest ones. The algorithm terminates when no parameters have improved over the best recorded validation error for some fixed number of iterations, called the patience parameter. This way, the optimal number of epochs is determined by the model itself.
In Figure 14 we can see the effect of early stopping with $patience = 30$, making the algorithm terminate at epoch ≈ 150 instead of allowing it to overfit.



(a) Loss



(b) Accuracy

Figure 14: Loss and accuracy for two breeds classification with early stopping.

3.2.2 L2 regularization

The approach of $L2$ regularization, also known as weight decay, is to add an extra term in the loss function that can be thought of as corresponding to a soft constraint on the parameter values. The resulting loss function is thus

$$\tilde{L}(\theta; \vec{X}, \vec{y}) = L(\theta; \vec{X}, \vec{y}) + \lambda \vec{w}^2 \qquad (1)$$

where $\vec{w}$ are the weights excluding the bias, which is often not regularized for deep neural networks since it doesn't induce much variance and may instead provoke serious underfitting.

3.2.3 Dropout

Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks. Specifically, dropout trains the ensemble consisting of all subnetworks that can be formed by removing nonoutput units from an underlying base network, as illustrated in Figure 15. This is easily done by multiplying the output of the units to be removed by zero.

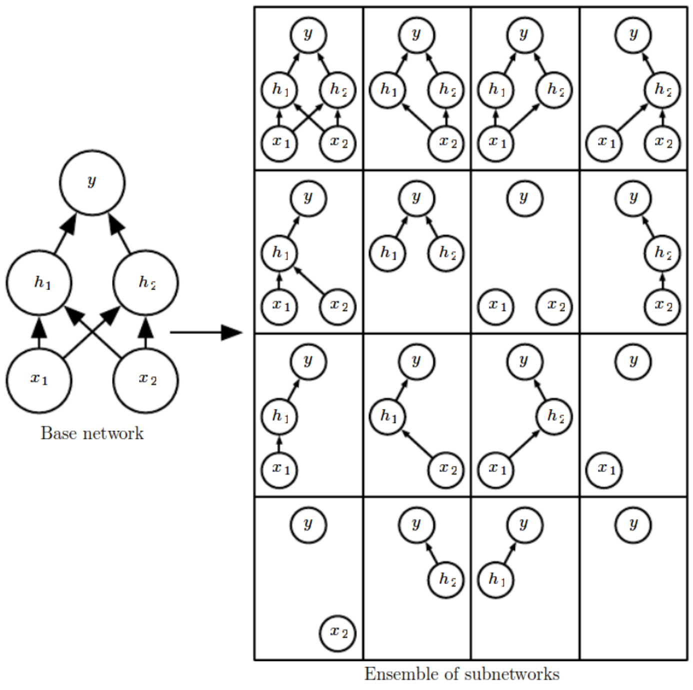
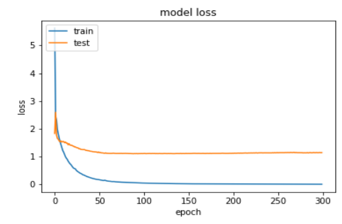


Figure 15: Dropout as an ensemble.

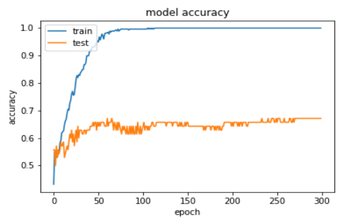
Recall that to learn with bagging, we define k different models, construct k different datasets by sampling from the training set with replacement, and then train model i on dataset i. Dropout approximates this process, but with an exponentially large number of neural networks. To train with dropout, a minibatch-based learning algorithm is used, like stochastic gradient descent. Each time an example is loaded into a minibatch, a randomly sampled binary mask is applied to all the input and hidden units in the network. The mask for each unit is sampled independently from all the others. The probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins. To predict in bagging we would have taken the average of the predictions of the base models, but in this case it would be unfeasible since the base models are exponentially many. Thus, the inference is approximated with sampling, by averaging together the output from many masks.

3.3 Results

In Figure 16 we can see the accuracy and loss of the model prior to data augmentation and model tuning for binary classification.



(a) Loss



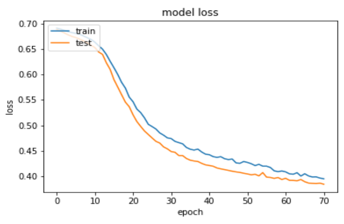
(b) Accuracy

Figure 16: Loss and accuracy for two breeds classification with no data augmentation nor model tuning.

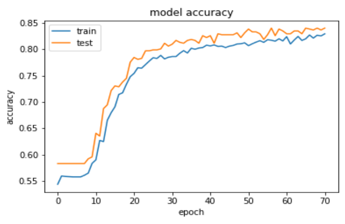
The model severely overfits and fails to reach an acceptable accuracy even for two class classification. After expanding the dataset with the techniques previously cited and regularizing the model with early stopping and dropout we instead obtain the following results.

3.3.1 Two breeds

Below are the results of the model for the case of two breeds, namely the Chihuhua and the Norwegian Elkhound.



(a) Loss



(b) Accuracy

Figure 17: Loss and accuracy for two breeds classification.

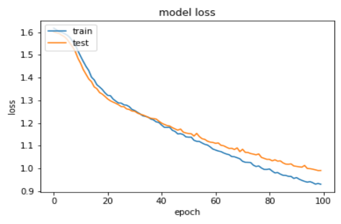
The obtained accuracy is reasonably good for a simple model; nevertheless, more complex models would need much more data in order to learn all the resulting parameters.

```
              precision    recall  f1-score   support

           0       0.89      0.81      0.85       248
           1       0.86      0.92      0.89       309

    accuracy                           0.87       557
   macro avg       0.87      0.86      0.87       557
weighted avg       0.87      0.87      0.87       557
```

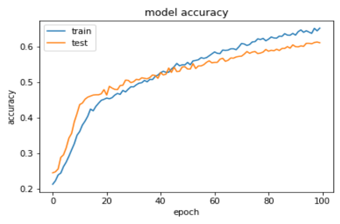
Figure 18: Classification report for two-breeds classification.

3.3.2 Five breeds

Below are the results of the model for the case of five breeds, namely Chihuhua, Basset, Bull Mastiff, Japanese Spaniel and Norwegian Elkhound.



(a) Loss



(b) Accuracy

Figure 19: Loss and accuracy for five breeds classification.

As the results are already unsatisfactory, it is not much meaningful to try a larger number of classes.

```
              precision    recall  f1-score   support

           0       0.59      0.59      0.59       285
           1       0.52      0.54      0.53       222
           2       0.56      0.33      0.42       258
           3       0.75      0.80      0.78       304
           4       0.63      0.78      0.70       314

    accuracy                           0.62      1383
   macro avg       0.61      0.61      0.60      1383
weighted avg       0.62      0.62      0.61      1383
```

Figure 20: Classification report for five breeds.

4 Transfer Learning

In this section we present a scalable and efficient approach giving optimal results at low computational cost, which is being really sucessfull in many practical applications.

4.1 Overview

A common and highly effective approach to deep learning on small image datasets is to use a pretrained network, which is a network that was previously trained on a large dataset typically on a large-scale image-classification task.

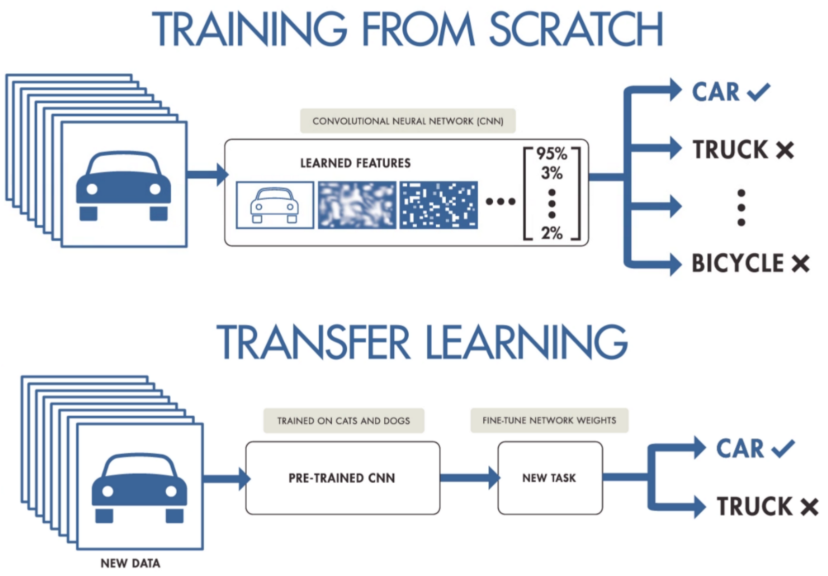


Figure 21: Transfer learning and training from scratch comparison.

If the original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pretrained network can effectively act as a generic model of the visual world, and hence its features can be useful for many different computer-vision problems, even though these new problems may involve completely different classes than those of the original task. Such portability of the learned features across different problems is a key advantage of deep learning compared to many, older, shallow-learning approaches, and it makes deep learning very effective for small-data problems. In this case, let's consider a large convnet trained on the *ImageNet*[1] dataset, called Xception.

4.1.1 Model

A convolution layer attempts to learn filters in a 3D space, with 2 spatial dimensions (width and height) and a channel dimension; thus a single convolu-

[1]A large dataset (1.4 million labeled images and 1000 different classes) where classes are mostly animals and everyday objects

tion kernel is tasked with simultaneously mapping cross-channel correlations and spatial correlations.
In Inception, a large network which has been proven very successfull, a typical module first looks at cross-channel correlations via a set of 1×1 convolutions, mapping the input data into 3 or 4 separate spaces that are smaller than the original input space, and then maps all correlations in these smaller 3D spaces, via regular convolutions. The fundamental hypothesis behind Inception is that cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly. [2]

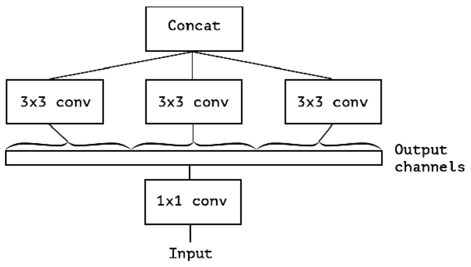


Figure 22: A typical Inception module.

An “extreme” version of an Inception module would first use a 1×1 convolution to map cross-channel correlations, and would then separately map the spatial correlations of every output channel.
The Xception architecture is based entirely on depth-wise separable convolution layers and it is built under the following hypothesis: the mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled. Because this hypothesis is a stronger version of the hypothesis underlying the Inception architecture, its name is Xception, which stands for “Extreme Inception”.

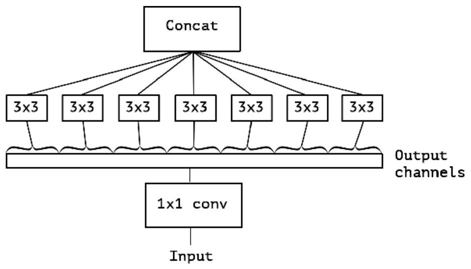


Figure 23: A typical Xception module.

A complete description of the specifications of the network is given in Figure 24. The Xception architecture has 36 convolutional layers forming the feature extraction base of the network. In our case we will do image classification and therefore our convolutional base will be followed by a logistic regression layer.

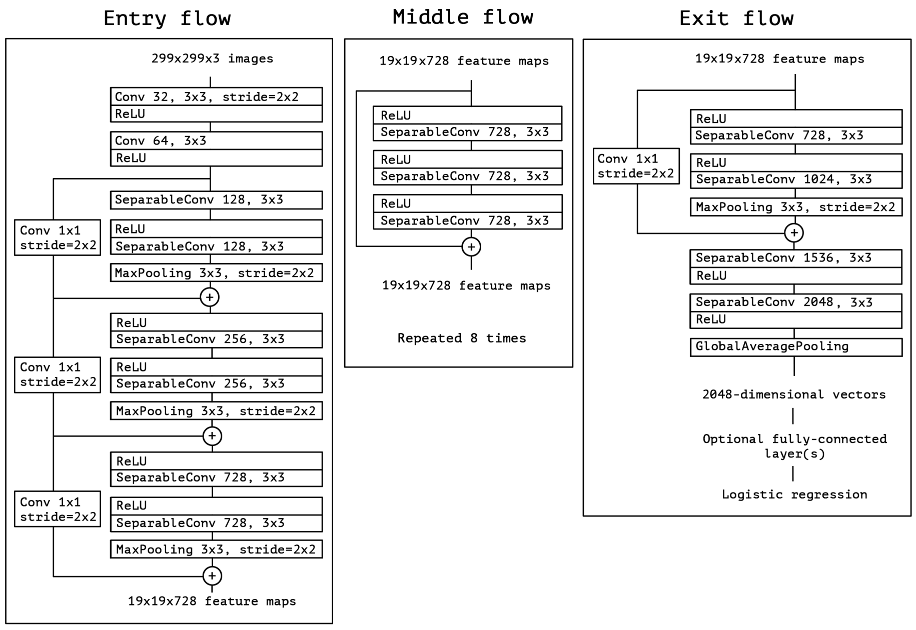


Figure 24: Architecture of the Xception convolutional neural network.

4.2 Visualization

The representation learned by convnets are highly amenable to visualization, in large part because they're representation of visual concepts. We present a couple of techniques developed for visualizing and interpreting what a convnet learns:

- Visualizing convnet filters: useful for understanding precisely what visual pattern or concept each filter in a convnet is receptive to;
- Visualizing heatmaps of class activation in an image: useful for understanding which parts of an image were identified as belonging to a given class, thus allowing you to localize objects in images.

Visualizing convnet filters This can be done following a simple process: build a loss function that maximizes the value of a given filter in a given convolution layer, and then use stochastic gradient descent to adjust the values of the image so as to maximize this activation value. [3]

Listing 3: Generate filter patterns

```
def generate_pattern(layer_name, filter_index, size):
    layer_output = model.get_layer(layer_name).output
    loss = mean(layer_output[:, :, :, filter_index])
    grads = gradients(loss, model.input)[0]
    grads /= (sqrt(mean(square(grads))) + 1e-5)
    iterate = function([model.input], [loss, grads])
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
    img = input_img_data[0]
    return deprocess_image(img)
```

Since the resulting image tensor is a floating-point tensor with values that may not be integers within [0, 255], we need to postprocess this tensor to turn it into a displayable image.

Listing 4: Postprocess the image

```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1
    x += 0.5
    x = np.clip(x, 0, 1)
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

For example, in the next image we can see the patterns learnt by the layer `block1_conv1` of the Xception model.

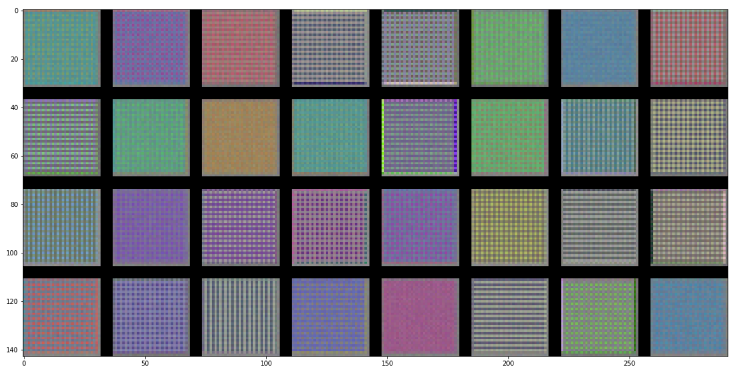


Figure 25: Filter patterns for layer `block1_conv1`

As expected, the features extracted at the first layers are simple building blocks for more complex features that will be extracted in the deeper layers.

Visualizing heatmaps of class activation This is helpful for debugging the decision process of a convnet, particularly in the case of a classification mistake. It allows us to locate specific objects in an image. It consists of producing heatmaps of class activation over input images. A class activation heatmap is a 2D grid of scores associated with a specific output class, computed for every location in any input image, indicating how important each location is with respect to the class under consideration.
Using the library *Keract*, we can easily obtain a visualization of the heatmaps, as seen in Figure 26.

Listing 5: Code to display heatmaps class application

```
img = cv2.imread(path)
resized_img = cv2.resize(img, (img_size,img_size))
preprocessed_img = xception.preprocess_input(np.expand_dims(resized_img
    .copy(), axis=0 ) )
yhat = model.predict(preprocessed_img)

label = decode_predictions(yhat)
label = label[0][0]
print('{}␣({})'.format(label[1], label[2] * 100))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
activations = keract.get_activations(model, preprocessed_img)
first = activations.get('block1_conv1/Relu:0')
keract.display_activations(activations)
```

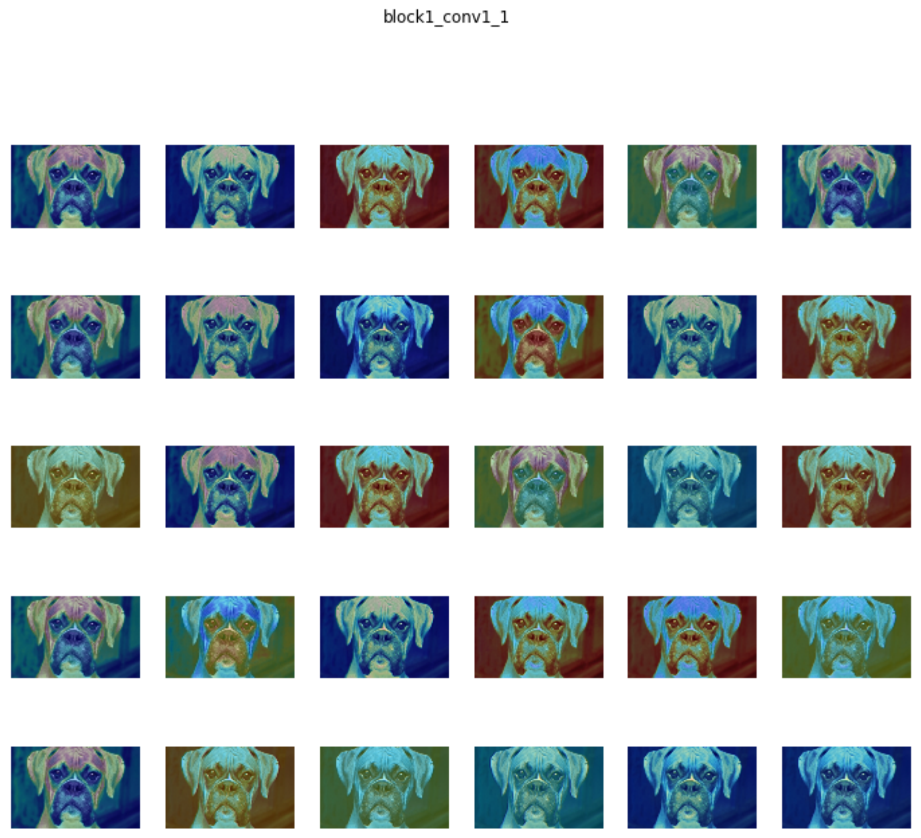
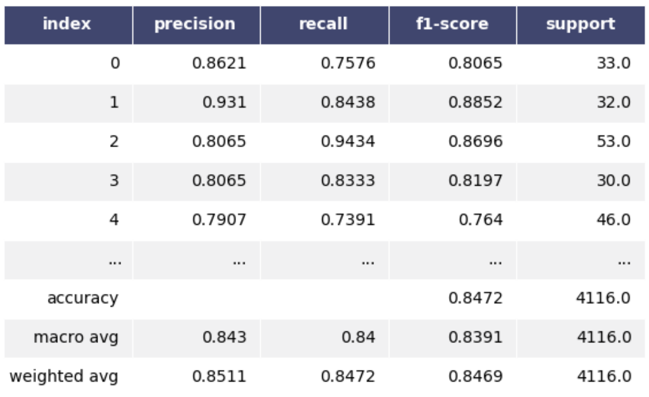


Figure 26: The class activation heatmap for an image.

4.3 Results

The Xception transfer-learning model has a really good performance; in fact, it obtains perfect accuracy for both the two-breeds classification and the five-

breeds classification, reaching 0.9786 for 10-breeds classification. Recall that the ad-hoc simple CNN already performed poorly for the five-breeds test, so using a pretrained convnet seems like a successful choice in small-datasets scenarios like this, due to the highly repurposable nature of the convnets.
Even feeding the model with all the 120 classes in the original dataset, we still obtain an acceptable accuracy of about 0.8472, as shown in the classification report in Figure 27.



index	precision	recall	f1-score	support
0	0.8621	0.7576	0.8065	33.0
1	0.931	0.8438	0.8852	32.0
2	0.8065	0.9434	0.8696	53.0
3	0.8065	0.8333	0.8197	30.0
4	0.7907	0.7391	0.764	46.0
...	...	...	...	...
accuracy			0.8472	4116.0
macro avg	0.843	0.84	0.8391	4116.0
weighted avg	0.8511	0.8472	0.8469	4116.0

Figure 27: Classification report for the Xception model.

5 Conclusion

As we have seen, the transfer learning approach outperformed the training-from-scratch one. This is mainly due to the limited size of the dataset which makes it impossible to build large networks, since there would be too many parameters to learn from few instances, resulting in overfit. Furthermore, transfer learning results in negligible computational costs, while training a deep convnet may take a lot of resources. Nevertheless, the ad-hoc CNN could still be much improved, a few ideas could be:

- design a smart hyper-parameter search to better tune the model, like Bayesian optimization;
- augment the dataset with modified environmental factors (weather, season etc) using Generative Adversarial Networks;
- try online data augmentation which, during the training, replaces the input batch with random modified images, making the model never encounter the same image twice.

The transfer learning performance could also be improved, for example by trying different networks.

References

[1] Data augmentation for deep learning. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[3] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.