Cats and dogs classification using CNN

Statistical learning with high-dimensional data, Fall-23

Axel Eriksson
Gustav Felländer

Project study

December 22, 2023

Contents

1 Introduction

One of the challenges for artificial intelligence is to discern and classify objects in images, a task that humans effortlessly perform. The aim of this project is to answer the fundamental question: Can a machine learn to differentiate between cats and dogs by training on a carefully curated dataset and subsequently apply this acquired knowledge to make accurate predictions on unseen data? To answer this question, we are going to use machine learning and the method of convolutional neural networks (CNN), which is a popular technique for image analysis. We will try two approaches, first using a vanilla CNN, which we build, and secondly, a Residual Neural Network (ResNet) which is a deep CNN using residual connections and is often used for computer vision applications. Lastly, we will compare the differences in performance between them.

2 Theory

2.1 Preprocessing

To solve this task, we gathered a dataset containing labeled images of cats and dogs. Then we prepared the data. First, we went through the dataset of 10,000 images to remove some images where the animals were not in focus. In fig. 3, some of the images that were removed from the dataset are shown. As we can see, there are some weird images, including one of Einstein classified as a dog. Keeping these images could lead to the model learning patterns and features that are not relevant to the target classes. For instance, if the training set contains images of unrelated objects or scenes, the model might start learning features from those images, leading to a less accurate and less specific model. Risk of overfitting, where the model becomes too specialized for the training data and fails to generalize well to new, unseen data. To be able to use the images in our model, they have to be the same size. We did this by transforming all the images to $200x200$ pixels and also maintaining the aspect ratio to avoid distorting the content.

2.2 Model Selection

CNNs are commonly used for this type of task due to their ability to capture spatial hierarchies, which is therefore the model we are using. The key components of a CNN include convolutional layers, pooling layers, and fully connected layers. Convolutional layers are responsible for feature extraction. They apply convolutional filters (kernels) to the input image to detect patterns and features. Each filter slides over the input, performing element-wise multiplication and summing the results to produce feature maps. Multiple filters capture different features, enabling the network to learn hierarchical representations. Pooling layers down-sample the spatial dimensions to avoid overfitting and also to decrease the complexity of the model. Fully connected layers are traditional neural network layers where each neuron is connected to every neuron in the previous layer. We are using each of these layers at least once to build a model optimal for our problem and to extract important information to distinguish between cats and dogs. One of the biggest problems with CNN is overfitting, and to

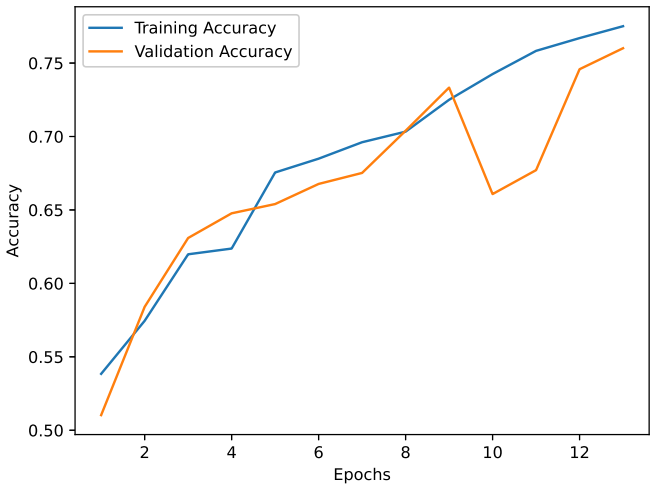


Figure 1: Training and validation accuracy over epochs.

avoid this, we have both used regularizing techniques and dropout layers. The combination of these layers and all the hyperparameters has been chosen from a manual grid search.

2.3 Training and validation

Once the model is finished, we start to train it using the labeled dataset, where the model learns to map input images to their corresponding labels, which in this case are cats or dogs. During training, the model adjusts its internal parameters to minimize the difference between its predictions and the actual labels. Evaluate the model on a separate validation set to ensure it generalizes well to new, unseen data, and then use that information to fine-tune hyperparameters.

2.4 Testing

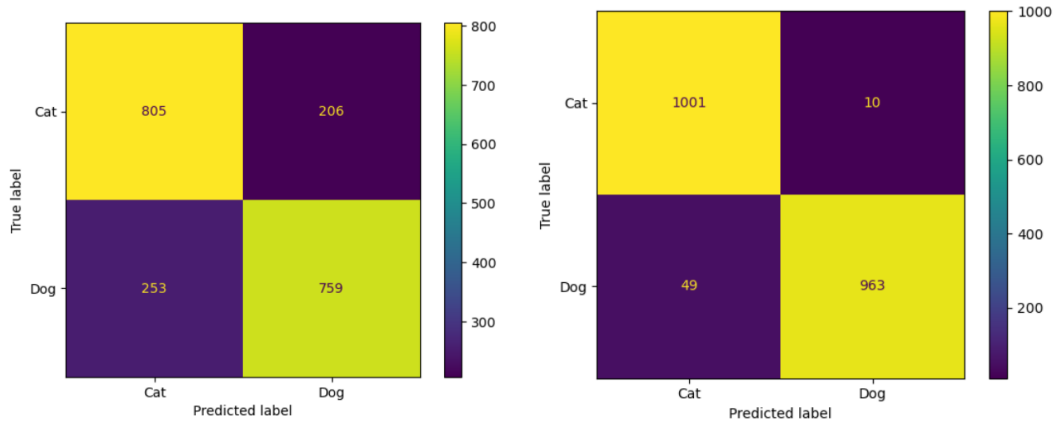
Once we are satisfied with our models, we can run them on the testing dataset. From this, we can extract the results of the models, such as a confusion matrix, accuracy, and recall.

3 Result/Discussion

3.1 Performance of the Vanilla CNN Model

The resulting CNN model, which was designed by us, reached a peak validation accuracy of 76.01%, which was the highest achieved during the epochs of training, as indicated in fig. 1. Also, a confusion matrix of the test results can be seen in fig. 2a, showing a test accuracy of 77.3%.

There, it is also clear that we avoided some overfitting. First, because of the different techniques we applied in order to prevent that, such as using L2-penalty and dropout layers.



(a) (b)

Figure 2: Confusion matrix for evaluation on the previously unseen test set of cats and dogs for the (a) vanilla CNN model and (b) ResNet50 pretrained model.

Also, by designing the architecture of the model to have the trainable weights more balanced throughout the network, we were able to avoid overfitting efficiently. Furthermore, we used the technique of early stopping to avoid going ”to far” and overtraining on the training set of images.

3.2 Performance of the ResNet50 Pretrained Model

Secondly, for the performance of the ResNet model, we achieved after only five epochs of training a validation accuracy of 96.13%, and when evaluating the model on the test set, we obtained the confusion matrix that can be seen in fig. 2b, which shows that the test accuracy was 97.1%.

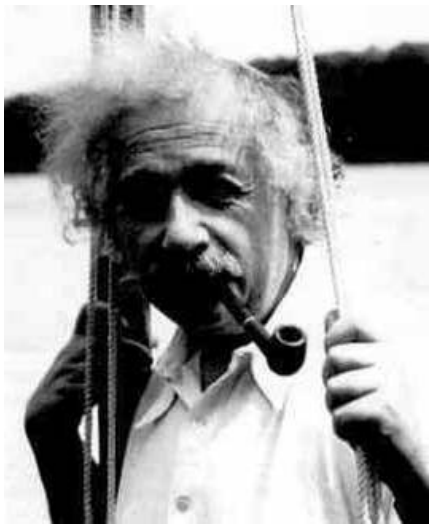
3.3 Model Comparison

Both in terms of validation and test accuracy, it is clear that the pretrained ResNet50 model outperforms our CNN model by comparing the results that we see in the confusion matrices in fig. 2.

The reason why this is happening is ResNet’s superior architecture, which is considerably deeper than the vanilla CNN model and allows for a better and more sophisticated feature extraction of the cats and dogs in the images. Moreover, the residual connections in the model facilitate the propagation of information throughout the network during the forward as well as the backward propagation steps when computing the gradient. Finally, the aim of the vanilla model was to obtain a cheap CNN model with few parameters and still obtain good results, which we argue is the case. However, compared with the ResNet model, with its way deeper architecture and many more trainable weights, the complexity of the captured image patterns is significantly increased, as is its performance in image recognition tasks.

A Appendix

(a) cat



(b) dog

(c) dog

(d) dog

Figure 3: Example of images that was removed from the dataset during the prepossessing

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 32)      896

 batch_normalization (Batch  (None, 198, 198, 32)      128
 Normalization)

 conv2d_1 (Conv2D)           (None, 196, 196, 8)       2312

 conv2d_2 (Conv2D)           (None, 194, 194, 32)      2336

 max_pooling2d (MaxPooling2  (None, 97, 97, 32)        0
 D)

 conv2d_3 (Conv2D)           (None, 95, 95, 8)         2312

 conv2d_4 (Conv2D)           (None, 93, 93, 32)        2336

 max_pooling2d_1 (MaxPoolin  (None, 46, 46, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 44, 44, 8)         2312

 conv2d_6 (Conv2D)           (None, 42, 42, 32)        2336

 max_pooling2d_2 (MaxPoolin  (None, 10, 10, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 8, 8, 4)           1156

 max_pooling2d_3 (MaxPoolin  (None, 4, 4, 4)           0
 g2D)

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 16)                1040

 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 2)                 34

=================================================================
Total params: 17198 (67.18 KB)
Trainable params: 17134 (66.93 KB)
Non-trainable params: 64 (256.00 Byte)
_________________________________________________________________
```

Figure 4: The image shows the architecture that we used in our designed vanilla CNN model.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_1 (Flatten)         (None, 100352)            0

 dense_2 (Dense)             (None, 1024)              102761472

 dropout_1 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 2)                 2050

=================================================================
Total params: 126351234 (481.99 MB)
Trainable params: 102763522 (392.01 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

Figure 5: The image shows the architecture that we used in the pretrained ResNet50 model which we used for transfer learning.