UNIVERSITÀ DEGLI STUDI DI MILANO

Data Science and Economics

F94-155.23.1 Statistical Methods for Machine Learning

CATS VS DOGS

ML Project: Neural Networks

Student Name	**Student ID**
Ruben Popper	963412

Professor:

Nicolò Cesa-Bianchi

Date : 14/01/2023

Contents

1 Introduction **2**

2 Theoretical Framework **2**
2.1 Convolutional Layer . . . 3
2.1.1 Kernel . . . 3
2.1.2 Stride . . . 4
2.1.3 Padding . . . 5
2.2 Pooling Layer . . . 6
2.3 Fully Connected Layer . . . 6
2.4 Activation Functions . . . 7
2.5 Regularization Methods . . . 8

3 Methodology **9**
3.1 Dataset Preparation . . . 10
3.2 Experimental Setup . . . 11

4 Results **12**

5 Conclusion **14**

1 Introduction

Artificial Neural Networks (ANN) are a class of machine learning algorithms inspired by the mechanics of the human brain. Similarly to the functioning of biological brains, ANN are structured as a collection of interconnected nodes called artificial neurons.

The building blocks of ANN are the layers, collections of nodes that operate together to process, transform and extract patterns from the data by passing information among themselves, according to a hierarchical structure. The way layers are disposed and signal to each other define what is known as the architecture of an artificial neural network.

The architecture can be thought as the backbone of the model which, according to a predefined set of rules, determines how the network learns the function that maps input data to output. Depending on the task at hand, the function to be learned can be very difficult, and the ANN's architecture can be enriched with additional components that better help uncover patterns in the data.

In this report, we analyzed several configurations of neural networks trained to perform image classification. Image classification is a computer vision task (the field of deep learning that aims at extracting valuable information from image data) whereby, given a pre-defined set of tags or classes, the model assign the most appropriate tag to the image. When working with image data, artificial neural network rely on the mechanics of a specific type of layer called convolutional layer, hence we refer to this class artificial networks as convolutional neural networks, or CNN.

The goal of this project is to present the building blocks of CNNs and to understand how different architecture setups affect the model results when performing binary classification. Specifically, we are interested in understanding how the number of convolutional and fully connected layers and the introduction of the dropout regularization strategy affects the network performancex.

We train ten different CNNs on the *Cats vs Dogs* dataset (Elson et al. 2007), a collection of human annotated images of shelter cats and dogs provided to Microsoft for research purposes. In particular, we work with a subset of the data made public by Kaggle[1].

2 Theoretical Framework

In this section we introduce the main components of CNN architecture. As mentioned earlier, CNNs are a specific class of neural networks characterized by an operation called "convolution", which reduces the dimensionality with limited information loss and captures the spatial structure within them. These types of model have been proved to work particularly well with time series (Wang et al. 2019), signal (Kiranyaz et al. 2019), audio and image data (Sharma et al. 2018), whose main characteristics is the presence of a spatial, or temporal structure, at least as perceived by the human brain.

By leveraging the principles of linear algebra, like matrix multiplication, the convolution operation considers not only the distance between measuraments, but also the direction of one with respect to the others. Therefore, similarly to how biological neurons allow humans to recognize depth, shapes, objects through

[1]`https://www.kaggle.com/c/dogs-vs-cats`

their sight, convolutional layers allow CNN to locate within the input data the presence of collection of edges, shapes, and collection of shapes, objects. Indeed, CNNs have found widespread application in computer vision tasks such as image classification, image segmentation, and object detection.

In the context of image classification, the input data fed to the model is represented by an $h \times w$ matrix, where h is the height and w the width of an image. Each matrix cell stores the value of a pixel, whose most common representation is RGB, a tuple of three digits storing the red, green and blue intensity on a scale from 0 to 255. Because images have a bidimensional representation, we focus the following discussion on the functioning of 2DConvolutional layers, albeit the concept can be generalized for the unidimensional and multidimensional case as well. Below we introduce the core elemets of a CNN, namely, the convolutional layer, the pooling layer and the fully connected layer, with a brief discussion on activation functions and regularization methods.

2.1 Convolutional Layer

The convolutional layer is the building block of a CNN. The number of convolutional layers through which the data passes determines the depth of the network. In general, as for the hidden layers of standard ANN design, convolutional layers are organized according to a sequential structure. The first convolutional layer takes as input the raw image data, while subsequent layers take as input the output of previous layers. In the context of image processing, the convolutional layer acts as a feature extractor. The raw input data are thus passed through the layer which generates a lower dimensional yet informative representation of the input data. In particular, low level features such as edges, dots and lines are identified and represented in the first convolutional layers of the newtork (Ahmed and Karim 2020). As the network grows deeper, higher level features such as shapes and object are also extracted. Below we illustrate the mechanics of a convolutional layer by presenting its core elements: the kernel, the stride, and the padding.

2.1.1 Kernel

A kernel (also known as convolution matrix) is an $n \times n$ square matrix describing a filter that is applied to the input image by means of matrix multiplication. In Figure1 below, an example of convolution operation between the input image and the the kernel. Matrices I, K, and O are the input, kernel and output matrix respectively.

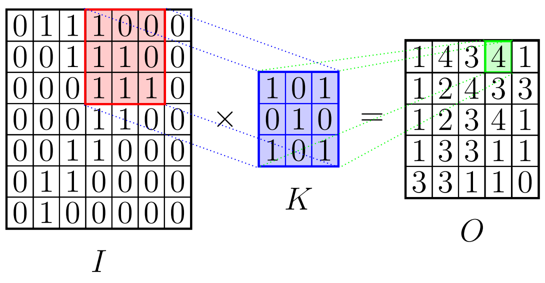


Figure 1: Convolution - Kernel 3 ×3

The name convolution comes from the iterative procedure that is applied. In fact, the kernel slides through the input with a left to right, top to bottom approach. Given a kernel of size $n \times n$ and single equal-sized portion of the input, an element wise multiplication between cells a^I_{ij} and a^K_{ij} is computed. The values obtained are summed to produce a "filtered pixel value". By sequentially applying the kernel to several regions of the input we obtained a convolved feature map, or filtered image O. The feature map can thus be interpreted as a condensed representation that captures the spatial patterns of the input.

Convolutional layers are built according to the input parameters; the image height and width, w & h and the number of input channels n^I. In the case of RGB coded image data $n^I = 3$, one for each element of the RGB tuple. The input of the layer consists of one $w \times h$ matrix per input channel n^I_i. Insted, the output channels n^O of a convolutional layer are determined by the number of filters applied n^K, such that $n^O = n^K \div n^I$. In particular, each filter K_i, when applied to input matrix I generates a feature map. The matrix of an output channel is computed as the sum of the convolved feature maps obtained from kernel K_i's application to each input channel. Therefore, the output of a convolutional layer consists in n^O lower dimensional matrices.

The entries of each filter are the trainable parameters of the CNN. During training, the values of the kernel are updated in the attempt to minimize the loss function governing the learning algorithm.

2.1.2 Stride

The way input data are filtered by the kernel is defined by the stride. Starting from the top left corner, the stride represents the number of columns to the right, or rows below, the kernel window slides through the image. Figure2 below illustrates an example where $stride = 1$ is used

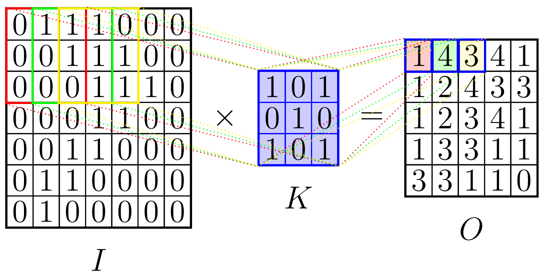


Figure 2: Stride = 1

The output cell represents the cross correlation between the kernel and the window on the image, both of size $n \times n$. By sliding the window over the input data, the information of specific regions in the input image are summarized and represented in the output matrix. The stride determines the pace at which the kernel slides through the image. The higher the pace, the less cross correlations are computed between the kernel and the input. As a result, the bigger the stride, the higher the dimension reduction performed. Specifically, for a squared input matrix of side i, a kernel of side k with stride value s, the side of the output matrix m is given by:

$$m = \left(\frac{i - k}{s} + 1 \right) \tag{1}$$

It is important to remark that, independently from the stride value, the location of the output matrix cells reflects the original position of the input matrix regions used to compute them. Therefore, the filtered image maintains the spatial information of the original image.

2.1.3 Padding

From Figure2, it can be noticed how the kernel sliding window captures cells located at the center of the input matrix more frequently. On the contrary, cells that are closer to the border are included less times in the output computation. As a result, the convoluted map, by construction, gives more weight to the information contained in the center of the image. Additionally, equation (1) shows that the filtered image has always a smaller dimension then the input one. Thus, the image shrinks every time a convolution operation is performed. This precludes networks from growing deeper as they are subject to an upper ceiling to the number of convolutional operation that can be performed before the image is reduced to nothing.

Padding partially solves this problem by adding one or more layers of zero entries to each matrix side. Figure 3 below illustrate the output of a convolution operation after *padding* $= 1$ has been applied to our example matrix.

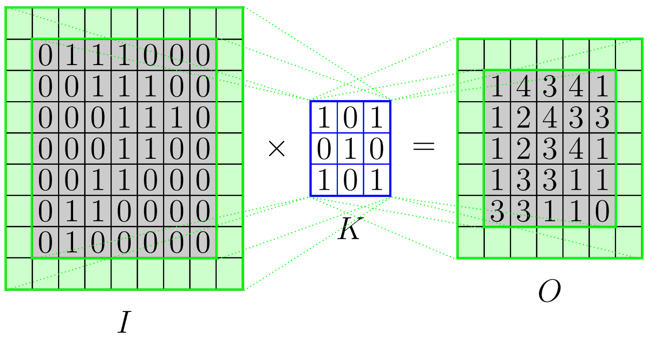


Figure 3: Padding = 1

From 3, we can see how the cells at the border of the original image (the grey area) have been brought towards the middle of the padded image. For instance, while in Figure2, the cells at the input corners entered the dot product operation with the kernel only once, they now entered the computation in four filtered pixel values.

Beside increasing the contribution of the outer cells of the input, the application of padding has brought the output size from 5×5 to 7×7. In our example the input image has maintained its original dimension despite going through a convolution. For squared inputs of side i with padding p, the convolution operation through a kernel of side k and stride s generates an output of size m where:

$$m = \left(\frac{i - k + 2p}{s} + 1 \right) \tag{2}$$

Therefore, the output size gets smaller as less padding is used and when a larger kernel or stride are applied.

2.2 Pooling Layer

A pooling layer is another type of layer commonly used in convolutional neural networks (CNNs). Its main purpose is to reduce the spatial dimensions of the input feature maps by down-sampling. A pooling operation is applied to the input feature maps, which involves taking the maximum or average value of a small window of the input and using that value to replace the entire window. The information contained in a region of the input is thus summarized in a smaller region of the output map.

Similarly to the convolutional layers, the pooling operation is applied with a kernel and a stride. In this case, the kernel defines the window size while the stride the distance between two subsequent windows. Figure 4 below illustrates the mechanics of the pooling operation.

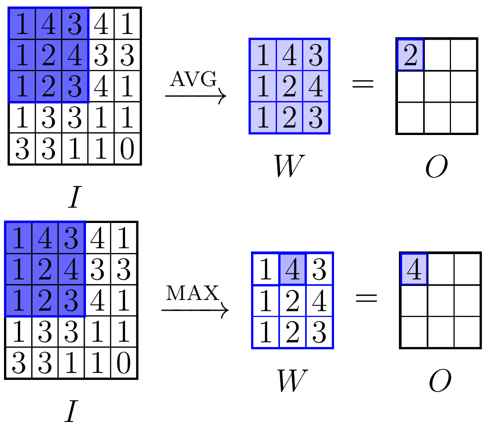


Figure 4: Pooling Layer - 3 ×3

The pooling operation aggregates the feature by taking the max or average if a window. This dimension reduction obtained reduces the number of parameters of the netowork, thus preventing overfitting (Lee et al. 2015). Indeed, by down-sampling the input feature maps, the network becomes more robust to changes in the images. Morover, by reducing the number of computations required, the network becomes more efficient and easy to train.

2.3 Fully Connected Layer

Fully connected layers (FC) are the core elements of standard ANNs. In the context of CNNs, fully connected layers take as input the reduced feature maps generated by the convolutional and pooling layers to serve a dual purpose: dimensionality reduction and classification (LeCun et al. 2015).

Fully connected layers map input data to output by performing an element-wise multiplication between input vectors and a weight matrix. Specifically, assuming an input vector of n, and a desired output vector size of m, the weight matrix will have size $n \times m$.

As shown in Figure 5, the number of columns in the weight matrix determines the size of the output vector, whose elements are the result of vector multiplication. The weight matrix represents the trainable parameters of the layer. As such, during training, the elements of the weights matrix are updated based on the loss function governing the learning algorithms.

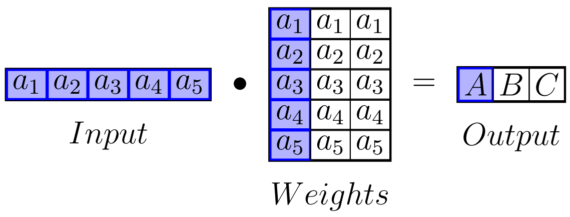


Figure 5: Fully Connected Layer

Besides performing dimensionality reduction, in the context of classification tasks, fully connected layer acts as predictor of the network. In fact, by choosing an output size for the last fully connected layer equal to the number of classes present in our learning domain, the weights trained according to the loss function will assign to the input its class group.

To summarize, convolutional layers extract salient features from the input, pooling layers help reducing the number of trainable parameters in the network by lowering the output dimension, and fully connected layers perform classification.

2.4 Activation Functions

So far we have introduced the different roles of convolutional neural networks' layers. Each layer applies simple linear algebra operations to map input data to output. Although each layer imposes a different way of performing such computations, the convolutional, pooling and fully connected operations are all types of linear operations. In the real world, he relationship between the input and output is often non-linear, and a linear model may not be able to accurately capture this relationship. Since linear models can only learn linear relationships between the input and the output, introducing non-linearity allows the networks to learn more complex relationships, such as non-linear decision boundaries and non-linear interactions between features. This is why we introduce the activation functions. Non-linerity is introduced in neural networks by activating the output neurons of a layer via element-wise application of these functions.

Below we introduce the most commonly used activation functions applied to the output neurons of the aforementioned layers:

- ReLU (Rectified Linear Unit): $f(x) = max(0, x)$

  For any input x, greater than 0, the output will be equal to x. For any input x is less than or equal to0, the output will be 0. One of the advantages of ReLU is its computational efficiency and its properties that help reduce the vanishing gradient problem.

- Leaky ReLU: $f(x) = max(\alpha x, x)$

  Leaky ReLU is an extension of ReLU that allows small negative values to pass through thanks to the α coefficient introduced.

- Sigmoid: $f(x) = \frac{1}{(1+e^{-x})}$

  Given input x and e, the base of the natural logarithm, the sigmoid function maps the input values between 0 and 1. It is used less frequently in CNNs due to the vanishing gradient problem. Nevertheless, it is very applicable in output layers performing binary classification tasks, where above a sigmoid output value of 0.5 label 1 is predicted, and label 0 otherwise

- Tanh (hyperbolic tangent): $f(x) = \frac{2}{1+e^{-2x}} - 1$

  The tanh function maps the input value x between -1 and 1. As for the sigmoid function, it is used less frequently in CNNs due to the vanishing gradient problem.

- Softmax: $f(z_i) = \frac{e^{z_i}}{\sum e^{z_j}}$

  Through the softmax activation function, where z is the input vector, z_i is the i^{th} element of z, and $e$is the base of the natural logarithm, the input values are squashed into a probability distribution between 0 and 1, such that the sum of all the output values is equal to 1. This makes it useful for multi-class classification problems, where the goal is to assign a probability of the input sample belonging to a certain class. Indeed, the softmax activation is used when the output layer is a fully connected layer. Specifically, it's not generally used in the middle of the network, as its main function is to convert the output into a probability distribution over the classes.

Introducing non-linearity to a neural network is important for several reasons. First, the network can learn more complex representations of the input data (Schmidt-Hieber 2020). Non-linearity allows the network to model more complex and realistic functions that better represent real-world problems, where the relationship between the input and output is often non-linear, and a linear model may not be able to accurately capture their relationship. Second, non-linearity helps to mitigate the vanishing gradient problem (Hochreiter 1998). In neural networks, parameters update is performed via backpropagation, which finds the gradients of the error with respect to the parameters layer by layer, starting from the output layer and proceeding backward. By the chain rule, the derivatives of each layer are multiplied down the network. This causes the gradient to decrease exponentially as it is backpropagated. A small gradient over the error causes the weights not to be updated effectively in each training session. This is a concerning problem as it leads to overall inaccuracy of the network, as initial layers are crucial in identiying the core elements of the input data. Non-linear activation functions, such as ReLU, can help to mitigate this problem by introducing non-linearity to the model, allowing the gradients to flow more easily through the layers. For this reasons, the introduction of non-linearity helps the network learn more accurate models.

2.5 Regularization Methods

In the context of neural networks, regularization methods are techniques used to improve the performances and stability of the model. For any learning algorithm, the complexity of the model defined in terms of the number of trainable parameters, must be consistent with the amount of data available. Complex models trained on few amounts of data will cause the parameters to adapt too well to the training data, thus limiting their generalization capabilities due to the well-known phenomenon of overfitting (Jing and Guanci 2018).

Regularization techniques can be applied to different elements of the network. Although governed by different logics, the general principle is the same: they either reduce the number of parameters or the variability among them. Below is an overview of the most commonly used approaches at regularization, listed according to the target of their application.

- Input Data:
  - Data augmentation: the synthetic creation of new training samples by applying random transformations to the existing data. By enlarging the training set, more parameters can be accurately trained, reducing the risk of overfitting.
- Loss Functions and Weights Update:
  - $L1$ and $L2$ regularization: the addition of a term to the loss function that penalizes large weights according to a coefficient lambda λ that controls the strength of the regularization. While L1 regularization adds the absolute value of the weights to the loss function, L2 regularization adds the square of the weights to the loss function.
  - Weight decay: the addition of L2 regularization in the model. It differs from the aforementioned technique as it is implemented directly in the weights update rule rather than the objective fucntion of the network.
  - Early Stopping: the automatic interruption of the training process determined by the speed of change in performances. Training is interrupted when the training loss reaches a plateau and performances on the validation set start to degrade,
- Layers:
  - Dropout: randomly setting to zero a certain percentage of the activations of a layer during training. The risk of overfitting is reduced as dropout limits the dependence of the activations of one layer on the activations of the previous layer. Specifically, the network is forced to learn multiple independent representations, making the model more robust to the changes in the input data The network generalization capabilities are thus improved.
  - Batch Normalization: exploiting the first and second moments of the activations of a layer (mean and standard deviation) to normalize the neurons. The internal covariate shift, defined as the change in the distribution of the inputs to a layer caused by the change in the parameters of the previous layer is thus reduced. Normalization renders the network training process less sensitive to the weights initialization. Moreover, by reducing the span of activation values, it helps the network to converge faster. Lastly, batch normalization provides zero centered inputs to the activation functions. This allows the gradients to be much higher and stable during backpropagation, thus helping to solve the vanishing gradient problem.

3 Methodology

As mentioned in the introduction, the primary goal of the project is to analyze differences in performances of different CNNs' architectures setups. Training of deep neural networks usually requires greater amount of computational resourcs when compared to classic supervised learning algorithms. In particular, GPU accelerated runtimes provide a faster solution when developing CNNs. For this reason, the python code executing the various stages of the project is ran on GoogleCollab[2], which provides 12GB GPUs environments.

The first step in the deployment of a deep neural network model consists in preprocessing the dataset in order to make it suitable for the task at hand. Second, the network structure and its components have to be programmed to build the final architecture. In the following subsections we illustrate the data preparation strategy and the experimental setup.

3.1 Dataset Preparation

The initial *Cats&andDogs* dataset consists of 25000 imagese, with 12500 samples from each class. In order for images to be processed, we convert all files from JPG extension to RGB, where each pixel is represented by a 3-tuple indicating the three primary colors (red, green and blue) intensity on a scale from 0 to 255.

The first stage of preprocessing eliminates corrupt image files. In particular, we found 2 cases of image files with incomplete Exif data, and 1588 corrupted image files that standard image processing python libraries like OpenCV and PIL fail to read.

In the second stage, we inspect the quality the image files by analyzing their number of pixels. Intuitively, the lower the image quality the less information the RGB representatin carries. In the attempt to avoid noisy data in our training set, we discarded images with exceptionally low pixel counts. Figure 6 below shows the distribution of pixel count in the dataset

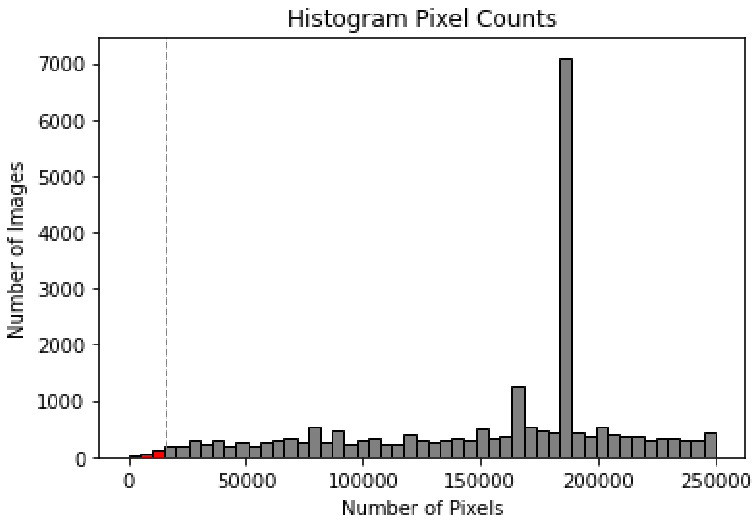


Figure 6: Pixel Count Distribution

The last step in the dataset preparation is the creation of the train, validation and test sets. Hence, we proceed by creating a test set with 10% images. The remaining samples are split according to a 80-20% rule. The removal of some images from the initial dataset raises a concern regarding the possibility of class imbalance. In classification problems, huge class imbalances could dampen the accuracy of the model, which would be accurate in the prediction of the majority class only. Thus, we proceed by analyzing the class distributions in the three

[2]`https://colab.research.google.com/`

subsets. As shown in Figure 7, although the cats category counts few more observatin than the other, overall we conclude that the dataset remains balanced despite the removal of >1500 images.

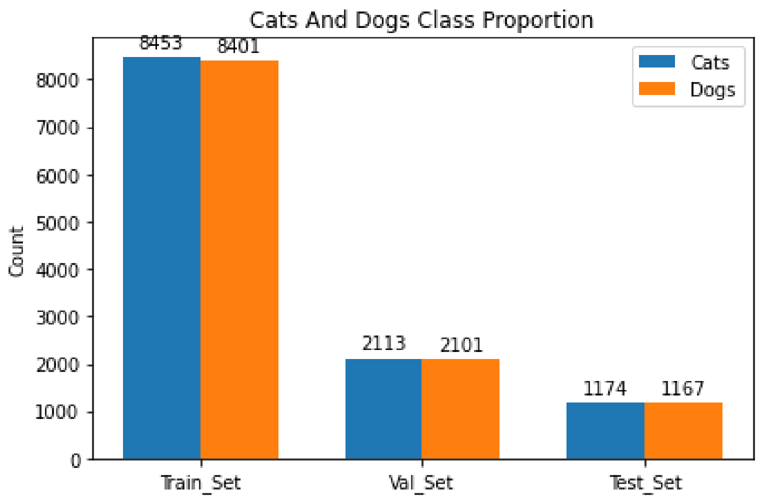


Figure 7: Train-Validation-Test Sets: Class Distribution

3.2 Experimental Setup

As anticipated in the introduction, the goal of the project is to analyze performance differences based on the number of Conv2D and FC layers and the utilization of the dropout regularization strategy in the networks. To do so, a scalable strategy is designed to grow deeper network keeping some of the network parameters constant. In this way, changes in performances can be attributed only to variations in the aforementioned network components.

First, the datasets are transformed into tensors, the standard input of Python deep learning libraries. Data augmentation is applied in order to increase the number training samples and reduce the risk of overfitting. This technique is applied randomly to the training set only. The image transformations introduced consist in rescaling, flipping, shearing, shifting, and zooming. Figure 8 below shows an example of the transformations applied.

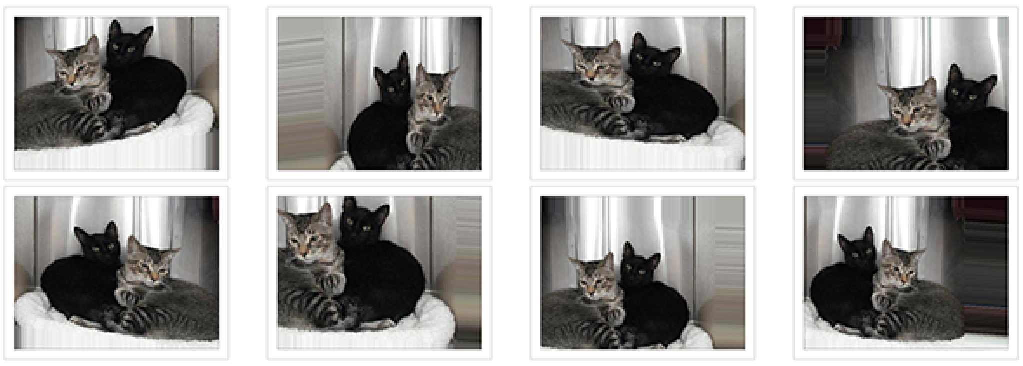


Figure 8: Data Agumentation Transformations

Second, for all models we fix the main training parameters as specified below. Also, we include earlystopping and learning rate reduction if the metrics do not improve in five subsequent epochs.

- Input Size: *128 × 128*
- Batch Size: *32*
- Epochs: *30*
- Kernel Size: *3 × 3*
- Stride: *1*
- Pooling Size: *2 × 2*
- Loss Function: *categorical cross-entropy*
- Activation Function: *ReLU*
- Optimizer: *RMSprop*

Next, we consider all combinations of models with 1 to 4 Conv2D and FC layers, excluding those where the number of FC layers exceeds that of Conv2D. Each configuration is trained with and without dropout. For Conv2D layers, the number of filters applied, which is initially set to 32, is doubled at each additional layer, and dropout is applied to 25% of the activations. For FC layers, the output size is doubled starting from the second last layer with size 512 proceeding backward and dropout is applied to 50% of the activations.

4 Results

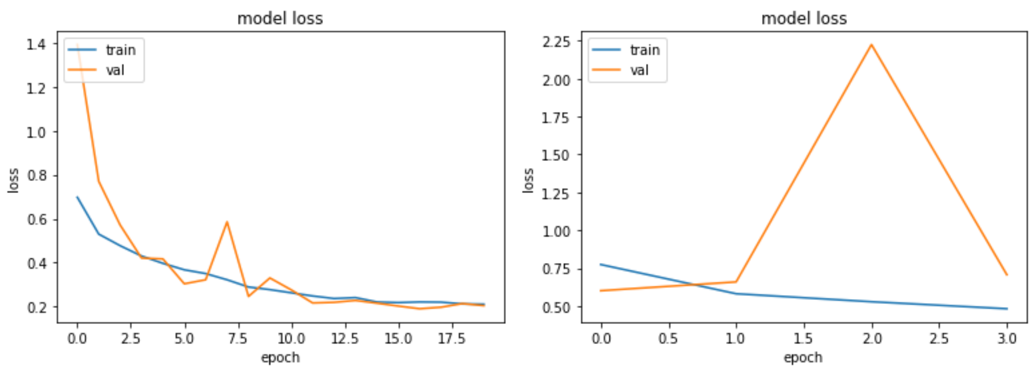
We considered the 10 configurations of CNN according to the strategy devised in the previous section. For each architecture, test loss and accuracy are computed with and without dropout.

Performance	**Architecture**		**No Dropout**		**Dropout**	
	Conv2D	FC	Loss	Accuracy	Loss	Accuracy
Model1	1	1	*0.612*	*0.642*	0.627	0.629
Model2	2	1	0.601	0.695	*0.518*	*0.751*
Model3	2	2	*0.287*	*0.877*	0.327	0.860
Model4	3	1	*0.236*	*0.911*	0.531	0.741
Model5	3	2	*0.216*	*0.914*	0.356	0.839
Model6	3	3	*0.156*	*0.936*	0.603	0.667
Model7	4	1	*0.213*	*0.916*	0.284	0.886
Model8	4	2	**0.090**	**0.966**	0.150	0.946
Model9	4	3	*0.151*	*0.942*	0.200	0.917
Model10	4	4	*0.187*	*0.918*	0.634	0.650

Table 1: Results: architecture designs expressed in terms on number of Conv2D and FC layers, presence or absence of dropout. Performances evaluated on test dataset using zero-one loss

Table 1 above reports the experiments results. The worst architecture generated is represented by Model 1. A single Conv2D layer fails to capture the discriminant image features necessary for making accurate predictions. Instead, Model 8 configuration achieves the best performances in terms of loss and accuracy (>0.96).

With the exception of Model 9 and 2, the application of dropout regularization to both convolutional and fully connected layers dampens the predictive power of the network. There are two main explanation for this phenomenon: first, random dropout of neurons may cause the network weights to generalize poorly on test data when the size of the training set is too small relative to the number of parameters to be trained; second, introducing dropout in the final (FC) layers forces the model to generalize, preventing the network from learning useful representations of the input data. This phenomena leadas to increased overfitting, as shown in the plots reported in Figure 9 below, where validation loss departs from the training loss as dropout is included.



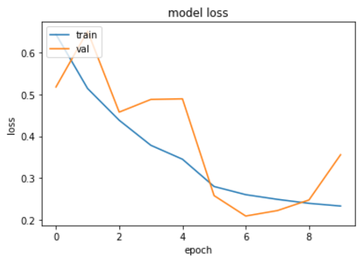
(a) (b)

Figure 9: Loss plots for 3 Conv2D - 3 FC layers with (a) and without (b) dropout

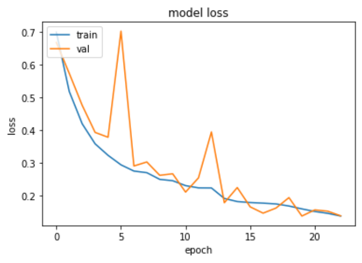
When looking at performance differences determined by changes in Conv2D layers, we consider together the following models: 1-4-7, 3-5-8, 6-9. Notice how an extra convolutional layer consistently improves the network performances. This indicates that feature maps generated by an additional pass through the layers' filters become better representations of the information contained in the input data.

On the other hand, when looking at the effect of extra FC layers on the network, we conclude that, fixing the number of Conv2D layers, network performances are steadily improved when passing the feature maps obtained by the convolutional layers through additional FC layers. Besides improved accuracy, a higher number of FC layers seems to better generalize the convolution maps information on test data. Figure 10 provides a visual evidence for the ability of extra FC layers of reducing overfitting and improving the accuracy of the network predictions in models 7-8-9.

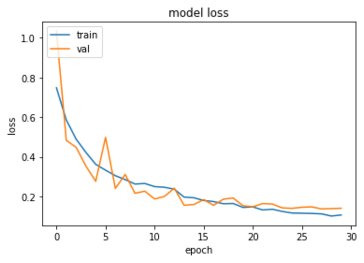
Nonetheless, from Table 1 we conclude that the main performance improvements arise from the additional Conv2D layers. Moreover, by looking at the detailed summaries of the models, an extra Conv2D layer guarantees over 17% performance improvement compared to an extra FC layer with only one third of the trainable parameters of the second.



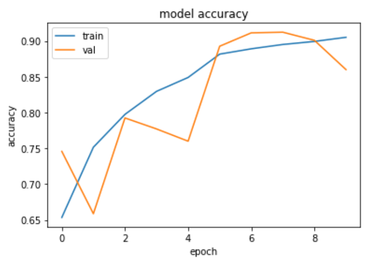
(a)



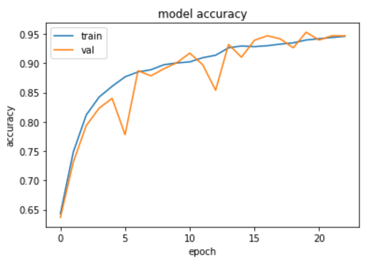
(b)



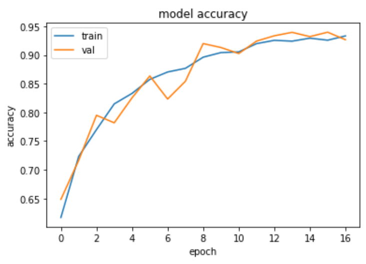
(c)



(d)



(e)



(f)

Figure 10: Performance comparison based on the number of FC layers (no dropout)

For this reason, when computational efficiency and resource availability is a concern, limiting the number of FC layers represent the optimal solution, as slight performance improvements are obtained at the cost of a significant increase in training time.

5 Conclusion

We have designed an experimental strategy to understand how different components of CNNs, namely the number of Conv2D layers, the number of FC layers and the implementation of dropout regularization strategy affect the network accuracy and risk of overfitting. We trained ten different model configurations on the *Cats vs Dogs* dataset after a careful preprocessing phase. We computed our performance measures on 10% of the images, which were excluded from training thus previously unseen from the model.

The dropout regularization techniques failed to reduce the risk of overfitting. Limited availability of training samples and the utilization of the techniques in the final layers were the main explanation of the increased overfitting of the network.

The representations of input data become better and better as more convolutional layers are added. Also, the generalization capabilities of the network are improved by the forward passes through extra FC layers. However, when considering architecture designs outside the scope of the experiment, networks with fewer FC layers may be preferred for they are easy faster to train. Indeed, most of the permance improvements arise from the extra Conv2D layers rather than the mapping to output classes performed by FC layers.

References

W. S. Ahmed and A. a. A. Karim. The impact of filter size and number of filters on classification accuracy in cnn. In *2020 International Conference on Computer Science and Software Engineering (CSASE)*, pages 88–93, 2020. doi: 10.1109/CSASE48920.2020.9142089.

J. Elson, J. J. Douceur, J. Howell, and J. Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007.

S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

Y. Jing and Y. Guanci. Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer. *Algorithms*, 11:28, 03 2018. doi: 10.3390/a11030028.

S. Kiranyaz, T. Ince, O. Abdeljaber, O. Avci, and M. Gabbouj. 1-d convolutional neural networks for signal processing applications. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8360–8364, 2019. doi: 10.1109/ICASSP.2019.8682194.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree, 2015. URL `https://arxiv.org/abs/1509.08985`.

J. Schmidt-Hieber. Nonparametric regression using deep neural networks with ReLU activation function. *The Annals of Statistics*, 48(4):1875 – 1897, 2020. doi: 10.1214/19-AOS1875. URL `https://doi.org/10.1214/19-AOS1875`.

N. Sharma, V. Jain, and A. Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.05.198. International Conference on Computational Intelligence and Data Science.

K. Wang, K. Li, L. Zhou, Y. Hu, Z. Cheng, J. Liu, and C. Chen. Multiple convolutional neural networks for multivariate time series prediction. *Neurocomputing*, 360:107–119, 2019. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.05.023.