Convolutional Neural Network to classify Cats and Dogs

Sara Paolini - 08400A

May 28, 2023

Abstract

This project focuses on the study of Convolutional Neural Networks (CNN) for the classification of images depicting Cats and Dogs. The analysis begins by introducing the fundamental theory behind Convolutional Neural Networks to provides a basis for understanding the inner workings and principles of CNNs.Next, the study delves into the impact of regularization methods, such as Data Augmentation and Dropout, on mitigating the effects of overfitting. Additionally, the study investigates the influence of increasing the network's depth and the number of filters on the overall performance of the model. Finally, the analysis addresses the evaluation of the best-performing model's risk using a 5-fold Cross-Validation procedure.

1 Convolutional Neural Network Theory

Neural Networks (NNs) encompass a diverse range of predictive models suitable for different tasks, including categorization, prediction, and regression. Various types of NNs exist, such as Feed Forward Neural Networks, Multilayer Perceptrons, and Convolutional Neural Networks. Among these, the Feed Forward Neural Network represents the simplest form of an artificial neural network. Its structure forms an acyclic graph, facilitating unidirectional information flow—from the input nodes through any hidden nodes and finally to the output nodes.

Inspired by the intricate network of neurons in the human brain, the computational structure of a Feed Forward Neural Network consists of interconnected nodes (or "neurons") and edges (or "links"). Each neuron receives input from connected neurons via incoming edges, where the inputs are weighted, summed, and then passed through an activation function. The activation function plays a crucial role in transforming the output value of a neuron into a more meaningful form for the neural network's operations. A bias term is commonly added before applying the activation function and it allows the shifting of the activation function by incorporating a constant value (the bias) to the input. By including a bias, the network gains the ability to adjust the activation threshold and enhance its learning capabilities. It is possible to see an example in Figure 1

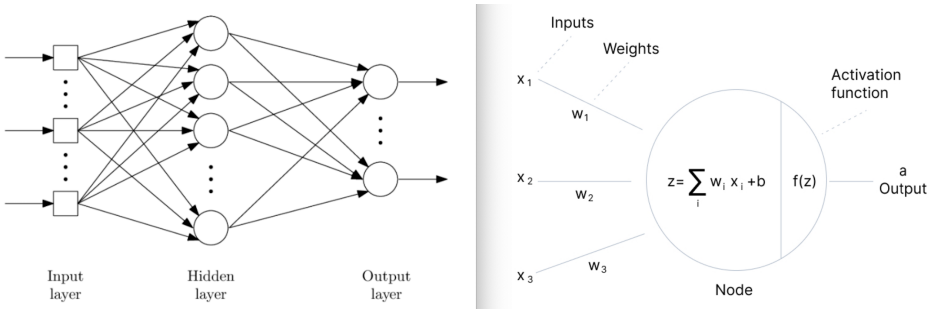

Figure 1: One-Hidden-Layer FNN and Structure of activation Function

The Convolutional Neural Network (CNNs) it's distinguished from other neural networks by their superior performance with image classification. The identification of patterns and features is possible thanks to the ability of CNNs to keep important information about the data analyzed.

This is done through its special architecture and tends to increase with each layer.Indeed other convolution layer can follow the initial convolution layer and when it happens the structure of the CNN

can become hierarchical. The hierarchical learning process allows the network to gradually assemble more sophisticated and abstract representations of the input data.

In the initial layers, the network primarily focuses on learning basic features such as colors and edges. As the image data propagates through the deeper layers of the CNN, the algorithm becomes capable of recognizing more complex elements or shapes associated with the object of interest for more accurate object recognition and classification enables the network to ultimately achieve the task of identifying the intended object. As an example, we're trying to determine if an image contains a dog. You can think of the dog as a sum of parts since it is comprised of a paws, tail, ears, muzzle,... Each individual part of the dog makes up a lower-level pattern in the neural network, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.

1.1 Convolutional Neural Network Architecture

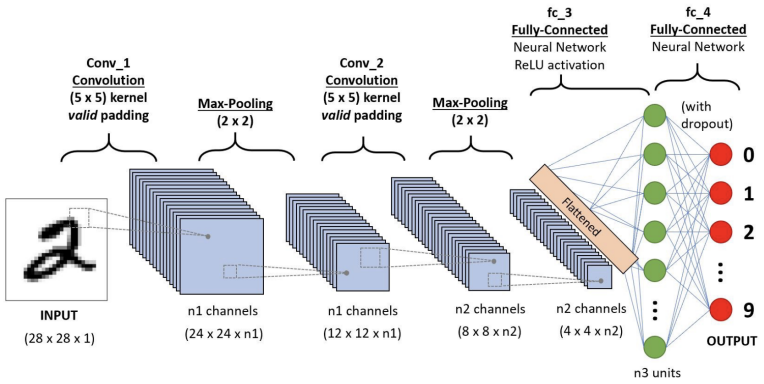

Figure 2: Convolution Neural Network Architecture

1.2 Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. A Convolution is a mathematical operation that describes how a function, typically represented by a filter and in this case denoted as g, modifies another function, the input function denoted as f, to produce a third function c.

$$c = (f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

It involves integrating the two functions, where the filter function, is shifted by a parameter τ across the principal function f at each point t and that produce a Convolutional function c.

The Convolutional Layer requires a few components, which are input data, a filter, and padding.

1.2.1 Input

Input: In the present analysis, the input consists of color images, which can be represented as three-dimensional matrices composed of pixels. Specifically, these images possess three dimensions: height, width, and depth, which correspond to the RGB (Red, Green, Blue) channels in an image. Each channel represents an independent array of color values, and when combined, they produce a composite array that results in a colored image. Figure 3 provides a visual representation of an RGB image and illustrates how it interacts with convolutional filters.

1.2.2 Filters and kernel size

Filters, also referred to as output channels, represent the number of distinct feature maps produced after the convolution operation. These feature maps capture specific patterns and characteristics within

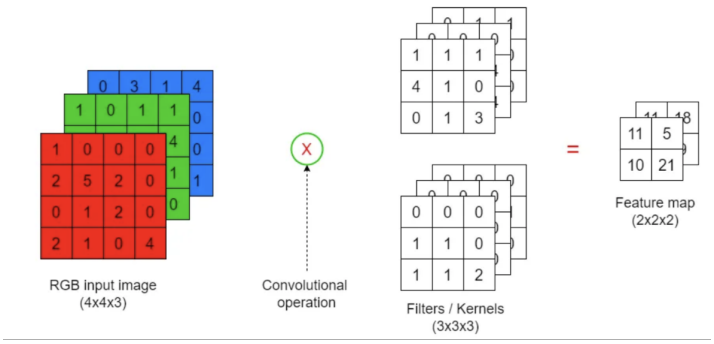

Figure 3: Example of RGB image rappresentation and covolutional operation

the input data. In a CNN, the number of filters in a convolutional layer can vary and commonly takes values such as 16, 32, 64, or 128, depending on the complexity and depth required by the network. On the other hand, the kernel represents the size of the convolution filter used to perform the convolution operation on the input data. The kernel is typically represented as a square matrix, such as 1x1, 3x3, or 5x5. In the case of a 3x3 matrix, the kernel moves across the input data, usually by a stride of one pixel at a time, scanning from left to right and top to bottom. As the kernel traverses the input data, it performs a dot product operation with the corresponding sub-region of the input, generating an output matrix of dot products.

1.2.3 Padding

Padding is a technique to address the border problem that occur during convolution operations. Padding involves adding extra pixels around the borders of an image before applying the convolution (Figure 4). The border problem arises because when a kernel is applied to pixels near the edges of an image, there are fewer neighboring pixels available for the kernel to operate on. As a result, the information at the borders can be lost or distorted, so to ensures that each pixel in the input image has an equal opportunity to contribute to the feature map, padding is introduced. In this analysis I used *padding = SAME"* such that input image or feature map is padded with zeros (zero-padding) and the layer's outputs will have the same spatial dimensions as its inputs. In other words, the spatial size of the input is preserved. [1]

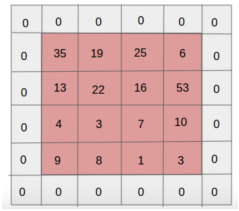

Figure 4: Zero-Padding

1.3 Activation Function

After each convolution operation, a CNN applies a Activation Functions which purpose is to determine whether a neuron should be activated or or not. By applying a mathematical operation, the activation function transforms the pre-activation value into an output value, which is then passed to the next layer or used as the final output of the network. It introduces non-linearity into the network, allowing it to learn complex relationships between the input and output values. Without activation functions, the network would simply be a linear combination of linear operations.

There are several types of activation functions used in neural networks, including the *Sigmoid* , *Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh),Leaky ReLU, ect.* Each activation function

[1] Reference:https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

has its own characteristics and influences the network's learning behavior, convergence speed, and ability to handle different types of data. (Figure 5)

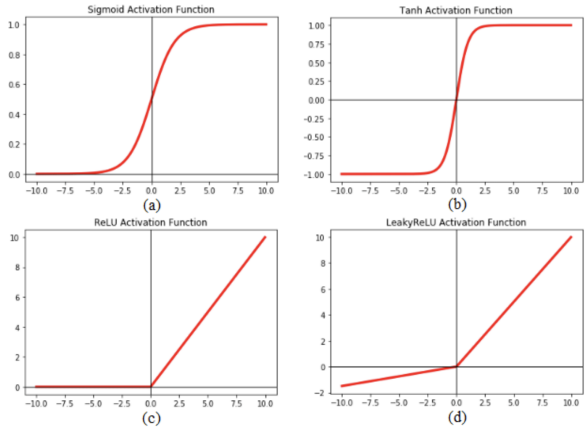

Figure 5: Different Structure of Activation Function

In this analysis, the activation functions used are the ReLU activation function and the Sigmoid activation function for the dense layers (since the task is a binary classification)

Sigmoid function: Its monotonically increasing property guarantees that the output will be in the range between 0 and 1, making it suitable for probability-based predictions in the analysis.

Mathematically is is expressed as:

$$\mathrm{S}(x) = \frac{1}{1 + e^{-x}}$$

ReLU function: It returns 0 for any negative input value, so it effectively "turning off" the neuron if the input is negative. For positive input values, the it returns the input value itself.

Mathematically is is expressed as:

$$f(x) = \max(0, x)$$

but it can also be written as:

$$\mathrm{f}(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

Where X is the input value.
[2]

1.4 Pooling Layers

Pooling layers are valuable components in CNNs for downsampling feature maps, reducing dimensionality, and extracting important features while maintaining spatial information. Pooling is applied by summarizing or subsampling patches of the feature map. The pooling operation involves moving a pooling window or filter across the feature map. Different pooling window sizes, such as 3x3 or 4x4, can be used depending on the specific requirements of the model and the nature of the data (in this analysis a 2x2 pooling window is used).

Two most popular methods of pooling are:

1. **Max pooling**: In each 2x2 patch, the maximum value among the features is selected as the representative value for that patch. The maximum value captures the most dominant or salient feature within the patch.

[2]Reference:https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh$_{fig4_339991922}$.$Reference : https://www.ml - science.com/sigmoid - activation - function$
$Reference : https://www.researchgate.net/figure/Plot - of - different - activation - functions - a - Sigmoid - activation - function - b - Tanh_{fig4_339991922}$

2. **Average pooling**: In each 2x2 patch, the average of the feature values is computed. This provides a more smoothed or averaged representation of the features within the patch.

(Figure 6)

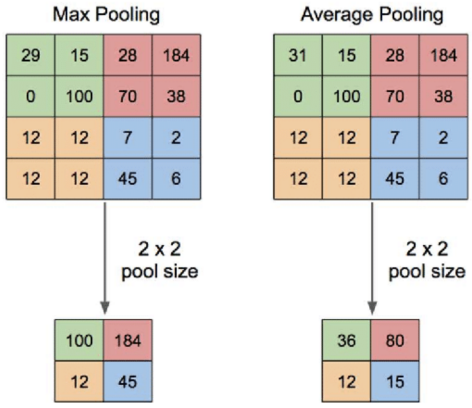

Figure 6: Max-Pooling and Average-Pooling

1.5 Flatten Layer

After applying pooling and padding operations to the feature map, the resulting feature map is typically a multi-dimensional matrix. However fully connected layers in a neural network require inputs to be in a one-dimensional format. The flatten layer is responsible for reshaping the multi-dimensional feature map into a single column or one-dimensional vector. This allows the neural network to treat the entire feature map as a sequence of individual features and enables further processing in fully connected layers.

1.6 Dense Layer

The previous vector is forwarded to the dense layer(also known as a fully connected layer). In a dense layer all neurons are connected to every neuron of its preceding layer and the output of each neuron is determined by performing a matrix-vector multiplication between the input from the preceding layer and a weight matrix.

$$A\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

Since the aim of the analysis is a binary classification the output will represents the probability of belonging to a particular class. [3]
[4]

1.7 Drop out Layer

A regularization method to mitigate overfitting and that help to improve performance of neural network,is the dropout technique.This layer randomly drop nodes in the architecture according to a dropout probability (denoted as p). All the connection with the node will be temporarily removed

[3]Reference: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks
[4]Reference:https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9.
Reference:https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/
Reference:https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/
Reference:https://www.analyticsvidhya.com/blog/2021/03/image-augmentation-techniques-for-training-deep-learning-models/

and, by doing so, it introduces noise and uncertainty into the network, forcing it to become more robust. The result is a modified network architecture.

1.8 Data Augmentation

Another method to reduce overfitting is by incorporating Data Augmentation layers that introduces more randomness into the training set. It enables the model to reduce sensitivity due to changes in the input data or noise, while continues to produce reliable predictions. It will challenge the classification of images since it will make harder to detected the subject because of the variations and transformation. Data Augmentation will also encourage the model to focus on the essential features of the object rather than being overly reliant on specific details or patterns that may be present in the original images.

In this analysis I used two layers :

1. **RandomFlip "horizontal"**: it performs a horizontal random flip transformation on the image, inverting the image with respect to the horizontal axis, creating a mirror image.
2. **RandomRotation**: it performs a random translation of the image both horizontally and vertically, with a maximum translation of 1 percentage with respect to the image dimensions.

This will enhance the model's ability to handle variations in object generalization and become more robust to different angles and orientations of objects present in the images.

1.9 Compiling the neural network

1.9.1 Optimizer and learning rate

An optimizer is a computational algorithm used to improve the performance of a neural network by adjusting its parameters, such as weights and learning rates. The primary goal is to minimize the overall loss of the network and improve its accuracy in making predictions. There are various optimization algorithms available in deep learning, including Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Adagrad, Adam, and more. It determines the step size taken during parameter updates and affects the model's ability to adapt to the problem. Setting a learning rate that is too large can cause the model to converge towards a sub-optimal solution or even diverge. On the other hand, using an excessively small learning rate may lead to slow convergence or the training process getting stuck in a local minimum. In this analysis the *Adaptive Moment Estimation (Adam)* optimization algorithm was employed, since this was specifically designed for training deep neural networks. The learning rate in this algorithm is a configurable hyper-parameter that can be set within the range of 0.0 to 1.0, with a default value of 0.001.
[5]

1.9.2 Loss function

Binary cross-entropy is a commonly used loss function for binary classification tasks. It measures the dissimilarity between the predicted probability distribution and the true binary labels.
[6]

The formula for binary cross-entropy can be expressed as:

$$\text{Binary Cross-Entropy} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

where p_i is the probability of class 1, and $(1-p_i)$ is the probability of class 0 and where $\hat{y}_i$ is the i-th scalar value in the model output, and y_i is the target value. When the observation belongs to class 1 the first part of the formula becomes active and the second part vanishes and vice versa in the case observation's actual class are 0. When the predicted probability aligns with the true label, the loss value is minimized. During the training process, the goal is to minimize the binary cross-entropy loss by adjusting the model's parameters since minimizing the loss helps the model improve its predictions and ultimately achieve better performance on binary classification tasks.

[5] Reference:https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
[6] https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

1.9.3 Metrics

The accuracy metric is used to evaluate the performance of the model. It calculates the proportion of correct predictions compared to the total number of predictions made.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions Made}}$$

In addition to binary cross-entropy, other metrics such as precision, recall, and F1 score can provide valuable insights into the performance of a model. These metrics offer a more comprehensive evaluation of the model's ability to classify the data accurately. In this analysis, these metrics will be used to evaluate the performance of the best model.

2 Data and Preprocessing

In this section, I am going to illustrate the dataset that I use for this project and the pre-processing phase . The dataset initially contained 25,000 images in JPG format, divided into two classes: Cats and Dogs. However, during the data preprocessing stage, it was discovered that 1,590 images were corrupted and had to be removed from the dataset to ensure the accuracy of the analysis. (Figure 7)

Figure 7: Dataset as a graph of 9 images

The RGB values of the images were rescaled to a range of 0 to 1 by dividing them with 255. The dataset was then split into training, validation, and testing subsets. Each image was resized to a shape of 128x128 pixels with three color channels.

3 Model description and results

In this section it will be describe Convolutional Neural Network models that are sequentially evolving from a baseline model to a model which guarantee higher accuracy.

3.1 Model 1

The baseline model is kept simple in order to be used as a benchmark for further analysis.

3.1.1 Architecture

1. **Convolutional layer**: one Covolutional layer of filters size 32, Kernel size 3x3, input shape 128x128x3;
2. **Activation layer**: ReLU Activation function;
3. **Max Pooling**: size 2x2;

4. **Flatten Layer**
5. **First dense layer**: Filter size 64;
6. **First dense layer**: Filter size 1;
7. **Activation function Layer** Sigmoid;

Total Parameters: 8,389,633
Test results : [0.46245765686035156, 0.7836394906044006]

3.1.2 Results

The results of the baseline models indicate the presence of overfitting. During training, the accuracy reaches an impressive 99%, while the validation accuracy starts at 72% but then decreases. This substantial gap of nearly 30% is clearly depicted in Figure 8. Moreover, the validation loss exhibits unfavorable behavior, increasing instead of decreasing as the epochs progress, in contrast to the loss computed on the training set.

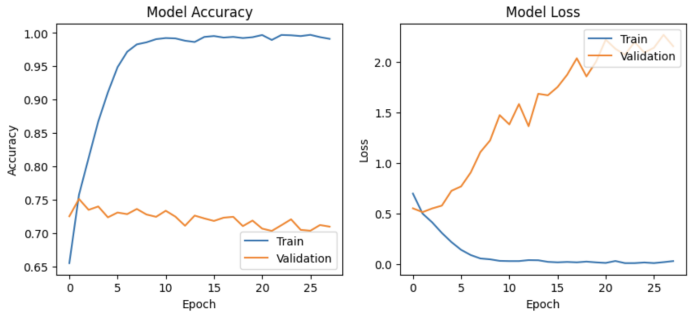

Figure 8: Model 1 Accuracy and Loss

3.2 Model 2

The purpose of creating the model was to investigate the effects of regularization techniques in order to prevent overfitting. To achieve this, the same model structure is maintained while introducing a slight dropout layer.

3.2.1 Architecture

1. **Convolutional layer**: one covolutional layer of filters size 32, Kernel size 3x3, input shape 128x128x3;
2. **Activation layer**: ReLU Activation function;
3. **Max Pooling**: size 2x2;
4. **Drop out**: rate 10%;
5. **Flatten Layer**
6. **First dense layer**: Filter size 64;
7. **First dense layer**: Filter size 1;
8. **Drop out**: rate 20%;
9. **Activation function Layer** Sigmoid;

3.2.2 Results

The implementation of regularization techniques has led to a slight reduction in overfitting. The accuracy of the training set stands at 87%, whereas the validation accuracy is 72%. As a result, the gap between the two has decreased to less than 20% (Figure 9). However, despite this improvement, the validation loss continues to exhibit a significant increase, while the training loss remains at 19%. However, the model's performance on the test set is not satisfactory, as indicated by the notably high loss.

Total Parameters: 8,389,633
Test results: [0.9822589755058289, 0.7411362528800964]

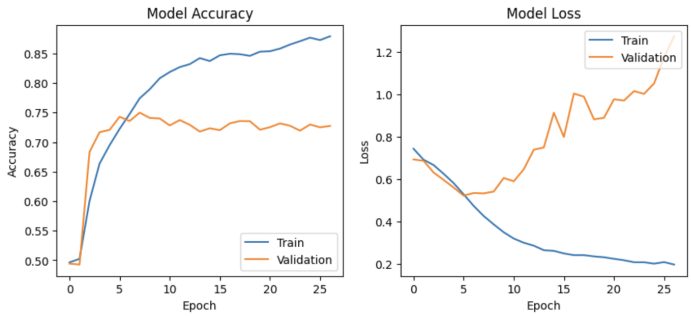

Figure 9: Model 2 Accuracy and Loss

3.3 Model 3

To further decrease overfitting in the previous model, another potential solution is to incorporate Data Augmentation. By applying various transformation layers, we can introduce variations in the training data, thereby increasing its diversity and reducing overfitting.

3.3.1 Architecture

1. **Horizontal flip**
2. **Rotation**: 20%
3. **Convolutional layer**: one covolutional layer of filters size 32, Kernel size 3x3, input shape 128x128x3;
4. **Activation layer**: ReLU Activation function;
5. **Max Pooling**: size 2x2;
6. **Flatten Layer**
7. **First dense layer**: Filter size 64;
8. **First dense layer**: Filter size 1;
9. **Activation funciton Layer** Sigmoid;

3.3.2 Results

The implementation of Data Augmentation techniques appears to have successfully resolved the overfitting problem. The validation accuracy, as well as the loss, now converge with the training accuracy, reaching an impressive level of nearly 75%. The training loss stands at 46%, while the validation loss is at 50% (as shown in Figure 10). This indicates that the model has significantly improved its ability to learn and generalize from the data, likely attributable to the effectiveness of Data Augmentation.

In addiction to that, there is a positive outcome observed in the test results as well: the loss on the test set has shown a decrease.

Total Parameters: 8,389,633
Test results: [0.49661019444465637, 0.7652712464332581]

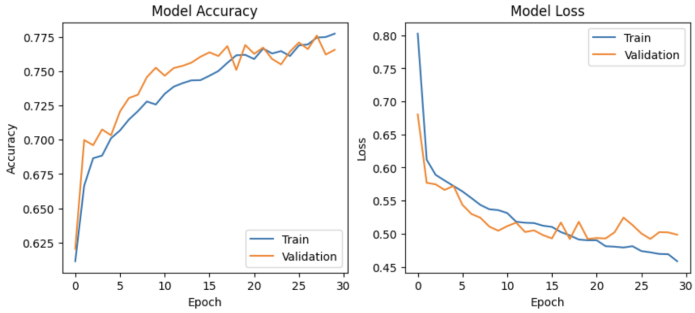

Figure 10: Model 3 Accuracy and Loss

3.4 Model 4

In order to further enhance the model's performance, two additional blocks of convolutional layers were incorporated into the existing architecture while retaining the Data Augmentation techniques.

3.4.1 Architecture

1. **Horizontal flip**
2. **Rotation**: 20%
3. **Convolutional layer**: 3 covolutional layer of filters size 32, Kernel size 3x3,input shape 128x128x3;
4. **Activation layer**: ReLU Activation function;
5. **Max Pooling**: size 2x2;
6. **Flatten Layer**
7. **First dense layer**: Filter size 64;
8. **First dense layer**: Filter size 1;
9. **Activation funciton Layer** Sigmoid;

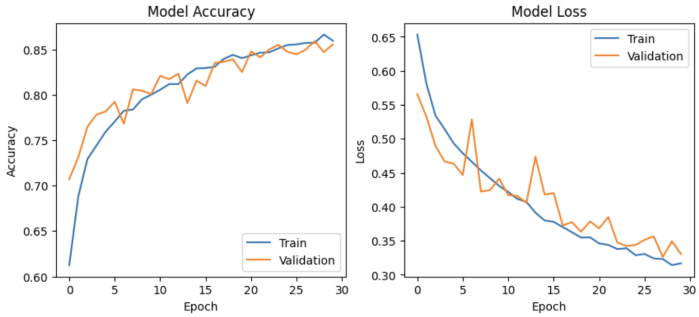

Figure 11: Model 4 Accuracy and Loss

3.4.2 Results

The inclusion of an additional Convolutional Layer has resulted in significant improvements in both the accuracy and loss curves, as demonstrated in Figure 11.With the introduction of the extra Convolutional Layer, the training and validation accuracies have reached a comparable level of 85%. This suggests that the model is now exhibiting consistent performance across both the training and validation datasets. The training loss stands at 31%, while the test loss is at 33%. By increasing the complexity and depth of the architecture, the model is now capable of capturing and representing more intricate features present in the data. Consequently, it can make more accurate predictions and generalize better to unseen instances.

Total parameters: 543,809
Test results are: [0.28830060362815857, 0.8720632195472717]

3.5 Model 5

Other two Convolutional Layers have been added to investigate how the increased depth and complexity would influence the model's ability to learn and extract features from the data.

3.5.1 Architecture

1. **Horizontal flip**
2. **Rotation**: 20%
3. **Convolutional layer**: 5 covolutional layer of filters size 32, Kernel size 3x3, input shape 128x128x3;
4. **Activation layer**: ReLU Activation function;
5. **Max Pooling**: size 2x2;
6. **Flatten Layer**
7. **First dense layer**: Filter size 64;
8. **First dense layer**: Filter size 1;
9. **Activation function Layer** Sigmoid;

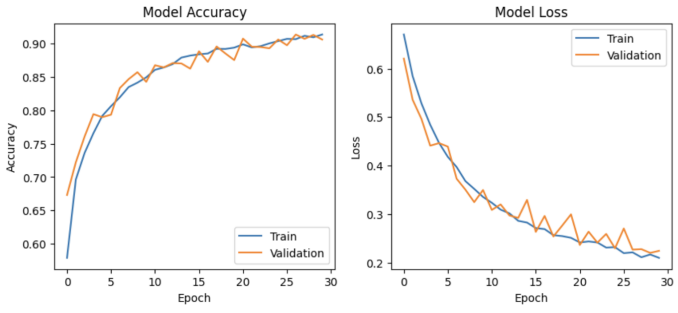

Figure 12: Model 5 Accuracy and Loss

3.5.2 Results

The inclusion of an additional Convolutional Layer has resulted in significant improvements in both the accuracy and loss curves, as demonstrated in Figure 12. These improvements have led to highly favorable results. The training loss has notably decreased to 20%, indicating that the model has become more adept at minimizing errors and discrepancies between predicted and actual values during the training phase. Similarly, the validation loss stands at 22%, reflecting the model's ability to generalize well and achieve good performance on unseen data.

Total parameters: 70,785
Test results are: [0.19437438249588013, 0.92011958360672]

3.6 Model 6

To further explore the impact of filter size on the model's performance, an experiment was conducted by reducing the filter size from 32 to 16 while utilizing the previous model architecture.

By decreasing the filter size, the model aims to capture more fine-grained details and intricate features within the input data. This modification allows for a more focused and localized analysis of the data, potentially leading to improved performance and better representation of the underlying patterns.

3.6.1 Architecture

1. **Horizontal flip**
2. **Rotation**: 20%
3. **Convolutional layer**: 5 covolutional layer of filters size 16, Kernel size 3x3, input shape 128x128x3;
4. **Activation layer**: ReLU Activation function;
5. **Max Pooling**: size 2x2;
6. **Flatten Layer**
7. **First dense layer**: Filter size 64;
8. **First dense layer**: Filter size 1;
9. **Activation funciton Layer** Sigmoid;

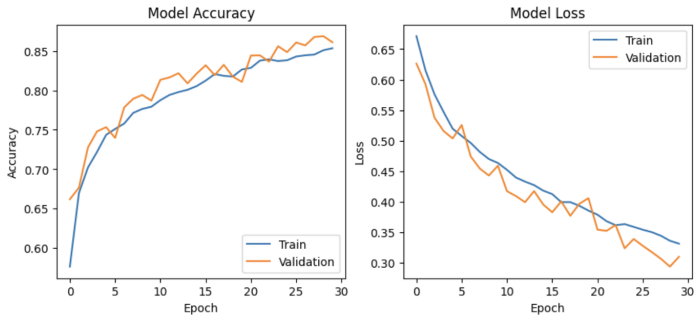

Figure 13: Model 6 Accuracy and Loss

3.6.2 Results

Despite reducing the filter size from 32 to 16 in the existing model, the resulting performance did not surpass the previous model. However, it is noteworthy that the model still achieved a commendable performance with significantly fewer parameters. (Figure 13)

Total parameters: 30,865
Test results: [0.31115537881851196, 0.8654420971870422]

3.7 Model 7

Continuing the exploration of different filter sizes, an additional experiment was conducted by increasing the filter size to 64, surpassing the previous model architecture.

3.7.1 Architecture

1. **Horizontal flip**
2. **Rotation**: 20%
3. **Convolutional layer**: 5 covolutional layer, filters size 64, Kernel size 3x3, input shape is 128x128x3;
4. **Activation layer**: ReLU Activation function;
5. **Max Pooling**: size 2x2;
6. **Flatten Layer**
7. **First dense layer**: Filter size 64;
8. **First dense layer**: Filter size 1;
9. **Activation funciton Layer** Sigmoid;

3.7.2 Results

The results of the experiment utilizing a filter size of 64 reveal that the model achieved a training loss of 23% and a validation loss of 22%. The accuracy metrics also indicate that both the training and validation accuracies reached 90%.

Comparing these results with the previous models, it is evident that this model performs better than Model 6 but falls short of the performance achieved by Model 5. However, it is important to note that the difference in performance can be attributed to the significant difference in the number of parameters between the models.

Total parameters: 215,169
Test results are: [0.23230355978012085, 0.9006834626197815]

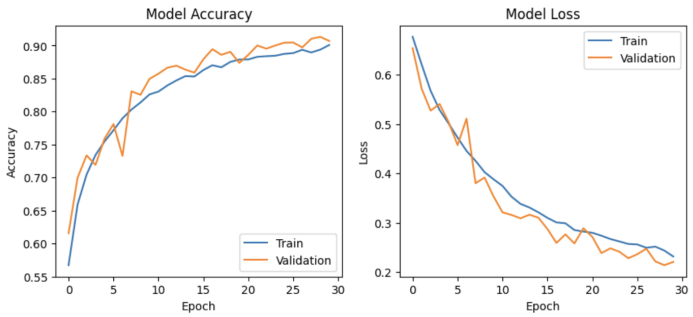

Figure 14: Model 7 Accuracy and Loss

4 Testing result

4.1 Evaluation metrics

To evaluate the performance of the model, various commonly used evaluation metrics in machine learning were considered. The evaluation process involved comparing the true labels of the test dataset with the predicted labels generated by the model.

4.2 Precision score

It is a measure of how often the model correctly predicts a positive instance (i.e., true positive) out of all the instances it predicted as positive (both true positives and false positives). It is calculated as the ratio of the number of true positives to the total number of predicted positives.

$$\text{Recall} = \frac{TruePositive}{TruePositive+FalsePositive}$$

4.3 Recall score

It is a measures of the ability of the model to correctly identify all positive instances out of all actual positive instances in the data.

$$\text{Recall} = \frac{TruePositive}{TruePositive+FalseNegatives}$$

4.4 F1 score

It combines both precision and recall to provide an overall measure of the model's ability to correctly classify positive and negative instances. F1 score ranges from 0 to 1, with a value of 1 indicating perfect precision and recall, and a value of 0 indicating that the model fails to correctly classify any of the positive instances.

$$\text{F1 score} = 2\frac{precision*recall}{precision+recall}$$

4.5 Results

	precision	recall	f1-score	support
Dogs	0.92	0.93	0.92	2345
Cats	0.93	0.92	0.92	2337
accuracy			0.92	4682
macro avg	0.92	0.92	0.92	4682
weighted avg	0.92	0.92	0.92	4682

Figure 15: Model 5 Evaluation matrix

4.6 Confusion Matrix

For Class 0, 2181 as been predicted correctly as Cats and 164 as Dogs. While for the images beloging to Class 1, 2147 are be predicted correcly while 190 have been predicted as cats.

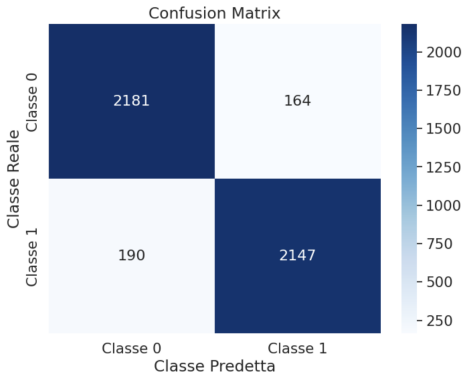

Figure 16: Confusion Matrix

5 Cross-Validation

This paragraph describes the usage of 5-Fold Cross-Validation technique to assess a model's performance on unseen data(not used during training). It helps to obtain a more reliable estimate of the model's performance and reduces the impact of randomness or bias in the evaluation. The technique is shown in the Figure 17, which provides a visual representation of how cross-validation is performed. It involves the following steps:

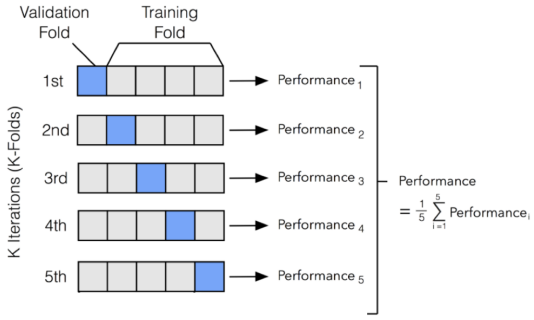

Figure 17: Cross Validation Example

The entire dataset is divided into k equal-sized folds or subsets; the model is trained and evaluated k times; in each iteration, one of the k folds is used as the validation set, while the remaining k-1 folds are used as the training set; the model is trained on the training set and evaluated on the validation set; the performance metrics are calculated for each iteration.

The algorithm's performance is more likely realistic because train and test were performed on all data. The Cross-Validation was performed on a `Model 3` since it is the one with better performance.

5.1 Result

It is possible to see that, using `Model 3`, the cross-validation technique has yielded excellent average results (Figure 20)

6 Conclusion

The CNN model developed to classify pictures of leaves has shown the potential to achieve even higher accuracy on the test set. Although different architectures could have been explored, the model's performance is commendable considering that only two parameters, namely filter size and number of convolutional layers, were tuned.

The experimental results of the exercise reveal three key conclusions:

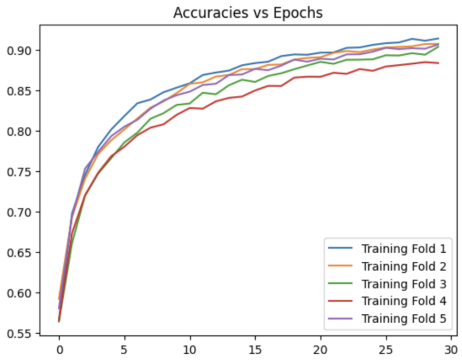

Figure 18: Loss of CV.

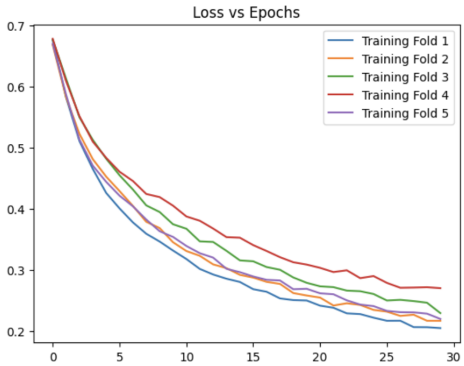

Figure 19: Accuracy of CV.

```
Average scores for all folds:
> Accuracy: 0.8916702270507812 (+- 0.011738493987953345)
> Loss: 0.26110116243362425
```

Figure 20: Average Results.

- Regularization techniques, such as Data Augmentation and Dropout, prove effective in mitigating overfitting issues.
- The network's learning capability improves depending on the depth of the Convolutional layers, as I previously explained.
- Techniques like hyperparameter tuning can significantly enhance the testing performance of the model.

To further elucidate this matter, additional experiments should be conducted.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.