Università degli Studi di Milano

Data Science and Economics

Convolutional Neural Networks for automated recognition of cats and dogs images

Course: Statistical Methods for Machine Learning
Student: Marta Magnani (961071)

Academic year 2021-22

Contents

List of Figures

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 I 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.

1 Introduction

In today's era of advanced technology, Deep Learning has emerged as a powerful and transformative field within Artificial Intelligence. With its ability to learn intricate patterns and representations directly from data, deep learning has revolutionized numerous domains, including image classification. Image classification is a task of Computer Vision, a field of Machine Learning, which addresses the task of automatically categorizing images into predefined classes or categories. This useful ability to analyze and understand visual data finds a wide range of applications, such as autonomous driving, medical diagnosis, facial recognition, and content organization.

One of the most common methods for conducting image recognition is training Convolutional Neural Networks (CNNs), which are recognized for being capable of effectively extracting features from images and learning to recognize patterns. Indeed, CNNs are a specialized class of deep neural networks inspired by the functioning of the animal visual cortex, where small regions of input activate neurons. By applying convolutional filters to local regions of an image and gradually aggregating information, CNNs can effectively discern intricate features and patterns within images.

This report specifically addresses the issue of binary Image Classification, considering a dataset composed of dogs and cats images. The goal is to classify the images by assigning them to the correct label (Cat or Dog). In this setting, the proposed methodology consists in building some Convolutional Neural Network architecture. The first part of this work breifly covers the theory regarding Feedforward Neural Networks and more specifically Convolutional Neural Networks (Section 2). The used environment for conducting the Machine Learning implementations and the relative packages are summarized in Section 3. The dataset structure and relative pre-processing phases are then presented in Section 4 together with the CNNs architecture of the models. Finally, Section 5 reports the results of the analysis along with some considerations to improve the performances of the aforementioned models. All the code is available on GitHub[1].

2 Neural Networks architecture

2.1 Feedforward Neural Networks

Artificial Neural Networks (ANNs) are computational structures that emulate the organization and functionality of biological neural networks. These networks consist of interconnected artificial neurons, each capable of receiving input signals, processing them, and transmitting output signals to other neurons through weighted connections. The strength of these connections, represented by coefficients known as weights, is adjusted during the training phase to determine the importance of each input signal. Neurons in ANNs are organized into layers, forming groups of interconnected nodes. The arrangement and function of these layers define the type of neural

[1]GitHub repository: https://github.com/mmartamagna/Machine-Learning-Project-CatsDogs

network being used. One widely recognized type is the Feed Forward Neural Network (FFNN), where neurons in each layer are connected exclusively to neurons in the subsequent layer. This architecture creates an acyclic graph, enabling the flow of information from the input layer, through the hidden layers, and finally to the output layer. When an FFNN incorporates multiple hidden layers, it is referred to as a Multilayered Feedforward Neural Network (MFFNN). This expanded architecture allows for the extraction of increasingly complex features and representations as information traverses through the network. By stacking layers and adjusting the weights, MFFNNs possess the ability to learn and model intricate relationships within input data, making them highly effective in tasks such as pattern recognition, classification, and regression. Figure 1 shows an example of the general architecture[2].

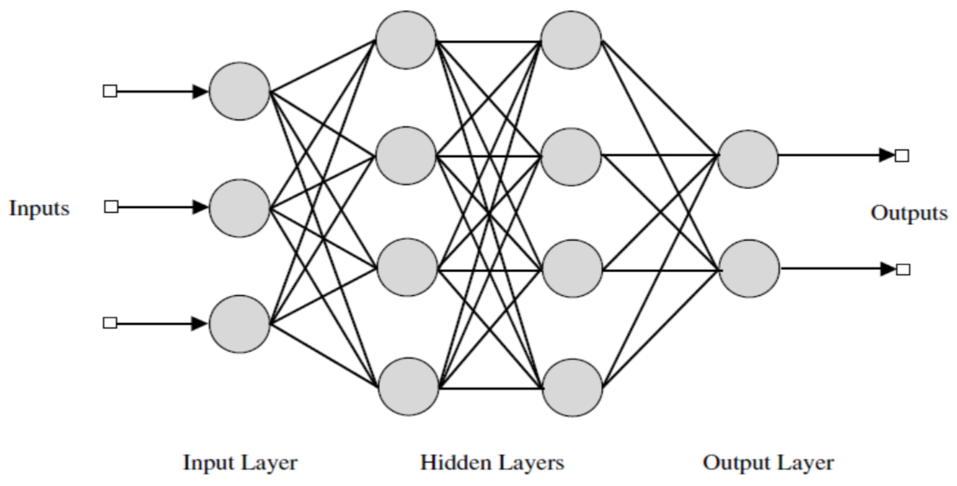


Figure 1: Multilayered Feedforward Neural Network architecture

2.2 Convolutional Neural Networks

This project focuses on the implementation of Convolutional Neural Networks (CNNs) for the binary classification of images depicting dogs and cats. CNNs are a specialized Feed Forward Neural Networks class that excels in visual tasks and find extensive application in various domains such as spatial data analysis, computer vision, natural language processing, signal processing, and more. The architecture of CNNs is inspired by the functioning of the visual cortex in animals, where neurons respond selectively to specific regions of their input. In CNNs, the key idea is to apply a kernel, represented by a square matrix, to localized regions of the input image. This kernel performs the convolution operation by sliding across the entire input matrix, enabling the extraction of relevant features resulting in activation maps or feature maps. This process is repeated systematically, allowing the network to discover intricate patterns and spatial relationships within the input data. This approach offers a significant advantage compared to traditional fully connected topologies, as it enables the network to capture spatial correlations within the inputs. In contrast, conventional networks require the linearization of the input data.

[2]Source: https://cse22-iiith.vlabs.ac.in/exp/forward-neural-networks/theory.html

Every convolutional layer in the network plays a pivotal role in downsizing the input matrix by consolidating a region into a singular output for the subsequent layer. Typically, the final segment of a Convolutional Neural Network (CNN) encompasses a fully connected network. As depicted in Figure 2[3], the convolutional part of the network focuses on extracting patterns from the input, encompassing even complex patterns. These patterns are then propagated as features to a traditional network, which specializes in learning classification tasks based on these extracted features. Consequently, the overall structure of the CNN can be divided into two distinct sections: the initial section dedicated to feature learning and extraction, and the subsequent section devoted to the classification task.

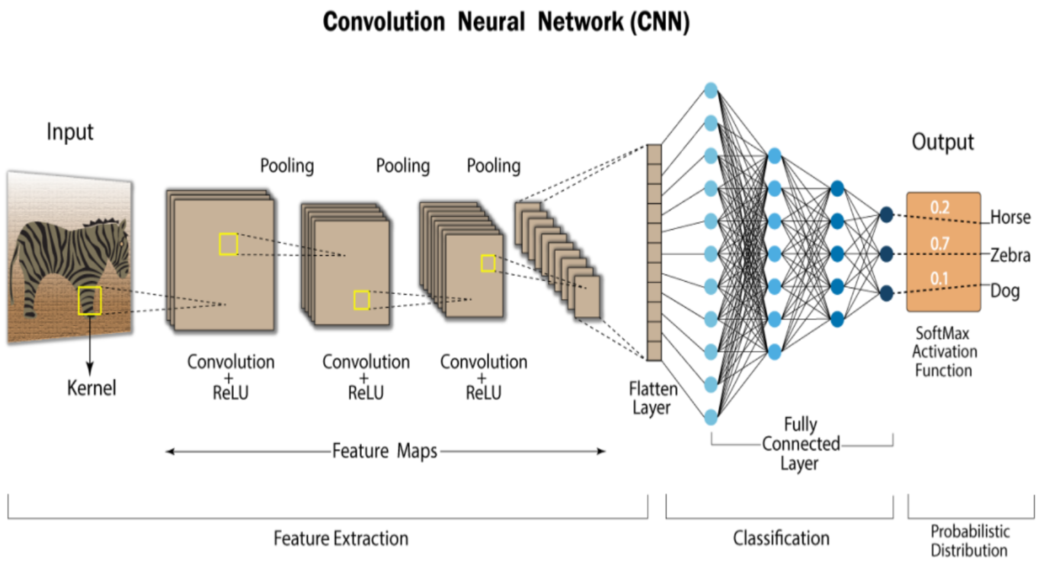


Figure 2: Convolutional Neural Network architecture

2.2.1 CNN architecture

One of the crucial aspects of the CNN implementation consists of the choice of layers. Going into more detail, a CNN generally has three layers: a convolutional layer, a pooling layer, a flattening layer and a fully connected layer.

- **Convolutional layer**. This is the first type of layer commonly found in CNNs and applies a set of learnable filters (also known as kernels) to the input image, performing a convolution operation that extracts local features. The filters slide over the input, capturing spatial patterns and creating feature maps or activation maps that highlight the relevant information.
- **Pooling layer**. After the convolution, the pooling layer further modifies and down-sample the output of the layer. The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps reduce the

[3]Source: https://developersbreach.com/convolution-neural-network-deep-learning/

representation's spatial size, reducing computational costs and allowing the learning of high-level patterns by deeper layers of the network. Common pooling operations include max pooling, which selects the maximum value in each pooling region, and average pooling, which calculates the average value.

- **Flattening layers**. After the convolutional and pooling layers, the resulting feature maps are typically in a multidimensional format. Flattening is the process of converting these multidimensional feature maps into a one-dimensional vector.
- **Dense layers.**The one-dimensional vector created by the Flatten layer can then pass through one or a series of Dense Layers, where each neuron performs a weighted sum of the inputs received from the previous layer. These weights are learned during the training process. After the weighted sum, an activation function is applied to introduce non-linearity into the network. Common activation functions include the sigmoid function, ReLU (Rectified Linear Unit), and others. The output of each neuron in the Dense layer is then passed to the next layer, or in the case of the output layer, it represents the final prediction or value.
- **Output layer**. The output layer is the final layer of a neural network that produces the desired output based on the specific task at hand. It is responsible for generating the predictions or classifications for the input data. The configuration of the output layer depends on the nature of the problem being solved. For binary classification tasks, where the goal is to classify input data into two classes, the output layer typically consists of a single neuron with a sigmoid activation function. The output of this neuron represents the probability or confidence of the input belonging to the positive class.

Additional layers that can be inserted are Dropout and Batch Normalization layers, which consist of regularization techniques that can improve the generalization performance of a network.

- **Dropout layers**. Dropout layers are typically added after fully connected layers in a network and they nullify the contribution of some neurons towards the next layer and leave all others unmodified. During training, dropout layers randomly set a fraction of the input units (neurons) to zero at each update, effectively "dropping out" those units. This prevents the network from relying too much on specific neurons and encourages the network to learn more robust and generalized features. These layers are usually implemented to prevent overfitting the training data.
- **Batch normalization layers**. Batch normalization layers are usually placed after a network's convolutional or fully connected layers and it is a technique used to improve the stability and speed of training deep neural networks. It is a method used to make the training of artificial neural networks faster and more stable through the normalization of the layers' inputs (i.e., it normalizes the input of a layer by subtracting the mean and dividing by the standard deviation of the mini-batch). This helps to alleviate the problem of internal covariate shift and ensures that the input to each layer remains within a suitable range.

3 Environment and tools

The code for this project was written in Python 3 on Google Colab, an online environment allowing the use of free cloud-based Jupyter Notebooks, a well-suited tool for machine learning and data analysis work. This platform also provides access to computing resources, including some powerful GPUs, which are useful when training ML models. All the code is available on GitHub.

During the development of the project and the implementation of the models, we primarily used the following libraries:

- **OpenCV**, useful for dealing with images in the preprocessing phase;
- **Python Imaging Library - PIL**, to help OpenCV recognizing more image formats;
- **Numpy**, which allows large array manipulations;
- **Tensorflow**, open source library widely used in Machine Learning, since it provides a framework capable to build, train, and finally use deep learning models;
- **Matplotlib**, which allows to create plots;
- **Seaborn**, a data visualization library which refines plots obtained through Matplotlib.

4 Dataset and data preprocessing

The original dataset is a collection of pictures of cats and dogs containing 12500 images for each category and provided through Unimibox. The whole folder has a size of 809 MB: the dog folder has a size of 439 MB, while the cat one has a size of 370 MB. The preprocessing procedure involved the following operations:

1. Misleading images removal

Cats and Dogs folders contain misleading images that do not represent animals (e.g., people, words, drawings, forests, etc.) or simultaneously contain cats and dogs. These misleading images can introduce noise and inconsistencies into the dataset, affecting the quality and representativeness of the data. By removing these irrelevant images, we can improve the overall integrity of the dataset and ensure that the Neural Network models focus solely on learning the distinctive features and patterns associated with cats and dogs. This process helps prevent any biases from being introduced into the training data and ultimately enhances the performance of the models in accurately classifying cat and dog images. After the wrong images removal, the remaining cats' images are 12468 and the remaining dogs' images are 12465. Below are some examples of misleading images in the cats' folder (Figure 3) and the dogs' folder (Figure 4).

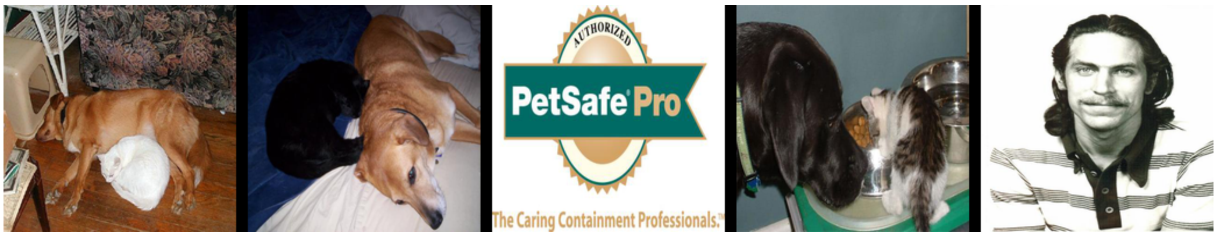


Figure 3: Sample of misleading images in Cats folder

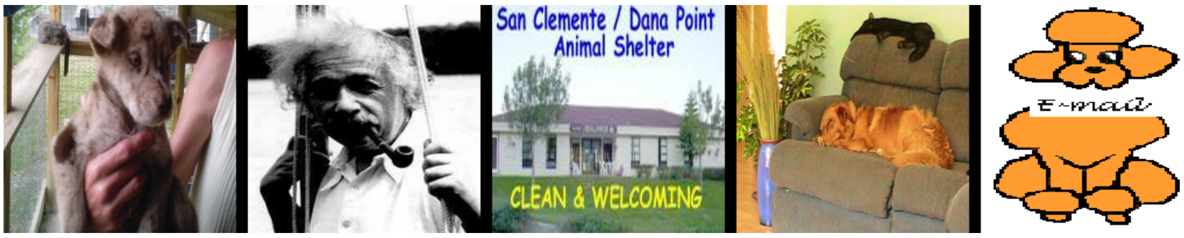


Figure 4: Sample of misleading images in Dogs folder

2. Resize (128x128) and colour conversion The next step was to resize all the images to 128x128 pixels to give more homogeneity and uniformity to the dataset. Furthermore, all the images were converted from JPG format to greyscale (Figure 5). This colour scale was preferred to multiple colour channels (e.g., RGB scale) to make the CNNs learning process less computationally intensive.

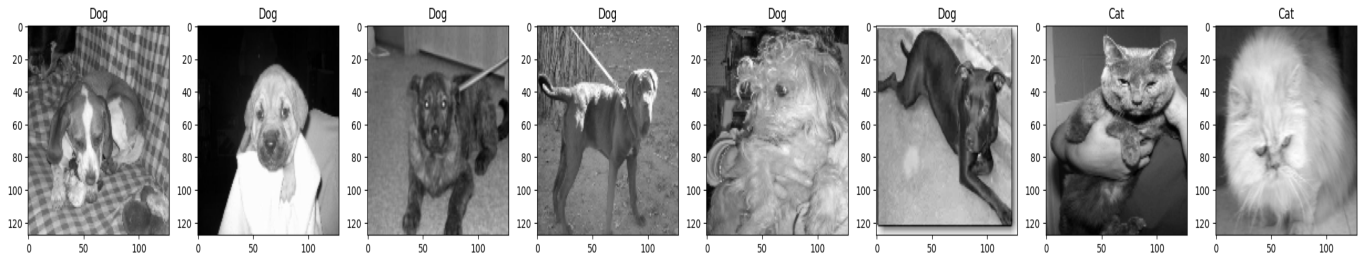


Figure 5: Sample of 128x128 greyscale images

3. Normalizing pixel values To improve model performances the gray-scale values have been re-scaled from the [0, 255] range to the [0, 1] range.

5 Implemented techniques

This section presents the Convolutional Neural Network model used for the binary classification task.

5.1 CNN models definition

Model 1

The first CNN model consists of multiple layers for extracting features from input images and making predictions. Here is a description of each layer: 1. First Convolutional Layer: It has 32 filters of size 3x3. It uses the ReLU activation function to introduce non-linearity. The input shape of the layer is (128, 128, 1), indicating the size and number of channels of the input images (1 for greyscale images). 2. Max Pooling Layer: It performs max pooling with a pool size of 2x2, reducing the spatial dimensions of the feature maps by half. 3. Dropout Layer: it randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting. In this case, it sets 25% of the units to 0 (one in four inputs will be randomly excluded from each update cycle). 4. Second Convolutional Layer: It has 64 filters of size 3x3. It uses the ReLU activation function. 5. Max Pooling Layer: Same as before, performs max pooling with a pool size of 2x2. 6. Dropout Layer: As before, 25% of the units are set to 0. 7. Third Convolutional Layer: It has 128 filters of size 3x3. It uses the ReLU activation function. 8. Max Pooling Layer: Same as before, performs max pooling with a pool size of 2x2. 9. Dropout Layer: Same as before, sets 25% of the units to 0. 10. Flattening Layer: It flattens the tensor output from the convolutional layers into a 1D vector. 11. Dense Layer: It has 128 units with the ReLU activation function. This layer is fully connected to the previous layer. 12. Dropout Layer: Same as before, sets 50% of the units to 0. 13. Output Layer: It has 1 unit with the Sigmoid activation function, which gives the binary classification prediction. The whole architecture of Model 1 can be seen in Figure 6 (Appendix).

Model 2

This second CNN model is similar to the previous one but includes Batch Normalization layers. Here is a description of each layer: 1. First Convolutional Layer: It has 32 filters of size 3x3. It uses the ReLU activation function and takes an input shape of (128, 128, 1), indicating the size and number of channels of the input images. 2. Batch Normalization Layer: It normalizes the activations of the previous layer along the channel axis, which helps in stabilizing and accelerating the training process. 3. Max Pooling Layer: It performs max pooling with a pool size of 2x2, reducing the spatial dimensions of the feature maps by half. 4. Dropout Layer: It randomly sets a fraction of input units to 0 at each update during training, helping prevent overfitting. In this case, it sets 25% of the units to zero. 5. Second Convolutional Layer: It has 64 filters of size 3x3. It uses the ReLU activation function. 6. Batch Normalization Layer:

Same as before, normalizes the activations of the previous layer. 7. Max Pooling Layer: Same as before, performs max pooling with a pool size of 2x2. 8. Dropout Layer (rate 25%) 9. Third Convolutional Layer: It has 128 filters of size 3x3. It uses the ReLU activation function. 10. Batch Normalization Layer: Same as before, normalizes the activations of the previous layer. 11. Max Pooling Layer: Same as before, performs max pooling with a pool size of 2x2. 12. Dropout Layer: As before, 25% of the units are set to 0. 13. Flattening Layer: It flattens the tensor output from the convolutional layers into a 1D vector. 14. Dense Layer: It has 128 units with the ReLU activation function. This layer is fully connected to the previous layer. 15. Batch Normalization Layer: Same as before, normalizes the activations of the previous layer. 16. Dropout Layer (rate 25%). 17. Output Layer: It has 1 unit with the sigmoid activation function, which gives the binary classification prediction.

Both models have a similar structure with multiple convolutional layers, max-pooling, dropout, and dense layers. However, the second model includes Batch Normalization layers after each convolutional layer to further enhance the training process by normalizing the activations and improving the stability of the model. Figure 7 shows the whole architecture of Model 2 (Appendix).

5.2 K-fold Cross Validation

K-fold Cross Validation (CV) is a valuable technique used to assess the performance of a machine learning model by estimating the expected risk. The primary objective of CV is to evaluate the model's ability to accurately predict new, unseen data, which were not used during the model's training phase. This process helps identify potential issues such as overfitting or selection bias and provides insights into how the model will generalize to independent datasets. CV involves iteratively training and evaluating the model on different subsets of the initial dataset. The dataset is first randomly shuffled and divided into k parts, or folds. The model is then trained on k-1 folds and tested on the remaining fold. This process is repeated k times, with each fold serving as the test set once. The errors obtained from evaluating the model k times are averaged to obtain the cross-validation estimate for the expected risk of the model. By employing K-fold CV, it is possible to obtain a more robust assessment of the model's performance as it is tested on multiple subsets of the data. This approach provides a reliable estimation of the model's ability to generalize to unseen data, enhancing our understanding of its predictive capabilities and helping us identify any potential limitations or areas for improvement.

To the aim of this project, the **5-fold Cross-Validation** technique was implemented. As previously said, the primary goal of using 5-fold cross-validation is to obtain a more robust and reliable estimate of the model's performance. Using k=5 can be beneficial also in terms of the optimal use of data. By dividing the dataset into five equal parts (or folds), it is possible to use 80% of the data to train the model and the remaining 20% to evaluate its performance. This allows to make efficient use of the available data and obtain a more accurate estimation of the model's performance.

5.3 Hyperparameters and performance metrics

The two models are compiled with the following hyperparameters:

- Epochs: 30
- Mini-batch size: 64
- Optimizer: Adam optimizer
- Metrics: Binary accuracy
- Loss function: Binary cross-entropy.

During the training of the data, I decided to use the Binary Cross Entropy (or Log Loss) as the training loss function. This loss function increases when the prediction diverges from the target label. As the prediction approaches 1, the loss function slowly decreases. On the other side, when the prediction decreases, the log loss increases quickly. Therefore, there is higher penalization for those errors with high confidence.

For what regards the risk estimate, instead, we employed the zero-one loss function, which is defined as follows:

$$l(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise,} \end{cases}$$

where y is the target label and $\hat{\mathbf{y}}$ the prediction made by the model.

6 Results and visualization

Results tables

After performing the experiments, the results regarding the performances were organized within tables and graphs. The following tables respectively contain the average accuracy, the zero-one loss and the binary cross-entropy for each k-fold in Model 1 and Model 2. Looking at the average performances of the two models, it is possible to see that the two CNNs performed very similarly in terms of accuracy and zero-one loss, but diffently in terms of cross entropy loss.

Starting from the accuracy, it is a common evaluation metric that measures the percentage of correctly classified instances out of the total number of instances in a dataset. A model with a lower accuracy compared to another model means that it has a higher number of misclassifications or incorrect predictions. In this case, Model 2 has a slightly greater accuracy (0.8628), indicating that that the model made correct predictions for 86.28% of the instances.

For what concerns the Zero-one loss, it represents the risk or error incurred by a model in making incorrect predictions, therefore indicating the proportion of incorrectly classified samples in a dataset. A smaller zero-one loss means that the model has made fewer misclassifications and has a better performance in terms of classification accuracy. In this setting, Model 1 has a smaller zero-one loss (0.1231), indicating that the model is making incorrect predictions for approximately 12.31% of the instances. In other words, the model has a misclassification rate of 12.

Finally, the binary cross-entropy quantifies the dissimilarity between the predicted probability distribution and the true binary labels of a dataset. When comparing models, a lower binary cross-entropy loss indicates that the model has a better fit to the data and provides more accurate probability predictions. In this case, Model 2 has a lower cross entropy than Model 1, meaning that Model 1's predicted probabilities are closer to the true labels, resulting in a lower average loss value. Overall, Model 2 seems to be the best perfoming in terms of accuracy and loss cross-entropy, but not in terms of estimated risk.

Model 1	Accuracy	Zero-one loss	Cross entropy
Fold 1	0.8658	0.1377	0.2127
Fold 2	0.8769	0.1373	0.2405
Fold 3	0.8608	0.1586	0.2216
Fold 4	0.8552	0.1403	0.2319
Fold 5	0.8489	0.1510	0.2072
Average	0.8549	0.1231	0.2228

Model 2	Accuracy	Zero-one loss	Cross entropy
Fold 1	0.8732	0.1267	0.1688
Fold 2	0.8692	0.1307	0.1535
Fold 3	0.8708	0.1291	0.1505
Fold 4	0.8662	0.1337	0.1498
Fold 5	0.8345	0.1655	0.1637
Average	0.8628	0.1371	0.1573

Graphical interpretation

The reported images support the CNNs models' performance interpretation by giving fundamental insights in terms of overfitting and underfitting detection. Starting from Model 1 (Figure 8, Appendix), it is evident that in all the five folds of Cross-Validation there is a severe situation of overfitting. This problem occurs when the model becomes too specialized and preferably memorizes the training data instead of learning unseen data (i.e. validation set). This problem is recognizable by the fact that the validation loss function is dramatically separated after the 5th epoch by the train loss curve. This suggests that around epoch 5-6 there is the best fit, but it dramatically gets worse after that point.

For what concerns Model 2 (Figure (9, Appendix), in each fold there is a overfitting behavior as well. Starting from the accuracy curves, the starting level of the validation set is dramatically low in all five folds and the curves fluctuate in an unordered way until the 30th epoch. This behaviour is reflected in the loss function trend, which starts at a very high point in correspondence of the first epochs and then keeps on fluctuating far from the train curve.

The reasons why overfitting occurred may reside in the model complexity: the models used for image classification could be excessively complex and the presence of too many parameters relative to the available training data could have easily overfitted by memorizing the training samples rather than validation ones. Another problem could be the lack of regularization techniques or their wrong use. In both models, regularization techniques such as dropout and batch normalization were added to prevent this problem. However, if these techniques were not appropriately applied, the model might have become overly sensitive to the training data, leading to overfitting. In general, the elements that confirm the presence of overfitting in the current images classification are: i) high training accuracy together with low validation accuracy, indicating that the model achieves high accuracy on the training set but performs poorly on unseen validation or test images; ii) rapidly decreasing training loss, while validation loss stagnates or increases: it indicates that the model optimizes its parameters well on the training data, but fails to generalize to new examples, resulting in a higher loss on validation or test sets.

7 Conclusion

This project aimed to evaluate and compare different Neural Network architectures for a binary classification task, with the objective of identifying the best predictive model. Two deep network structures were employed, both incorporating convolutional layers along with regularization techniques such as Dropout and Batch Normalization. The key distinction between the models lies in the presence of Batch Normalization layers in Model 2, intended to enhance the network's generalization capabilities. Upon training the models and assessing their performance using 5-fold Cross-Validation, the results indicate that Model 2 outperforms Model 1 in terms of Accuracy and Binary Cross-Entropy loss. However, it is noteworthy that Model 1 exhibits a lower estimated risk, as indicated by the Zero-One Loss metric. This suggests that although Model 2 achieves higher overall accuracy and better loss optimization, it may incur a slightly higher misclassification rate on certain instances. Analyzing the accuracy and loss graphs, it becomes apparent that both models suffer from overfitting. This is evident from the distinct behavior of the training and validation curves in the accuracy and loss plots. To mitigate this issue and improve the models' generalization capabilities, various strategies can be explored. These include implementing data augmentation techniques, applying additional regularization methods, simplifying the model architecture, or utilizing early stopping to prevent excessive training. Furthermore, enriching the study by incorporating hyperparameter tuning would be valuable. Hyperparameter tuning involves systematically exploring different combinations of hyperparameters to identify the optimal configuration that yields the best performance. By employing this approach, it is possible to further optimize the models and potentially enhance their predictive capabilities. In conclusion, although Model 2 demonstrates superior performance in terms of accuracy and loss, the presence of overfitting in both models warrants further investigation and the implementation of additional strategies to enhance generalization. Additionally, exploring hyperparameter tuning techniques can contribute to the refinement and optimization of the models, ultimately leading to improved predictive performance.

Appendix

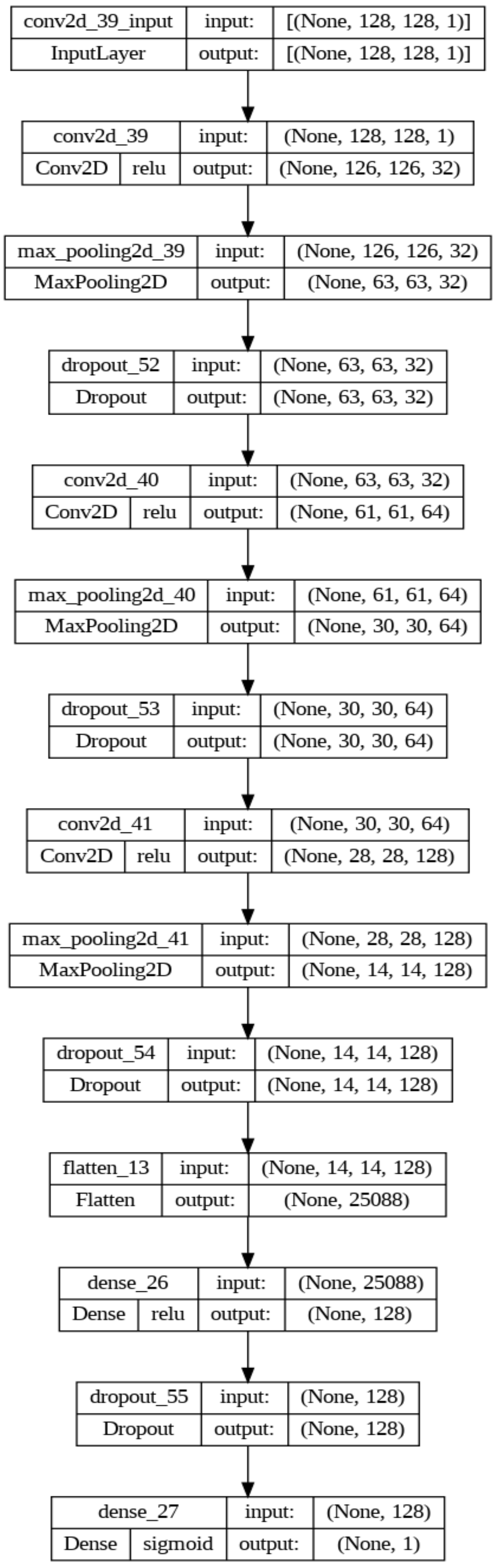

conv2d_39_input | input: | [(None, 128, 128, 1)]
InputLayer | output: | [(None, 128, 128, 1)]
conv2d_39 | input: | (None, 128, 128, 1)
Conv2D | relu | output: | (None, 126, 126, 32)
max_pooling2d_39 | input: | (None, 126, 126, 32)
MaxPooling2D | output: | (None, 63, 63, 32)
dropout_52 | input: | (None, 63, 63, 32)
Dropout | output: | (None, 63, 63, 32)
conv2d_40 | input: | (None, 63, 63, 32)
Conv2D | relu | output: | (None, 61, 61, 64)
max_pooling2d_40 | input: | (None, 61, 61, 64)
MaxPooling2D | output: | (None, 30, 30, 64)
dropout_53 | input: | (None, 30, 30, 64)
Dropout | output: | (None, 30, 30, 64)
conv2d_41 | input: | (None, 30, 30, 64)
Conv2D | relu | output: | (None, 28, 28, 128)
max_pooling2d_41 | input: | (None, 28, 28, 128)
MaxPooling2D | output: | (None, 14, 14, 128)
dropout_54 | input: | (None, 14, 14, 128)
Dropout | output: | (None, 14, 14, 128)
flatten_13 | input: | (None, 14, 14, 128)
Flatten | output: | (None, 25088)
dense_26 | input: | (None, 25088)
Dense | relu | output: | (None, 128)
dropout_55 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_27 | input: | (None, 128)
Dense | sigmoid | output: | (None, 1)

Figure 6: Architecture of Model 1

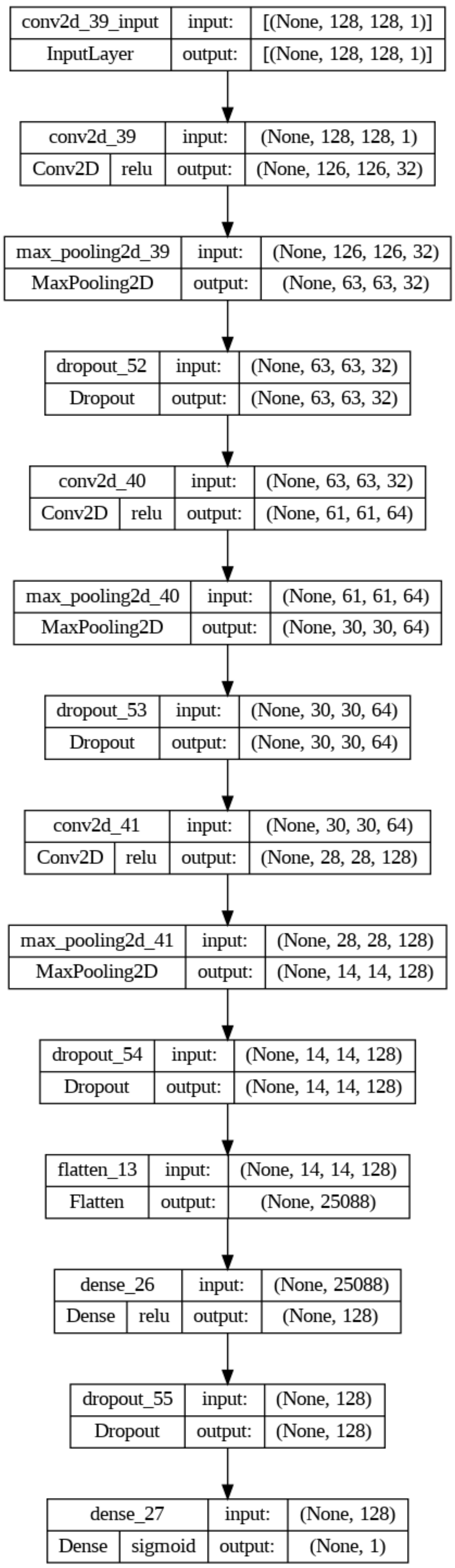
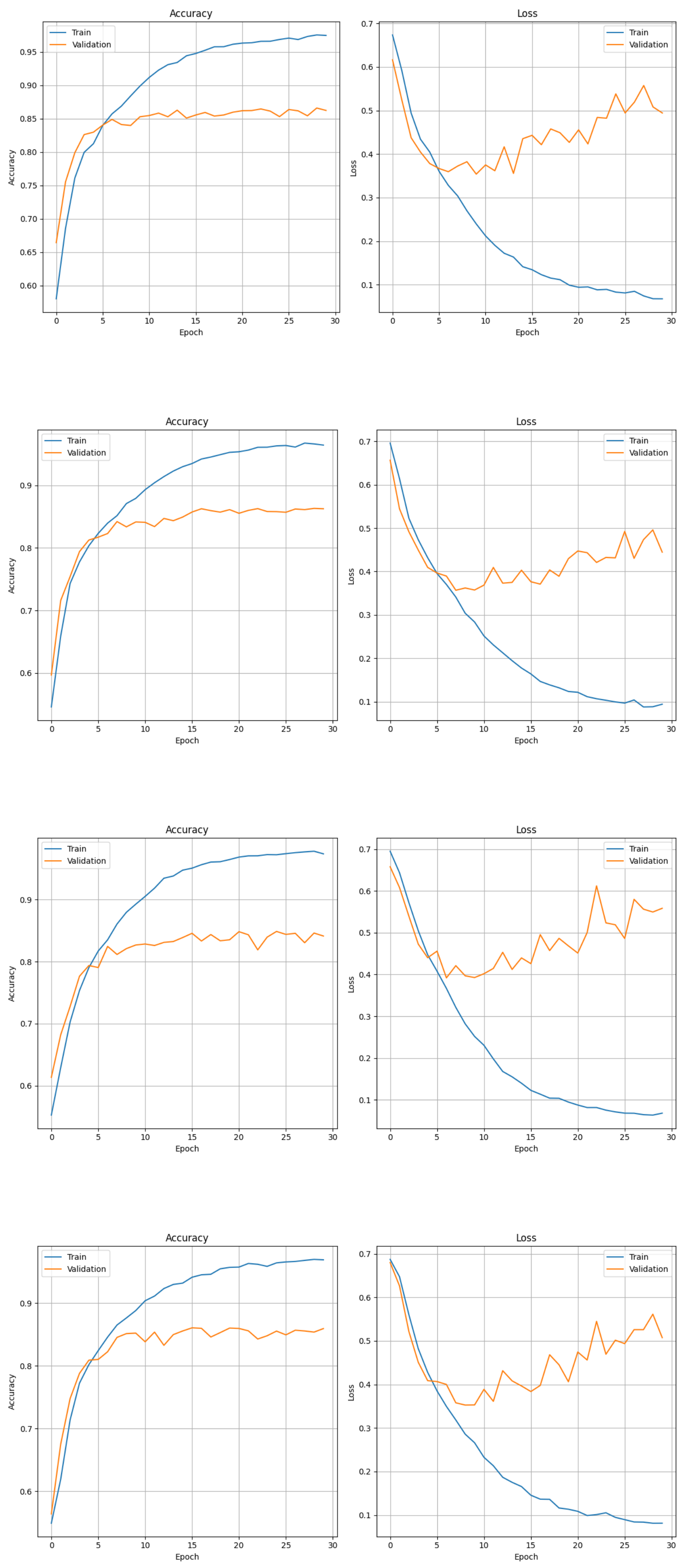

conv2d_39_input | InputLayer | input: [(None, 128, 128, 1)] | output: [(None, 128, 128, 1)]
conv2d_39 | Conv2D | relu | input: (None, 128, 128, 1) | output: (None, 126, 126, 32)
max_pooling2d_39 | MaxPooling2D | input: (None, 126, 126, 32) | output: (None, 63, 63, 32)
dropout_52 | Dropout | input: (None, 63, 63, 32) | output: (None, 63, 63, 32)
conv2d_40 | Conv2D | relu | input: (None, 63, 63, 32) | output: (None, 61, 61, 64)
max_pooling2d_40 | MaxPooling2D | input: (None, 61, 61, 64) | output: (None, 30, 30, 64)
dropout_53 | Dropout | input: (None, 30, 30, 64) | output: (None, 30, 30, 64)
conv2d_41 | Conv2D | relu | input: (None, 30, 30, 64) | output: (None, 28, 28, 128)
max_pooling2d_41 | MaxPooling2D | input: (None, 28, 28, 128) | output: (None, 14, 14, 128)
dropout_54 | Dropout | input: (None, 14, 14, 128) | output: (None, 14, 14, 128)
flatten_13 | Flatten | input: (None, 14, 14, 128) | output: (None, 25088)
dense_26 | Dense | relu | input: (None, 25088) | output: (None, 128)
dropout_55 | Dropout | input: (None, 128) | output: (None, 128)
dense_27 | Dense | sigmoid | input: (None, 128) | output: (None, 1)

Figure 7: Architecture of Model 2

Figure 8: Accuracy and loss of Model 1 during 5-fold CV



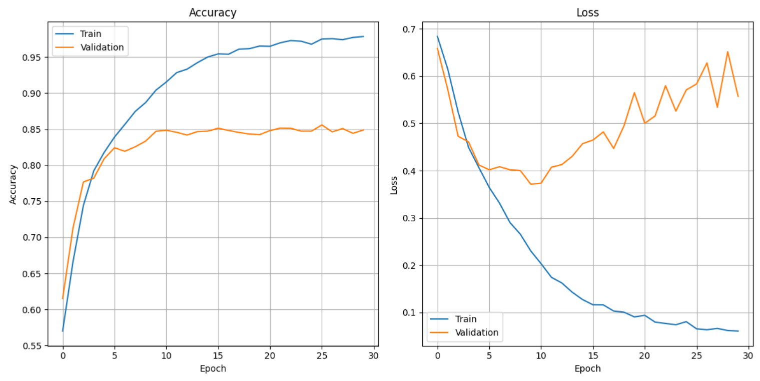
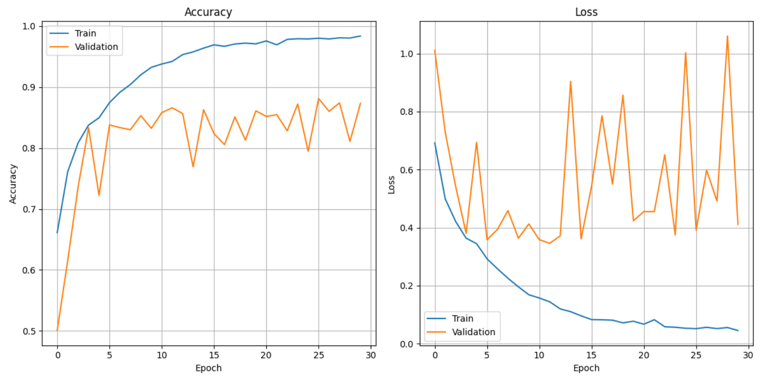
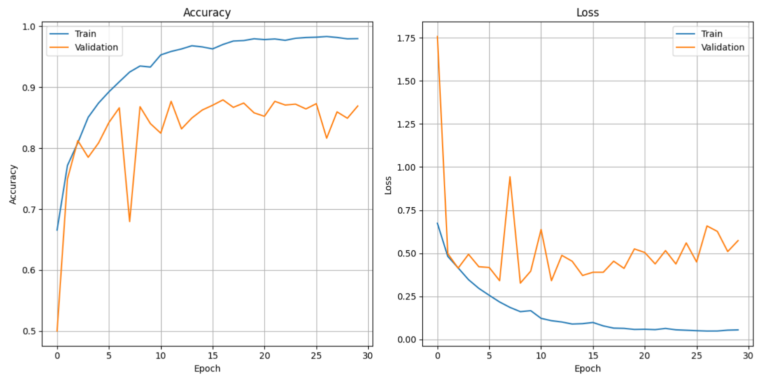
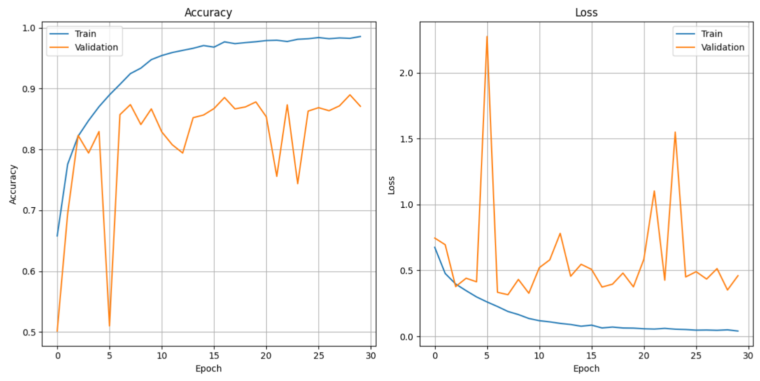
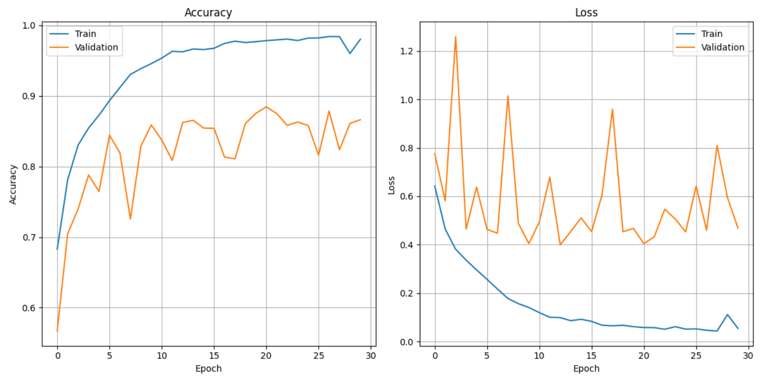


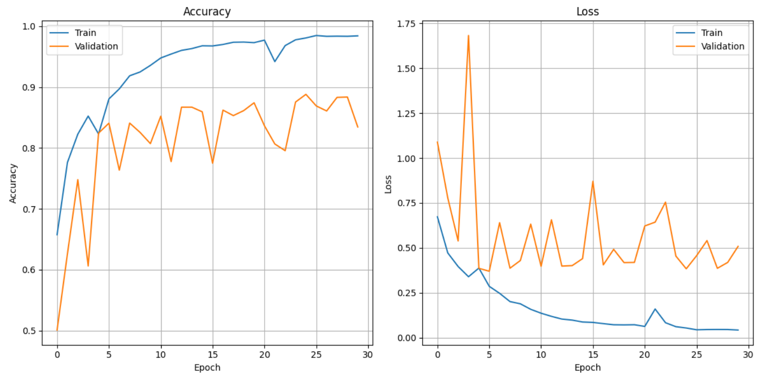
Figure 9: Accuracy and loss of Model 2 during 5-fold CV








Accuracy
Train
Validation
Accuracy
Epoch
Loss
Train
Validation
Loss
Epoch


Accuracy
Train
Validation
Accuracy
Epoch
Loss
Train
Validation
Loss
Epoch