Neural Networks Project Report: Binary Classification of Muffins and Chihuahuas

Francesca Ferraro - 16125A

July 21, 2023

Contents

Description : "Use Keras to train a neural network for the binary classification of muffins and Chihuahuas based on images from this dataset. Images must be transformed from JPG to RGB (or grayscale) pixel values and scaled down. The student is asked to: experiment with different network architectures (at least 3) and training hyperparameters, use 5-fold cross validation to compute your risk estimates, thoroughly discuss the obtained results, documenting the influence of the choice of the network architecture and the tuning of the hyperparameters on the final cross-validated risk estimate. While the training loss can be chosen freely, the reported cross-validated estimates must be computed according to the zero-one loss. "

Declaration : *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.*

1 Introduction

The goal of this project was to train a neural network for the binary classification of muffins and Chihuahuas based on images from a given dataset. I preprocessed the images by converting them from JPG to RGB format and scaling them down. I experimented with different network architectures and training hyperparameters, using 5-fold cross-validation to compute risk estimates. I then discussed the obtained results, documenting the influence of the choice of network architecture and hyperparameter tuning on the final cross-validated risk estimate.

2 Methodology

2.1 Dataset Loading and Preprocessing

I first set up the Kaggle API Key, downloaded the dataset, and unzipped it. This was necessary to access the dataset and extract the images for further processing.

Next, I loaded the images from the dataset. Loading the images is a crucial step in any computer vision task as it provides the raw data on which the model will be trained. After loading the images, I resized them. Resizing the images is important to ensure that all images have the same dimensions. This is necessary because neural networks require input data to have consistent shapes. Resizing the images also helps to reduce the computational complexity of the model.

Additionally, I converted the images to RGB format. RGB format represents images using three color channels: red, green, and blue. This format is widely used in computer vision tasks and is compatible with most neural network architectures.

To organize the data for training and testing, I created DataFrames for the train and test sets. DataFrames are a convenient data structure that allows for easy manipulation and analysis of tabular data. By creating DataFrames, I could associate each image with its corresponding label (muffin or Chihuahua) for both the train and test sets.

To feed the images into a neural network, I converted them into NumPy arrays. NumPy arrays are efficient data structures for numerical computations and are widely used in machine learning tasks. Converting the images into NumPy arrays allowed me to perform various operations on the images, such as normalization and feeding them into the network for training.

Finally, I normalized the pixel values of the images. Normalization is a common preprocessing step in machine learning tasks. It scales the pixel values to a standard range (usually between 0 and 1) to ensure that the model can learn effectively. Normalization also helps to mitigate the impact of differences in pixel intensity across different images.

Overall, these preprocessing steps were necessary to prepare the dataset for training a neural network. They ensured that the images were in the correct format, had consistent dimensions, and were normalized for optimal model performance.

2.2 Network Architectures

For this project, I experimented with four different network architectures: a custom from-scratch CNN, VGG16, MobileNet, and ResNet50. Each of these architectures has been pre-trained on the ImageNet dataset and has learned to recognize a wide range of features from a large dataset.

To adapt these pre-trained models to the specific binary classification task, I created a general model creation function. The function added additional layers on top of the pre-trained models, including a Flatten layer, a Dense layer with ReLU activation, and a final Dense layer with sigmoid activation. These layers were added to allow the models to learn task-specific representations and patterns on top of the pre-trained features learned by the base models.

By experimenting with these four different network architectures and using cross-validation to estimate the risk of each model, I was able to identify the best-performing architecture for the specific binary classification task at hand. This approach allowed me to leverage the pre-trained features learned by the base models and adapt them to the specific task, resulting in a more efficient and accurate model.

2.2.1 Custom CNN

For the "from-scratch" step of the project, I decided to create a custom CNN with some of the characteristics layers of a general neural network architecture, that are:

- A convolutional layer with 32 filters of size 3x3, with ReLU activation.

- A max pooling layer with a pool size of 2x2.
- Another convolutional layer with 64 filters of size 3x3, with ReLU activation.
- A max pooling layer with a pool size of 2x2.
- A third convolutional layer with 128 filters of size 3x3, with ReLU activation.
- A max pooling layer with a pool size of 2x2.
- A flattening layer to convert the 2D feature maps to a 1D vector.
- A dense layer with 512 neurons, with ReLU activation.
- A final dense layer with 10 neurons, with softmax activation for multi-class classification.

The convolutional layers extract features from the input images, applying a small filter on it. The max pooling layers reduce the size of the feature maps while retaining the most important features. The dense layers perform classification or regression tasks. It takes an input vector and applies a weighted sum to it. The output of the dense layer is a vector of predictions.

2.2.2 VGG16

VGG16 is a convolutional neural network architecture that was proposed by the Visual Geometry Group at the University of Oxford. The architecture consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. This architecture is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth. The network also includes max-pooling layers and ReLU activation functions. VGG16 has been pre-trained on the ImageNet dataset, which is a large-scale image recognition dataset, containing millions of images and thousands of object categories.

This pre-training allows for transfer learning in our classification task, where we can use the pre-trained weights as a starting point for our own classification task, fine-tuning the weights to our specific problem.

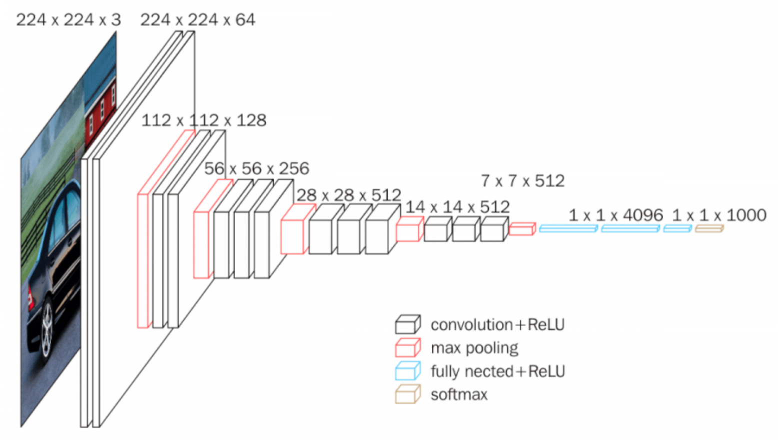


Figure 1: VGG-16 Architecture - image taken from online sources

2.2.3 MobileNet

MobileNet is a convolutional neural network architecture specifically designed for mobile and embedded vision applications. It addresses the challenge of deploying deep neural networks on devices with limited computational resources, such as mobile phones and embedded systems.

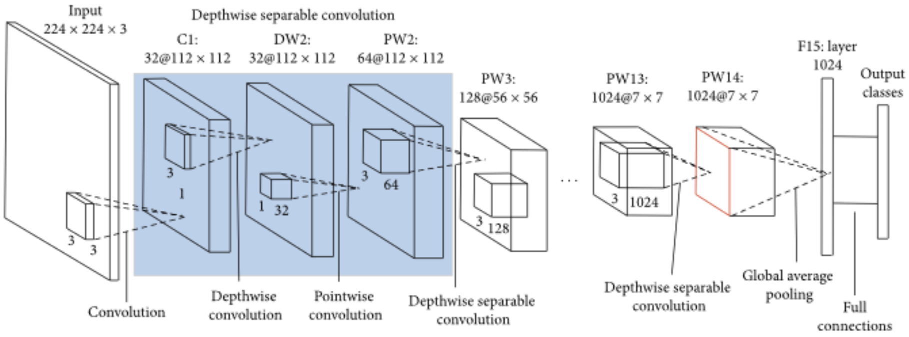


Figure 2: MobileNet Architecture - image taken from online sources

The key innovation of MobileNet is the use of depthwise separable convolutions. Traditional convolutional layers perform both the spatial convolution and the channel-wise convolution in a single step, resulting in a large number of parameters and high computational cost. In contrast, depthwise separable convolutions split the spatial convolution and the channel-wise convolution into separate layers, significantly reducing the number of parameters and computational cost.

Depthwise separable convolutions consist of two main steps. First, a depthwise convolution is applied independently to each input channel, using a separate filter for each channel. This step captures spatial information within each channel. Then, a pointwise convolution is applied to combine the outputs of the depthwise convolution, using 1x1 filters. This step allows for the transformation of the channel-wise information. MobileNet has been pre-trained on the ImageNet dataset, which contains millions of labeled images from a wide range of categories. Pre-training on ImageNet allows MobileNet to learn general features and patterns that can be transferred to our specific classification task. By leveraging transfer learning, we can benefit from the knowledge and representations learned by MobileNet on a large-scale dataset, which can improve the performance of our classification model.

The architecture of MobileNet is characterized by its efficiency and low computational cost. It strikes a balance between model size and accuracy, making it suitable for real-time applications and devices with limited computational resources. MobileNet has been widely used in various computer vision tasks, including image classification, object detection, and semantic segmentation.

In my project, I utilize the MobileNet architecture as one of the options for binary classification of muffins and Chihuahuas. By leveraging the pre-trained MobileNet model, we can benefit from its efficient design and transfer learning capabilities to achieve accurate and efficient classification on our specific dataset.

2.2.4 ResNet50

ResNet50 is a deep residual network architecture proposed by Microsoft Research.

It consists of 50 layers, including convolutional and fully connected layers. The key innovation in ResNet is the introduction of residual connections, which allow the network to learn residual functions and alleviate the vanishing gradient problem in deep networks. This is achieved by adding skip connections that bypass one or more layers, allowing the network to learn the residual mapping between the input and output.

ResNet50 has been pre-trained on the ImageNet dataset, which contains millions of labeled images from a thousand different classes. This pre-training

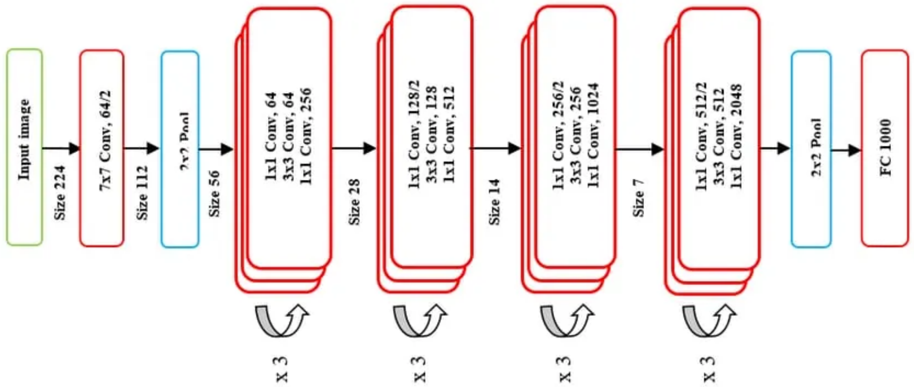


Figure 3: ResNet50 Architecture - image taken from online sources

enables transfer learning, where the knowledge learned from the ImageNet dataset can be transferred to our classification task. By leveraging the pre-trained weights, the network can already recognize low-level features such as edges and textures, allowing it to focus on learning higher-level features specific to our task.

The architecture of ResNet50 is characterized by its depth and ability to learn complex features. The network is composed of residual blocks, each containing multiple convolutional layers and shortcut connections. These shortcut connections allow the network to learn the residual functions, which are the differences between the input and output of each block. By learning these residual functions, ResNet50 can effectively model complex relationships and capture fine-grained details in the images.

Due to its depth and ability to learn complex features, ResNet50 is suitable for a wide range of computer vision tasks, including image classification, object detection, and image segmentation. Its performance has been demonstrated in various benchmark datasets and competitions, making it a popular choice in the computer vision community.

2.3 Training hyperparameters

In this experiment, the main goal was to investigate the impact of learning rate, the combination of the number of epochs and batch size, and the parametrization of the activation function and the number of neurons in the Dense layer (added to the base models in the general creation function) on the performance of the four different CNN architectures: custom CNN, VGG16, MobileNet, and ResNet.

Two sets of hyperparameters were used for each architecture:

- Set 1: learning rate = 0.001, epochs = 4, batch size = 8, activation function = ReLU, number of neurons = 1024

- **Set 2: learning rate = 0.0001; epochs = 8; batch size = 16, activation function = tanh, number of neurons = 512**

The learning rate is a hyperparameter that controls how much the model adapts to the problem. A larger learning rate allows the model to learn faster but may result in unstable training, while a smaller learning rate may lead to a slower training process that could get stuck.

The number of epochs is the number of complete passes through the training dataset, and it influences how many times the weights of the network are changed. As the number of epochs increases, the boundary of the neural network goes from underfitting to optimal to overfitting.

The batch size is another important hyperparameter that affects both the training time and the generalization of the model. A smaller batch size allows the model to learn from each individual example but takes longer to train, while a larger batch size trains faster but may not capture the nuances in the data.

Activation functions introduce non-linearity into the output of a neuron, enabling the neural network to learn complex patterns in the data.

The number of neurons in a Dense layer determines the output size and affects the layer's capacity to learn and represent features from the input data.

2.4 Training the Model with 5-Fold Cross-Validation

In order to train and evaluate the model, I experimented with different activation functions, number of neurons, learning rates, epochs, and batch sizes for each architecture.

For the optimization of the model I used the Adam optimizer, that is a stochastic gradient descent method that is computationally efficient, has little memory requirement, and is well-suited for large-scale problems. The loss function that I choosed for the training is the binary cross entropy loss, that measures the difference between the predicted probability distribution and the true probability distribution of the binary classification problem.

Then I used the zero-one loss as the performance metric for cross-validated estimates. The zero-one loss is a binary classification performance metric that measures the proportion of misclassifications. It is defined as the fraction of misclassified examples over the total number of examples.

The 0-1 loss is formalized as follows :

$$Loss = \sum_{i=1}^{n} \mathbb{I}\left(y_i \neq \hat{y}_i\right)$$

I used 5-fold cross-validation to compute risk estimates. Cross-validation is a technique used to evaluate the performance of a model on an independent dataset. It involves partitioning the dataset into k equally sized folds, training the model on k-1 folds, and evaluating the model on the remaining fold. This process is repeated k times, with each fold used exactly once as the validation data. The final performance metric is the average of the performance metrics obtained across the k folds.

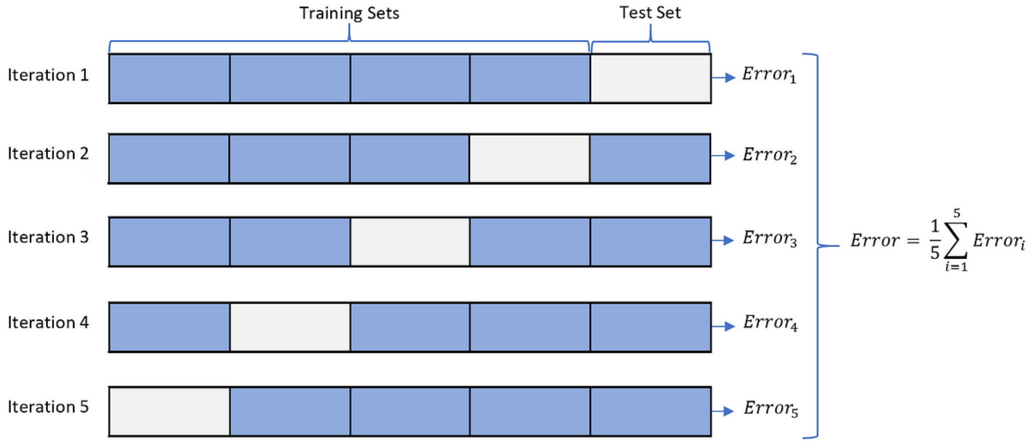


Figure 4: 5-fold Cross Validation - image taken from online sources

During the training process, I monitored the loss and accuracy for the training and validation sets. If the loss value decreases while the validation loss value starts to increase, it may indicate that the model is overfitting to the training set and is not generalizing well to new data. In this case, I would consider reducing the model's complexity or applying regularization techniques to prevent overfitting.

3 Results

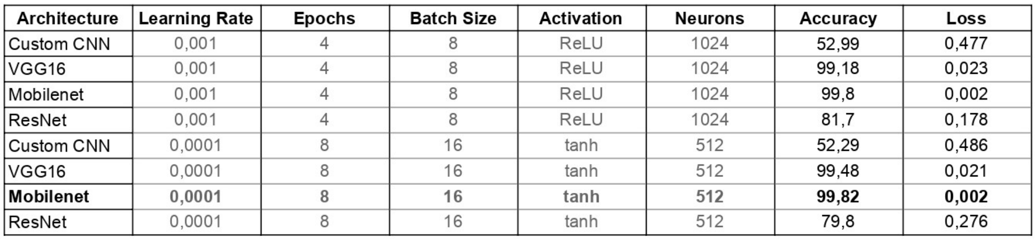
Accuracy is one of the most important metrichs used to evaluate the performances of a model.

It's a measure of how well the model correctly classifies the test data, obtained by dividing the number of correctly classified images by the total number of images in the test set.

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

I calculated the best accuracy and loss for each architecture and determined the architecture with the best accuracy and best loss.

Here we have the table of the results of each configuration :



Architecture	Learning Rate	Epochs	Batch Size	Activation	Neurons	Accuracy	Loss
Custom CNN	0,001	4	8	ReLU	1024	52,99	0,477
VGG16	0,001	4	8	ReLU	1024	99,18	0,023
Mobilenet	0,001	4	8	ReLU	1024	99,8	0,002
ResNet	0,001	4	8	ReLU	1024	81,7	0,178
Custom CNN	0,0001	8	16	tanh	512	52,29	0,486
VGG16	0,0001	8	16	tanh	512	99,48	0,021
Mobilenet	**0,0001**	**8**	**16**	**tanh**	**512**	**99,82**	**0,002**
ResNet	0,0001	8	16	tanh	512	79,8	0,276

These are the best results :

- Custom CNN: Best accuracy - 52.99%, Best loss - 0.477
- VGG16: Best accuracy - 99.48%, Best loss - 0.0211
- **MobileNet: Best accuracy - 99.82%, Best loss - 0.002**
- ResNet50: Best accuracy - 81.70%, Best loss - 0.178

The general best accuracy results 99.82%, and the general best loss results 0.002, performed by MobileNet architecture with this set of hyperparameters:

- learning rate = 0.0001
- number of epochs = 8
- batch size = 16
- activation function = tanh
- number of neurons = 512

4 Improvements

Even if the results obtained by the CNNs used in this project are very good, there are some possible signs of overfitting in the models.

In effect, the validation loss and accuracy curves did not always decrease or increase as quickly as the training curves.

To verify if the models overfit and eventually reduce overfitting, there are a lot of improvements that can be made, for example by experimenting with other hyperparameters, increasing the number of epochs and/or the batch size or decreasing the learning rate to help the model to learn more slowly and avoid overfitting the training data, or using some regularization techniques, such as L2 regularization or dropout. These techniques help to prevent the model from becoming too complex and overfitting the training data.

Moreover, a deeper analysis of the results can be done by extending the sets of hyperparameters to experiment with a larger set, such as testing all the possible combinations of the parameters.

5 Conclusion and discussion

In this project, I demonstrated the importance of selecting appropriate network architectures and hyperparameters for achieving optimal model performance in the binary classification of muffins and Chihuahuas. The experiments showed that the choice of network architecture and hyperparameter tuning significantly impacted the final cross-validated risk estimate.

The first architecture, that is computed from scratch, shows the worst results and doesn't compare well to the other models that I've considered.

The VGG16, MobileNet, and ResNet50 architectures exhibited different performance characteristics, with the best accuracy and loss achieved by the MobileNet architecture.

The choice of activation function, number of neurons in the dense layer, learning rate, epochs, and batch size also played a crucial role in determining the model's performance. I trained and evaluated the model using different learning rates, epochs, and batch sizes for each architecture. I used the zero-one loss as the performance metric for cross-validated estimates.

Regardless of the hyperparameter choices, there may be several reasons why I obtained the best performances from MobileNet.

Analyzing the 3 pretrained architectures I experimented with, we can notice that :

- MobileNet is a smaller model than VGG16 or ResNet, which means that it has fewer parameters. This makes it faster to train.
- MobileNet is designed for mobile devices, which means that it is optimized for inference on small devices with limited computational resources.

In conclusion, this project highlights the importance of selecting appropriate network architectures and hyperparameters for achieving optimal model performance in the binary classification of muffins and Chihuahuas. Future work could explore additional architectures, hyperparameter optimization techniques, and strategies for improving training speed.