CNN for Cats/Dogs recognition

DSE Unimi - Statistical Methods for Machine Learning

Schiavoni Cosimo - 964563

August 6, 2022

Abstract

The Project consists in the realization of a Convolutional Neural Network (CNN) for the cats/dogs recognition problem published on Kaggle. The code has entirely been created in the GoogleColaboratoy environment using Python 3.9, with the introduction of TensorFlow 2, published in the GitHub repository. The data set, consisting of 25.000 images, is publicly available and can automatically be downloaded during code execution. The experiment foresees the creation of a solution that is able to recognize the images and point out the correct label (cats vs dogs). To this extent, a single CNN structure has been conceived and it has been improved through the implementation of different layer compositions: 3 Convolutional/Pooling layers model (with 256 units in the Dense layer), 2 Convolutional/Pooling Layers model (with 256 units in the Dense layer), 1 Convolutional/Pooling Layer model (with 256 units in the Dense layer) and a 3 Convolutional/Pooling layers model (with 128 units in the Dense layer). The results showed two important insights: firstly, the most the complexity the most the accuracy of the model, secondly, greater differences are registered in the Zero-One Evaluation loss scores rather than in training and validation performances.

1 Declaration of authenticity

I declare that this material, which I now submit for assessment, is entirely my/our 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.

2 Introduction

The project consists in the realization of a Convolutional Neural Network algorithm, employed for the recognition and classification of Cats versus Dogs.
The algorithms have been developed in Python version 3.9, with the implementation of Keras (high-level API in the Tensorflow library).

3 Theory Backgroung

Convolutional neural networks (ConvNets or CNNs) are a type of neural network utilized in machine learning for classification and computer vision tasks. They leverage principles from linear algebra, specifically matrix multiplication, to identify patterns within an image.

They are comprised by three main types of layers, which are:
- Convolutional layer
- Pooling layer
- Dense Layer
- Fully-connected layer

The convolutional layer is typically the first layer of a convolutional network, additional convolutional and pooling layers can follow before feeding the data into a fully connected neural network. With each layer, the CNN increases its complexity, identifying every time greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional Layer
The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires 3 main components, which is to say input data, a filter, and a feature map. For the extent of this project, the input is a color image, which is converted into a 3-D matrix of pixels. The first two dimensions are height and width, which represent image size. The third is depth, due to the presence of 3 RGB image color layers.
The filter (or kernel) is a 2-D array of weights that can vary in size (typically a 3x3 matrix). It is applied iteratively (shifting across the entire image, each time by a stride) to specific areas of the image called receptive fields, which is to say

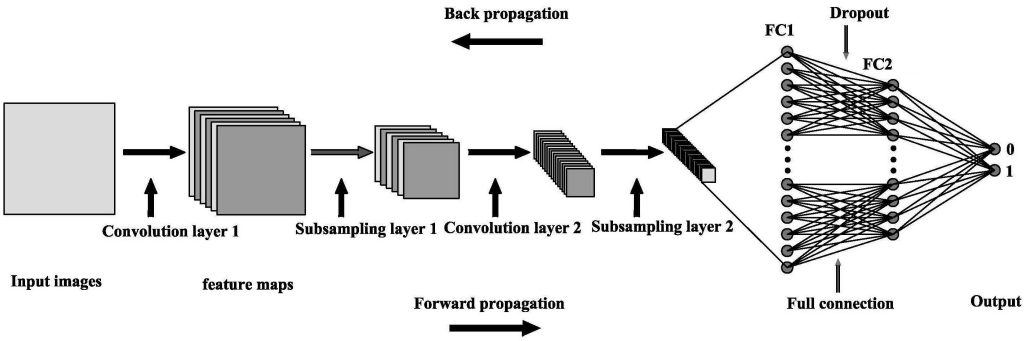

Figure 1: Structure of a Convolutional Neural Network.

portions of the image having the same size as the filter. At each movement, a dot product is calculated between the input pixels of the specific receptive field and the filter. The final output from the series of dot products is known as a feature map, activation map, or convolved feature. This entire process is known as a convolution.

Before training a network, it can be useful to initialize the values of the weights, a good practice is to set limits to these weights. One of the most popular techniques is the Glorot weight initialization algorithm, which basically uses limits that are based on the number of nodes in the network.

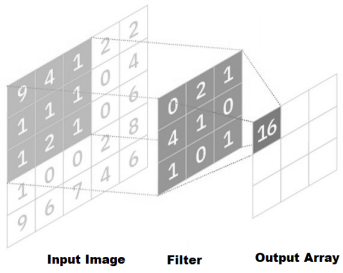

Figure 2: Numerical example of a convolution operation.

As can be noticed in the image above, each output value in the feature map does not directly connect to each pixel value in the input image. It only connects to the receptive field, where the filter is being applied. For that reason, convolutional (and pooling) layers are commonly referred to as "partially connected" layers. This characteristic can also be described as local connectivity. Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. This behavior is different with respect to other parameters like the weight values in NN, which adjust

during training through the process of backpropagation and gradient descent. During the convolution process, there are three hyperparameters that affect the volume size of the output that needs to be set before the training of the neural network begins:
1. The number of filters: defines the depth of the output (e.g. 3 filters produce 3 feature maps)
2. Stride: is the number of pixels that the kernel takes into consideration during iterative shifts across the image (typically the stride is set to 1 pixel, note that a larger stride yields a smaller output).
3. Padding: sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. It is usually employed when the filters do not fit the input image or when it is useful to receive a feature map of the same size as the input image.

There are three types of padding:
- Valid padding (No-padding): drops the last convolution if dimensions do not align;
- Same padding: ensures that the output layer has the same size as the input layer;
- Full padding: adds zeros to the border of the input image in order to increase the size of the output.

As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers.

The convolution process is basically conducted through linear transformations, in order to introduce nonlinearity to the model CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map afterward.

Pooling Layer
Pooling layers (or Subsampling), conduct dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter iteratively across the entire input. However, the filter does not have any weights, the kernel just applies an aggregation function to the values within the receptive field, populating the output array. As parameters, it only needs the shape of the pooling filter.

There are two main types of aggregation that a pooling layer can carry out:
- Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array.
- Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

The main drawback of Pooling is that a lot of information is lost, on the

contrary, it helps to reduce complexity, improve efficiency, and limit risk of overfitting.

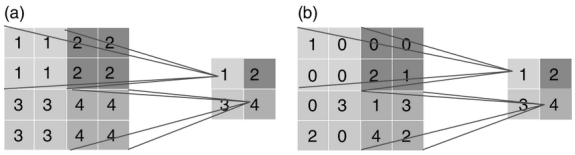

Figure 3: Numerical example of Average Pooling and Max Pooling operations.

Dense Layer
The Convolutional and pooling layers convert the image into numerical values in 2-D matrixes, which are re-organized in a 1-D array by a dense layer in order to allow the neural network to analyze data in the Fully-Connected Layer.

Fully-Connected Layer
Node layers contain an input layer, one or more hidden layers, and an output layer. In the fully-connected layer, each node in the output layer connects directly to a node in the previous layer and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, Fully-connected layers usually leverage a Softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

$$\hat{y} = sgn(\theta^T x) \tag{1}$$

Where θ^T is the transpose of the weight matrix and x is the example. The Softmax function is more suitable in classification tasks, with respect to the Sigmoid function, whenever the outputs are mutually exclusive, which is to say that an image cannot represent at the same time a cat and a dog.

For $x, \theta \in R_p$, feedforeward process in the fully connected layer can be written as:

$$E = \sigma(\theta_L \cdot \sigma(\theta_{L-1} \cdot \sigma(\dots \sigma(\theta_1 \cdot x + b_1) \dots) + b_{L-1}) + b_L) \tag{2}$$

Where b_i represents the bias vector, θ_i represents the weights and σ represents the Softmax activation function.This process in a neural network can be conceived as multiple instances of logistic and multivariate regression stacked on each other, so that the input of each sigmoid function is a linear combination of outputs of other sigmoid functions.

4 Empirical Analysis

1. Dataset: the dataset consists of 25.000 images and it's composed by two sets, the first one containing 12.500 cats, the second one containing 12.500 dogs. Each set of images is organized in a specific folder that is used to point out the label of the examples (in the specific case "Cats" and "Dogs"), all the variable dimensioned and color-scaled pixel images are composed by 3 color layers (RGB). Values are triplets of eight-bit numbers, contained in a range from 0 to 255, which point out the intensity of each color layer inside the pixel. During the RGB conversion process, 3 images (2 from cats dataset and 1 from dogs dataset) are deleted because corrupted.

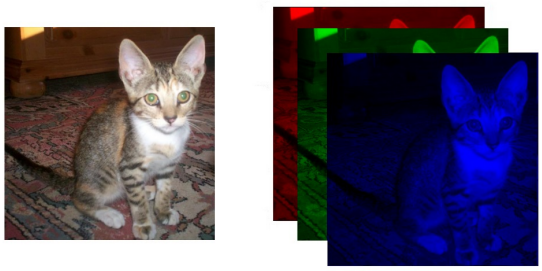

Figure 4: Example of the RGB composition of an image extracted from the original dataset.

2. Data Organization: The dataset is firstly collected into batches, the size of each batch is set as 32 images. Secondly, a split is carried out in order to organize the batches into three different groups, which is to say training, validation and test sets. The split allows for the creation of different input pipelines X that will be used for the training phase (train and validation sets) and for the test phase. The output is the binary class label, represented by a vector $Y = (Y_0, Y_1)$ of 2 dummy variables, let's say 1 for cats and 0 for dogs. To this extent, the training set amounts to 70% of the total dataset, which is to say 17.500 images collected into 547 batches. The rest of the data is collected into the validation set, which amounts to 25% (one-fifth as defined in the K-Fold Cross Validation algorithm) of the total (3.400 images collected into 107 batches). The test set is composed by the resulting data and amounts to 30% of it (7.497 images collected into 235 batches).

   A validation set is extremely important as far as it is used during training, to approximately validate the model on each epoch. This helps to update the model by giving "hints" as to whether it's performing well or not. Additionally, is not necessary to wait for an entire set of epochs to finish

Figure 5: Example of a random permutation from Training Set.

to get a more accurate glimpse at the model's actual performance. Using a validation set it is possible to interpret whether a model has begun to overfit significantly in real-time, and based on the disparity between the validation and training accuracies, is possible to trigger responses, such as automatically stopping training or updating the learning rate. The three input pipelines are prefetched in order to prepare subsequent elements simultaneously, hence speeding up batch iterations by exploiting the available computational memory in a more efficient way. The buffer size parameter, which defines the quantity of memory allocated for the computation, is set automatically.

3. Data Pre-processing: different data pre-processing techniques are employed in order to avoid overfitting, hence increasing the capacity of the model to intercept the right output even when images represent the same subjects in different conditions. At first, the image shape is reduced in size to 160 x 160 pixels, and shuffle is applied in order to retrieve a random permutation of images in each dataset. Secondly, Keras preprocessing has a class called ImageDataGenerator. It generates batches of tensor image

data with real-time data augmentation. It is possible to add any form of data augmentation technique and specify the validation split size.

Figure 6: Example of data augmentation generated by all the functions introduced.

Several different functions are applied in order to transform the images:

(a) Random Horizontal Flip: which randomly flips images in a horizontal sense;

(b) Random Rotation: which applies a random rotation;

(c) Random Shearing: which randomly applies shearing transformation;

(d) Random Rotation: which randomly rotates the image;

(e) Random Zoom: which randomly zooms in or zooms out;

(f) Random Shift: which randomly shifts the height and the width of the image;

(g) Image Normalization: which divides each RGB triplet value of a pixel by 255, changing the range to 0 - 1.

The reason why it should be used preprocessing is that in the real world pictures can be taken from any angle. For instance, by using flipping the model is possible to generalize on different scenarios. If it were to say, train on horizontal images of hands it might not predict hands in a vertical orientation.

5 Algorithms and implementation

For the extent of this project, the following structure of a CNN algorithm has been employed:

Layer (type)	Output Shape	Param num
Sequential	(None, 160, 160, 3)	0
Conv2D	(None, 78, 78, 32)	2432
MaxPooling2D	(None, 39, 39, 32)	0
Conv2D	(None, 19, 19 32)	9248
MaxPooling2D	(None, 9, 9, 32)	0
Conv2D	(None, 4, 4, 32)	4128
MaxPooling2D	(None, 4, 4, 32)	0
Flatten	(None, 512)	0
Dense	(None, 256)	131328
Dense	(None, 1)	257

Total params: 147,393
Trainable params: 147,393
Non-trainable params: 0

In order to check for different possible performances, the model has been analyzed step by step, introducing each time new layers. In the beginning, the experiment started by coding a basic CNN mode (1 Convolutional/Pooling Layer CNN model with 256 units in the dense layer, more convolutional-pooling layers have been introduced in order to improve the complexity and receive better results afterward. In the final experiment, the model passed from a 1 convolutional-pooling layer structure to a 3

convolutional-pooling layers structure. Moreover, the final model composed by 3 Convolutional/Pooling Layers with 256 units in the dense layer has been compared with the same model with 128 units in the dense layer. The computation foresees the implementation of the K-Fold Cross Validation algorithm.

K-Fold Cross Validation:
Ideally, if enough data are available, it could be set aside a validation set in order to use it to assess the performance of the prediction model. Since data are often scarce, this is usually not possible (The Elements of Statistical Learning, T. Hastie et al. - 2012). To finesse the problem, K-fold Cross Validation uses part of the available data to fit the model, and a different part to test it. Split the data into K roughly equal-sized parts; for example, when $K = 5$, the scenario looks like this:

Figure 7: The image shows the functioning of the K-Fold Cross Validation Algorithm composed by 5 folds.

For the kth part (gray section in the image above), fit the model is fit to the other $K - 1$ parts of the data, and the prediction error of the fitted model is calculated when predicting the kth part of the data. This process is repeated for $k = 1, 2, \ldots, K$ and the K estimates of prediction error are combined.
Let $k : \{1, \ldots, N\} \to \{1, \ldots, K\}$ be an indexing function that indicates the partition to which observation i is allocated by the randomization. Denote by $\hat{f} - k(x)$ the fitted function, computed with the kth part of the data removed. Then the Cross Validation estimate of prediction error is:

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{n} L(y_i, \hat{f}^{-k_i}(x_i)) \tag{3}$$

The case $K = N$ is known as leave-one-out Cross Validation. In this case

$k(i) = i$, and for the ith observation the fit is computed using all the data except the ith. Given a set of models $f(x, \alpha)$ indexed by a tuning parameter α, denote by $\hat{f} - k(x, \alpha)$ the αth model fit with the kth part of the data removed. Then for this set of models can be defined:

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{n} L(y_i, \hat{f}^{-k_i}(x_i, \alpha)) \tag{4}$$

The function $CV(\hat{f}, \alpha)$ provides an estimate of the test error curve, and it can be found the tuning parameter $\hat{\alpha}$ that minimizes it. The final chosen model is $f(x, \hat{\alpha})$, which is then fit to all the data.

The training phase is an optimization problem in terms of the difference between the true value, y, and the actual value, $\hat{y}$. This difference is often referred to as the loss, or the risk, denoted by L and is to be minimized with respect to the model parameters:

$$\min_{\{W_i\}_{i=1}^{L}, \{b_i\}_{i=1}^{L}} L(y, \hat{y}) \tag{5}$$

In this model, the "binary cross-entropy" loss function has been employed, given that a binary classification problem is meant to be solved. The formula can be expressed as follows:

$$L_{log} = -\frac{1}{N} \sum_{i=1}^{n} (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{6}$$

Where $\hat{y}_i$ is the prediction of class 1, and is the prediction of class 0. When the observation belongs to class 1 the first part of the formula remains and the second part vanishes and vice versa in the case observation's actual class is 0.

Backpropagation is a technique of weight adjustment employed during the training phase when an error occurs. At every step the model makes predictions, it finds out the difference between the actual output and the prediction, then if the difference is not the minimum the model must make another prediction closer to the output. In backpropagation, it is necessary to express $\frac{\delta L}{\delta \theta}$ for all elements θ in all weight matrices and bias vectors, then the optimization problem can be solved by setting the derivative to zero for all parameters. One of the simplest algorithms for finding the zeros of the derivative is Gradient Descent, an iterative algorithm, where in each iteration every parameter is updated a small step, α, called the learning rate, in the direction of the negative gradient. In this specific project, the Adam optimizer (adaptive moment estimation) has been employed in order to optimize the computation. The Adam optimizer involves a combination of two gradient descent methodologies and it can be expressed as

follows:

1) Momentum:
the algorithm computes the parameters after taking the gradient of the loss function at each step and accelerates the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients:

$$\theta_{t+1} = \theta_t + \alpha m_t \tag{7}$$

Where

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta \theta_t} \right] \tag{8}$$

Having, m_t the aggregate of gradients at time t (at t_0, $m_t = 0$), θ_t weights at time t, α the learning rate, β the moving average parameter, δL the derivative of the loss function and $\delta \theta_t$ the derivative of weights at time t.

2) Root Mean Square Propagation (RMSP):
Root mean square prop or RMSprop is an adaptive learning algorithm that considers the exponential moving average.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \left[\frac{\delta L}{\delta \theta_t} \right] \tag{9}$$

Where

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta \theta_t} \right]^2 \tag{10}$$

Having, v_t the sum of square of past gradients (i.e $\sum_{t=0}^{T} (\frac{\delta L}{\delta \theta_{t-1}})$. At t_0, $v_t = 0$) and ϵ a small positive constant to ensure that the denominator is always greater than 0. The combination of the two methods allows Adam optimizer for reaching the global minimum efficiently. Indeed a simple Gradient Descent algorithm has more probability to reach a local minimum rather than a global one during backpropagation.

Since m_t and v_t are initialized at t_0 as 0, that normally show a tendency to be biased towards 0. For that reason, it is useful to introduce bias-corrected algorithms:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{11}$$

and

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{12}$$

In this way, the algorithms adapt to gradient descent after every iteration so that they can remain controlled and unbiased throughout the process. Finally, introducing the bias-corrected weight parameters $\hat{m}_t$ and $\hat{v}_t$ into the backpropagation process, the new optimizer is:

$$\theta_{t+1} = \theta_t - \hat{m_t} \left(\frac{\alpha}{\sqrt{\hat{v_t}} + \epsilon} \right) \tag{13}$$

Finally, in order to test the efficiency of the model, the evaluation phase is carried out. To this extent, the loss generated by the untrained data (test dataset) employing the trained model is computed. During this specific phase, the Zero-One Loss Function has been implemented:

$$Ł(y, \hat{y}) = I(y \neq \hat{y}) \tag{14}$$

Where I is an indicator function, which is to say the function that maps all the results that respect the condition (expressed as its inner argument) equal to 1 and the rest equal to 0. In this case, all the predicted labels different from the actual values are taken into account.

It is important to notice that the Zero-One Loss Function is non-differentiable, hence it does not allow for optimizing the computation through a Gradient Descent algorithm during backpropagation. For this reason, in the model presented above, it is not possible to implement the Zero-One Loss Function also during the training and validation phase.

6 Comments and Discussion

The image below shows the experimental results for both training and validation sets for all the models. Comparing the curves, it's possible to appreciate the differences for every methodology. The 3 Convolutional/Pooling Layers with 256 units in the dense layer is the one that reaches the highest/lowest levels of accuracy/error (86% accuracy and 32% loss), while the 1 Convolutional/Pooling Layers with 256 units in the dense layer model returns the worse performance (76% and 45% respectively). Reducing the number of units in the dense layer by 50%, the performance of training and validation accuracy and loss are not very different, which means that simplifying the model does not apparently result in valuable benefits. However, considering the Zero-One Evaluation loss score, the first model is the one that reaches better results: 23% versus 34%. This result suggests that the most complex model (3 Convolutional/Pooling Layers

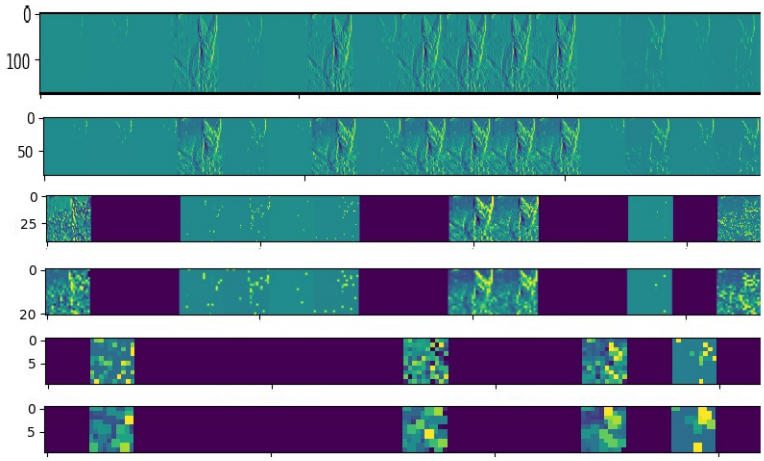

Figure 8: The image shows the feature maps resulted by each step of convolution and pooling processes.

with 256 units in the dense layer) is more accurate in making predictions, hence preferable with respect to the second one. Similarly, the reduction of Convolutional/Pooling layers from 3 to 2 layers or 1 layer has severe drawbacks in the Zero-One Evaluation loss score, in fact, the results are respectively 35% and 42%. As a consequence, the most the complexity, the most the precision. This principle is generally valid, however, it is a Data Scientist's task to define the optimal trade-off between the complexity of the model and its computing capacity, in order to avoid overfitting and, simultaneously, to obtain acceptable learning times. To this extent, in this experiment, the most complex model is the best one because it performs better than the others and is able to complete the learning process in acceptable times. In order to reach acceptable learning times, the input shape of the image in the model has been changed from 350 x 350 to 160 x 160 pixels, reducing the learning time for each iteration by almost 50%. Such a parameter is of great importance in CNN models.

Running the test on a small sample of 9 random images extracted from the test set, the 3 Convolutional/Pooling Layers with 256 units in the dense layer model performs pretty well. In fact, 7 over 9 of the randomly sorted images are correctly classified.

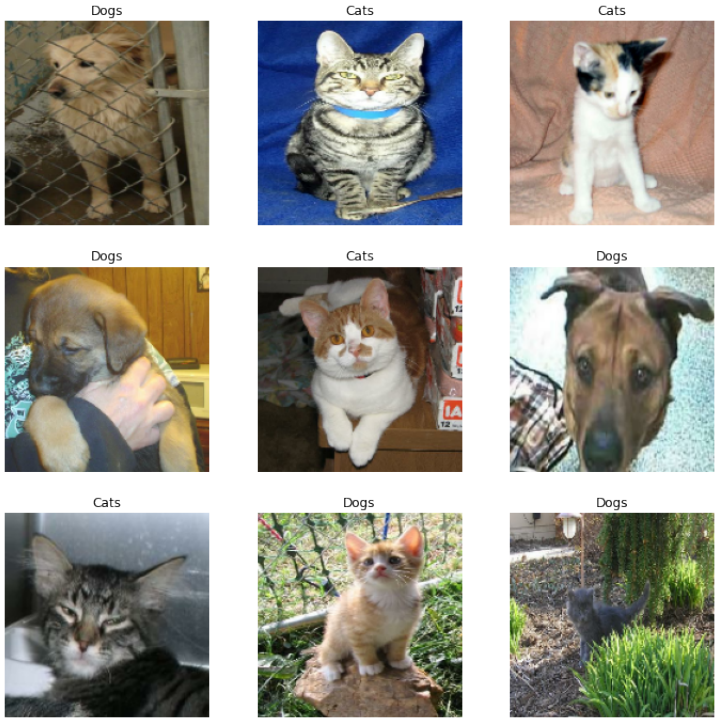

Figure 9: Results of the 3 Convolutional/Pooling Layers CNN Model (Dense layer at 256 unit) over a subset of the testing set images.

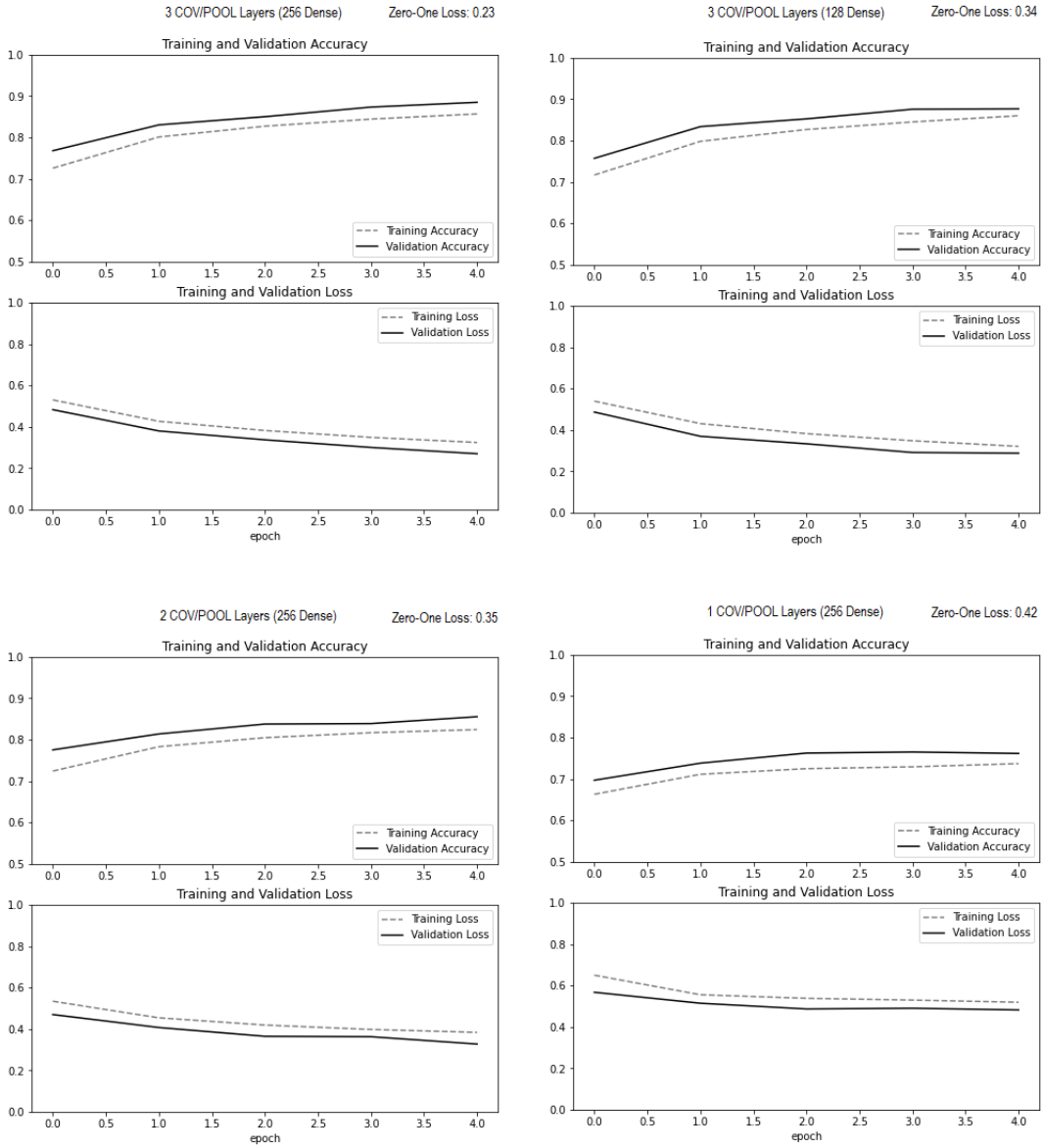

Figure 10: Graphs of 5 Fold Cross-Validation process repeated for 15 epochs. Accuracy and loss rate detail for each model.