Statistical Methods for Machine Learning

Experimental Project on Neural Networks

Paolo Cortis – 09539A

Task Summary

The goal of this machine learning project is to experiment and provide different solutions employing various neural network architectures in the context of a binary classification task. In particular, the dataset at hand is composed of 24946 images representing cats (12476) and dogs (12470), a standard image classification dataset for composing and improving neural network models.

All the different phases of the work are developed in python with the tensorflow machine learning library via Jupiter Notebook.

Data loading and data pre-processing

The first step is to load the dataset into the Jupiter notebook to start working on it. Assuming the images are stored locally, this is achieved via the os python library.

A crucial first step for all types of machine learning applications is to check thoroughly the dataset and apply some kind of pre-processing, in fact, even the best model cannot learn much from a poor, noisy and inconsistent dataset. Given the experimental nature of the project, all images present in the original dataset are being considered, without any kind of manual removal of noisy samples. Anyhow, the images have different sizes and are stored in jpg format. Hence, in this case, pre-processing consisted in a resize of all images and in a color conversion, resulting in 100x100 gray scale images. These procedures aim at aligning the dataset, speeding up the training and reducing the memory requirements without affecting too much the final results. This is achieved via a single custom function adopting the computer vision library openCV.

Not much else is applicable to images in terms of data pre-processing, however data normalization can be employed. It consists of adjusting feature values so to have them on a similar scale, this is proven to increase the performance and the training stability of the model, in particular of neural networks. In this case gray scale pixel values ranging from 0 to 255 are normalized in the 0-1 range.

After this phase, images are stored in an array with shape (24946, 100, 100) and labels are represented in a separate array with shape (24946, 2). Labels are represented via one-hot encoding since classes are not related in any way and distance has no meaning.

Risk estimate via k-fold cross validation

To compute the risk estimate of the different models 5-fold cross validation is used. This allows to train the model on different portions of the dataset, generate five different predictors from a specific network architecture and evaluate the overall model performance by considering the average loss and accuracy across all predictors. This is achieved via a custom function that shuffles in unison the images and their labels, creates the folds, trains the model over four folds and tests it over the remaining one for five times. Of course, the model weights are reset at the start of each training phase, meaning that effectively five different predictors are generated. The function then returns the average loss and accuracy of the predictors as an overall evaluation of the learning algorithm with that fixed structure and hyperparameters.

Note that a neural network is a stochastic learning algorithm, meaning that a certain element of randomness is in play during training. This results in the algorithm generating slightly different predictors even when trained over the same data. This variation however does not invalidate the results presented in this report and the discussion over the ranking of the network architectures in terms of performance.

Model set-up and metrics

Each model is constructed in tensorflow.keras and compiled using the same training loss function and optimizer.

The loss function is crucial for the overall performance of the model since it’s the only element that the algorithm is trying to optimize. In this case binary cross-entropy is used, a standard loss function for the binary classification task.

Equally important is the optimizer, which influences the weight update of the network. In this case Adam is used, an extended version of stochastic gradient descent which has become standard for deep learning and computer vision tasks.

Other relevant training parameters are the batch size and the number of epochs. The batch size defines the number of samples the model passes through before updating the internal parameters. An epoch is a full pass over the entire training set, and the number of epochs defines the number of times the algorithm goes through it during the training procedure.

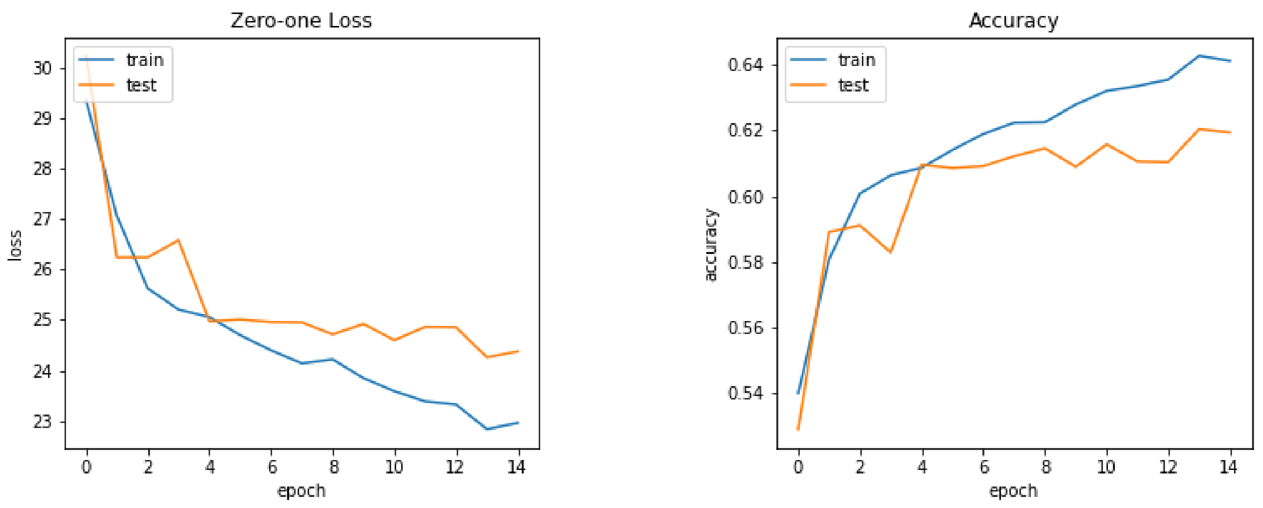
The impact of these parameters over the algorithm performances will be discussed later in the report, for now note that the batch size is set to 64 and the number of epochs is set to 15.

Additional metrics provided when analyzing each model are the accuracy and the zero-one loss. The accuracy returns the number of correct predictions over all samples considered, usually expressed in percentages. The zero-one loss, inserted as a custom metric, simply counts the number of mistakes over the total number of samples considered. Note that the zero-one loss is not differentiable nor convex, so it is not suited as a training loss, however it can still be used as an additional evaluation metric. An historical report of these parameters is kept during the training of each model, this permit plotting the evolution of these metrics over the epochs for both training and test set, ultimately allowing a better evaluation of the model behavior.

Model 1 – Multilayer perceptron

The first neural network architecture tested on the task is a multilayer perceptron. It is a feedforward neural network consisting of six layers. The first layer is a flattening layer, this turns the 2D input array with shape 100x100 into a 1D array with shape 10000, allowing computations for the next layers. The next four layers are basic densely-connected layers, the first has 100 neurons the next have 64 neurons. The activation function of these layers is the Relu function, this is the most commonly used for hidden layers since it avoids the vanishing gradient problem during training. The output layer has only two neurons, corresponding to the two different outcomes of the prediction, here, a sigmoid activation function is used, which outputs two values between zero and one summing up to one, representing the likelihood (can be interpreted as a probability) of the input image belonging to the first or second class (cat or dog). The results:

```
MLP Metrics - 5-fold cross validation estimate:
0-1 LOSS : 24.843589782714844 (over batch size = 64).
BCE LOSS : 0.6621504426002502
ACCURACY : 0.6122026085853577
```



1 - MLP Results

Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend.

The MLP achieves an accuracy of about 61.2% with a zero-one loss of about 24.84 misclassifications over a batch size of 64 samples. A high loss for both train and test is observed leading to a very poor performance. This is a case of underfitting: the model is too simple with respect to the task and it cannot understand the relationship between the input data and the labels. These results are without a doubt very disappointing but not surprising at all. In fact, a simple feedforward neural network is not suited for image classification. For as much as the network tries to capture patterns in the image and associate them to the label, it cannot understand spatial relationships within the image. For example, it may recognize a patter associated with "pointy ear" in a certain location and create a sort of "ear detector", but it cannot see the same pattern when present on a different portion of the image.

Convolutional neural networks

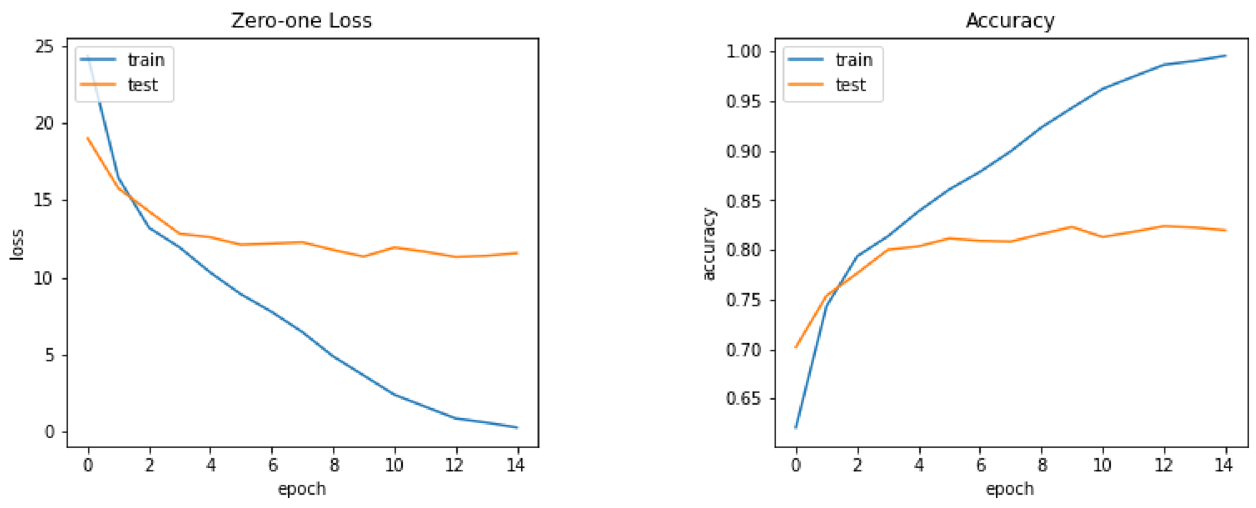
To tackle computer vision problems such as image classification a Convolutional Neural Network (CNN) is the way to go. This type of network is based around the operation of convolution, consisting in the application of a filter to the image in order to create a feature map that identifies the presence of particular features. This allows the network to automatically extract relevant features from the image, in the first layers those may be as simple as edges and corners, later they can be more specific to the task, in this case for examples ears and muzzles. The deep learning model will set not only connection weights between neurons, but also filter coefficients in order to capture as many useful features as possible, these results will pass through a dense layer afterwards. Because the filter slides across the image, convolutional neural network are able to find these details independently from their specific position. The Relu activation function is used for these layers as well, in order to keep only the portion of the input where the kernel is responsive, negative values are set to zero.

This allows clear identification of patterns and reduction of noise, making learning simpler. After each convolutional layer a max pooling layer is necessary to downsample, keeping only the maximum values within the feature map region, thus, the output is a feature map consisting of the most prominent features of the previous one.

Model 2 – Convolutional neural network (A)

The resulting building block of convolutional neural network consists of a convolutional layer, followed by a max-pooling layer; multiple blocks of this kind compose the full network. When defining a convolutional layer the number of filters applied must be specified, as well as the size of the kernel, padding is also used to avoid changing the shape of the image when operating the convolution. For this model, three blocks were used with an increasing number of 3x3 filters (16, 32, 64). Thereafter a flattening layer is required to reduce the dimensionality of the output and feed it to a dense layer composed of 512 neurons. The output layer has, again, just two neurons with a sigmoid activation function. The results:

```
CNN_A Metrics - 5-fold cross validation estimate:
0-1 LOSS : 11.207692337036132 (over batch size = 64).
BCE LOSS : 0.7810935974121094
ACCURACY : 0.8246613502502441
```



2 - CNN_A Results

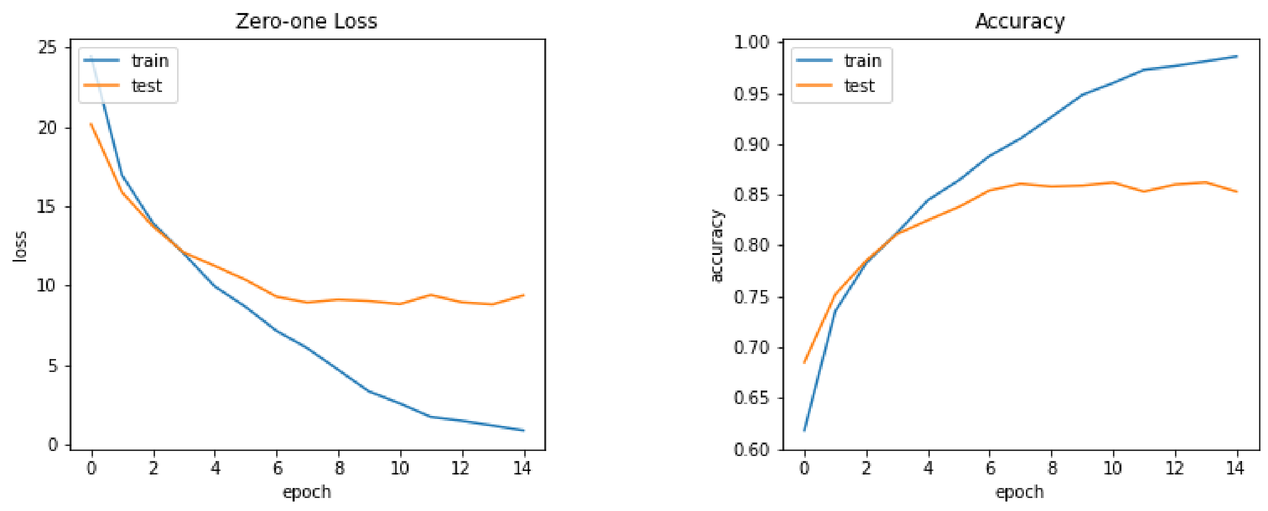
Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

The CNN_A achieves an accuracy of about 82.4% with a zero-one loss of about 11.21 misclassifications over a batch size of 64 samples. Overall a much better performance than the MLP, as expected.

Model 3 – Convolutional neural network (B)

To experiment further with this powerful architecture a second CNN can be defined, this time it is composed of four blocks generating 32, 64, 128, 128 filters, the rest of the network is left unchanged. This should provide the network with extra power and the ability to capture more details. The results:

```
CNN_B Metrics - 5-fold cross validation estimate:
0-1 LOSS : 8.458974361419678 (over batch size = 64).
BCE LOSS : 0.46755859851837156
ACCURACY : 0.8678743243217468
```

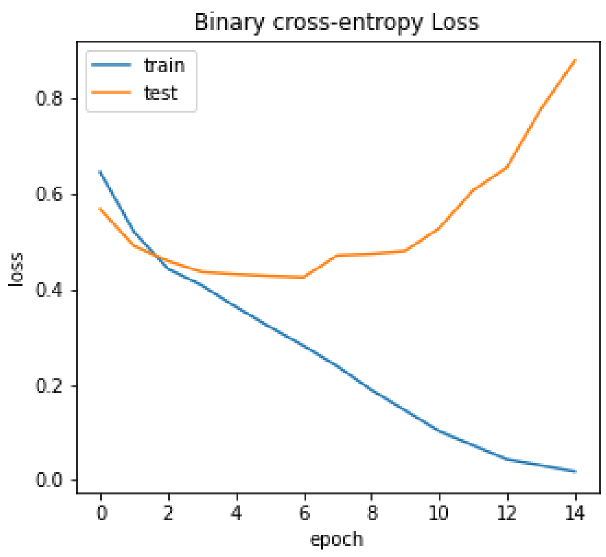


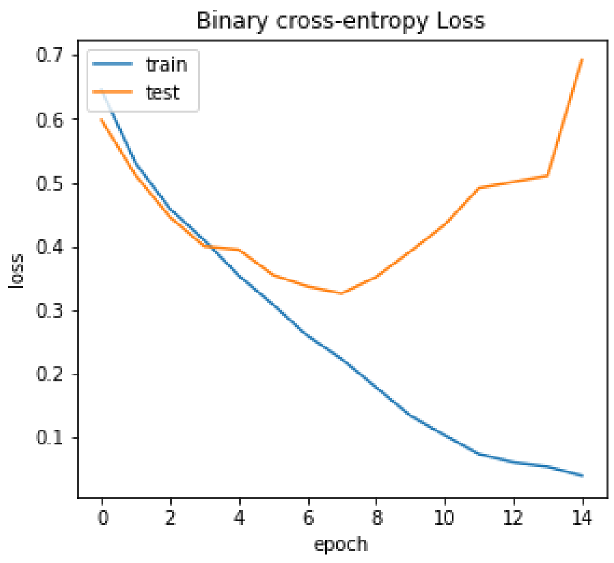
3 - CNN_B Results

Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

The CNN_B achieves an accuracy of about 86.7% with a zero-one loss of about 8.46 misclassifications over a batch size of 64 samples. This shows that more filters, hence more power, were indeed required for the network to better understand the task and, in particular, to better extract important features from the images.

Even if the model is performing decently, both the loss and accuracy curve show clear signs of overfitting. This becomes even clearer when analyzing the plot of the binary cross-entropy loss for the models:





4 - CNN_A and CNN_B Binary Cross-Entropy Loss

Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

These plots show a steady decrease in the training loss, it almost reaches zero in fact, however, the test loss decreases only up to a certain point, then it stops improving, and, in the case of the binary cross-entropy, it starts increasing. This means that the model is learning too faithfully the training set, even the noise, while losing generalization capabilities.

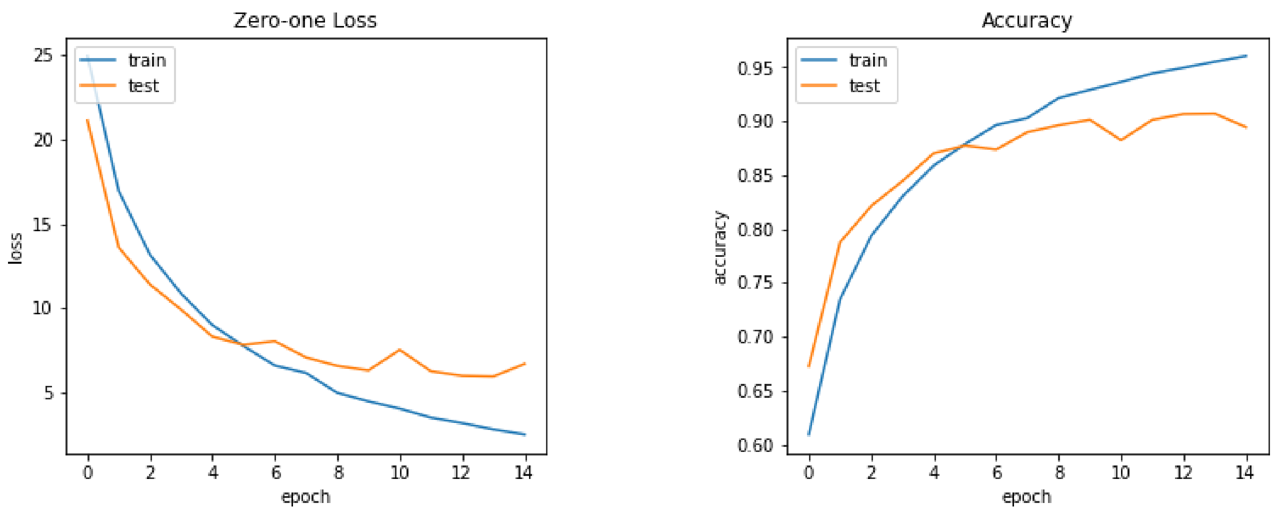
Overfitting happens because the model has more capacity than what is necessary for the task or, in this case, because the model is trained for too long without any regularization. In order to avoid it, an idea could be to re-train the model with a lower number of epochs; in particular, the training should be stopped right when the model stops improving, around the 6-8th epoch in this case. Alternatively, there are a number of so-called regularization techniques, which intend to reduce the generalization error of the model. A possibility is to operate over the value of weights, in particular, in the weight decay technique, the model is forced to work with weights of smaller magnitude this encourages generalization and avoids learning noise.

The solution proposed in this report is dropout. The idea is to partially limit the network capabilities during training by "shutting off" some neurons. This forces the rest of the network to learn different aspects of the input and pick up more robust features, which in turn helps generalization. To implement dropout within the model dropout layers need to be added in the architecture. These layers apply dropout to the input, randomly setting some of them to zero, the rate to which this happens can be specified, for example 0.5 means that there is a 50% chance for each input of this happening. Regarding the application of dropout to convolutional neural networks there is a debate on where to actually position the dropout layer within the architecture. In fact, the effect of a dropout layer is well defined over dense layers but less understandable over convolutional ones. To tackle this discussion, two models are proposed in this project, implementing varying degrees of dropout over the last dense layer and over the convolutional layers.

Model 4 – Convolutional neural network + Dense dropout

Let's start by positioning the dropout layer right before the dense layer, in particular the reference model is CNN_B and the dropout rate is set to 0.2. The results:

```
CNN_dropout_dense -0.2- Metrics - 5-fold cross validation estimate:
0-1 LOSS : 7.105128192901612 (over batch size = 64).
BCE LOSS : 0.3653428316116333
ACCURACY : 0.8887594223022461
```



5 - CNN Dropout over Dense layer, rate = 0.2

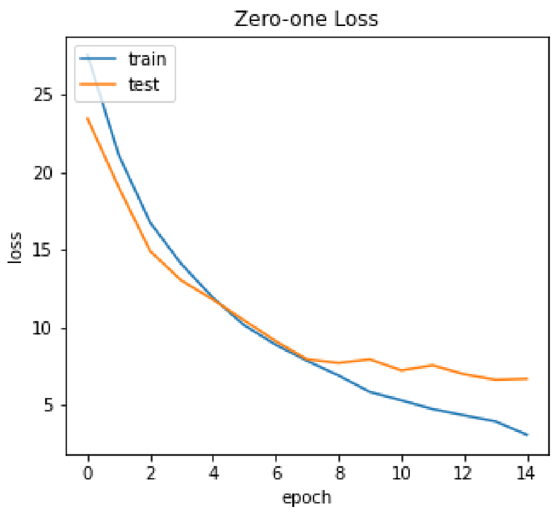
Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

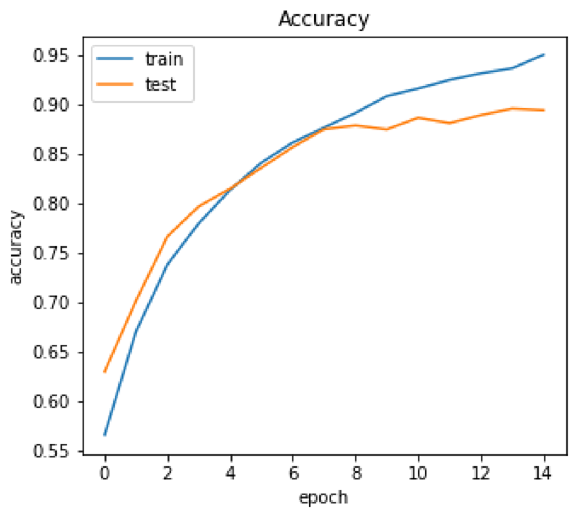
When comparing these results with the ones of CNN_B it is clear that overfitting has been reduced, but not entirely avoided. In CNN_B the test loss stopped improving around the 7th epoch, with dropout this happens around the 9th epoch. Furthermore, the k-fold cross estimate confirms this as the zero-one loss went from 8.46 to 7.10.

In the attempt to completely remove overfitting the dropout rate is increased to 0.5 in this second test. The results:

```
CNN_dropout_dense -0.5- Metrics - 5-fold cross validation estimate:
```

```
0-1 LOSS : 6.57179479598999 (over batch size = 64).
BCE LOSS : 0.2822296440601349
ACCURACY : 0.8973781704902649
```





6 - CNN Dropout over Dense layer, rate = 0.5

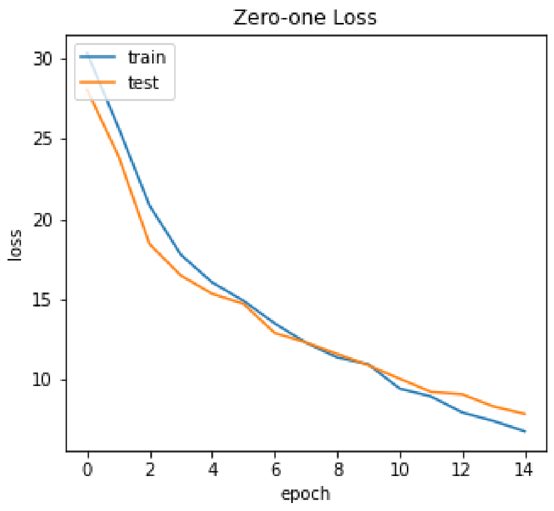
Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

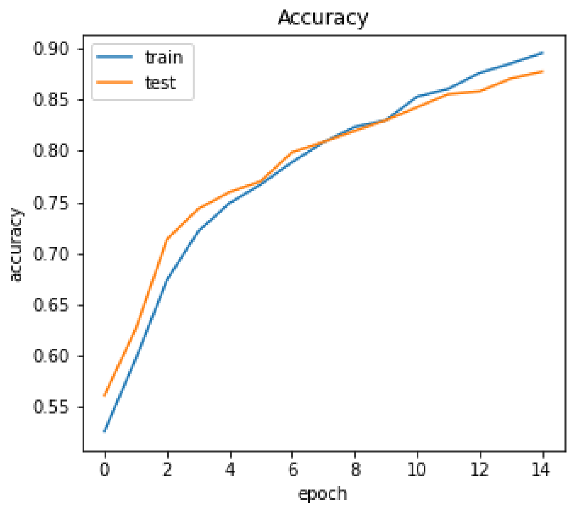
The results improved, the model achieves an accuracy of about 89.7% with a zero-one loss of about 6.57 misclassifications over a batch size of 64 samples. Overfitting has been further limited as shown by the plot, even if the loss and the accuracy slow down in their improvement, they don't get worst.

Model 5 – Convolutional neural network + Convolutional dropout

To experiment further with dropout and understand his effects based on the layer positioning a second dropout model is defined. This time four dropout layers are added to the CNN_B, each after the max-polling layer, with a lower dropout rate of 0.2. The results:

```
CNN_dropout Metrics - 5-fold cross validation estimate:
0-1 LOSS : 6.989743614196778 (over batch size = 64).
BCE LOSS : 0.2681012094020844
ACCURACY : 0.8904432535171509
```





7 – CNN Dropout over convolutional layers

Graphs referring to a single step of the k-fold CV, graphs from other steps show a similar trend

These graphs show no clear sign of overfitting and the evaluation metrics of this model are comparable to the previous one with about 89.0% accuracy and about 6.98 misclassification over a batch size of 64.

As final thoughts about the dropout technique it is fair to say that, based on these results over the binary classification task, this method proves to be extremely efficient at limiting overfitting. In particular, when considering convolutional neural networks, dropout is effective when applied over the latter dense layer. Furthermore, it has been shown that when dropout is used over convolutional layers with a lower rate, specifically after the max pooling of each block, it can still provide good results.

Hyperparameters

Now let's discuss the hyperparameter choises made and their influence on the overall performance of the algorithm.

Number of hidden layers and neurons

The number hidden of layers and neurons per each layer define the architecture of the network and influence its overall predictive capabilities. There is no specific rule concerning the choice of these values but the general guideline is: the more complex the problem, the deeper and more powerful the network needs to be, hence, the more neurons and layers it requires. However, it is known that, in case more power is needed (underfitting), it's always better to add more layers before adding more neurons to each layer.

Activation function

The activation function is what introduces non-linearity into the model, allowing the learning of non-linear functions. It has already been mentioned that the Relu function is ideal as an activation function for hidden layers and, for binary classification, sigmoid is a good choice for the output layer.

Loss function and optimizer

As previously stated, the loss function is critical for the correct training of the model and its choice depends on the problem at hand. In this case binary cross-entropy is used, a standard loss function for the binary classification task. The same can be said about the optimizer, which determines the way in which the network weights are updated. Adam was used for these networks since it is a standard optimizer for deep learning and computer vision tasks. As for the learning rate, the value determining how much weights are updated with respect to the estimated error, the default of 0.001 was maintained as larger values resulted in the model missing the optimal weight values and consequent poor performance.

Number of epochs

The number of epochs represents the number of times the whole training data is shown to the network during training, this parameter alone can determine whether the model underfits or overfits. In fact, a too-small number of epochs results in underfitting because the network did not have the time to properly learn the data and its patterns. On the other hand, too many epochs lead to overfitting as the model can predict the training data very well, noise included, but cannot understand new unseen data. There are no hard rules for selecting the number of epochs, and there is no guarantee that increasing the number of epochs provides a better result than a lesser number. However, as stated before in the report, in case of overfitting, stopping the training process around the epoch where results stop improving is a sound reasoning. Since an epoch requires the network to go through the whole dataset, the more epochs there are, the more time training will require.

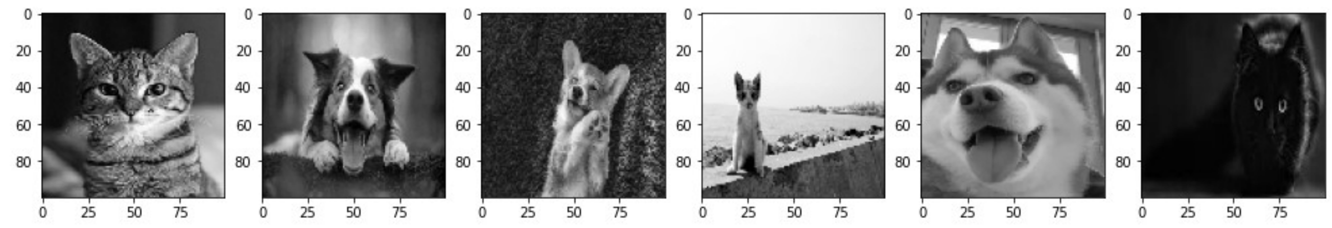
Batch size

The batch size is the number of samples given to the network before the error is computed and the network parameters are updated. This parameter can also greatly affect the final performance of the model. Since the backpropagation algorithm is in use, the weights adjustments are proportionate on how wrong the predictions of the samples in the batch were. The more training examples considered, the more correct the gradient estimate and the adjustments, the higher the likelihood of a performance increase. On the other hand, by considering fewer training examples, there will be a higher number of smaller, less precise, adjustments, which, nonetheless, can result in faster learning and a more robust model. This is an essential trade-off, in fact considering the whole dataset allows to tune parameters a few times but in a more precise way, whereas evaluating smaller batches guarantees many more updates, even if slightly more noisy. Anyhow, it is known that too large of a batch size will lead to poor generalization. This makes sense as more noisy but frequent tuning favors a sort of regularizing effect. In terms of computational efficiency, a small batch size requires much longer and the training procedure demands less memory, on the other hand, higher batch size values allow for a faster training but higher memory is required to fit all these examples.

These results and considerations were derived from testing with different batch sizes over the previously defined networks, in particular the differences in training time were noticeable as well as the memory requirements. Training these convolutional networks over a 64 batch size would require from 5-10 minutes, when lowering the batch size to 4 over 30 minutes were needed with no noticeable improvement on the performance of the model. Setting the batch size to higher values like 256 or 512 would make training impossible with the current gpu (even if above average) as the memory requirements were not met. In conclusion, under no computational constraints, it is advised to start with a smaller batch size and steadily grow it through training, in this case 64 was found to be the optimal value.

Predicting

The goal of this project was to experiment with different neural network architectures and understand how to conduct statistically sound experiments and evaluations using k-fold cross validation. That said the last models almost reached an accuracy of 90%, it's now time to classify new unseen images. To do so a custom function for prediction was defined, taking in input the folder containing the new images and the model to use, it normalizes the data and returns the predicted labels output by the model. The results:



```
Predictions: ['cat', 'dog', 'dog', 'cat', 'dog', 'cat']
```

8 - Predicting new images

Striving for more

Other interesting options to better the model not explored in this project are data augmentation and transfer learning.

Data augmentation looks at the problem of overfitting with a different prospective: instead of changing the model, or lowering its complexity, one simply provides more data to help a correct parameter tuning of the network. In fact, overfitting is also caused by a complex model, with many parameters, but too little data to tune them, which implies an erratic behavior of parts of the network. When gathering more samples is not an option, slightly modified copy of existing data can be included in the dataset, in the case of images, changes in position, scale, rotation and resolution can be considered, as well as the addition of noise and so on. The goal is to enlarge the dataset with pre-existing data.

Transfer learning instead makes use of a pre-trained network, proved to have great results over a similar task, and tunes it for the job at hand. These adjustments, in case of convolutional neural networks, are mostly operating over the final layers, for example, it’s common practice to freeze the feature extraction portion of the network and tune the output layer, e.g. going from a face recognition task to an image classification one. This is motivated by the observation that the earlier layers of a convolutional network extract more generic features (edge, color, blob detectors) that are in fact useful for many tasks, whereas latter layers become progressively more specific to the details of the classes contained in the original dataset. In this case an option would be to use the feature extraction part of a different model and add a new classifier part that is specifically made for the cats and dogs dataset. Specifically, the weights of all of the convolutional layers fixed during training can be kept, and new fully connected layers only are trained, in order to learn to interpret the features extracted from the model and make a binary classification.

References

These were consulted during the development of the project, guided the construction and deployment of the models and helped forming the more general considerations reported.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep learning, MIT Press, 2016, chs. 6-9

Tensorflow documentation and tutorials -
https://www.tensorflow.org/tutorials/images/classification?hl=en

Jason Brownlee, Recommendations for Deep Learning Neural Network Practitioners, 2019