Computer Vision.
In the name of Deep Learning

Ignacio Garrido Botella
Abel Rodríguez Romero

KU Leuven — June 10, 2019

Contents

Introduction

In the recent years deep learning is drastically changing image processing, bringing with it numerous new applications. For example, it has transformed the way problems like autonomous driving or image creation are faced. In this project some of the most relevant topics for which deep learning is used, such as feature extraction, image classification and segmentation will be studied.

The models known as auto-encoders, in particular those using convolutional networks, as well as their possible applications, will be analysed first. The auto-encoders have been successfully applied to dimensionality reduction tasks, noise elimination and even as part of generative models. Among these, the behaviour of the auto-encoders for reduction of dimensionality in the case of image processing will be seen. The use of auto-encoder elements, particularly the encoder, for the extraction of features as part of more complex networks will also be studied.

Another application in which neural networks have increased their use in recent years is image classification, with great relevance in both academic literature and industry. Approaches to this problem will be studied using convolutional networks, in which different techniques will be applied to improve the performance of the systems analyzed. This comparison will allow to examine the impact of various architectures, hyperparameters and training techniques in classification tasks.

In addition, image segmentation based in auto-encoder's architecture has proven to have a great performance. Thus, pixel-wise binary segmentation of images will be studied in the last section of this report. For so, several architectures based in the auto-encoder's structure will be designed, and two different loss functions, binary cross entropy and dice coefficient, will be tested for each of the networks. Additionally, the performance of the well-known segmentation network, U-Net [1], will be evaluated and compared to the rest of the designed networks.

Data processing

For the elaboration of the training and validation sets, some modifications were made to the proposed script. Firstly, the spatial dimensions of all the images to be used were set at 128x128 pixels, in order to find a compromise between the computational impact of processing larger images and using data with sufficient quality to appreciate the distinctive features of the image.

The classes considered were also modified. It was chosen to use 4 classes for the development of the different tasks described in the document, with the exception of the segmentation section, which will be detailed later. The classes chosen are *aeroplane, dog, bird* and *car*. The choice of these classes was motivated by the differences in the features of the objects represented by these classes, to facilitate the tasks of classification.

The models used generally require high volumes of training data to exhibit good performance, so it was attempted to increase the amount of training images. For this reason it was made the decision to use a different distribution to the one present by default in the training and validation sets, which contained approximately the same amount of images in both cases. For this purpose, all the available images of both sets of the classes to be used were gathered and a redistribution was made to a new training and validation set taking random samples. The ratio chosen was 80:10:10 between training, validation and test respectively, commonly used in this type of applications, and that presents a good compromise between having more material for training and enough validation data to draw conclusions about the performance of the models reliably. As a result of this approach 1832 training images, 232 validation images and 232 test images were obtained. As mentioned above, the use of this modified dataset was used consistently throughout the different cases analyzed.

1 Auto-encoders

1.1 PCA vs auto-encoder

PCA is a well-known technique used for dimensionality reduction of data. The basic idea is orthogonally projecting the data on those directions in which there is more variation, i.e., more information can be retrieve. This is done by orthogonally projecting the data in the directions of the eigenvectors with higher eigenvalues (the bigger is the eigenvalue, the more "information" of the original data is described in the direction of its corresponding eigenvector).

Auto-encoders can be used in a similar way to reduce the dimensionality of data, as well. An auto-encoder is a neural network that is trained to learn the principal features of the input data. The basic operation of the auto-encoder consists in making an approximation to an identity function, that is, reconstructing at the output the data present in the input, using an intermediate abstract representation, which, in case of being inferior in terms of dimensions to the input data, will allow for the dimensionality reduction. In this case, two different parts can be identified in the auto-encoder: the encoder and the decoder. The encoder will transform the input into an intermediate representation, called code, while the decoder will reconstruct at the output the initial data starting from the code.

The main difference between PCA and auto-encoders is that PCA does a linear transformation of the data, while an auto-encoder can do non-linear transformations of the input data if the activation function is a non-linear function, like sigmoid. Furthermore, an auto-encoder with one layer and with n components in the code layer can span the same subspace than the n principal components of a PCA if it uses linear activation functions. However, if the auto-encoder uses a more complex structure and non-linear activation functions, it can extract much more complex features, as shown in Figure 1, taken from [2], in which numbers of the MNIST set are reduced to two principal components.

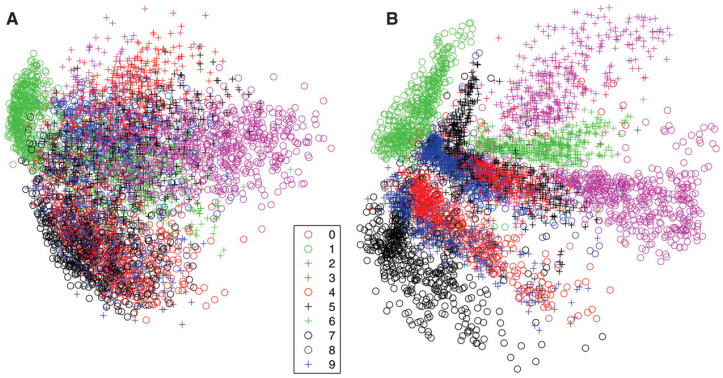

Figure 1: A: PCA - B: 784-1000-500-250-2 auto-encoder [2]

1.2 Convolutional auto-encoder for dimensionality reduction of images

In this section an attempt will be made to implement a functional auto-encoder to analyze its behavior and design variables, as well as the results obtained. As stated above, the use of auto-encoders is linked to applications of dimensionality reduction. In this context, the use of auto-encoders using convolutional network architectures, known as convolutional auto-encoders, has been successfully used in unsupervised learning for feature extraction in a hierarchical manner, an approach that will be used later in classification tasks [3]. The use of convolutional layers, by considering spatial information of the image, usually provides better results in these cases in comparison with the alternatives using fully-connected networks.

1.3 Architecture

Auto-encoders present a symmetrical architecture between the encoder and the decoder, since the latter performs the inverse function to the first. Due to this, the design decisions are made on the encoder, and then reverse operations are made for reconstruction. Due to the use of a convolutional architecture, the traditional approach comprised of subsequent layers of convolution and pooling, which has already been employed in various applications [4, 5], will be utilized.

The chosen architecture is similar to Zhang's proposal in [6]. It is composed of three convolutional layers, each followed by a max pooling layer, and the corresponding inverse layers of transposed convolution for the decoding process. The decoder features an additional transposed convolution layer for image reconstruction to three color channels. Figure 2 further illustrates the complete architecture.

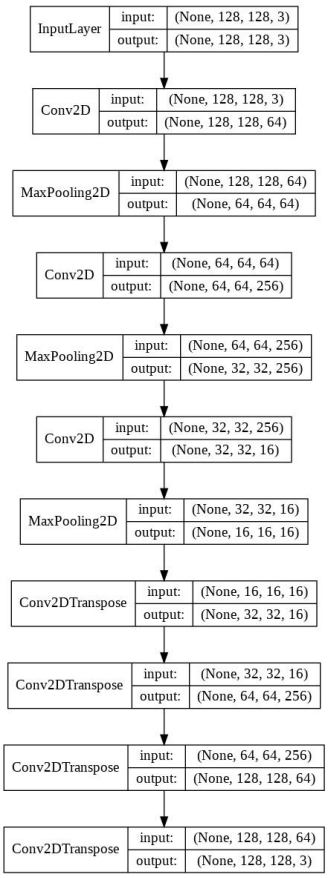

Figure 2: Proposed architecture of auto-encoder

The ReLU activation function was used in all layers as well as filters of dimension (2,2). The number of filters of the convolutional layers was chosen through various tests, trying to find a compromise between reconstruction quality and the ratio of the compression of the original image to the generated code, which is given by the values of the parameters used and the number of layers. The pooling layers with the stride used present at their output data with spatial dimensions half of the dimensions at the inputs, and the convolutional layers return the same spatial dimensions at their input, but with the number of filters used in the layer.

With this data, the size of the code will be given by a total of 4096 elements, which are given by the output of the encoder, 16 filters of size 16x16. With these values, and considering that the dimensions of the image at the input are given by a total of 49152 parameters, corresponding to an image of 128x128 with three color channels, a compression in a factor of 12:1 is obtained. The compression in this case is lossy, as is logical, since it operates on data already compressed with JPEG compression, but allows to identify the objects present in the image.

1.4 Training

For the training, the training images were defined as input and target of the model, and the Mean Squared Error (MSE) was used as an error metric to determine the distance between the reconstruction and the original image. The optimization algorithm chosen was Adam [7], which in the preliminary tests performed offered good results. Early stopping and adaptive learning rate mechanisms present in Keras, external to the optimization algorithm were used in cases where it was observed that there was no improvement after a certain number of epochs. Additionally, in each iteration, the order of the input images was modified.

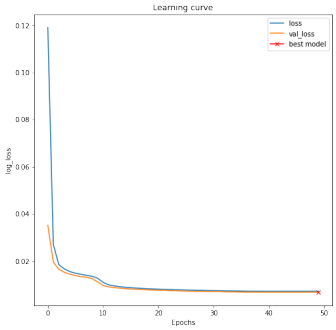

Figure 3: Model loss. Convolutional filters 16-32-16.

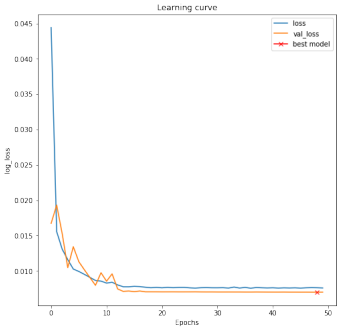

Figure 4: Model loss. Convolutional filters 32-128-16.

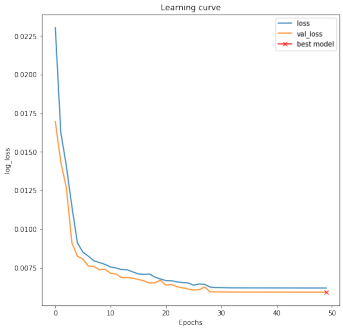

Figure 5: Model loss. Convolutional filters 64-256-16.

Various tests were carried out to evaluate the impact of the number of filters on the final results, some of which are shown in Figures 3-5. The results obtained in the validation set were similar, although it was observed that higher values corresponded to slightly better performance. However, the differences in the number of epochs considered were very small.

1.5 Results

The results obtained with the best model, corresponding to the use of 64, 256 and 16 filters in the convolutional layers respectively, present an average MSE in the validation set of 0.0055. With these values, two examples of auto-encoder reconstructions are shown below.

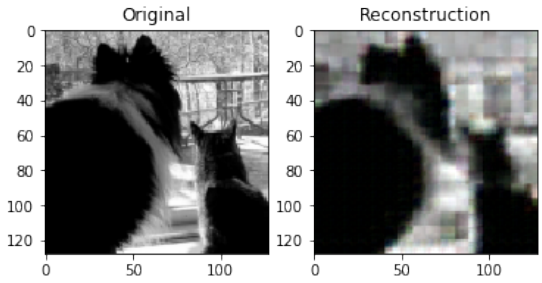

Figure 6: Example of auto-encoder reconstruction.

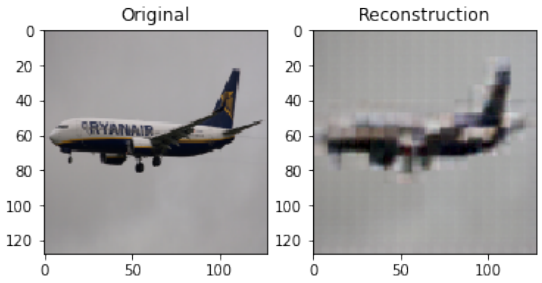

Figure 7: Example of auto-encoder reconstruction.

It can be seen in the images the loss of resolution in the reconstructed images, although the general shapes of the objects can be inferred. In the tests carried out, no better reconstruction qualities were achieved that were not associated with a code with a greater number of parameters in the encoder stage.

2 Classification

In this section different approaches for carrying out classification tasks using CNNs will be discussed, based on elements present in the auto-encoder architecture presented above. In general, the architecture of a deep convolutional network for classification consists of two parts. The first layers, composed by series of convolutional and pooling layers as previously explained, are used to elaborate a feature map with the most relevant elements of the image in question, that is, they are used as a feature extraction mechanism. The second part of the network, comprised of dense layers, is in charge of finding the relations between the extracted features and assigning them the desired classes [8]. This type of architecture, consisting of convolution and pooling layers first and then dense layers at the end, has been described extensively in scientific literature, and used in networks such as AlexNet [9] or VGG-16 [10].

This approach will therefore be used here, albeit on a smaller scale than that of these networks. The first part of the developed auto-encoder, i.e. the encoder, will be used to implement the feature extraction of the images, and at its output dense layers will be connected to perform the classification tasks.

In all cases it is intended to implement a classifier that can distinguish between 4 classes. The problem to address is a complex problem, which without a very good optimization of hyperparameters or low volumes of data does not show good performance, as noted in the preliminary tests conducted. The results obtained, although superior to random guessing, were far from those present in the state of the art networks or from presenting low error values. However, the aim of the project is more focused on analyzing different approaches used and their impact.

With the conditions imposed on the project, the volume of data present in the dataset is inferior in comparison to what is normally used. This situation leads to weak system performance and overfitting in many situations. Therefore, in all the cases analyzed, strategies were used to combat these adverse effects. In the first place, due to the impossibility of obtaining more data for the training phase, the dataset was restructured to include more training images, as explained in the first part of this document, and a mechanism was implemented to generate synthetic data from the original images. This mechanism, known as Data Augmentation, modifies the training images to generate new ones based on rotations, reflections, changes in brightness and color, etc., which the network identifies as different instances, and which has been shown to present good results [11]. This has been implemented using functionalities included in Keras, through its ImageDataGenerator class, and and example of a generated image can be seen in figure 8. Other mechanisms have included the use of Dropout [12] and Batch Normalization [13] layers to combat overfitting when a significant difference was observed between the values obtained in the training and the validation set. The dropout rate of each layer was adjusted based on observations, but the values used are within the interval [0.2,0.6] being 0.5 the default rate.

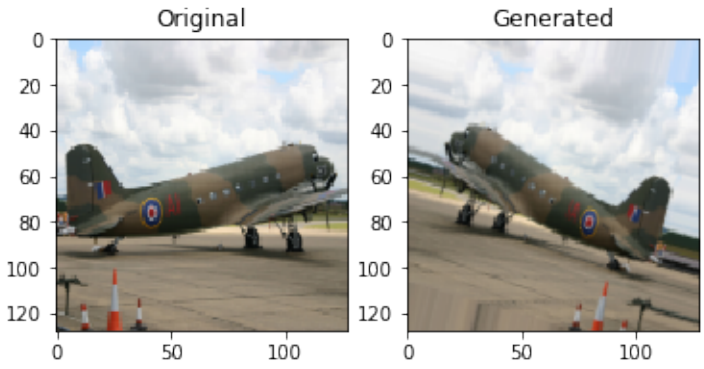

Figure 8: Example of generated image using Data Augmentation techniques.

2.1 Sequential classifier

The classifier used in these experiments has two distinct parts as discussed above. The first one is the encoder, in charge of extracting the features from the images. At the output of the encoder a flatten layer is placed, so that the data can be processed by the 3 dense layers thereafter. At the input of each dense layer a dropout layer has been placed in order to reduce overfitting. The resulting global architecture, which will be used in the following three experiments, is shown in Figure 9.

The output of the classifier is given by the output of the last dense layer. This layer has 4 outputs, according to the representation of the labels, so that the output is represented by a binary string in which the bit in the column corresponding to the class to which the instance in question belongs is at 1, while the rest are at 0. This representation is known as one hot encoding. In views of the use of this categorical format, the categorical cross entropy is used as a function of loss.

As an activation function, the softmax function is used, given the multiclass scope of the classifier, and due to the fact that the values at the network output of the softmax function form a vector with the probability distribution of the possible classifications, that is, its output can be considered as the certainty that the instance belongs to each class. In the rest of the layers, the activation function used was ReLU, and Adam is again used as an optimizer with the mechanisms of adaptive learning and early stopping.

In the first experiment considered, the encoder will be used as a feature extractor and the dense layers present at the end will be trained for classification. The encoder used will be previously trained and its weights will have been selected as fixed, therefore, when training the net only the dense layers will be modified.

The idea behind this layer-based training approach is that it simplifies the optimization problem, and therefore the training process, by allowing an intelligent, more advantageous initialization of the parameters [14]. However, in this case no adjustments can be made on the first part of the feature extractor, so it is necessary to operate based on the results of an encoder not trained for this case in question. Later the impact of a complete training process with a pre-trained encoder will be analyzed.

The results obtained are shown in figure 10. It can be noted that in this case the performance of the system is not on par with the models presented in the literature, as is natural. In terms of accuracy, the best model obtained 62.22% accuracy in the validation set, quite far from the networks currently used, although higher than random guessing with balanced classes. For the reasons stated above, the classification must be done with the code generated at the output of the encoder, which may not be optimal for extracting the most relevant features for classification. The high fluctuations are influenced by the use of the Data Augmentation mechanism, which presents different images in each epoch.

In the second experiment with this architecture the previous encoder training approach, similar to the first experiment, was used. In this case, however, it is possible to adjust the parameters of the first layers, with the result that the whole network can be trained. This can be approximated to a previously trained system with a fine-tuning phase.

The performance using this method is superior to the previous case, presenting an accuracy in the validation set of 65.05% for the best model. This is the expected behavior, since the initialization of the weights of the auto-encoder layers is not fixed as in the previous case, and the layers can be trained, so the parameters are closer to local optima, and can converge to those values. The results in terms of accuracy are shown in figure 11.

The last experiment analyzes the case in which all parameters are randomly initialized at the beginning of the training phase. The results obtained, presented in figure 12, resemble the previous case, although they are slightly better, obtaining 66.38% accuracy in the validation set. This is due to the fact that the number of parameters to be adjusted is not very large quantitatively, so the optimization algorithm allows converging to similar optimal states after a few iterations. In all the cases the early stopping mechanism was used to avoid non relevant iterations.

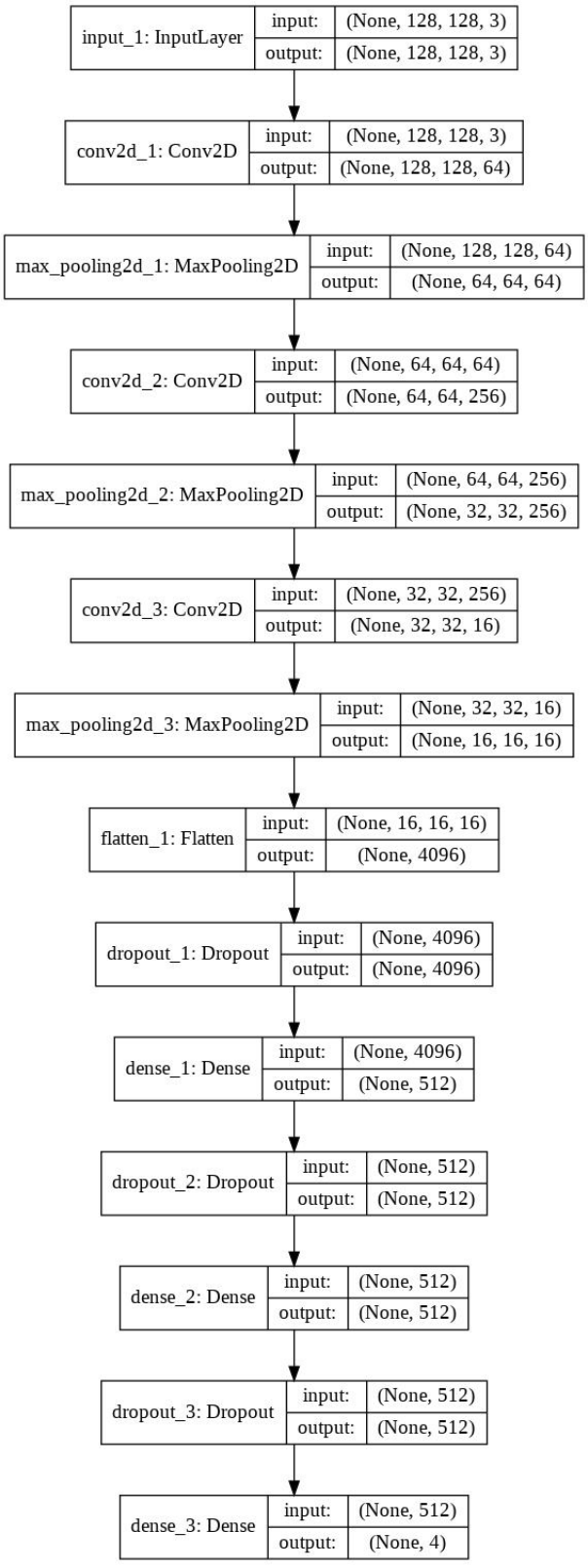

Figure 9: Architecture for the sequential classifier.

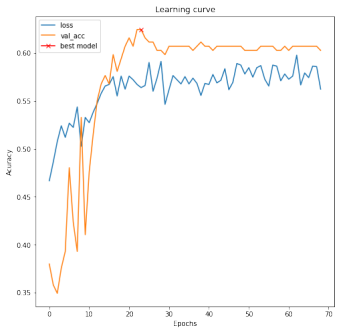

Figure 10: Learning curve. Sequential classifier with pre-trained non-trainable encoder.

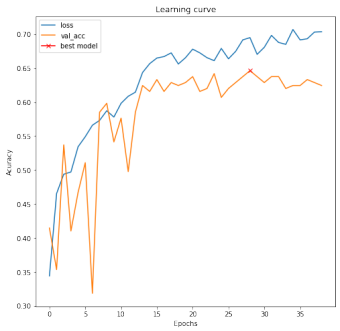

Figure 11: Learning curve. Sequential classifier with pre-trained trainable encoder.

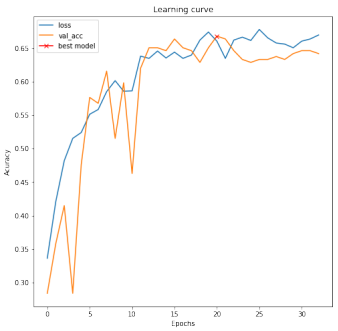

Figure 12: Learning curve. Sequential classifier with random initialization.

2.2 One vs. All (OVA) classifier

In order to study whether it was possible to improve the classification obtained with the previous models, it was decided to implement a more complex architecture composed of various binary classifiers. It was decided to use a One vs All (OVA) classifier for this task, which is in most cases easier to implement than other composite architectures, and can present good results.

An OVA classifier consists of separately training several binary classifiers capable of recognizing only one class, trained with a different class each, and then predicting on the basis of the combined result of these. This approach is known as binary relevance [15]. For this purpose, 4 binary classifiers were implemented in the case in question, one per class. The objective is for this composite model to make predictions with results at least similar to those analyzed in the previous cases.

The output of each classifier is given by a dense layer of only one output with sigmoid activation function. The rest of the architecture is similar to the sequential classifier presented above, formed by dense layers with ReLU activation function. Figure 13 illustrates the resulting architecture. It is observed that despite the apparent complexity, it is only 4 sequential classifiers in parallel, followed by the dense layers mentioned. For simplicity, it is opted for simply replicating the same architecture for all models, although better results can be obtained by performing a deeper tuning of the different architectures and values of hyperparameters.

In order to train these classifiers, it was necessary to generate for each one a new set of labels composed of the categorical representation of each class to classify, that is to say, only the column of the class in the categorical representation. Because the number of instances belonging to a particular class is less than the rest (each class represents approximately 25%), the dataset used for training is unbalanced. This can cause the binary classifier not to generalize and simply always classify the majority class as correct. To avoid this, it was decided to balance the dataset by taking a number of random samples of the negative class that was similar to the positive class, an approach known as undersampling [16].

Subsequently, the four classifiers were trained separately with the vector of the corresponding class. The learning curves for the four binary classifiers are shown in Figures 14-17. As expected, the performance of each one separately was higher than those presented above. Although there are differences in the accuracy obtained for the best model, in general the values exceed 70%. This is due to the fact that the problem of binary classification is less complex, and the feature extractor of each model presents more specialization. The differences in the number of epochs are due to the early stopping mechanism.

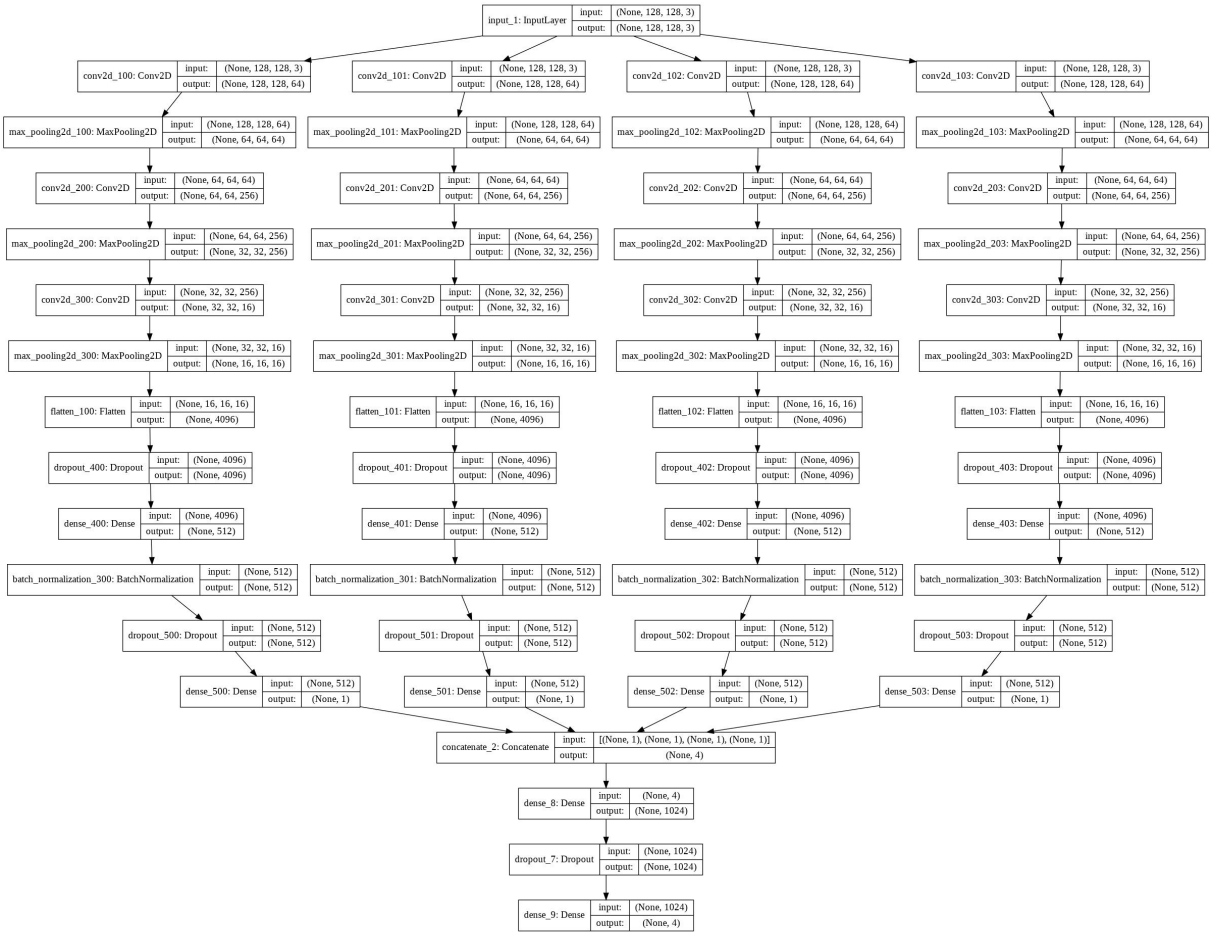

Figure 13: Architecture for the OVA classifier.

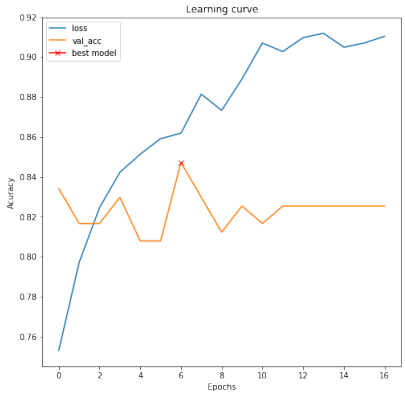

Figure 14: Learning curve. Binary classifier of class *aeroplane*.

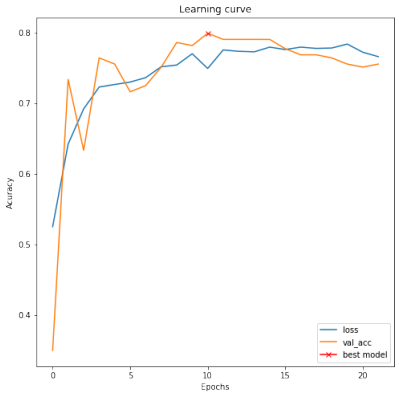

Figure 15: Learning curve. Binary classifier of class *car*.

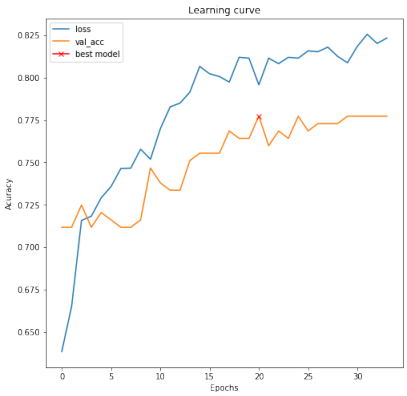

Figure 16: Learning curve. Binary classifier of class *dog*.

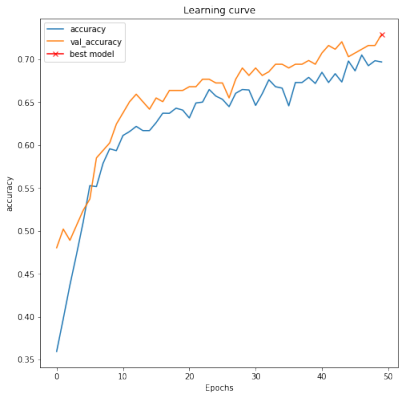

Figure 17: Learning curve. Binary classifier of class *bird*.

In a similar way to the experiments developed with the sequential classifier, different cases are studied depending on the parameters that can be trained in the network. First, only the parameters associated with the dense output layers are trained, leaving the binary classifiers fixed. In the second experiment, the fine tuning approach is analyzed, that is, it is possible to train the whole network, with the binary classifiers being previously trained. Finally, the case employing random initialization and training of all network parameters is evaluated. The results obtained are shown in figures 18, 19 and 20 respectively.

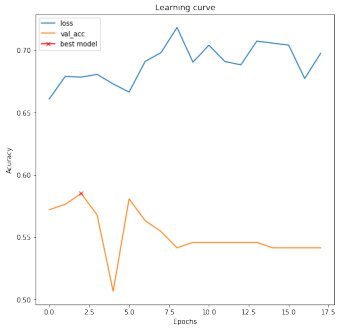

Figure 18: Learning curve. OVA classifier with pre-trained non-trainable binary classifiers.

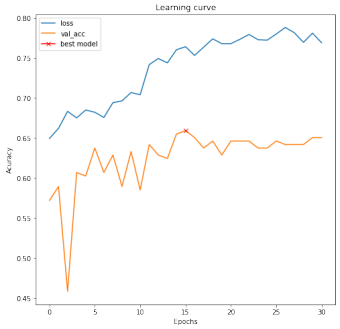

Figure 19: Learning curve. OVA classifier with pre-trained trainable classifiers.

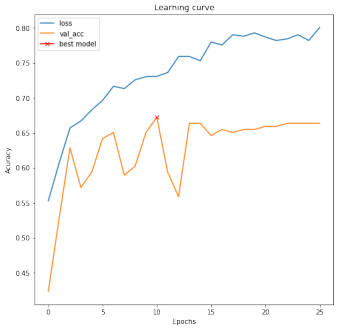

Figure 20: Learning curve. OVA classifier with random initialization.

2.3 Results

In order to compare the approaches used, the best models were analyzed in the test set. In addition, they were compared with a model using the feature extraction part of the VGG16 [10] network previously trained with the parameters present in Keras of the ImageNet dataset. To this model two dense layers were added at the output, one for feature recognition and another layer for the prediction of the four classes. The resulting network was trained in two cycles. In the first one, the values of the weights of the VGG layers were kept fixed, and only the dense layers were trained, and in the second one a fine tuning phase was carried out training the whole network.

Table 1 shows the values measured in the test set for the following models:

- **SF**: Sequential classifier with pre-trained non-trainable encoder.
- **ST**: Sequential classifier with pre-trained trainable encoder.
- **SR**: Sequential classifier with random initialization.
- **OF**: OVA classifier with pre-trained non-trainable binary classifiers.
- **OT**: OVA classifier with pre-trained trainable classifiers.
- **OR**: OVA classifier with random initialization.
- **VGG16**: VGG16 feature extractor with dense layer on the given set of classes.

	SF	ST	SR	OF	OT	OR	VGG16
Accuracy	0.5347	0.6014	0.6128	0.5924	0.6277	0.6049	0.5124

Table 1: Accuracy on the test set of the evaluated models.

It can be observed that in general the performance is similar to that observed in training in terms of generalization. The performance of the approach using the VGG16 network extractor feature is worse than that observed in the rest of cases, probably due to the fact that it is a model trained in a much more complex dataset, with more classes, and with the relatively small data volume of the dataset used it is more challenging to find relationships between the identified features.

3 Segmentation

The main idea of image segmentation consists in recognizing objects and assigning each of the pixels of the image to one of the recognized objects or background. In the recent years and with the rise of deep learning, fully convolutional neural networks, based in the encoder-decoder architecture, have been used to perform image segmentation. For example, one of the most well known image segmentation networks is the one created by Ronneberger et al., U-Net [1].

In this section of the project, a pixel-wise binary segmentation based in the auto-encoder's architecture of section 1 will be studied. For doing so, all the images of the segmentation dataset have been transformed to binary images, distinguishing only between foreground and background and not between objects. The training set is composed by 1190 images in gray-scale with their corresponding segmented images, all of them of size 128x128, being the rest of the images used as test and validation sets.

A neural network for binary segmentation based in the auto-encoder architecture designed in section 1, and in the exponential increment of number of filters' principle in which is based U-Net has been implemented. Notice that U-Net has a similar encoder-center-decoder structure as well. The architecture of the network is the one shown in Figure 21. The encoder part of the network is composed by three convolutional layers, each of them with filters of size 3x3, stride of 1, zero padding of 1 (keep the same size of the image at the output) and with ReLU activation function, being each of them followed by max-pooling filter that do a downsampling of 2x2. The center part of the network is composed by a convolutional layer, with a filter of size 3x3, stride of 1, zero padding of 1 and with ReLU activation function. The decoder part of the network is composed by three convolutional layers , each of them with filters of size 3x3, stride of 1, zero padding of 1 and with ReLU activation function, being each of them followed by upsampling layers of size 2x2. The output layer is a convolutional layer with one filter of size 3x3, stride of 1, zero padding of one and with sigmoid activation function. In addition, and due to the high overfitting because of the small size of the training set, it has been added batch normalization layers to each convolutional layer and dropout layers with a dropout rate of 0.6 after the encoder's max-pooling layers. Furthermore, in order to avoid overfitting it has been tested L2 regularization in the convolutional layers. However, the performance of those networks without L2 regularization was always better. With a finer tuning of the hyperparameters of L2 regularization, better results could have been achieved.

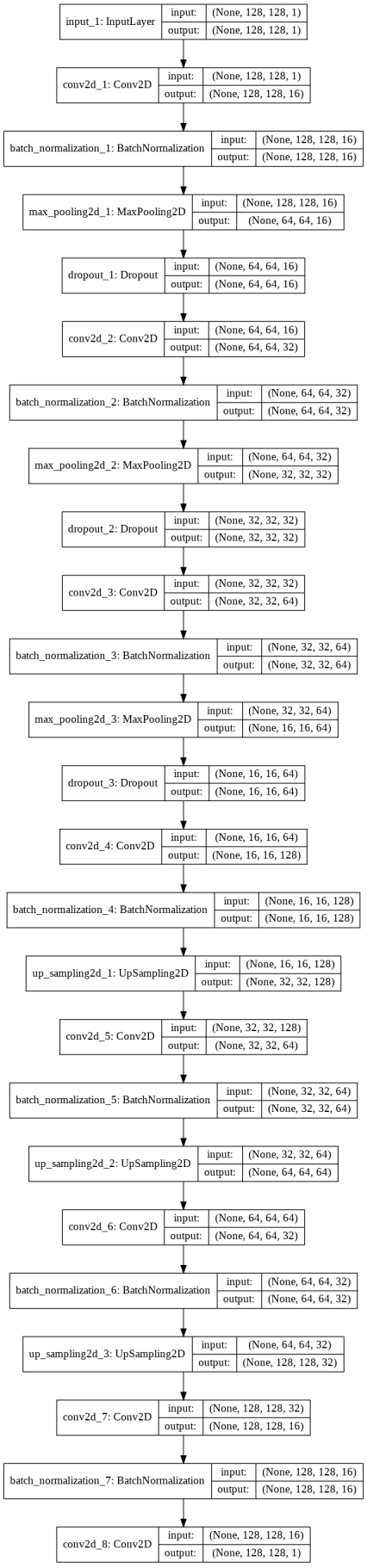

Figure 21: Architecture used for binary segmentation

As the nature of the problem is binary segmentation, it has been decided to test two different loss functions. First, it has been tested the network trained with binary cross-entropy [17] [18]. The problem of binary cross-entropy is that the output is not hardly pushed to be binary, but to have values between 0 and 1, obtaining a segmentation far from being smooth. Thereby, it has been tested a second loss function based in dice coefficient loss [19] [20] [21] [22]. Dice coefficient loss is used for class imbalance problems, and it has the advantage of obtaining a smoother output by pushing it to be binary. However, the borders of the images are less fine shaped when using this last loss function.

In addition, due to the big size of the network and to overcome the vanishing gradient problem, it has been implemented a greedy layer-wise training of this network. First, it has been trained separately each of the three convolutional layers of the encoder, and then all the network has been fine-tuned.

Both loss functions provide desired characteristics in the segmented images. Binary cross-entropy loss function drives the network to detect better the margins of the objects that are being segmented. Dice coefficient loss function drives the network to have a smoother image, filling better the interior of the segmented objects. In an attempt to join these desired features of both models it has been designed a combined model that combines in its input the two outputs of the auto-encoder trained with both, binary cross-entropy loss and dice coefficient as loss function. For so, the combined model combines and process the two images of the output of the segmenting neural networks (one for the auto-encoder trained with binary cross-entropy loss and the other with dice coefficient loss), concatenating them in the z-axis, i.e., its input is of size 128x128x2.

This combined model's architecture is shown in Figure 22. It consists in a merging layer of the two outputs of the auto-encoders, followed by two convolutional layers. The first one is composed by 8 filters of size 3x3, stride of 1, zero padding of 1 and ReLU activation function, and the second one (output) is a convolutional layer with one filter of size 3x3, stride of 1, zero padding of 1 and sigmoid activation function.

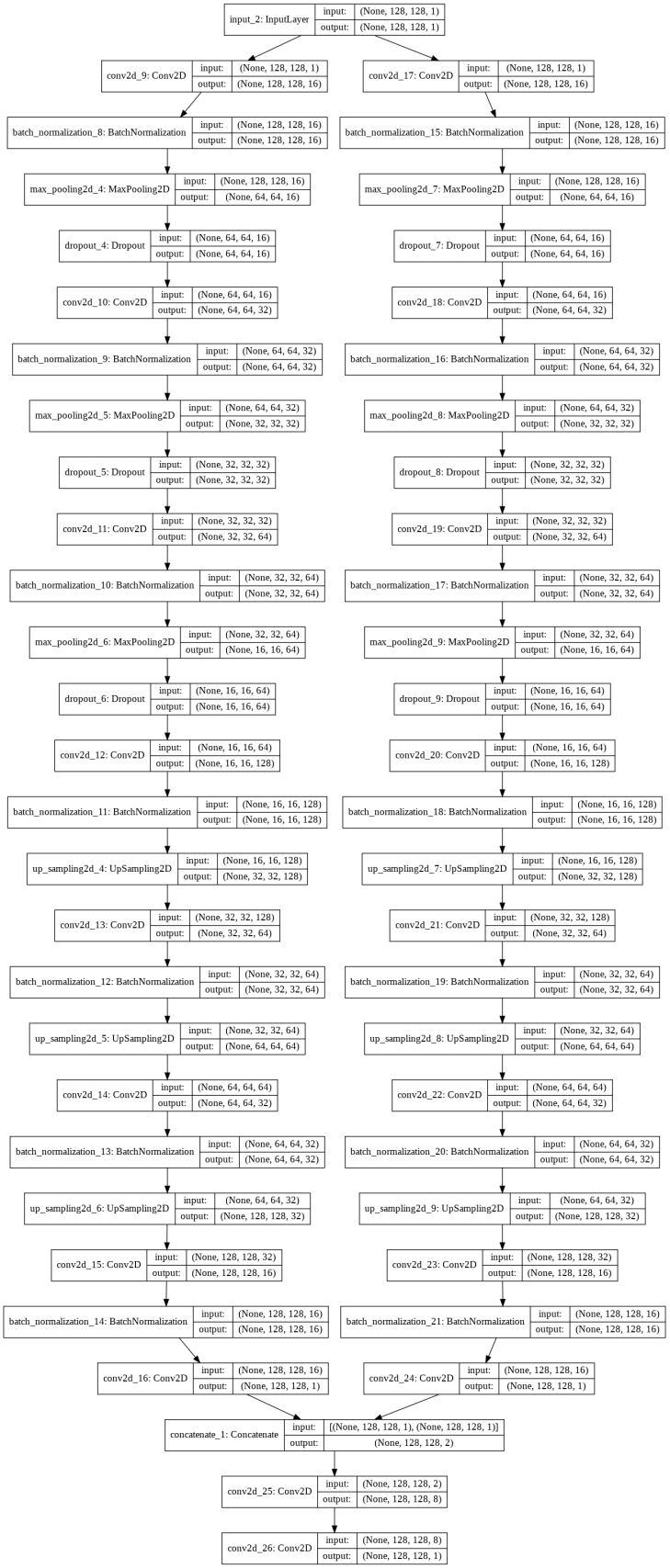

Figure 22: Architecture used for combining both outputs of the segmentation networks

In addition to this, it has been reshaped the auto-encoder architecture with the concept of skip connection, in which some connections skip deeper layers and are feed as input to the decoder's higher layers. This makes the architecture more similar to the architecture of U-Net [1]. The purpose of this is to use some concepts that were learned in the previous layers for the reconstruction in the upsampling layers. The new architecture is shown in Figure 23. For comparison purposes, the network has been trained with both, binary cross-entropy and dice coefficient as loss functions.

Finally, and for comparing the previous architectures with a well-known model, the architecture of U-Net [1] has been trained. Again, the network has been trained with both, binary cross-entropy and dice coefficient as loss functions.

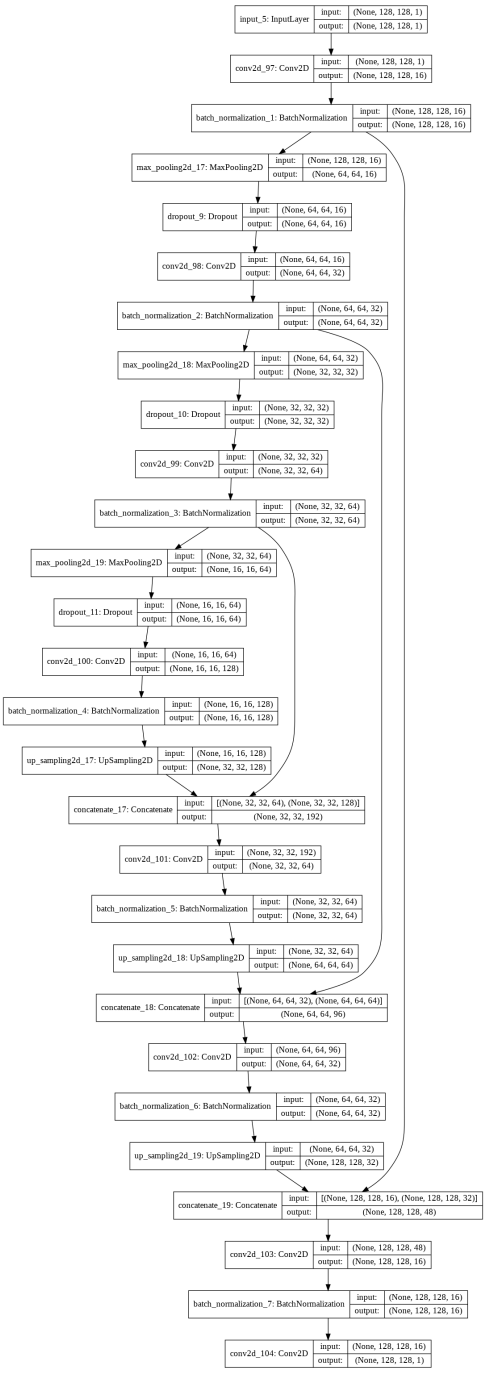

Figure 23: Architecture used applying the skip connection concept

3.1 Auto-encoder trained with binary cross-entropy loss function

First, it has been trained the whole architecture of the auto-encoder with the training set of 1190 segmented binary images and binary cross-entropy as loss function. The structure of the model is shown in Figure 21 and it has been trained with 150 epochs, batch size of 16, Adam optimizer and random initialization of the weights. The results of the tests can be seen in the next figures.

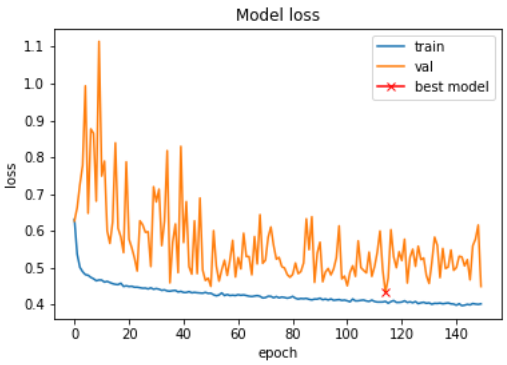

Figure 24: Model loss - Auto-encoder and BCE

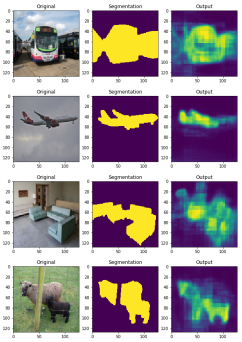

Figure 25: Output of the segmentation

In Figure 24 it can be seen the evolution of the loss function over 150 epochs. Even though the training loss is smaller than the validation loss, the network hasn't overfitted a lot. However, if more epochs are executed, it overfits dramatically increasing the loss value of the validation set, and obtaining much worse results.

In Figure 25 the segmentation done for four images is shown. In the first column, the original image, in the second column, the ground truth, and in the third column the segmentation done by the neural network can be observed. As it can be noticed, despite the shapes and borders of the objects being correctly detected, the obtained segmented images are not smooth.

3.2 Auto-encoder trained with dice coefficient loss function

In this second case, the whole architecture of the auto-encoder with the training set of 1190 segmented binary images and dice coefficient as loss function has been trained. The structure of the model is shown in Figure 21 and it has been trained with 150 epochs, batch size of 16, Adam optimizer and random initialization of the weights. Notice that the only difference with the model of section 3.1 is that this one has been trained with dice coefficient as loss function instead of binary cross-entropy. The results of the tests are shown in the next figures.

In Figure 26 it can be seen the evolution of the loss function over the 150 epochs. As in the previous section, more epochs will result in a highly overfitted network.

In Figure 27 it can be noticed the segmentation done for the same four images than in the previous section. As it was mentioned before, the dice coefficient loss drives the network to have smoother outputs, but the shapes are more poorly taken out than when used binary cross-entropy as loss function.

3.3 Greedy layer-wise training of the auto-encoder trained with binary cross-entropy loss function

In this section it is presented the results when training the encoder layers separately with the training set of 1190 segmented binary images and binary cross-entropy as loss function in all the stages of the training. First it has been trained the highest convolutional layer of the encoder with 20 epochs, batch

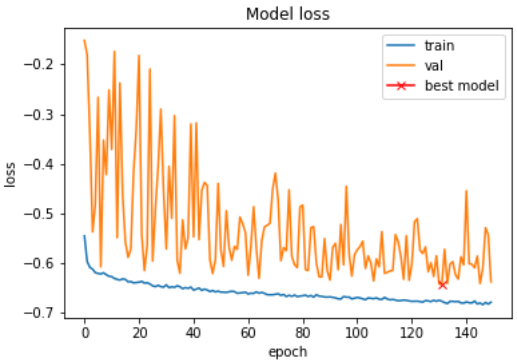

Figure 26: Model loss - Auto-encoder and DC

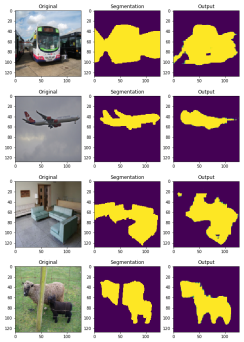

Figure 27: Output of the segmentation

size of 16 and optimizer. Then, it has been trained the second convolutional layer of the encoder with 25 epochs, batch size of 16 and Adam optimizer. Afterwards, it has been trained the third convolutional layer of the encoder with 5 epochs, batch size of 16 and Adam optimizer. Finally, it has all been joined to the center and decoder layers and it has been fine-tuned with 100 epochs, batch size of 16 and Adam optimizer. The structure of the whole network can be seen in Figure 21, and it is the same that the structure of the networks of previous sections, 3.1 and 3.2. The results of the tests are shown in the next figures.

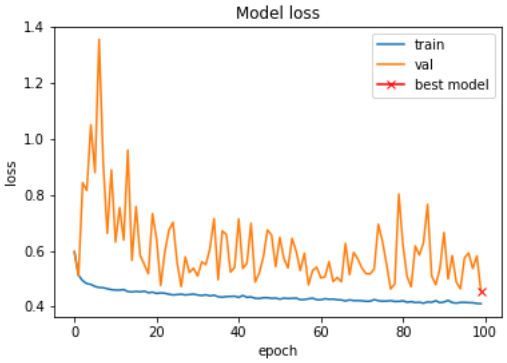

Figure 28: Model loss - Greedy layer-wise and BCE

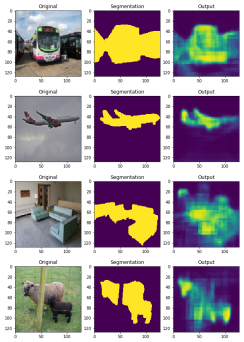

Figure 29: Output of the segmentation

In Figure 28 it can be seen the evolution of the loss function over 100 epochs for the fine-tuning stage of all the network. Comparing it with its homologous, Figure 24, it can be said that the improvement is practically non-existent.

In Figure 29 it is exposed the segmentation done for the same four images that in the previous sections. As well, and comparing it with its homologous, Figure 25, it can be seen that the segmented images are very similar, recognizing correctly the shapes of the objects, but not smoothing the segmentation.

In addition, in the next figures it is shown the training of each of the three layers of the encoder.

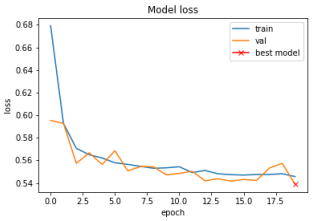

Figure 30: Model loss - Greedy layer-wise and BCE - L1

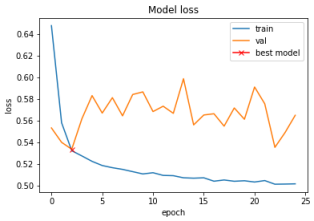

Figure 31: Model loss - Greedy layer-wise and BCE - L2

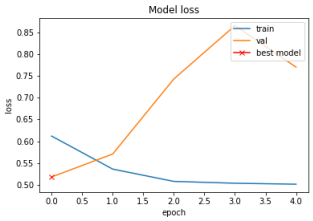

Figure 32: Model loss - Greedy layer-wise and BCE - L3

Figure 30 shows the loss function evolution of the first convolutional layer of the encoder. Figure 31 presents the loss function evolution of the second convolutional layer of the encoder. Figure 32 shows the loss function evolution of the third convolutional layer of the encoder. From these figures it can be inferred that a network architecture with only either one or two convolutional layers in the encoder will not overcome the performance of the whole network fine-tuned.

3.4 Greedy layer-wise training of the auto-encoder trained with dice coefficient loss function

In this section it is shown the results when training the encoder layers separately with the training set of 1190 segmented binary images and dice coefficient as loss function in all the stages of the training. First it has been trained the highest convolutional layer of the encoder with 20 epochs, batch size of 16 and Adam optimizer. Then, it has been trained the second convolutional layer of the encoder with 25 epochs, batch size of 16 and Adam optimizer. Then, it has been trained the third convolutional layer of the encoder with 5 epochs, batch size of 16 and Adam optimizer. Finally, it has all been joined to the center and decoder layers and it has been fine-tuned with 100 epochs, batch size of 16 and Adam optimizer. The structure of the whole network can be seen in Figure 21, and it is the same that the structure of the networks of previous sections, 3.1 and 3.2. Notice that the only difference with the model of section 3.3 is that this model has been trained with dice coefficient as loss function instead of binary cross-entropy. The results of the tests are shown in the next figures.

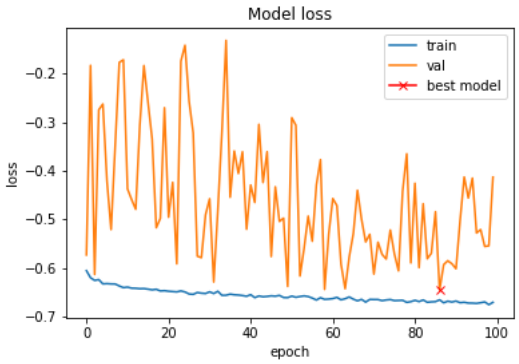

Figure 33: Model loss - Greedy layer-wise and DC

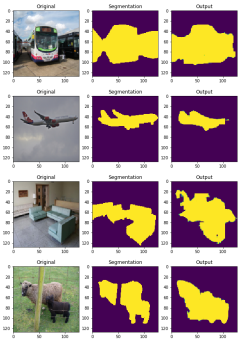

Figure 34: Output of the segmentation

In Figure 33 it can be seen the evolution of the loss function over the 100 epochs for the fine-tuning stage of all the network. Comparing it with its homologous, Figure 26, it can be said that the improvement is, again, practically non-existent.

In Figure 34 it is exposed the segmentation done for the same four images than in the previous sections. As well, and comparing it with its homologous, Figure 27, it can be see that the segmented images are

very similar, giving a poorer segmentation than the obtained with binary cross-entropy loss function, but obtaining a much smoother result.

In addition, in the next figures it is shown the training of each of the three layers of the encoder.

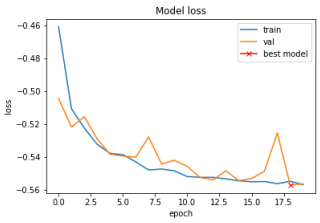

Figure 35: Model loss - Greedy layer-wise and DC - L1

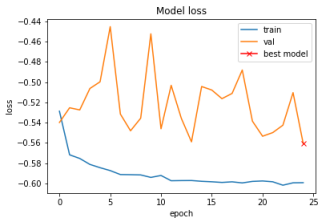

Figure 36: Model loss - Greedy layer-wise and DC - L2

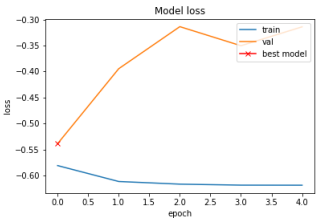

Figure 37: Model loss - Greedy layer-wise and DC - L3

Figure 30 shows the loss function evolution of the first convolutional layer of the encoder. Figure 31 presents the loss function evolution of the second convolutional layer of the encoder. Figure 32 shows the loss function evolution of the third convolutional layer of the encoder. From these figures it can be inferred, again, that a network architecture with only either one or two convolutional layers in the encoder will not overcome the performance of the whole network fine-tuned.

3.5 Combined model trained with binary cross-entropy loss function

In this section it is exposed the results obtained when it is trained a new model that combines the output of the both previous models, the greedy layer-wise trained with binary cross-entropy loss function, and the one trained with dice coefficient loss function.

The structure of the model is exhibited in Figure 22. As explained before, the new model combines both outputs of the the greedy layer-wise networks and feed it to two common convolutional layers. The first one is a convolutional layer with 8 filters, stride of 1, zero padding of one and ReLU activation function. The second one is a convolutional layer with one filter of size 3x3, stride of 1, zero padding of 1 and sigmoid activation function. The model has been trained with 50 epochs, batch size of 16, Adam optimizer, random initialization of the weights and, in this case, binary cross-entropy as loss function. The results can be seen in the next figures.

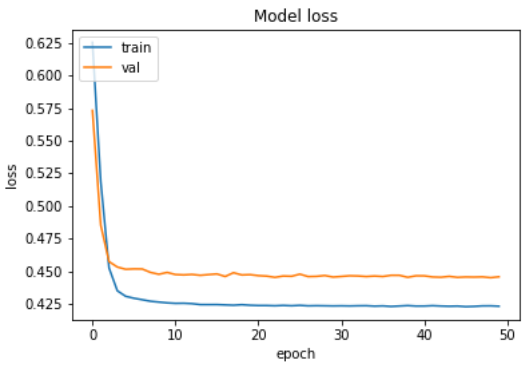

Figure 38: Model loss - Combined model and BCE

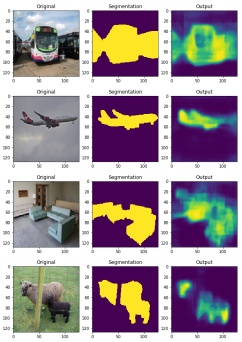

Figure 39: Output of the segmentation

In Figure 38 it is presented the evolution of the loss function for 50 epochs. The loss of the validation set rapidly decreases to a constant level of 0.45, and then it barely changes. This is because of the simplicity of the network. However, when comparing it with Figures 24 and 28 it can be seen that it slightly improves the performance of those networks.

In Figure 39 it is shown the segmentation done for the same four images than in the previous sections. As well, and comparing it with Figures 25 and 29 it can be seen that the results are slightly better, obtaining smoother images (they are "filled" in a more continuous way) than in the homologous segmentations. This improvement is specially highlighted in the couch image, in which the shape is much better defined than in the homologous segmentations of sections 3.1 and 3.3.

3.6 Combined model trained with dice coefficient loss function

In this section it is exposed the results obtained when it is trained a new model that combines the output of the both previous models, the greedy layer-wise trained with binary cross-entropy loss function, and the one trained with dice coefficient loss function. The difference with the model of section 3.5 is that this neural network has been trained with dice coefficient as loss function instead of binary cross-entropy.

The structure of the model is shown in Figure 22 and it has been trained with 50 epochs, batch size of 16, Adam optimizer, random initialization of the weights and, in this case, dice coefficient as loss function. The results can be seen in the next figures.

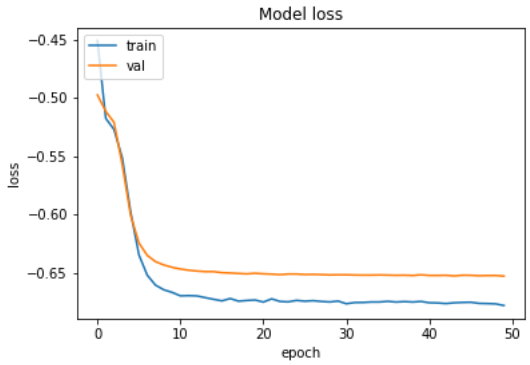

Figure 40: Model loss - Combined model and DC

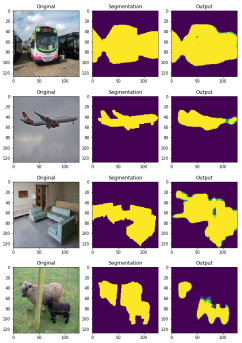

Figure 41: Output of the segmentation

In Figure 40 it is shown the evolution of the loss function for 50 epochs. As in the previous section, the loss of the validation set rapidly decreases to a constant level of -0.64, and then it barely changes. As well, when comparing it with Figures 26 and 33 it can be seen that it slightly improves the performance of those networks.

In Figure 41 it is presented the segmentation done for the same four images than in the previous sections. As well, and comparing it with Figures 27 and 34 it can be seen that the results are, again, slightly better, obtaining more accurate definitions of the borders of the objects than in the homologous segmentations of sections 3.2 and 3.4.

3.7 Auto-encoder with skip connection layers trained with binary cross-entropy loss function

In this section it is exposed the results of training the auto-encoder with connections between the encoder and decoder layers that skip the deeper layers (skip connection) with the training set of 1190 segmented binary images and binary cross-entropy as loss function. The model architecture is shown in 23 and it has been trained with 150 epochs, batch size of 16, Adam optimizer and random initialization of the weights. The results of the tests can be seen in the next figures.

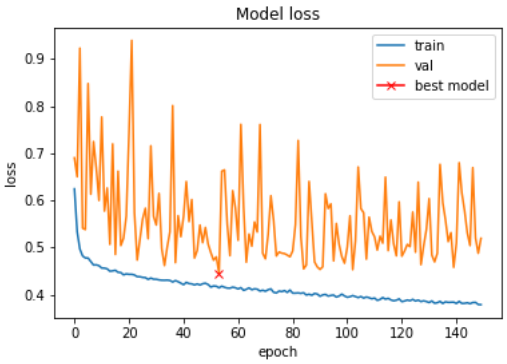

Figure 42: Model loss - Skip connection and BCE

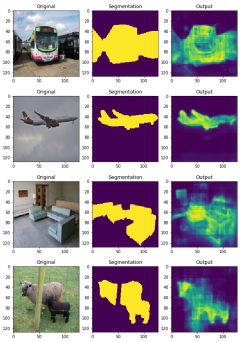

Figure 43: Output of the segmentation

In Figure 42 it can be seen the evolution of the loss function over the 150 epochs. In Figure 43 it is exposed the segmentation done for the same four images than in the previous sections. Apparently, this is the model that performs the best segmentation among all the models based in the auto-encoder trained with binary cross-entropy as loss function, i.e., the models of sections 3.1, 3.3 and 3.5. In this case, the images are smoother than its homologous images in the previous sections. However, the improvement is not big.

3.8 Auto-encoder with skip connection layers trained with dice coefficient as loss function

In this section it is exposed the results of training the auto-encoder with connections between the encoder and decoder layers that skip the deeper layers (skip connection) with the training set of 1190 segmented binary images and dice coefficient as loss function. The model architecture similar to the one of section 3.7, it is shown in 23 and it has been trained with 150 epochs, batch size of 16, Adam optimizer and random initialization of the weights. Notice that the only difference with the models of section 3.7 is that this one has been trained with dice coefficient as loss function instead of binary cross-entropy. The results of the tests are exposed in the next figures.

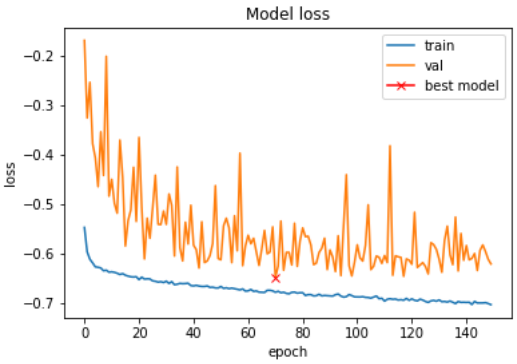

Figure 44: Model loss - Skip connection and DC

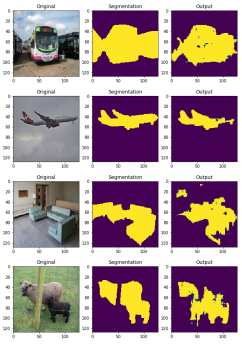

Figure 45: Output of the segmentation

In Figure 44 it can be seen the evolution of the loss function over the 150 epochs.

In Figure 45 it is shown the segmentation done for the same four images than in the previous sections.

Apparently, this is the model that performs the best segmentation among all the models based in the auto-encoder trained with dice coefficient as loss function, i.e., those models of sections 3.2, 3.4 and 3.6. In this case the borders of the objects are better taken. However, and like in the previous section, the improvement is not very big.

3.9 U-Net trained with binary cross-entropy loss function

In this section it is shown the results of training U-Net [1] with the training set of 1190 segmented binary images and binary cross-entropy as loss function. The model has been taken from [23] and it has been trained with 150 epochs, batch size of 16, SGD optimizer and random initialization of weights. The results of the tests are shown in the next figures.

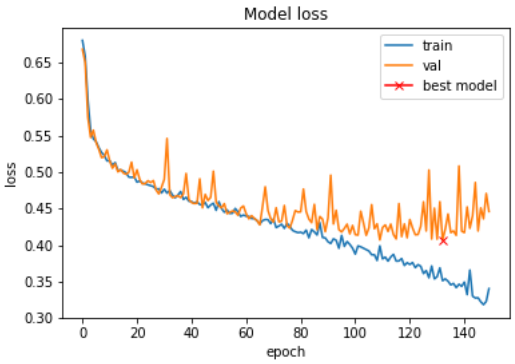

Figure 46: Model loss - U-Net and BCE

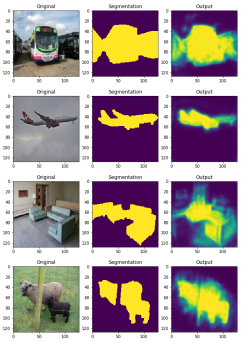

Figure 47: Output of the segmentation

In Figure 46 it can be seen the evolution of the loss function over 150 epochs. As it is shown, the network tend to overfit in the last epochs. This is probably because of the small number of training images and the big size of the network.

In Figure 47 it is exposed the segmentation done for the same four images than in the previous sections by the best model of the training (the best for the validation set).

Apparently, this model outperforms all the previous models trained with binary cross-entropy as loss function. It not only has a lower loss value for the best model, but also it obtains the smoothest segmented images.

3.10 U-Net trained with dice coefficient loss function

In this section it is exposed the results of training U-Net [1] with the training set of 1190 segmented binary images and dice coefficient as loss function. The model has been taken from [23] and it has been trained with 150 epochs, batch size of 16, SGD optimizer and random initialization of the weights. Notice that the only difference with the models of section 3.9 is that this model has been trained with dice coefficient as loss function instead of binary cross-entropy. The results of the tests are shown in the next figures.

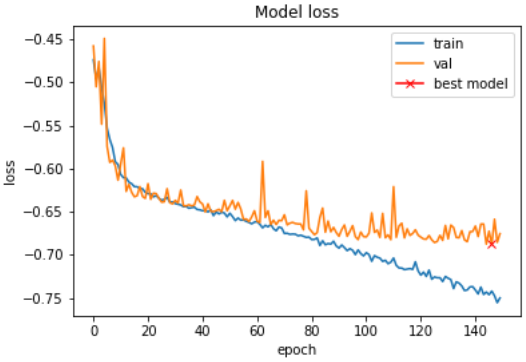

Figure 48: Model loss - U-Net and DC

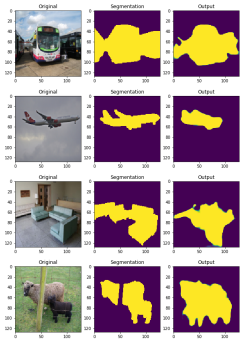

Figure 49: Output of the segmentation

In Figure 48 it can be seen the evolution of the loss function over 150 epochs. Again, and probably due to the small size of the training set, the network tends to overfit in the last epochs.

In Figure 49 it is shown the segmentation done for the same four images than in the previous sections by the best model of the training.

Again, as in section 3.9, this model outperforms all the previous models trained with dice coefficient as loss function. It not only has a lower loss value for the best model, but also it obtains the most defined border segmentations.

Conclusion

The project presented has allowed contact with different applications of neural networks to complex problems, their particularities, and the benefits they can bring. It has also allowed to present the strengths and weaknesses of these approaches, and different strategies that can be used to maximize performance in the task in question at hand.

First of all, the use of convolutional auto-encoders for dimensionality reduction has been analyzed. With the results obtained, it is possible to make a representation of the target image using a codification 12 times smaller than that of the dimensions of the image, obtaining a reconstruction with losses, but that allows to appreciate the content of the image, nevertheless. This compression of 12:1 can be considered a good result, given that these images make use of the JPEG standard, which is already a method of compression with losses, so that the redundant information present in the image has already been possibly eliminated.

The use of such encoding to encode features in images has allowed the encoder section to be used as part of network architectures for classification. It has been demonstrated in several experiments, however, that the approach that provides best results is to use the encoder architecture, either previously trained or with random initialization, as part of a fully trainable model, as opposed to operating on the coded output of an encoder trained for another purpose.

For classification tasks, different models have been presented that can be used in multi-class classification environments, obtaining similar results once a good adjustment of the design variables of the model was achieved. We have also shown the approaches used to improve network performance, as well as to solve the problems encountered. These strategies, such as the use of Data Augmentation mechanisms, regularization, adaptive learning rate or redistribution of training data in the dataset have allowed the improvement of the models with respect to the initial case, in an iterative way.

Finally, in this project different network architectures for image segmentation have been studied and two loss functions, binary cross entropy and dice coefficient loss, have been tested. The first loss function has proven to get more defined shapes, while the second gets smoother segmented figures.

Five different networks have been tested, all trained with both loss functions. First, a neural network similar to the auto-encoder has been designed. In order to get better results, this network has been trained in a greedy layer-wise manner, and certainly, better results were obtained. Furthermore, in an attempt to get smooth images with well-defined borders, the greedy layer wise architectures trained with binary cross entropy loss and dice coefficient loss have been combined, merging them and adding two shared output convolutional layers. However, the combination of these two architectures has not supposed significant improvements. In addition, pursuing building the previous network architecture in a similar way to U-Net's architecture, some skip connections layers between the encoder and the decoder have been added, obtaining the best results of our self-defined segmentation networks. Finally, and in an attempt to compare the best designed network with a state of the art network, U-Net architecture has been tested. Despite the better results obtained with this last network, it can be said that the designed networks were not far from it's performance.

References

[1] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351 of *LNCS*, pp. 234–241, Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks and Machine Learning – ICANN 2011* (T. Honkela, W. Duch, M. Girolami, and S. Kaski, eds.), (Berlin, Heidelberg), pp. 52–59, Springer Berlin Heidelberg, 2011.

[4] B. Leng, S. Guo, X. Zhang, and Z. Xiong, "3d object retrieval with stacked local convolutional autoencoder," *Signal Processing*, vol. 112, pp. 119 – 128, 2015. Signal Processing and Learning Methods for 3D Semantic Analysis.

[5] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, "Deep features learning for medical image analysis with convolutional autoencoder neural network," *IEEE Transactions on Big Data*, pp. 1–1, 2018.

[6] Y. Zhang, "A better autoencoder for image: Convolutional autoencoder,"

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[8] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 international interdisciplinary PhD workshop (IIPhDW)*, pp. 117–122, IEEE, 2018.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[14] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, pp. 153–160, 2007.

[15] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: an overview," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 191–202, 2018.

[16] S.-J. Yen and Y.-S. Lee, "Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset," in *Intelligent Control and Automation*, pp. 731–740, Springer, 2006.

[17] T. Sterbak, "U-net for segmenting seismic images with keras," 2018.

[18] S. A. Taghanaki, K. Abhishek, and G. Hamarneh, "Improved inference via deep input transfer," *arXiv preprint arXiv:1904.02307*, 2019.

[19] M. Jocic, "Deep learning tutorial for kaggle ultrasound nerve segmentation competition, using keras," 2017.

[20] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pp. 240–248, Springer, 2017.

[21] F. Sancinetti, "U-net convnet for ct-scan segmentation," 2018.

[22] A. A. Novikov, D. Lenis, D. Major, J. Hladvka, M. Wimmer, and K. Bühler, "Fully convolutional architectures for multiclass segmentation in chest radiographs," *IEEE transactions on medical imaging*, vol. 37, no. 8, pp. 1865–1876, 2018.

[23] Zhixuhao, "Implementation of deep learning framework – unet, using keras."