POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Bridging the gap between natural and medical images through deep colorization

Supervisors

Prof. TATIANA TOMMASI

Dr. LIA MORRA

Prof. FABRIZIO LAMBERTI

Candidate

LUCA PIANO

April 2020

Summary

Nowadays, Deep Neural Networks are widely used in medical image analysis, and they are the state-of-the-art in many applications. However, performances are often limited from the scarcity of labeled data since generating appropriated annotations requires medical experts and is very time-consuming. A large number of techniques are used to tackle this problem, and one of the most popular is to transfer learning from models pre-trained on ImageNet. Due to the different characteristics of the natural and medical domain, this strategy may be suboptimal. This thesis proposes three different colorization modules that learn how to map gray-scale medical images to three-channels colorful ones exploiting the classification loss generated during the training of a pre-trained model. Several experiments have been conducted to compare different transfer learning strategies from the RGB to the medical domain, with and without the proposed colorization module. Experiments were conducted on different X-ray datasets, including CheXpert, ChestX-ray14, and MURA. Results on the CheXpert dataset show that the colorization modules can improve the results up to 8% when the pre-trained model is frozen except for the last layer. This outcome suggests that the colorization module effectively compensates for the differences between the ImageNet and medical images. In general, fine-tuning the entire network obtains the best results, with or without the colorization module. The proposed strategy is most effective when the number of available samples is small (less than 512), whereas fine-tuning the entire network is the most effective strategy for large scale datasets.

"Alla mia famiglia e ai miei amici che mi hanno sempre supportato e sopportato. A mio padre, che è stato la mia roccia finche ha potuto, e a mio nonno, che sognava questo giorno. Sarete sempre al mio fianco."

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since it was possible to scan and load medical images into computers, researchers have studied and built systems to perform automatic analysis. Until the 90s, medical image analysis exploited low-level pixel processing and mathematical modeling to solve particular tasks. But, at the end of the same decade, supervised techniques, that use training data to develop models, were becoming very popular in this field. This marked the beginning of a shift from systems that are entirely designed by humans to others trained by computers. Convolutional neural networks (CNNs) are among the most successful models for image analysis to date. The story of this type of architecture started at the beginning of 1980 with the works of Fukushima et al.[1], while one of the first applications in medical image analysis was in 1995 thanks to the work of Lo et al.[2] that uses CNN to detect nodules in the lungs. However, convolutional neural networks gained popularity after the end of 2012, when the model proposed by Krizhevsky et al.[3], called AlexNet, wons the ImageNet competition by a large margin. In the last years, medical image analysis has gained huge advantages from machine learning and deep learning. A large number of various tasks are nowadays almost exclusively done by machine learning methods, for instance, segmentation, in which each pixel is assigned to different tissues or anatomical structures. Like other techniques, even if there are enormous advantages, these methods bring some problems that must be tackled. The scarcity of labeled data is one of the most frequent and critical obstacles that machine learning methods have in medical image analysis, even when large sets of unlabeled data are available. The demand for highly specialized work to collecting annotations for medical images represents one of the primary reasons for this lack. Moreover, manual labeling of the instances is a costly and time-consuming process. Some approaches are used to tackle this problem as Transfer Learning, Data augmentation, and Semi-supervised Learning. Transfer Learning

has a central role in several works, and it is common to exploit models trained on ImageNet to take higher results in small datasets. On the other hand, some recent work shows that perform Transfer Learning from ImageNet does not help to reach better outcomes than training from scratch. These results can be justified due to the diversity between the two domains of natural and medical images. This thesis aims to investigate how to learn a colorization method that helps to mitigate these differences. Since standard deep learning colorization methods are not applicable due to the lack of color references, it is proposed a technique that exploits the classification loss generated from the pre-trained model frozen except for the last layer. Moreover, it was examined which is the best strategy of learning, if the best strategy changes when the number of samples in the dataset varies, and if the colorization module proposed are transferable.

1.1 Structure of the document

The work of this thesis is so structured:

- Chapter 1 contains an introduction to the thesis.
- Chapter 2 aims to introduce the reader to the Machine Learning and Deep Learning. This chapter is to consider only as a starting point for those who don't know these arguments.
- Chapter 3 reports a summary of Transfer Learning in medical image analysis and principal techniques of colorization.
- In Chapter 4 are presented the datasets used, the architecture proposed, the colorization modules and the pre-trained models chosen.
- Chapter 5 contains the description of all experiments over the different datasets with the results obtained.
- In Chapter 6, the results are discussed and the conclusion are presented. At the end, some possible future works are proposed.

Chapter 2

Background

This chapter aims to provide a brief introduction to machine learning techniques related to deep learning and the principal CNN models.

2.1 Machine Learning algorithms

Machine learning algorithms are systems that learn how to solve a problem from the data fed. One of the principal methods to categorize this kind of systems is according to their way to learn:

- **Supervised learning**: The system learns a mapping between input values and the target labels. So each training data must be a pair of an input object, e.g. a vector or an image, and the desired output value. After the training, if all the parameters chosen are "correct" the system should be able to predict the correct class also for the unseen instances (Figure 2.1). The two main supervised learning problems are:

  Classification: It requires that the system predict a class label.

  Regression: It requires that the system predict a numerical label.

  MNIST is one of the most famous supervised learning problems, where the inputs are handwritten digits (images) and the output is a class label for what digit represents (numbers from 0 to 9).

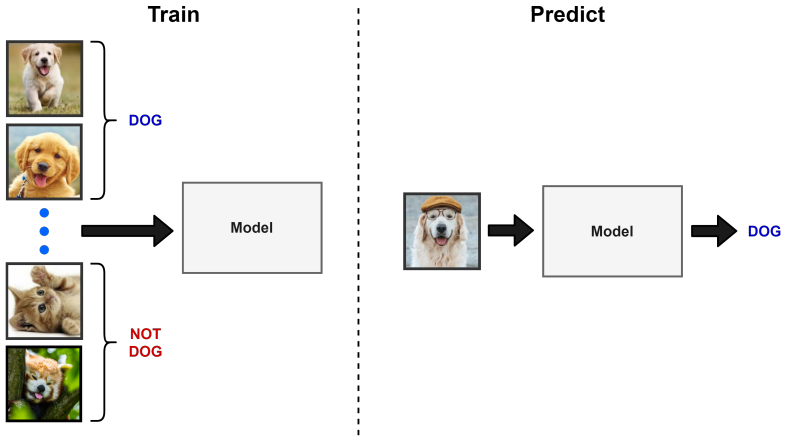


Figure 2.1: Supervised learning.

- **Unsupervised learning**: The system takes as input unlabelled data, and the algorithm tries to find structures in the data. The most common unsupervised learning problem is *Clustering* in which the algorithm attempts to separate data into groups without the guidance of labels. Other common types of unsupervised learning are *Density Estimation*, *Visualization*, and *Projection*.
- **Semi-supervised learning**: In this scenario, labeled data are few while the unlabeled examples are in a large number. This approach lies between unsupervised learning and supervised learning. To improve the performance these two types of data are used.
- **Reinforcement learning**: The system learns by receiving feedback from the environment. This means that there isn't a fixed dataset but the system must perform some steps and the environment gives back feedback about performance toward the goal. An example is playing a game where the system has the goal of winning or getting a high score. Each move performed by the algorithm received a positive or negative score.

2.2 Neural Network

A subcategory of machine learning algorithms is neural networks. These kinds of systems are inspired, not identical, to biological neural networks that constitute the animal and human brain. The neural networks are composed of simple computational units (called neurons) connected by link (called synapses).

A neuron can be generalized in:

- A group of inputs (Figure 2.2a).

- A group of weighted links (Figure 2.2b).
- A linear combiner which computes the weighted sum of the inputs (Figure 2.2c).
- An activation function for limiting the output amplitude of the neuron (Figure 2.2d).
- An output (Figure 2.2e).

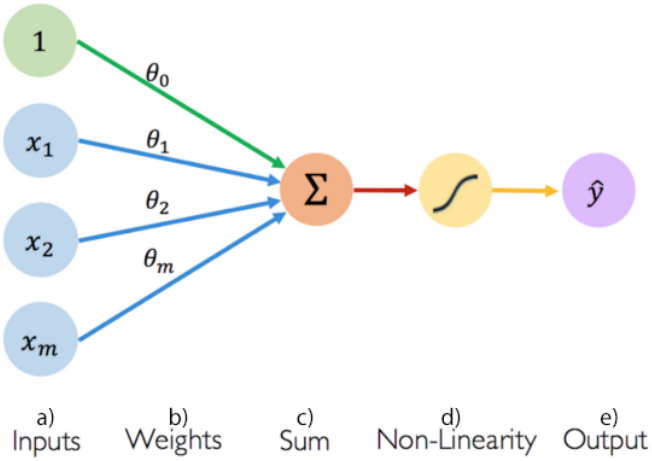


Figure 2.2: Neuron.[4]

A standard architecture is composed of:

- Input units.
- Hidden units (optional).
- Output units.

Classical neural networks have a high training cost and the training complexity increases with the number of hidden layers (Figure 2.3). For this reason, for a long time, neural networks with a large number of hidden layers and inputs, as for image classification, were considered unfeasible.

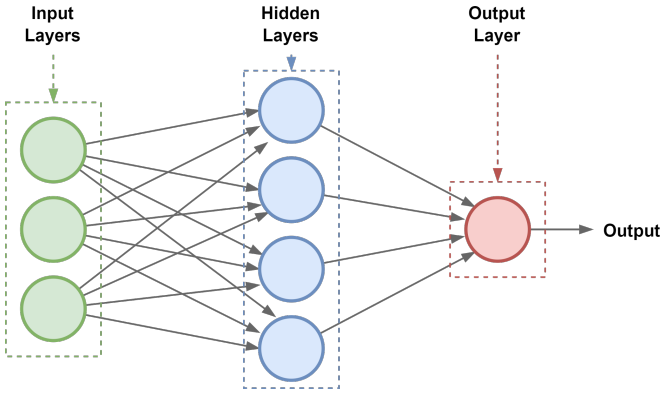


Figure 2.3: Example of a Neural Network.

2.3 Deep learning

Deep learning algorithms are based on neural networks and exploit a cascade of hidden layers for feature extraction. Each level learns how to transform the input data into a slightly more abstract and composite representation; in this way, the model learns a hierarchical feature structure in which low layers extract local features, like edges, and high layers extract global patterns. The most common architectures are convolutional neural networks (CNN) and recurrent neural networks. In image analysis, CNNs are widely used.

2.3.1 CNN

The Convolutional Neural Networks are models that take inspiration from the organization of the Visual Cortex. Each neuron is sensitive to the *Receptive Field* that is a small sub-region of the visual field. There are two principal differences between multi-layer neural networks (like multi-layered perceptron) and CNN:

1. While MLP is fully connected and each link has his weight, in CNN weights are shared thanks to the convolutional layers that are the basis of this type of architecture.

2. The second key difference is that CNN typically uses pooling layers that help to reduce the number of parameters.

The architecture of a CNN can change in different ways but three layers are characteristic of this type of network: Convolutional Layer, Non-Linear Layer, Pooling Layer.

Convolutional layer

This is the core layer of CNNs and consists of a set of filters (kernel) that are convolved (Figure 2.4) across the input data creating a multidimensional feature map. The network learns filters able to identify a specific type of feature over the image. Weights are shared across neurons and this increases learning efficiency by reducing the number of free parameters that must be learned. Moreover using this type of layer, the model doesn't require to learn different detectors for the same feature that can occur in different parts of an image.

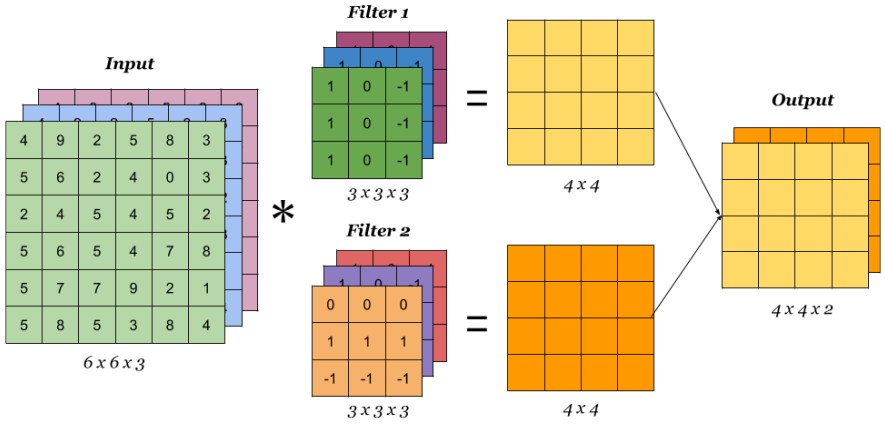


Figure 2.4: Convolution.[5]

Non-Linear Layer

Non-Linear Layer increases the non-linearity of the entire architecture without affecting the receptive fields of convolutional layers and helps to reduce the amplitude of parameters. There are different types of this kind of layers and the most used are:

ReLU Figure 2.5a

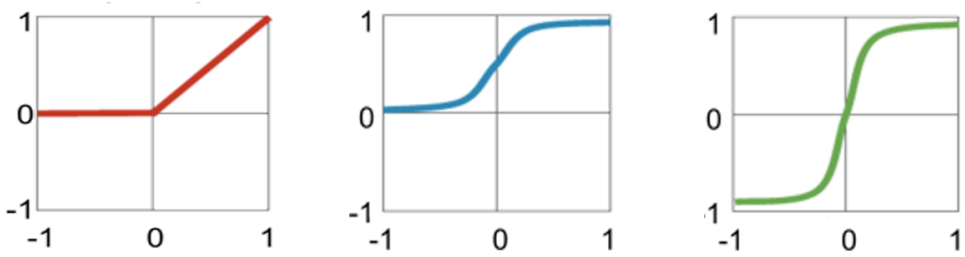
$$f(x) = max(0, x)$$

Sigmoid Figure 2.5b

$$f(x) = \frac{1}{1 + e^{-x}}$$

Tanh Figure 2.5c

$$f(x) = tanh(x)$$



(a) ReLU **(b)** Sigmoid **(c)** Tanh

Figure 2.5: Non-Linear function

Pooling Layer

The main purpose of this type of layer is to decrease the spatial size of the features map. This aims to reduce the computational power required to process the data through parameter reduction. Moreover, it is useful to extract features that are invariant over position and rotation, thus promoting the generalization of the model. Common pooling types are based on max and mean function:

- **Max Pooling Layer**: It returns the maximum value from the portion of the matrix covered by the kernel and acts as a noise suppressant.
- **Average Pooling Layer**: It returns the mean value from the section of the matrix covered by the kernel.

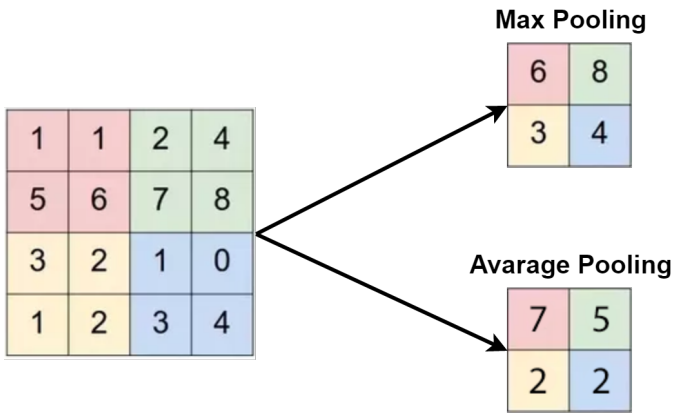


Figure 2.6: Pooling Layers.

Fully Connected Layer

At the end of the model, one or more fully connected layers, like in regular machine learning, are generally added, and in this section, the weights are no longer shared. This kind of layer allows the model to learn a non-linear combination of features extracted in previous layers.

2.3.2 Models

Several architectures have been submitted to the ImageNet[6] competition to reach better performance. After the breakthrough in this challenge achieved by AlexNet, other CNNs have obtained popularity. Between the most popular networks, it is possible to find Inception, VGG, GoogLeNet, ResNet, and DenseNet. In this thesis, the focus being on DenseNet and ResNet architectures.

ResNet

Deep convolutional neural networks are designed to extract from low to high-level features and the number of stacked layers can improve the "levels" of features. Consequently, the stacked layer is of primary importance. But when the networks start to converge, the deeper ones are exposed to a degradation problem: with the network depth increasing, accuracy gets saturated and then degrades rapidly. (Figure 2.7a) This problem is not caused by overfitting and shows that not all systems are easy to optimize. However, the deterioration of training accuracy can be mitigated with the introduction of a new type of layer - *The Residual Block* [7].

A ResBlock is defined as:

$$G(x) = F(x) + x$$

and can be achieved by feedforward neural networks with shortcuts connection that skip one or more layers so that perform as identity mapping. (Figure 2.7b)

Residual networks solve many problems as:

- ResNets are easy to optimize.
- They can easily gain accuracy from greatly increased depth, producing results that are better than previous networks.

To build the ResNet architecture, a plain structure, mainly inspired by the concept of VGG nets, is created and shortcut connections are inserted to turn the network

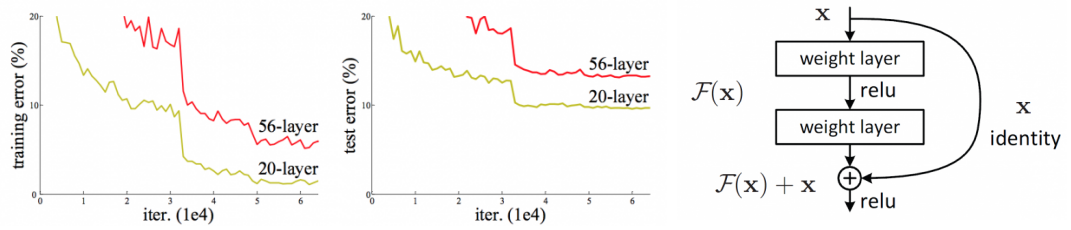


Figure 2.7: (Left) Degradation problem. (Right) Residual Block structure. [7]

into its counterpart residual version (Figure 2.8). The standard residual networks are ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152. In this thesis, the ResNet18 was used.

DenseNet

Dense Convolutional Network[8] is another widely used architecture that allows achieving high accuracy and reducing the number of parameters compared with ResNet. As seen previously, in ResNet, identity mapping is used to promote gradient propagation and element-wise addition is used to combine output by using ResBlock. Instead, DenseNet is composed of *DenseBlock* and these layers are densely connected. Each layer input is the concatenation of all previous layers' output (Figure 2.9). In this way, each layer receives knowledge from all preceding layer and each layer has direct access to the gradient of the loss function. This structure promotes "collective knowledge" and avoid to learn redundant feature maps. DenseBlock can be described as:

$$
\begin{aligned}
x_0 &= input \\
x_1 &= concat(x_0, F_0(x_0)) \\
x_i &= concat(x_0, F_0(x_0), .., F_{i-1}(x_{i-1})) \\
i &> 0
\end{aligned}
$$

This architecture brings some advantages:

- **Strong gradient flow**: The error signal can be easily propagated directly to earlier layers.
- **Fewer parameters:** For each layer, the number of parameters is directly proportional to $l \times k \times k$, where k is the *growth rate*, differently from ResNet where it is directly proportional to $C \times C$. Since $C >> k$, DenseNet is smaller than ResNet.
- **No features redundancy**: DenseNet tend to have richer patterns.

- The classifier use features of all complexity level.

The standard dense convolutional networks are DenseNet121, DenseNet169, DenseNet201, and DenseNet264. In this thesis, the DenseNet121 was used.

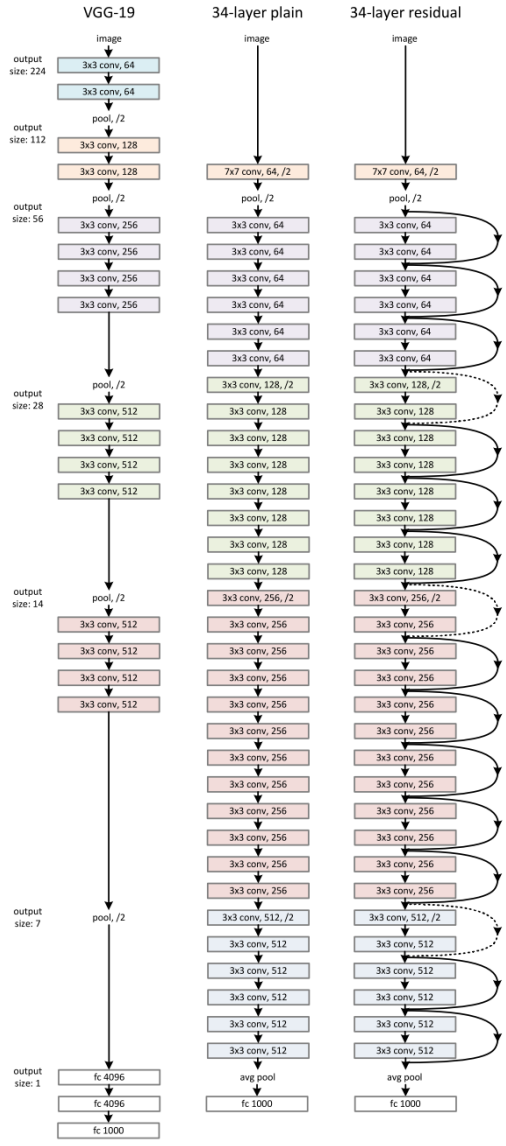


Figure 2.8: ResNet Architecture.[7]

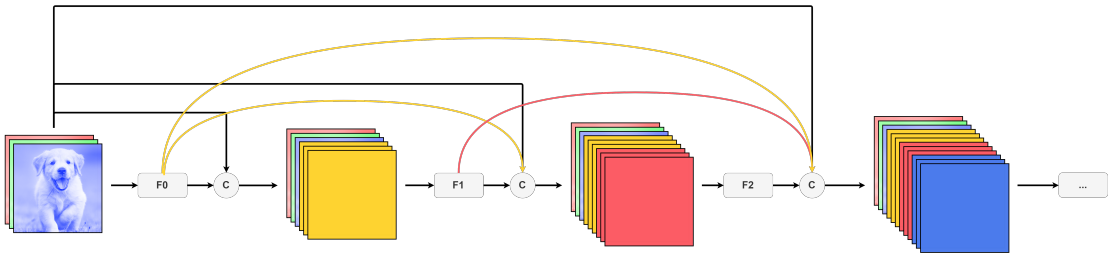


Figure 2.9: DenseNet structure.

2.4 Transfer Learning

One of the principal abilities of the human brain is the capacity to transfer knowledge between different tasks, and, after the early years of our lives, rarely we learn something from scratch. Instead, traditional machine learning and deep learning algorithms are designed to work in isolation and the models are rebuilt from scratch every time that the features-spaces distribution and tasks change. Generally, models that solve complicated problems require a large amount of data to gain high-grade performance but, for supervised models, retrieve a large amount of labeled data can be costly and difficult since the labels often require specialized human work. So the main idea of transfer learning is to exploit the previous knowledge (features, weights) acquired to solve a problem, to tackle another related-task that might have a smaller dataset or to help generalization (Figure 2.10).

Transfer learning can help in different situations:

- When the dataset used to learn a specific task is tiny, TL can help to avoid overfittingand to improve generalization. It is one of the most used techniques to take on the scarcity of labeled-data.
- To reduce the learning time ad cost.
- To improve performance.

2.4.1 Definition

In order to give a formal definition of transfer learning, we need to introduce some formal notations and the definition of domain and task [9]. A *domain* is defined as a two-element tuple:

$$D = \{\mathcal{X}, P(X)\}$$

where:

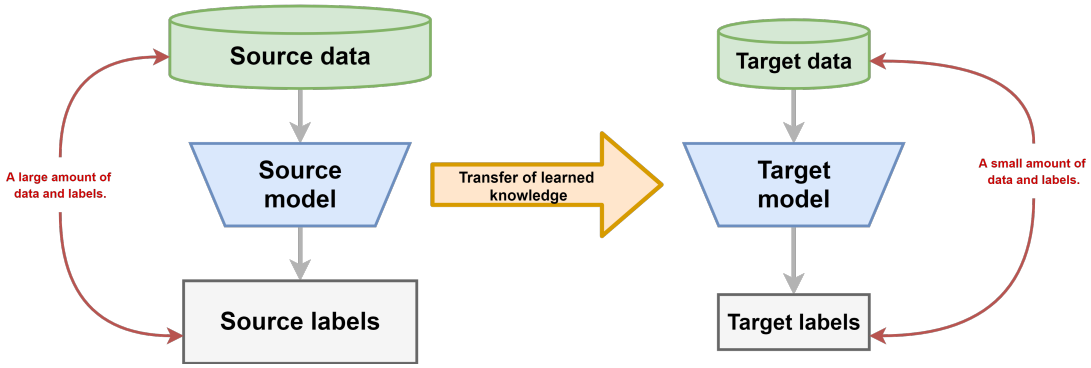


Figure 2.10: Transfer learning.

- Feature Space $\rightarrow \mathcal{X}$
- Marginal Probability Distribution $\rightarrow P(X), X = \{x_1, ..., x_n\}, x_i \in \mathcal{X}$

Two domain are different if have a different feature space or/and different marginal probability distribution.

Given a domain, **D**, a *task* is defined as:

$$T = \{\gamma, f(\cdot)\} = \{\gamma, P(Y|X)\}, Y = \{y - 1, ..., y_n\}, y_i \in \gamma$$

where:

- Label Space $\rightarrow \gamma$ is the set of all labels, which is $[True, False]$ for a classification task, and y_i can be $True$ or $False$.
- Objective predictive function $\rightarrow f(\cdot)$ that can be learned from training data denoted as $\{x_i, y_i\}$ and $x_i \in \mathcal{X}, y_i \in \gamma$. The objective function can be written, from a probabilistic point of view, as $P(Y|X)$.

Given a source domain D_S, a source task T_S, a target domain D_T, and a target task T_T, *transfer learning* aims to improve the learning of the target predictive function, $f_T(\cdot)$, in D_T by exploiting the knowledge in D_S and T_S where $D_S \neq D_T$ or $T_S \neq T_T$.

2.4.2 Transfer Learning Scenarios

The possible transfer learning scenarios can be:

- Different source and target feature spaces:

$$\mathcal{X}_S \neq \mathcal{X}_T$$

e.g. The images are taken from different devices as radiographs and photos.

- The marginal probability distributions of D_S and D_T are different:

$$P(X_S) \neq P(X_T)$$

e.g The images are taken at different hours of the day or anyway with different illumination.

- The label spaces of the two task are different:

$$\gamma_S \neq \gamma_T$$

e.g. Images need to be assigned different labels in the target task.

- There is a difference in the conditional probability distributions between the two task:

$$P(Y_S|X_S) \neq P(Y_T|X_T)$$

e.g. Source and target data are unbalanced about their classes. Often techniques, as oversampling and undersampling, are used since this scenario is quite common.

2.5 Transfer Learning Strategies

It is possible to categorize transfer learning strategies on the type of machine learning algorithm used [9]. Then, Transfer Learning can be divided in:

- **Inductive Transfer Learning**: This strategy aims to help improve the learning in the D_T using knowledge in D_S and T_S, where $T_s \neq T_t$. In this scenario, labeled data are available in the target domain, and depending upon if the source domain contains labeled data or not, this can be divided into two other subcategories.
- **Transductive Transfer Learning**: This setting requires that the T_S and the T_T are similar, but the domains may be different. This setting requires that the T_S and the T_T are the same, but the domains may be different. Additionally, some unlabeled data of D_T must be available at training time. If domains are different, this strategy takes the name of Domain Adaptation.
- **Unsupervised Transfer Learning**: No labeled data are presented in both domains, and the $T_S \neq T_T$.

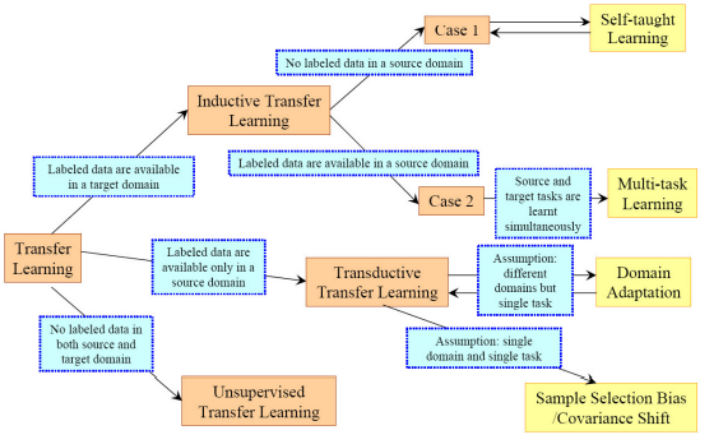


Figure 2.11: Transfer Learning Strategies [9]

2.6 How to perform transfer learning

A common way to perform transfer learning in the deep learning field is composed of these steps:

1. A pre-trained source model is selected from models available. Generally, models pre-trained on ImageNet are used. It is also possible to build and train a model over a large dataset.
2. The model for the current task is built. Then, the weights of the parts of the pre-trained model that are in common between the two networks are transferred.
3. The model obtained is finetuned on the target data.

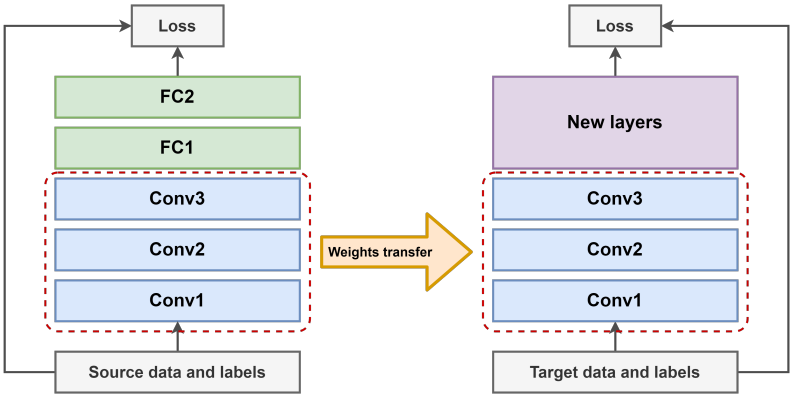


Figure 2.12: Example of Transfer Learning

2.7 Data Augmentation

State of the art deep networks have in the order of millions of parameters to be tuned, and they are profoundly dependent on big data to avoid overfitting. Moreover, a large dataset is also extremely important to obtain a good ability of generalization. Data Augmentation techniques try to solve the problem of data lacking in domains in which obtain more data is not feasible or too costly, e.g in medical image analysis. For instance, if the input data are images, to obtain, approximately, an infinite dataset, it is just needed to make some random transformations to our existing dataset as rotations and flips. Furthermore, Data Augmentation is also used to create CNNs that are invariant to illumination, translation, size, point of view or a combination of these transformations. This technique doesn't require that the transformations must be done beforehand, but it is possible to perform them on the fly. Common transformations are:

Flip: It's possible to flip images horizontally or vertically. It can increase the size of the dataset up to 4 times.

Rotation: It is possible to rotate the images randomly of an angle between $-\alpha$ and α.

Translation This operation helps the models to learn that the object to classify can be in different positions.

Noise For various reason is possible that some images can be affected by noise. To create a model resistant to the noise is possible to add it to some instance of the dataset generated through data augmentation.

Data augmentation is a powerful tool but requires some attention. Indeed, it is necessary to use only transformations that are useful in the target domain to avoid to generate images that the network would never see when deployed. Further, the transformations need to not alter the mapping function, i.e., exclusively transformations that not modify the labels should be applied. In Figure 2.13, C10+ and C100+ columns are the error rates achieved by using data augmentation, and they show that this technique achieves better results.

Method	Depth	Params	C10	C10+	C100	C100+
Network in Network [22]	-	-	10.41	8.81	35.68	-
All-CNN [32]	-	-	9.08	7.25	-	33.71
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57
Highway Network [34]	-	-	-	7.72	-	32.39
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73
ResNet [11]	110	1.7M	-	6.61	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58
	1202	10.2M	-	4.91	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07
	28	36.5M	-	4.17	-	20.50
with Dropout	16	2.7M	-	-	-	-
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33
	1001	10.2M	10.56*	4.62	33.47*	22.71
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**

Figure 2.13: Error rates of popular neural networks on the Cifar 10 and Cifar 100 datasets. Source [8]

Chapter 3

Related Work

This chapter introduces some recent works related to this thesis. The chapter contains a survey on transfer learnig in medical image analysis, on colorization methods, and few-shot learning.

3.1 Transfer Learning in Medical Image Analysis

In the last years in medical image analysis, there was a shift from systems designed entirely by humans to others based on machine learning. While in the past, the researcher needed to extract relevant features from data to solve problems, in the last years, this work is increasingly delegated to systems that can automatically perform this. Nowadays, the actual most powerful type of deep learning models is CNN. Even if this type of architecture achieves the top performance in most medical image analysis competitions, they present many challenges. One of these challenges is the lack of large labeled datasets. Indeed the number of medical images growing more and more, but the acquisition of appropriate annotations for these images is costly and requires specialized human effort. For these reasons, learning efficiently on a small dataset and create models robust to the noise of the labels is a crucial area of research. One of the solutions to the scarcity of labeled data is to reusing information learned from another task or different domains to improve the performance of the models, and this technique is known as *transfer learning*. According to the work of Cheplygina et al.[10], more than 50 different papers that use transfer learning have been published from 2012 to 2017. Menegola et al.[11] conducted some experiments in which compared training from scratch to fine-tuning models pre-trained on ImageNet or a dataset of retina images[12]. The dataset used was the Seven-Point Checklist Dermatology Dataset[13], and the results show that using transfer learning gives better results when a small

dataset is used. These experiments were performed over a too small scale to allows generalization. Raghu et al.[14] took too on this problem, and their work shows that transfer learning from ImageNet to medical image analysis offers only little benefit to performance and simple, lightweight models achieve comparable results to ImageNet architecture. According to their paper, on medical image analysis, transfer learning leads to convergence speedups (Figure3.1). Their experiments were conducted on two datasets: CheXpert[15] and a dataset of retina data[16]. In the work of Menegola et al., source and target dataset are composed of images in the RGB domain, while in Raghu et al.'s work the target dataset is composed of grayscale images. It may possible that the features learned on a different color space dataset are not useful to the new task. This may happen due to grayscale images in the RGB domain have the same data replicated on each channel, while colorful RGB images bring different information for each channel.

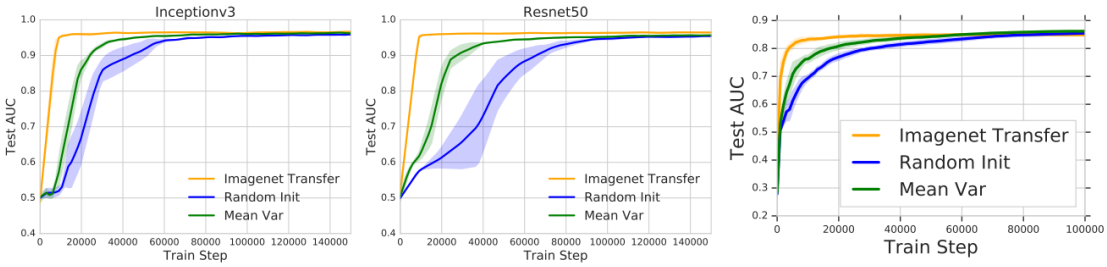


Figure 3.1: Convergence speedups with transfer learning. (a-b) Retina Dataset. (c) CheXpert Dataset. [14]

3.2 Image colorization

Since the difference between grayscale and colorful images may not help to use Transfer Learning among ImageNet dataset and grayscale medical ones, in this section, I'll go to talk about different techniques that are related to image colorization. These methods create a 3-channels picture from a 1-channel image, more known as grayscale. A quite common goal of these methods is to produce a plausible colorization, while for our aim, a colorization that can improve the performance of a pre-trained model. In the last years, the popularity of the colorization task is growing very quickly and various approaches and works were published, given us some incredible results (Figure 3.2). In

This problem was handled by different strategies:

- **Manual approach**: The colorization of the whole image is performed manually. This is costly because require specialized human work.

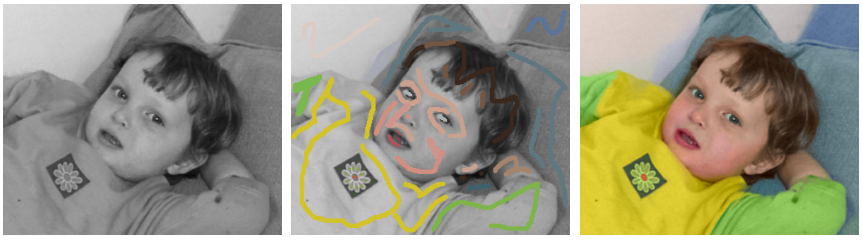
Figure 3.2: Example of colorization task.[17]

- **Manual human annotation**: This technique requires that a user puts some hints over the image. As for the method proposed by Levin et al.[18] that is based on the assumption that «neighboring pixels in space-time that have similar intensities should have similar colors» and they used a quadratic cost function. In Figure 3.3 is possible to see what steps this technique requires and that the number of hints to be given is high.
- **Automatic color transfer**: These methods expect the user to provide extra inputs as one or multiple images that the algorithm uses as a reference for colorization. An example of this approach is proposed by Gupta et al.[19] in which the user needs to supply a reference image that is semantically related to the target image. They extract features from the images chosen and exploit these features to perform the colorization process (Figure 3.4).
- **CNN based**: All the previous methods require human assistance that increases the time to generate the final image and the cost, while CNN based approaches aim to create a fully automatic method that can hallucinate a plausible colorization.

Since in this thesis, the target images are of a medical type, the focus has been on the last method that not require human interaction.

3.2.1 Colorful Image Colorization

In this paper, Zhang et al. [17] propose a fully automatic approach that produces «vibrant and realistic colorization». They trained their architecture (Figure 3.5) over ImageNet dataset in which images were converted to the Lab color space. The Lab color space is composed of three channels as the RGB format but the



(a) Input **(b)** Hints **(c)** Result

Figure 3.3: Example proposed in Colorization Using Optimization.[18]

(a) Input **(b)** Reference **(c)** Result

Figure 3.4: Example proposed in Image Colorization Using Similar Images.[19]

color information is encoded in this way:

- The **L channel** contains information about lightness intensity.
- The **a channel** encodes green and red.
- The **b channel** encodes blue and yellow.

By using this color space is possible to use the **L** channel as a grayscale image and the model must learn to predict only the other two channels. They treat the problem as a multinomial classification and quantize the **ab** output space with a grid size 10. After this step, only 313 **ab** pairs are used. At the end of the model, the two predicted channels **ab** are combined with the **L** channel to create the final output and the predicted image is compared to the ground truth, which is the original image in Lab color space. The image obtained can be reconverted to the RGB format.

3.2.2 ColorUNet

In their paper Billaut et al.[21] propose a lightweight architecture inspired by U-Net that, in general, is widely used in image segmentation. Similarly to Zhang et al. work, they approach the colorization problem as a classification task, working with a limitate set of colors (only 32). They work on the YUV color space that

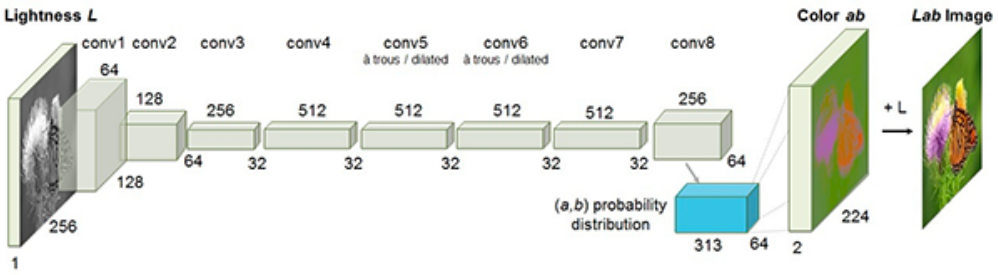
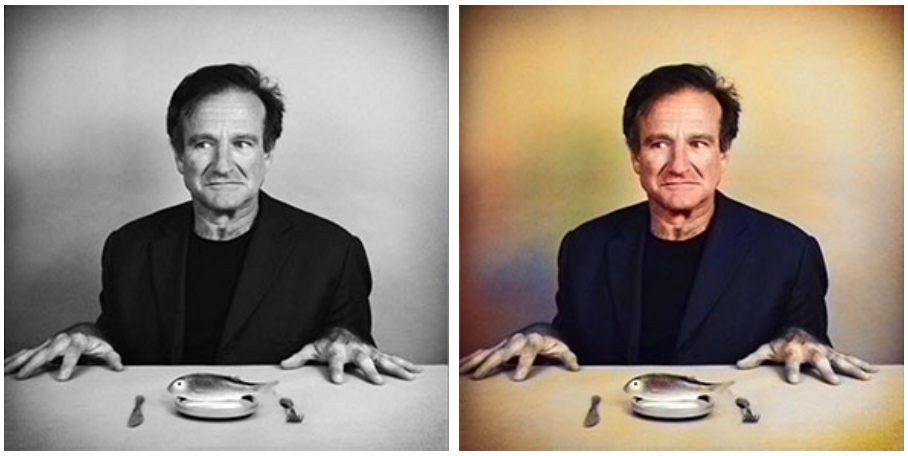


Figure 3.5: Architecture proposed by Zhang et al.[17]



(a) Input **(b)** Output

Figure 3.6: Example of an image generated from the architecture proposed by Zhang et al.[20]

allows separating the luminance information (**Y channel**) from the other chrominance components (**U and V channels**). The transformation between RGB and YUV color space is linear. Moreover working in this color space allows using the **Y channel** as a grayscale image and the model must predict only the chrominance components **UV**. Their architecture achieves quite good results (Figure 3.7) considering the limited number of colors that the model can predict.

3.2.3 (DE)2CO: Deep Depth Colorization

The previous papers train the networks by comparing the predicted images with the colorful version of the grayscale images. Since that is not possible with the medical images, because there is not a colorful version of them, other approaches have been searched. A possible solution was found in the work of Carlucci et al. [22]. They try to find an optimal colorization mapping for a given pre-trained architecture by connecting their colorization module to a standard CNN model

(a) Input **(b)** Prediction **(c)** Ground Truth

Figure 3.7: ColorUNet output.[21]

pre-trained over ImageNet. They train the colorization module freezing the pre-trained model and in this way, the model learns a good colorization that improves the performance (Figure 3.8). The main benefit of this work is that colorization is learned only by using labels.

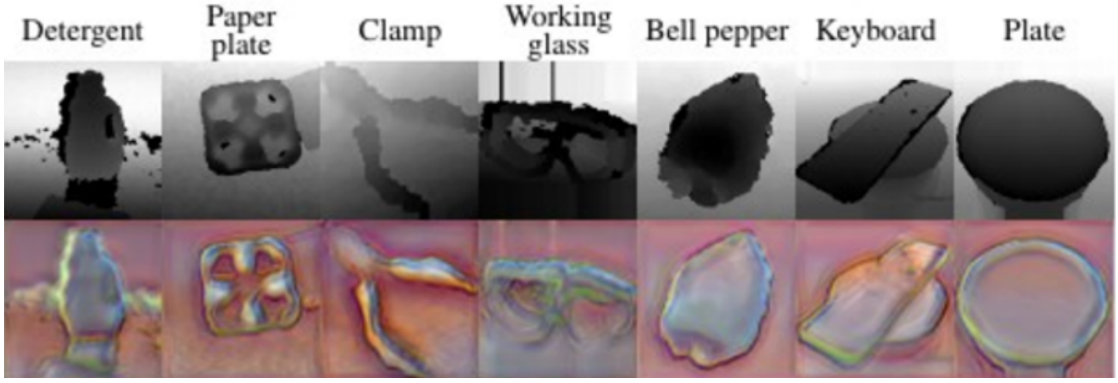


Figure 3.8: (DE)2CO results.[22]

3.2.4 Colorization in the medical domain

In medical image analysis, some studies use no-handcrafted colorization to improve performance. For instance, Teare et al.[23] have found that it is possible to significantly improve the classification accuracy of a network pre-trained on ImageNet by using genetic algorithms to discover an optimal set of preprocessing transformations for mammography false color enhancement. In the case of CT scans, pseudocolor images can be generated by applying different windows/level settings on each channel, similar to the process employed by radiologists to enhance the contrast of different tissues[24].

3.3 Few-Shot Learning

Few-Shot Learning (FSL) is a technique in which a model is fed with a small amount of data, which is the opposite of a standard machine learning approach. A variety of methods was proposed to tackle the few-shot learning task, that is possible to approximately divide into:

- **Meta-learning methods**: To generalize over new unseen tasks, these methods introduce a meta-learning paradigm to learn an across-task meta-learner. Commonly, Recurrent Neural Networks are used. Some example of these methods are proposed by Fin et al.[25], Lee et al.[26], and Jamal et al.[27]
- The **Metric-learning methods** use a simpler architecture to learn an embedding space in which the instances of the same category are close together. For instance, Wertheimer et al.[28], and Lifchitz et al.[29] work.
- **Data Augmentation methods**: The main idea is to learn a data augmentation to generate more examples from the few available. As the methods proposed by Chen et al.[30] and Zhang et al.[31].
- The **Semantics-based methods** use extra semantic information as category labels, attributes, and natural language description. For example, the method proposed by Schwartz et al.[32].

Chapter 4

Methods and materials

In this chapter, I'll describe the type of modules that were used, what experiments were done, and which datasets we have focused our attention on.

4.1 Datasets

An important aspect of this thesis has been to choose what datasets use.. In the beginning, I focused my attention on datasets of medical images with a quite large number of samples and some previous related works to compare our results with theirs. Later, the attention was shifted on some smaller datasets of various anatomy regions.

4.1.1 CheXpert

The first dataset selected was the CheXpert[15] that is a large public dataset for chest radiograph interpretation, consisting of 224,316 chest radiographs for the training and 234 for the validation. Each grayscale image is the version scaled of the original radiograph to have the smaller side equal to 320px. The images were labeled for the presence of 14 different observations as positive, negative, and uncertain. The labels are not mutually exclusive and the 14 observation are:

- No Finding
- Enlarged Cardiom.
- Cardiomegaly
- Lung Lesion
- Lung Opacity
- Edema
- Consolidation
- Pneumonia
- Atelectasis
- Pneumothorax
- Pleural Effusion
- Pleural Other
- Fracture
- Support Devices

As the current competition on this dataset, this thesis is focused only on five of this observation: Atelectasis, Cardiomegaly, Consolidation, Edema, and Pleural Effusion.

Labels policy

Since each observation can be uncertain, the paper related to the dataset[15] provides 5 different approaches to these labels:

- **Ignoring**: All images that have uncertain labels are ignored. (*U-Ignore*)
- **Binary mapping**: Each uncertain label is substituted with:
  - 0, in this way this image is considered negative. (*U-Zeros*)
  - 1, in this way this image is considered positive. (*U-Ones*)
- **Self-training**: This policy is used as an unsupervised technique. More in detail, before convergence this policy work as U-Ignore, after each uncertain label was substituted by the model prediction and the model is still trained. (*U-SelfTrained*)
- **U-MultiClass**: Uncertain labels are considered as an additional class. (*U-MultiClass*)

To simplify our work has been selected, for each observation, one policy between *U-Zeros* and *U-Ones* considering the best result obtained on the dataset paper. The policies chosen in this way are reported in Table 4.1.

	Atelectasis	**Cardiomegaly**	**Consolidation**	**Edema**	**PleuralEffusion**
Policy	U-Ones	U-Zeros	U-Zeros	U-Ones	U-Ones

Table 4.1: Labels policies chosen for CheXpert.

4.1.2 MURA

MURA [33] is a dataset of musculoskeletal radiographs consisting of 40,561 multiview radiographic images from about 12,000 patients. The dataset is subdivided into seven different radiographic study types:

- Elbow
- Finger
- Forearm
- Hand
- Humerus
- Shoulder
- Wrist

Each study was manually labeled as normal or abnormal by board-certified radiologists from Stanford Hospital.

4.1.3 ChestX-ray14

The ChestX-ray14[34] is a Chest X-ray dataset that contains about 112,000 X-ray images with the label for 15 different findings. The classes that describe each image are:

- Atelectasis
- Consolidation
- Infiltration
- Pneumothorax
- Edema
- Emphysema
- Fibrosis
- Effusion
- Pneumonia
- Pleural Thickening
- Cardiomegaly
- Nodule Mass
- Hernia
- No Finding

These labels have been extracted from the radiological report associated with the image by using Natural Language Processing. For this reason, the labels could be erroneous but the estimate accuracy is over 90%. Each image has a size of 1024×1024, and for our purpose, they were scaled to 320×320 to fit the size of the images of the CheXpert dataset. Furthermore, only 5 diseases were used and that are Atelectasis, Cardiomegaly, Consolidation, Edema, and Effusion. These classes were chosen to match the observation chosen in the CheXpert dataset to work on a similar task.

4.1.4 CheXpert subsets

To see how the performance change over different dataset sizes, some datasets have been created from the CheXpert dataset. The subsets have sizes of 10%, 5%, 1%, 0.5%, 0.2%, and 0.1% compared to the original one. For each dimension, three datasets were created by choosing randomly the images from the source dataset. Eighteen different datasets were built, and in Table 4.2 are reported their composition.

Size	Random seed	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion
223414 (100%)	-	67115	27000	14783	65230	97815
22341 (10%)	8	6647	2770	1509	6519	9796
22341 (10%)	8829	6665	2691	1437	6624	9684
22341 (10%)	2901	6698	2641	1535	6520	9682
11171 (5%)	8	3296	1415	788	3281	4874
11171 (5%)	8829	3338	1400	727	3330	4873
11171 (5%)	2901	3346	1346	793	3273	4925
2234 (1%)	8	664	289	154	654	986
2234 (1%)	8829	680	286	143	641	1004
2234 (1%)	2901	675	289	179	655	999
1117 (0.5%)	8	335	146	67	337	498
1117 (0.5%)	8829	348	144	79	324	501
1117 (0.5%)	2901	326	144	95	315	502
447 (0.2%)	8	131	57	30	118	202
447 (0.2%)	8829	134	53	28	148	193
447 (0.2%)	2901	144	58	41	124	213
223 (0.1%)	8	65	31	12	62	105
223 (0.1%)	8829	63	25	15	72	95
223 (0.1%)	2901	76	32	14	68	105

Table 4.2: Composition of CheXpert subsets. For each observation, the number of positive labels is reported.

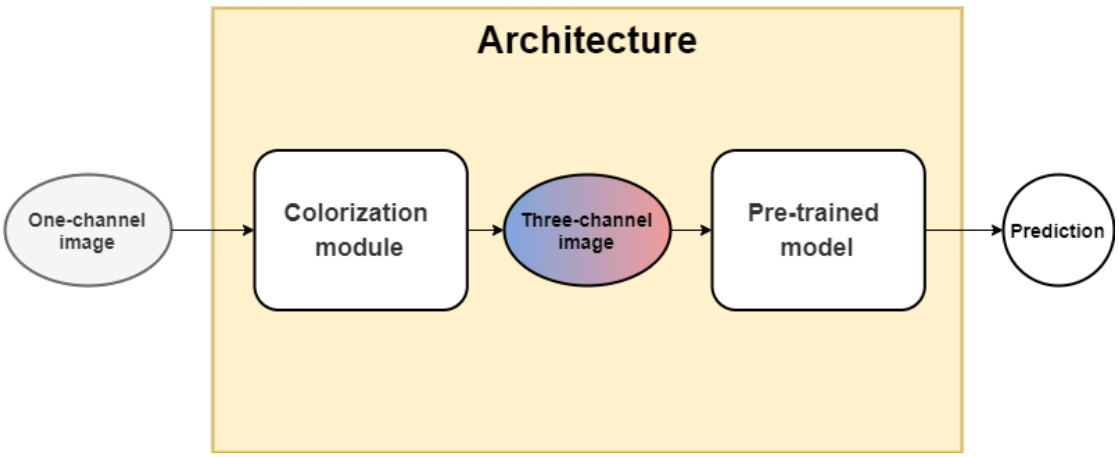


Figure 4.1: Schematic representation of architecture used.

4.2 Architecture

Intending to analyze different modules with diverse pre-trained models, I created a modular architecture that can be divided into two main parts (Figure 4.1):

1. **Colorization module**: The first one contains the module that learns how to transform the input gray-scale images to three-channels images.

2. **Pre-trained model**: The second part is designed so that it can contain any kind of model that accepts as input a three-channel image.

4.3 Colorization modules

As I just said previously, this thesis aim to create some modules that can transform medical images from grayscale to another domain so that they improve the performance of most-used pre-trained models. To achieve this result we have created and tested more than 20 different modules inspired by previous works. In this thesis, I'll describe the most interesting.

4.3.1 DECO

Since this thesis has been strongly inspired by the work of Carlucci et al.[22] because it is a technique that not requires a colorful reference during the training, I recreated and tested the model proposed in their paper. From their architecture, I have created three different modules that share the features extractor part, but use different up-sampling techniques. More in detail, the first part of the modules is equal to the $(DE)2CO$ module (Figure 4.2), while the up-sampling layer changes across modules.

Three different up-sampling approaches were tested:

1. **Transpose convolution (Deconvolution)**: It is very similar to the convolution operation, only that the convolution matrix is transposed (Figure 4.3). Therefore the result is that the output grows instead of reducing. Unfortunately, deconvolution can easily have an uneven overlap, and in particular when the kernel size is not divisible by the stride. In principle, the network could learn weights to bypass this behavior, but in practice, neural networks have difficulties to avoid it completely. This problem may generate some artifacts on a variety of scales, called Checkerboard Artifacts (Figure 4.4).

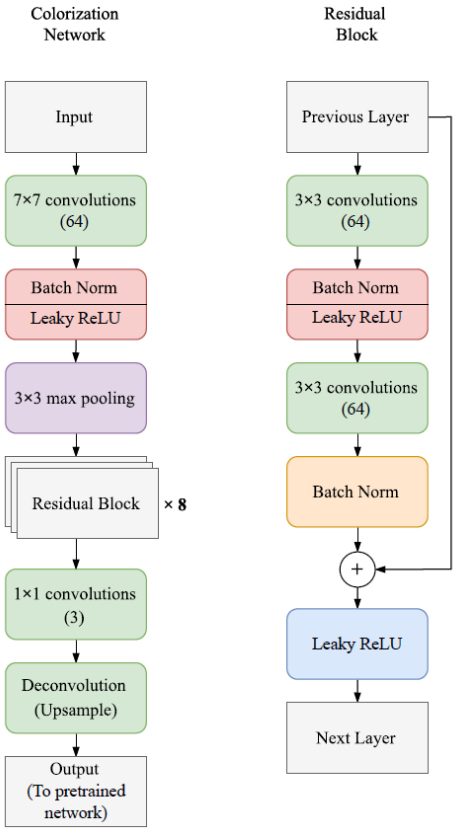


Figure 4.2: $(DE)^2CO$ colorization network. On the left, there is the overall architecture; on the right, there are the details of the residual block.[22].

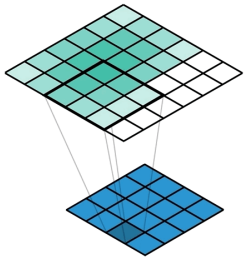


Figure 4.3: Deconvolution operation.[35]

Figure 4.4: Examples of checkerboard artifacts [36].

2. **Resized Convolution**: This technique is similar to the previous one but the up-sampling to a higher resolution and convolution are separated. Thus, the low-resolution image is up-sampled by using an interpolation algorithm, like nearest-neighbor interpolation, to become bigger than the original image, and after, the image is convolved [36].

3. **Pixel shuffle**: While the previous approach is based on the idea of generating a high-resolution image from a low-resolution one (low to high), this procedure uses the multi-layer feature map to recreate the image at the original size (map to high) (Figure 4.5). So the last part of the module takes an input of shape $H \times W \times C \cdot r^2$ which is rearranged to $rH \times rW \times C$. Where:

$$H = \text{height}$$
$$W = \text{width}$$
$$C = \text{number of output channel}$$
$$r = \text{resize factor}$$

This approach is used for high-resolution image generation and helps to remove checkerboard artifacts [37].

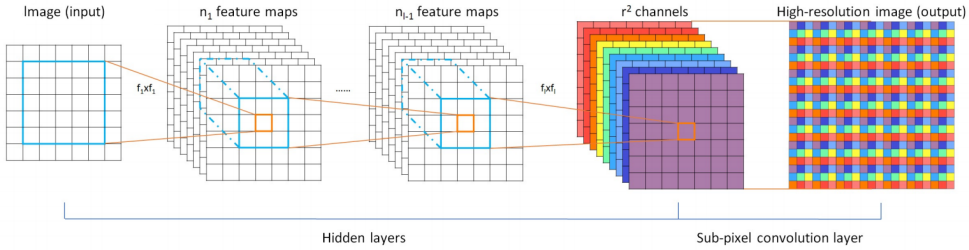


Figure 4.5: Pixel shuffle technique.[37]

4.3.2 U-Net

Since U-Net architectures in image colorization are widely used, in this thesis a module based on this architecture was created. Our module was mainly inspired by the work of V. Billaut et al.[21] because of their architecture is smaller than others and requires a comparable training time with other modules. The main components of our architecture are three (Figure 4.6):

1. **DownLayer**: Used in the first half of the module and is composed of Convolution, LeakyReLU, and BatchNorm. It aims to extract feature and reduce the spatial dimensionality.

2. **UpLayer**: Used in the second half of the module and is composed of Transpose Convolution, LeakyReLU, and BatchNorm. It gets as input the output of the previous UpLayer if exist, and from the DownLayer that is placed at the same high. Its purpose is to up-sampling the inputs.

3. **OutLayer**: Similar to the UpLayer and its goal is to create the output image. It doesn't use BatchNorm layers.

The blocks described above were used to create the final architecture of the ColorU module and Figure 4.7 shows how they were combined.

4.4 Pre-Trained models

The modules previously described are followed by a pre-trained model. In this thesis, two from the most widely used CNN architectures have been taken into consideration to see how different models interact with modules. The architectures used are ResNet18 and DenseNet121.

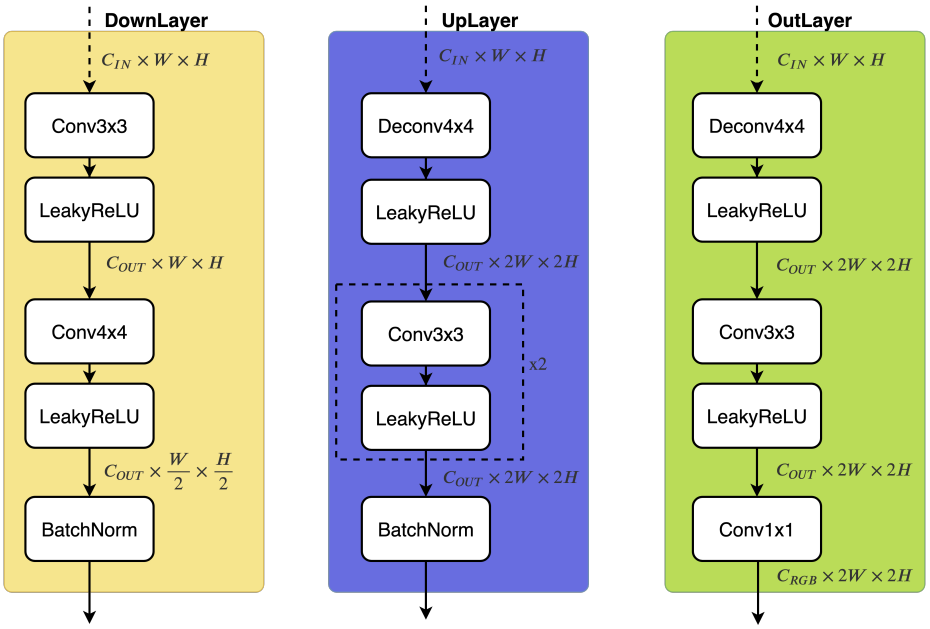


Figure 4.6: Components of the ColorU module.

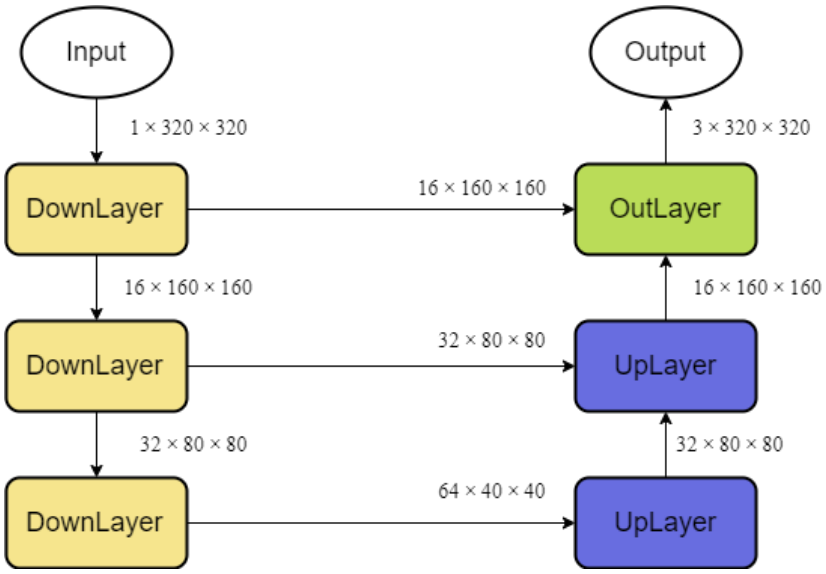


Figure 4.7: Structure of the ColorU module.

4.5 Training and evaluation

At the beginning of this thesis, a series of tests have been planned to reach our aim, and two different learning phases have been defined. To know how good are results is essential to have a baseline. In this thesis, the baselines are generated by finetuning the pre-trained models without colorization modules either by freezing all layers except the last one or by unfreezing all layers. The first part of the tests has been focused on training modules over the CheXpert dataset, and over how the performance change from the baseline and between the different modules. So the learning phase was divided into two main steps. In the first one, the module is trained while the pre-trained model is frozen except for the last layer (Figure 4.8a). Following the initial hypothesis, this phase should allow the module to learn how to convert a grayscale image to a three-channel image that improves the performance of the pre-trained model. Instead, in the second learning phase, all the architecture is trained after cleaning the last layer (Figure 4.8b), i.e., after reinitializing the weights of the last layer. Cleaning the last layer is helpful to escape from local minima. In the second part of the experiments, different transfer learning strategies were tested. So, the modules were tested on MURA and ChestX-ray14 datasets. In this stage, a new training strategy was introduced in which all the architecture was frozen except for the last layer (Figure 4.8c). Below, the experiments were reported in detail, while in Appendix B the parameters for replicate the results are reported.

4.5.1 Experiments on CheXpert

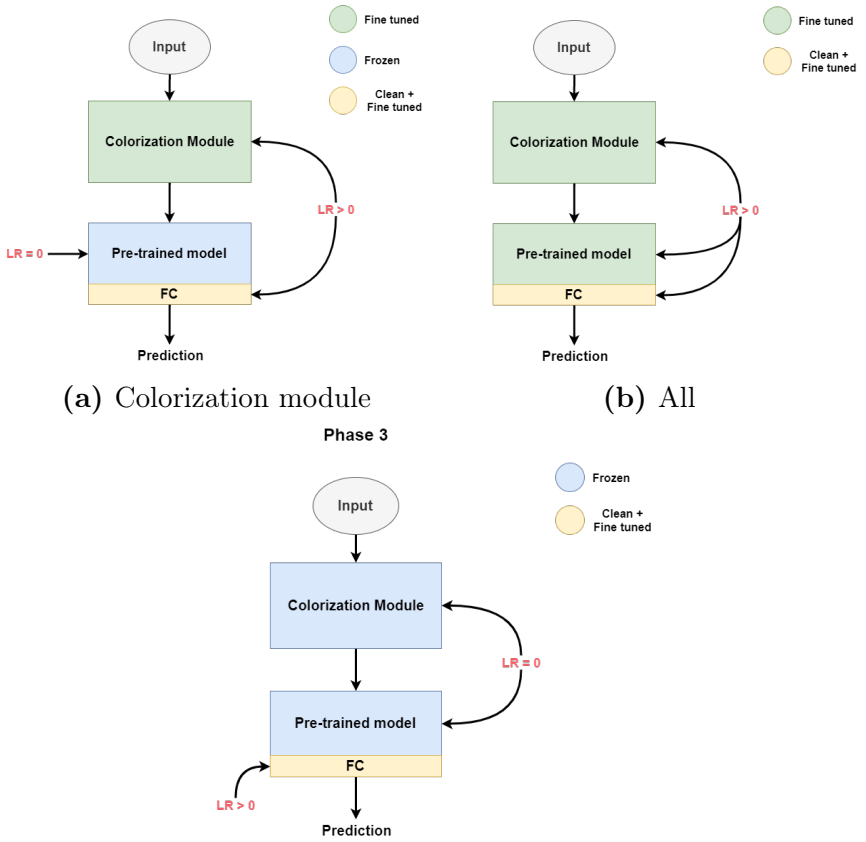
In this dataset, each input image can hold more than one positive observation. Consequently, it is possible to handle this problem by training one model for each disease as a binary classification task or, how it was done in this thesis, as a multi-label classification task by using as loss function the BCEWithlogitsLoss[38]. This loss is a combination of Sigmoid layer and the BCELoss and is defined as:

$$l(x, y) = mean(L) = \{l_1, \ldots, l_N\},$$
$$l_n = -w_n[y_n \cdot log\sigma(x_n) + (1 - y_n) \cdot (1 - \sigma(x_n))]$$

The last layer of the pre-trained model was substituted for a fully connected layer of five outputs (Figure 4.9). The tests on this dataset were conducted to find an answer to two questions:

1. What is the strategy that performs better?
2. Is the best strategy depending on the size of the dataset?

The experiments performed were:



(a) Colorization module **(b)** All

(c) Last layer

Figure 4.8: Learning strategies.

- Baseline 1: The pre-trained models without colorization modules were trained by taking frozen al the architecture except for the last layer.

- Baseline 2: All the layers of pre-trained networks without colorization modules were finetuned.

- A series of experiments in which only the colorization modules and the last layer was trained.

- A series of tests in which all architecture trained in the previous experiment were finetuned after cleaning the last layer.

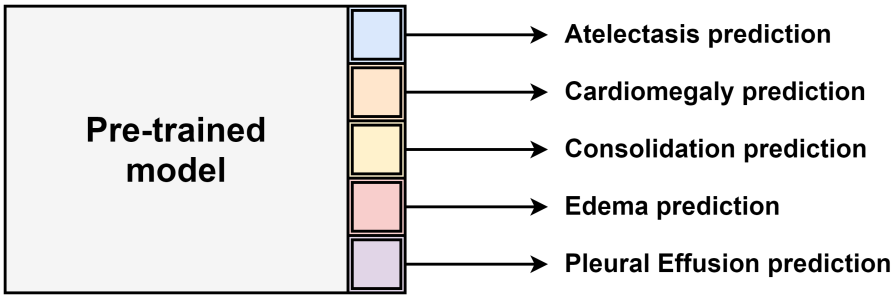


Figure 4.9: Changes to the pre-trained model.

4.5.2 Experiments on MURA

This dataset is composed of images of different body regions and with different tasks than CheXpert. The target choice is the shoulder, and the task is to identify the status of this body part (normal or not). Since it is a classical binary classification problem, the Cross-Entropy Loss was used [38].

$$loss(x, class) = -x[class] + log(\sum_{j} x[j])$$

On Mura was tested if it can be transferred the colorization knowledge learned on CheXpert to a smaller dataset with images in a different domain.

- The same experiments done on CheXpert were re-proposed on this dataset.
- A test in which networks composed of a colorization module trained on CheXpert and a ResNet18 model trained on ImageNet was taken frozen except for the last layer.
- A test in which networks composed of a colorization module trained on CheXpert and a ResNet18 model trained on ImageNet was taken frozen except for the last layer and the colorization modules.
- An experiments in which all architecture trained in the previous experiment were finetuned after cleaning the last layer.

4.5.3 Experiments on ChestX-ray14

Also, on ChestX-ray14 dataset was tested what type of improvements can be brought by transferring the colorization module pre-trained on CheXpert. Since the target domain and the Label Space is the same as the source dataset, the loss function used, and the last layer of the architectures are the same as used for CheXpert. Three different tests were conducted:

- The baseline in which only the last layer of a ResNet18 model pre-trained on ImageNet was finetuned.
- A test in which a network composed of a colorization module trained on CheXpert and a ResNet18 model trained on ImageNet was taken frozen except for the last layer.
- An experiment in which the architecture was composed of a module and a model pre-trained on CheXpert. Only the last layer of the network was finetuned.

4.5.4 Data augmentation and experimental setup

Each experiment was done by applying the same data augmentation that is composed in this way (Figure 4.11):

- **Random Crop**: Images that are not square was randomly cropped to a square form.
- **Resize**: If an image has a size greater of 320x320 it is scaled to this size.
- **Random Rotation**: Each image is rotated with a probability of 0.75 by an angle choose randomly between -10° and 10°.
- **Random Zoom:** Each image is scaled with a probability of 0.75 by a factor randomly choose between 1.0 and 1.1.

Other common parameters used were:

- Each image was normalized by using statistics calculated over the CheXpert dataset. Mean = 0.5028, std = 0.2902.
- The SGD was used as **optimizer** combined with a learning strategy called One Cycle Policy. In the policy proposed by Leslie et al.[39], the learning rate and the momentum vary over the training time, as reported in Figure 4.10.
- AUC was the **metric** chosen for comparing the various result. Since it is a binary metric and there are more label classes in the datasets chosen, for each observation was calculated the AUC and the mean of AUCs so obtained was used.
- Each experiment was done three times to observe how much the non-deterministic initialization influences the final result.
- Each N iterations the model is saved (The number of iteration changes across datasets. See AppendixB for details).
- The AUC reported is the best one over checkpoints.

The number of trainable parameters for different training strategies is reported in Table 4.3.

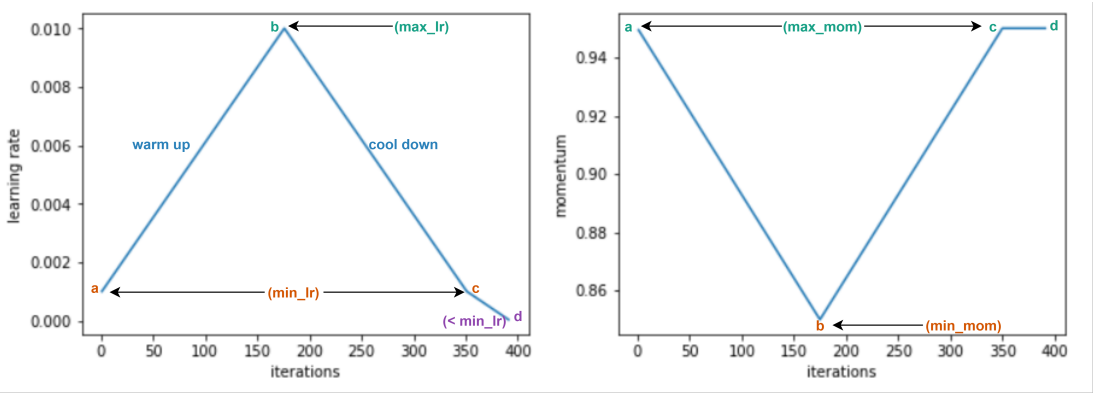
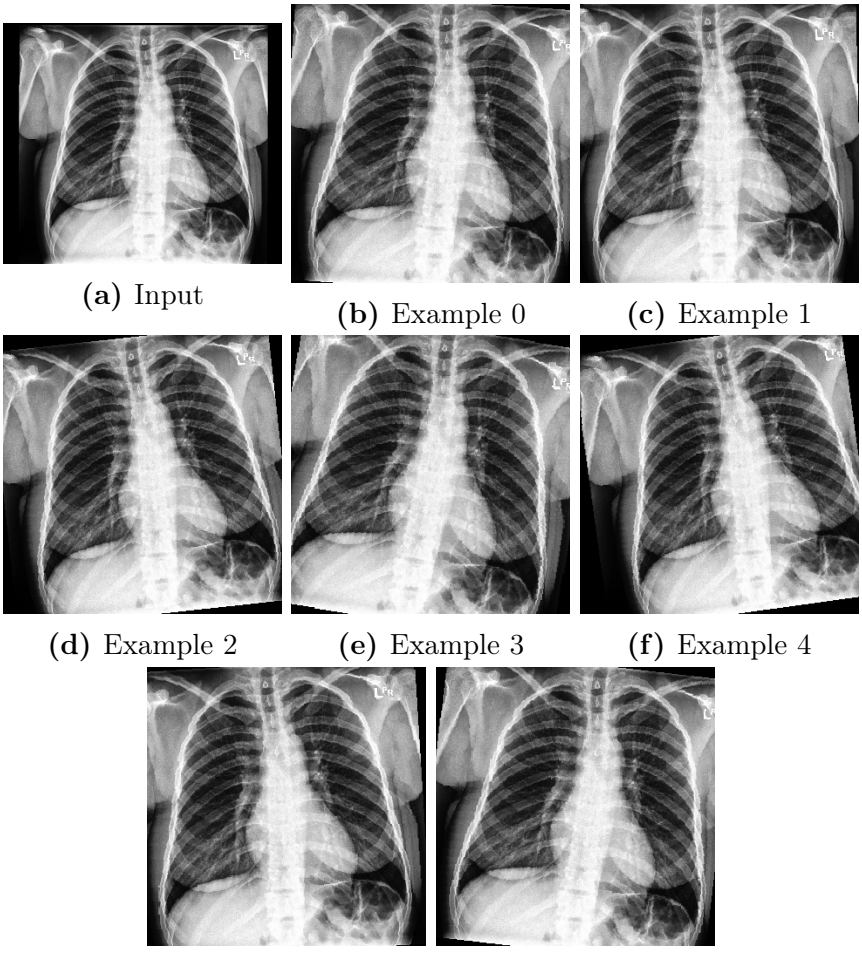


Figure 4.10: Example of learning rate and momentum by using One Cycle Policy.

Model	Colorization Module	Training Strategies	Trainable parameters
ResNet18	No	Last layer	2565 (0.02%)
ResNet18	DECONV	Colorization Module	599176 (5.36%)
ResNet18	PixelShuffle	Colorization Module	602005 (5.39%)
ResNet18	ColorU	Colorization Module	195224 (1.75%)
ResNet18	No	All	11179077 (100%)
ResNet18	DECONV	All	11775688 (105.34%)
ResNet18	PixelShuffle	All	11778517 (105.36%)
ResNet18	ColorU	All	11371736 (101.72%)
DenseNet121	No	Last layer	5125 (0.07%)
DenseNet121	DECONV	Colorization Module	601736 (8.65%)
DenseNet121	PixelShuffle	Colorization Module	604565 (8.69%)
DenseNet121	ColorU	Colorization Module	197784 (2.84%)
DenseNet121	No	All	6958981 (100%)
DenseNet121	DECONV	All	7555592 (108.57%)
DenseNet121	PixelShuffle	All	7558421 (108.61%)
DenseNet121	ColorU	All	7151640 (102.77%)

Table 4.3: The number of trainable parameters on different training strategies.



(a) Input **(b)** Example 0 **(c)** Example 1

(d) Example 2 **(e)** Example 3 **(f)** Example 4

(g) Example 5 **(h)** Example 6

Figure 4.11: Examples of data augmentation.

Chapter 5

Results

In this chapter, I'll describe the results of each test showing how the different modules modify the performance of the architecture.

5.1 Training modules over CheXpert

The first pre-trained model used was ResNet18 and in Table 5.1 are reported the result of tests (Figure 5.5). Experiments marked in blue are baselines and from these is possible to observe that when only the last layer is trained the model can not reach a high result. These could be provoked by the difference between the chest X-Ray images used and the standard images of ImageNet on which the model was trained. Instead by training entirely the model, it produces results that are better than 6.2-18.5% over observations and an increment of 12.8% over the mean. The results show that using colorization modules, the performance of the entire network increases by modifying the spatial domain of the input images. By using modules proposed in this thesis is possible to reach an increment of performance of about 6-7% compared to training only the last layer of the pre-trained model without colorization modules. This part of the tests aims to teach the modules on how to transform the one-channel input images into three-channels images. The output of the colorization modules was extracted to see what the colorization modules had learned.

CheXpert			
Model	**Colorization Module**	**Transfer strategy**	**AUC**
ResNet18	No	Last layer	0.784 ± 0.005
ResNet18	DECONV	Module	0.840 ± 0.003
ResNet18	PixelShuffle	Module	0.834 ± 0.004
ResNet18	ColorU	Module	0.839 ± 0.008
ResNet18	No	All	0.896 ± 0.002
ResNet18	DECONV	All	0.889 ± 0.003
ResNet18	PixelShuffle	All	0.889 ± 0.002
ResNet18	ColorU	All	0.893 ± 0.003
DenseNet121	No	Last layer	0.786 ± 0.002
DenseNet121	DECONV	Module	0.835 ± 0.003
DenseNet121	PixelShuffle	Module	0.843 ± 0.008
DenseNet121	ColorU	Module	0.839 ± 0.005
DenseNet121	No	All	0.898 ± 0.002
DenseNet121	DECONV	All	0.892 ± 0.002
DenseNet121	PixelShuffle	All	0.896 ± 0.001
DenseNet121	ColorU	All	0.897 ± 0.003

Table 5.1: Tests over CheXpert using ResNet18 and DenseNet121.

In Figure5.1 these outputs are reported and more precisely:

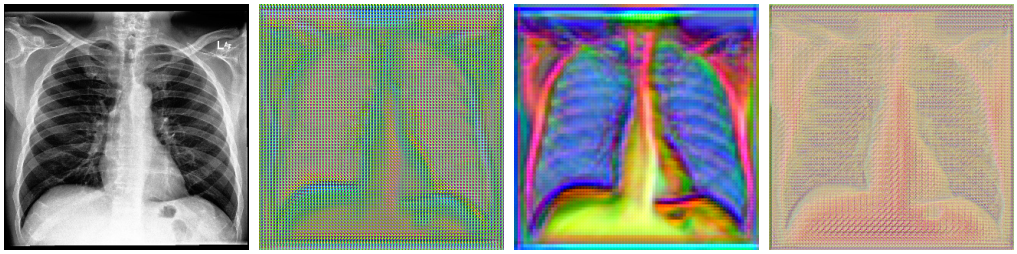
(a) It is the one-channel input image that enters in the architecture.

(b) It is the three-channels image that exits from the DECONV modules.

(c) It is the three-channels image that exits from the Pixel Shuffle modules.

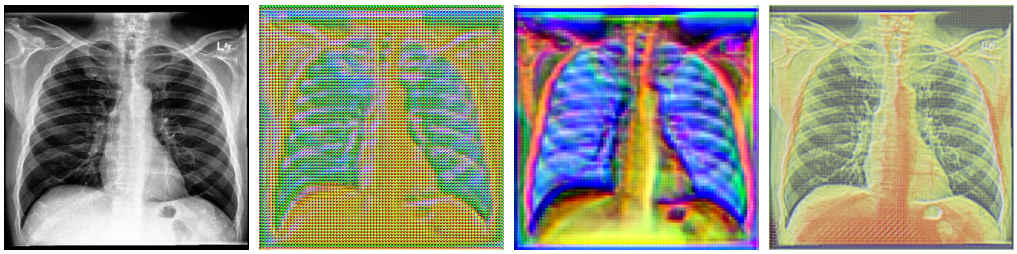
(d) It is the three-channels image that exits from the ColorU modules.

At first, it may seem that only the Pixel Shuffle module can convert the input image to a correct RGB image, but even if a CNN is based on how the mammal visual cortex work it doesn't *"see"* in the same way in which we are used to observing the world. Indeed looking at the AUCs after this first step, the DECONV module reaches the best performance even if it generates images that seem to be useless for human reading at first look. The last part of these experiments was focused on finetuning the entire architecture trained previously after cleaning the last layer. This is an important step since it allows the model to modify the weights learned on ImageNet that could be not the right one for this task. The results obtained from these tests show that after this step the whole architecture reaches slightly worst result comparing to training only the pre-trained model. As before the output of



(a) Input **(b)** DECONV **(c)** Pixel Shuffle **(d)** ColorU

Figure 5.1: Outputs of colorization modules after first training mode. (ResNet18)

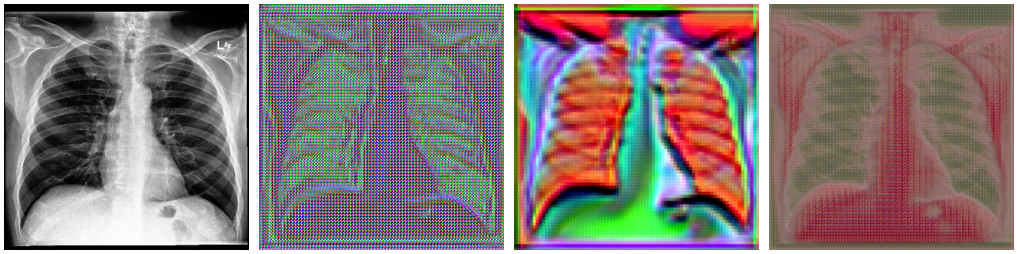


(a) Input **(b)** DECONV **(c)** Pixel Shuffle **(d)** ColorU

Figure 5.2: Outputs of colorization modules after finetuning all the architecture. (ResNet18)

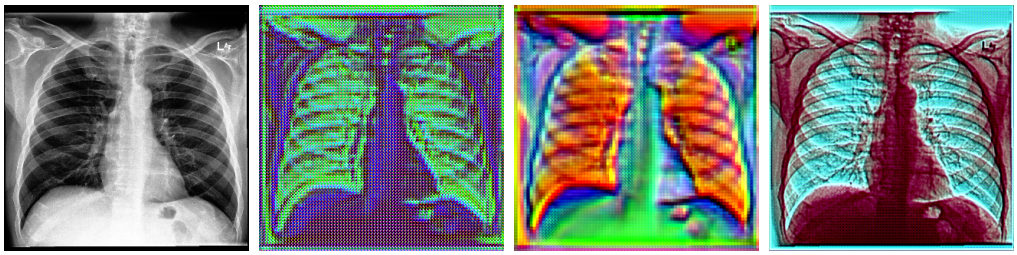
the modules has been saved and reported in Figure 5.2. Comparing the outputs of the modules before and after the last finetuning is possible to notice that the images produced contain more visual information. While the ColorU's output now is very similar to the input image, in the Pixel Shuffle the details over the ribcage are increased. Moreover, it is possible to notice that different parts of the images that share the same material have similar colorization, for example, regions filled with air.

After completing the experiments over ResNet18, it was tested how the results change by using a different pre-trained model. So the previous tests have been replicated by substituting the ResNet18 model with the DenseNet 121 one. Table 5.1 and Figure 5.6 contains the results that have been obtained and the behavior of this new architecture is comparable to the previous. Also by observing the output images (Figure 5.1 and Figure 5.3, Figure 5.2 and Figure 5.4), it is not possible to notice significant differences between the two architectures. It is quite interesting to notice that not only a random initialization leads to different final results, as expected, but also how it changes the final colorization. In Figure 5.7 is possible to observe this effect over the Pixel Shuffle module. In Appendix A is possible to find the results in detail, while in Figure 5.8 the ROC curves are reported.



(a) Input **(b)** DECONV **(c)** Pixel Shuffle **(d)** ColorU

Figure 5.3: Outputs of colorization modules after first training mode. (DenseNet121)



(a) Input **(b)** DECONV **(c)** Pixel Shuffle **(d)** ColorU

Figure 5.4: Outputs of colorization modules after finetuning all the architecture. (DenseNet121)

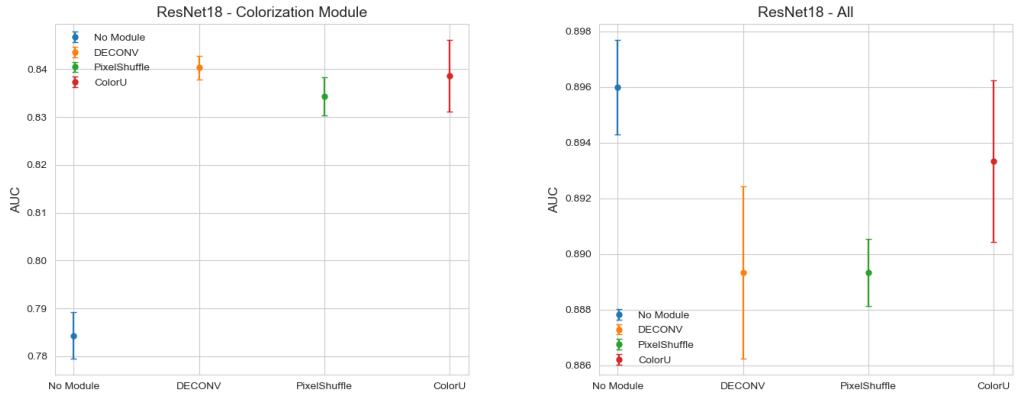


Figure 5.5: AUCs by using different modules. (ResNet18)

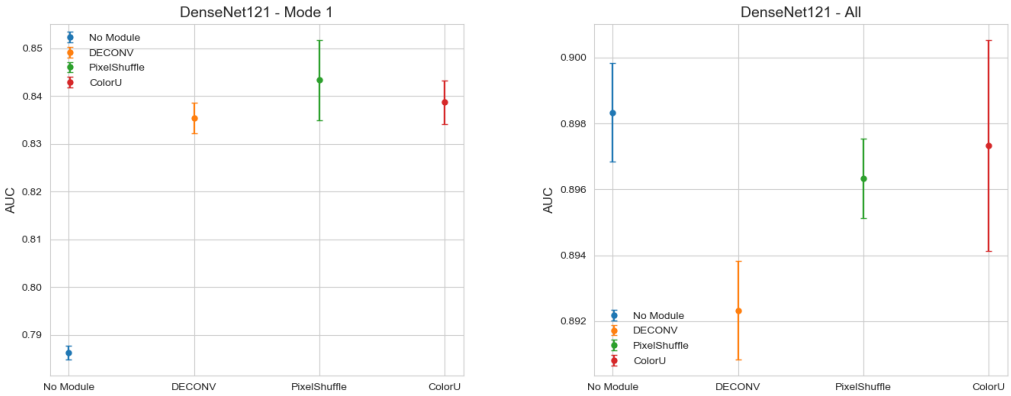


Figure 5.6: AUCs by using different modules. (DenseNet121)

Pre-trained model ResNet18

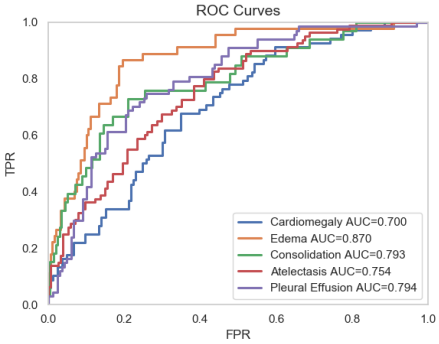
(a) Test1 **(b)** Test2 **(c)** Test3

Pre-trained model DenseNet121

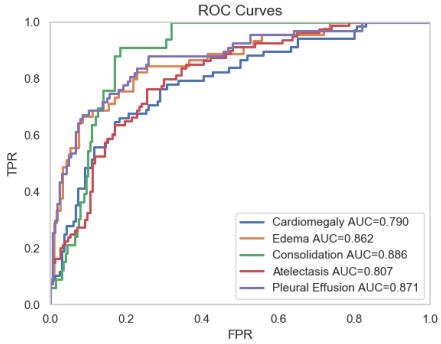
(d) Test1 **(e)** Test2 **(f)** Test3

Figure 5.7: Outputs of Pixel Shuffle modules from different initialization.

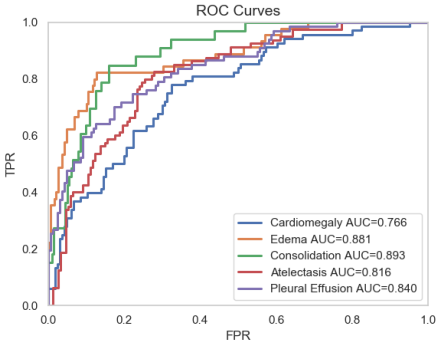
Colorization Module



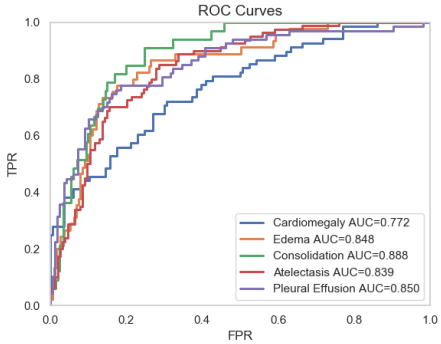
(a) No module



(b) DECONV

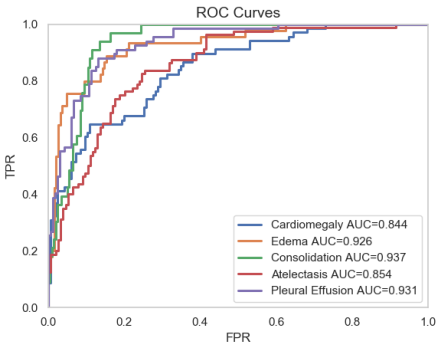


(c) PixelShuffle

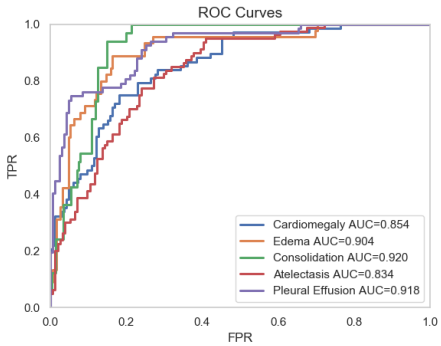


(d) ColorU

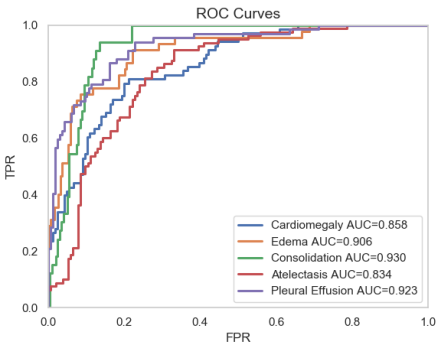
All



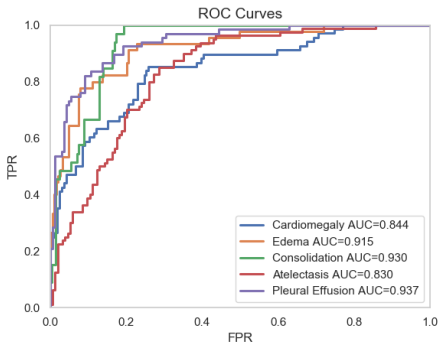
(e) No module



(f) DECONV



(g) PixelShuffle



(h) ColorU

Figure 5.8: Roc curves of tests with ResNet18.

5.2 Experiments over MURA

Over this dataset, it was tested how the modules perform over a smaller dataset and if it is possible to transfer the information for colorization learned on a different body region to another one. The same tests performed on CheXpert were redone and the results can be found in Table 5.2. The outcomes show the same trend seen in the CheXpert experiment, but when the network is not frozen, the results become worst. This behavior could happen due to the smaller dataset.

MURA			
Model	**Colorization Module**	**Transfer strategy**	**AUC**
ResNet18	No	Last Layer	0.730 ± 0.011
ResNet18	DECONV (Scratch)	Colorization Module	0.735 ± 0.009
ResNet18	PixelShuffle (Scratch)	Colorization Module	0.746 ± 0.012
ResNet18	ColorU (Scratch)	Colorization Module	0.765 ± 0.013
ResNet18	No	All	0.824 ± 0.004
ResNet18	DECONV (MURA)	All	0.792 ± 0.006
ResNet18	PixelShuffle (MURA)	All	0.790 ± 0.006
ResNet18	ColorU (MURA)	All	0.803 ± 0.007

Table 5.2: Tests over MURA using ResNet18 model (Baseline + Modules from scratch)

After this step, the architecture with modules pre-trained on CheXpert was tested by taking frozen all the architecture except the last layer and, after, by freezing all the layers except the output one (Table 5.3).

MURA			
Model	**Colorization Module**	**Transfer strategy**	**AUC**
ResNet18	DECONV (CheXpert)	Last Layer	0.709 ± 0.020
ResNet18	PixelShuffle (CheXpert)	Last Layer	0.671 ± 0.013
ResNet18	ColorU (CheXpert)	Last Layer	0.675 ± 0.039
ResNet18	DECONV (CheXpert)	All	0.790 ± 0.007
ResNet18	PixelShuffle (CheXpert)	All	0.775 ± 0.005
ResNet18	ColorU (CheXpert)	All	0.800 ± 0.013

Table 5.3: Tests over MURA using ResNet18 model with colorization modules pre-trained on CheXpert (Last layer and All).

The last set of experiments on this dataset is similar to the tests performed on CheXpert, but with modules pre-trained on CheXpert (Table 5.4). These tests seem to confirm that performing transfer learning from a dataset of a different body region not help to obtain better outcomes. Indeed, there aren't significant differences between results obtained by using colorization modules from scratch and pre-trained on CheXpert. Also, when a frozen pre-trained module was used, the worst outcomes were obtained, proving that the knowledge learned on the source dataset was not helpful.

MURA			
Model	**Colorization Module**	**Transfer strategy**	**AUC**
ResNet18	DECONV (CheXpert)	Colorization Module	0.745 ± 0.013
ResNet18	PixelShuffle (CheXpert)	Colorization Module	0.752 ± 0.011
ResNet18	ColorU (CheXpert)	Colorization Module	0.762 ± 0.010
ResNet18	DECONV (MURA)	All	0.781 ± 0.005
ResNet18	PixelShuffle (MURA)	All	0.796 ± 0.016
ResNet18	ColorU (MURA)	All	0.802 ± 0.004

Table 5.4: Tests over MURA using ResNet18 model with colorization modules pre-trained on CheXpert (Colorization Module and All).

5.3 Testing over ChestX-ray14

The following experiments have been focused on a smaller dataset than CheXpert but with a common task. On the ChestX-ray14 was tested if modules and architectures trained on CheXpert are transferable. For these experiments, the model chosen was the ResNet18, and the colorization module was the PixelShuffle. The outcomes (Table 5.5) show that the knowledge between datasets of the same body region can be transferred. Indeed, differently than what see on the MURA dataset, also reusing the colorization module frozen helps to achieve better results. However, transfer only the module leads to slightly worst performance when all layers are finetuned.

5.4 Effect of training set size

These experiments were done to see how the performance change over different dataset sizes. The subsets size chosen are 10%, 5%, 1%, 0.5%, 0.2%, and 0.1% of the original size. For each dimension, three different subsets have been created randomly. All these experiments were done by using ResNet18 as the pre-trained

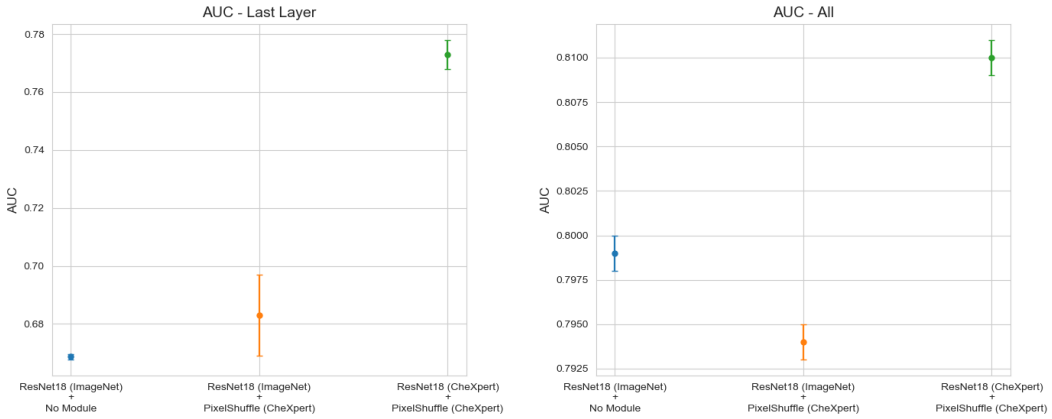


Figure 5.9: AUCs of networks over ChestX-ray14.

Table 5.5: Results of training different networks over the ChestX-ray14 dataset.

ChestX-ray14			
Model	**Colorization Module**	**Transfer strategy**	**AUC**
ResNet18 (ImageNet)	No	Last layer	0.669 ± 0.00
ResNet18 (ImageNet)	PixelShuffle (CheXpert)	Last layer	0.683 ± 0.01
ResNet18 (CheXpert)	PixelShuffle (CheXpert)	Last layer	0.773 ± 0.01
ResNet18 (ImageNet)	PixelShuffle (Scratch)	Colorization Module	0.729 ± 0.01
ResNet18 (ImageNet)	No	All	0.799 ± 0.00
ResNet18 (ImageNet)	PixelShuffle (CheXpert)	All	0.794 ± 0.00
ResNet18 (CheXpert)	PixelShuffle (CheXpert)	All	0.810 ± 0.00
ResNet18 (ImageNet)	PixelShuffle (ChestX-ray14)	All	0.792 ± 0.00

model and PixelShuffle as the colorization module. As for the original dataset, four different experiments were performed:

- The first baseline in which there is not colorization module and the pre-trained model is frozen except the last layer.
- The first test in which there is the colorization module and only the pre-trained model is frozen except for the last layer.
- The second baseline in which all the model is trained.
- The second test in which the architecture trained in the first test was finetuned in mode 2.
- In the last experiment, the colorization module was trained from scratch.

While results are summarized in Table 5.6, Figure 5.10 reports how the AUC

changes across different dataset sizes, and Figure 5.11 shows the difference between AUCs of different networks and modality took into consideration. The outcomes show that when the number of instances in the dataset decreases, training all the network or only the colorization module leads to better performance than training the architecture without the module.

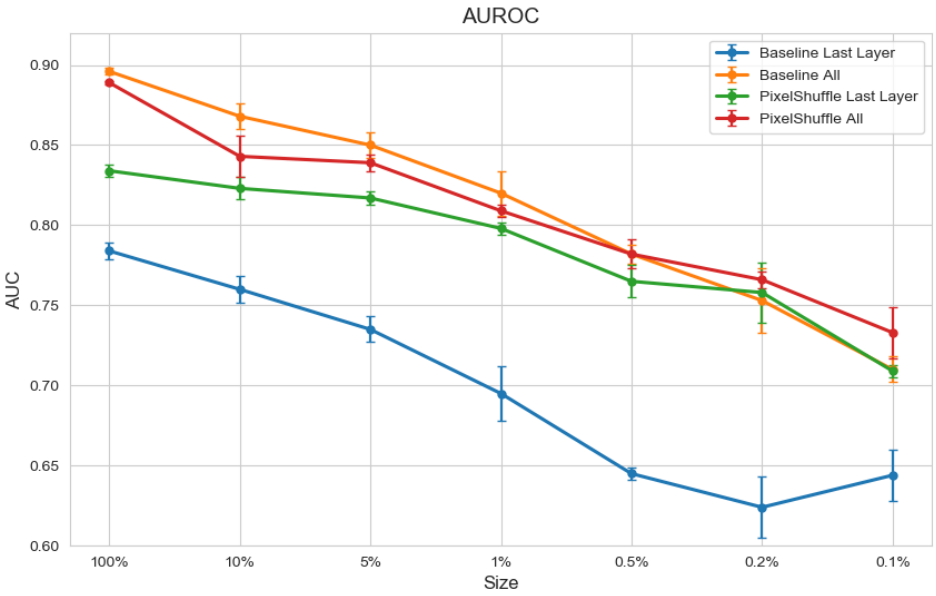


Figure 5.10: AUCs over different dataset size.

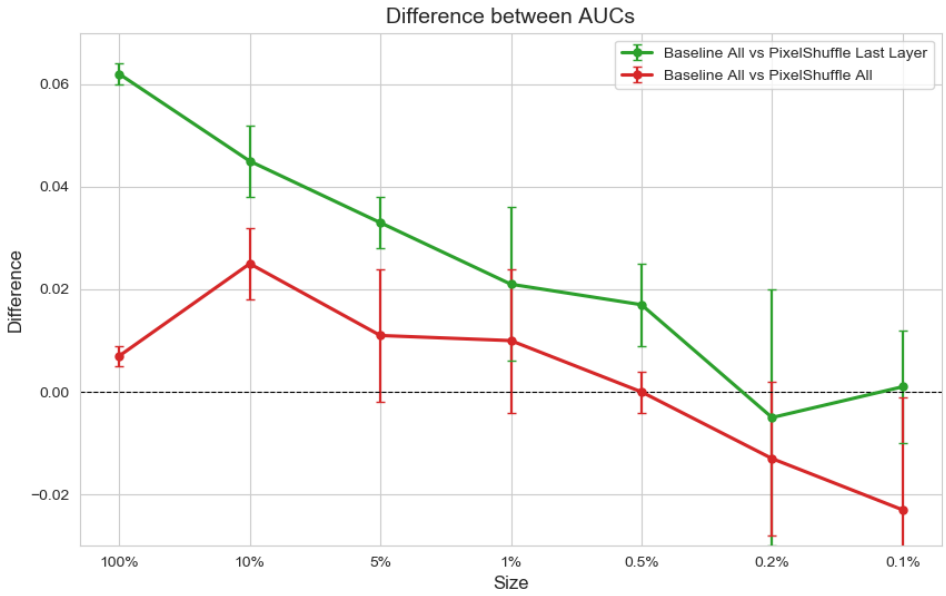


Figure 5.11: Difference between AUCs of different networks and training regime over subsets of CheXpert.

Table 5.6: Experiments over subsets of CheXpert. (ResNet18)

CheXpert				
Size	**Model**	**Colorization Module**	**Transfer strategy**	**AUC**
100%	ResNet18	No	Last layer	0.784 ± 0.005
100%	ResNet18	No	All	0.896 ± 0.002
100%	ResNet18	PixelShuffle	Colorization Module	0.834 ± 0.004
100%	ResNet18	PixelShuffle	All	0.889 ± 0.001
10%	ResNet18	No	Last layer	0.760 ± 0.008
10%	ResNet18	No	All	0.868 ± 0.008
10%	ResNet18	PixelShuffle	Colorization Module	0.823 ± 0.007
10%	ResNet18	PixelShuffle	All	0.843 ± 0.013
5%	ResNet18	No	Last layer	0.735 ± 0.008
5%	ResNet18	No	All	0.850 ± 0.008
5%	ResNet18	PixelShuffle	Colorization Module	0.817 ± 0.004
5%	ResNet18	PixelShuffle	All	0.839 ± 0.005
1%	ResNet18	No	Last layer	0.695 ± 0.017
1%	ResNet18	No	All	0.820 ± 0.014
1%	ResNet18	PixelShuffle	Colorization Module	0.798 ± 0.004
1%	ResNet18	PixelShuffle	All	0.809 ± 0.004
0.5%	ResNet18	No	Last layer	0.645 ± 0.004
0.5%	ResNet18	No	All	0.782 ± 0.006
0.5%	ResNet18	PixelShuffle	Colorization Module	0.765 ± 0.010
0.5%	ResNet18	PixelShuffle	All	0.782 ± 0.009
0.2%	ResNet18	No	Last layer	0.624 ± 0.019
0.2%	ResNet18	No	All	0.753 ± 0.020
0.2%	ResNet18	PixelShuffle	Colorization Module	0.758 ± 0.019
0.2%	ResNet18	PixelShuffle	All	0.766 ± 0.005
0.1%	ResNet18	No	Last layer	0.644 ± 0.016
0.1%	ResNet18	No	All	0.710 ± 0.008
0.1%	ResNet18	PixelShuffle	Colorization Module	0.709 ± 0.004
0.1%	ResNet18	PixelShuffle	All	0.733 ± 0.016

Chapter 6

Discussion

The experiments proposed aim to answer three main questions, and they were conducted over three different datasets. On the largest (CheXpert), it was tested what is the best strategy, and if it changes when the dataset sizes vary. Over the other datasets, it was tested if the modules and the architectures are transferable. From the experiments conducted on the CheXpert dataset is possible to assert that the colorization modules proposed can adapt the gray-scale inputs to exploit better the features learned on ImageNet. By looking at the architectures that use ResNet18 as the pre-trained model frozen, the module that reaches the best result is the DECONV one, while for the architectures that utilize DenseNet121, is the PixelShuffle module. However, the variation from the results obtained by the different modules is small. When the entire network is trained, the difference between using or not the modules decreases in a meaningful way. After this phase, the module that performs best is ColorU, and in combination with the DenseNet model, obtains the higher AUC over all experiments done. The outcomes of our baseline are slightly less to the results obtained by Irvin et al. [15], but they use an ensemble technique that could explain this difference.

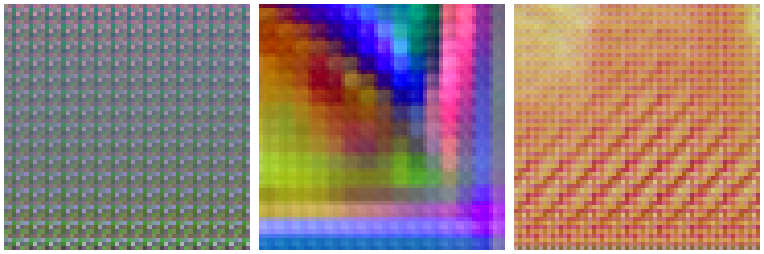
The experiments performed on the MURA dataset, in which the number of samples is about 8,000, confirm the behavior seen on the CheXpert dataset. However, the best approach to train the model on this dataset is to training only the pre-trained model without the colorization module. When the modules are pre-trained on CheXpert, and only the last layer was finetuned, the outcomes obtained are the worst. Furthermore, there are no differences between results obtained when modules from scratch and pre-trained are finetuned. This behavior confirms the work of Romero et al., which shows that using models pre-trained on different body regions not help to get better results. Reusing module or the whole architecture that was trained over the same anatomy region leads to better outcomes. The tests

over ChestX-ray-14 confirm these results; the improvements are of about 5% when only the module was transferred, and around to 16% when all the architecture was reused.

The series of tests conducted on subsets of CheXpert show that, when the size of the dataset decreases, the performance using the colorization modules reach higher results. If the number of samples is less than 512, training all the architecture with the colorization modules leads to achieving better results, with an improvement of about 3%. Moreover, training only the module allows reaching the same, or slightly performance better than finetuning all layers of the pre-trained model.

The images generated from the modules proposed are very different between them. The most colorful ones are the images that the PixelShuffle module produces. Each of these modules inserts some artifacts into the images as is possible to see in Figure 6.1. Since the purpose of these modules is to obtain a new representation of images that helps the pre-trained network, and not to generate realistic images, there may be different explanations for these artifacts:

- Some layers used can introduce this type of artifact, for instance, the Transpose Convolution Layers.
- It could indicate that some information as lost in the process.



(a) DECONV **(b)** PixelShuffle **(c)** ColorU

Figure 6.1: Artifacts generated by the modules.

6.1 Future works

At the end of this work, some future improvements are possible to perform. In this thesis, only two different pre-trained models were tested, but this work can be extended to other popular pre-trained networks, for instance, VGG. Moreover, due to the lack of time, only three different colorization modules have been chosen on which perform the experiments, but more than 20 modules were created. Also, different initialization for the modules could improve the results, for instance, by introducing a step in which the module is trained on ImageNet. An interesting

scenario could be to take advantage of the images created from the modules to help the radiologist interpretation. In Figure 6.2 some examples of the combination of output and input to achieve this purpose are reported.

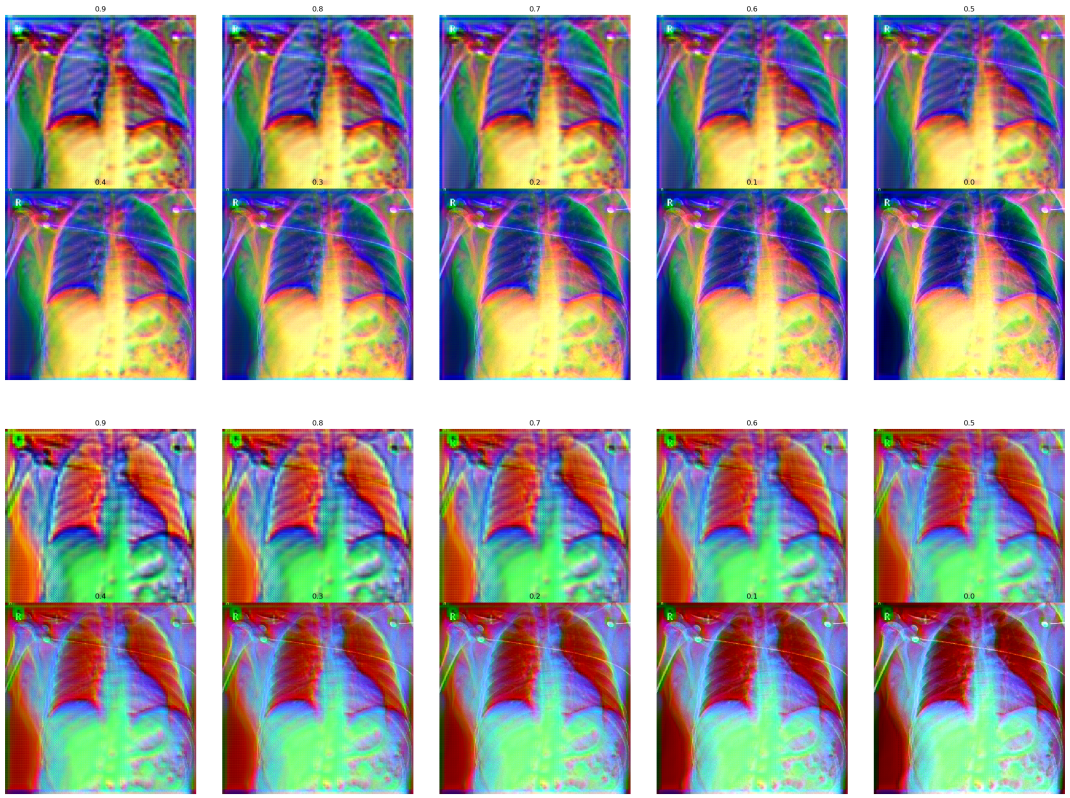


Figure 6.2: Examples of colorization for human work.

Bibliography

[1] Kunihiko Fukushima. «Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position». In: *Biological cybernetics* 36.4 (1980), pp. 193–202 (cit. on p. 1).

[2] S-CB Lo, S-LA Lou, Jyh-Shyan Lin, Matthew T Freedman, Minze V Chien, and Seong Ki Mun. «Artificial convolution neural network techniques and applications for lung nodule detection». In: *IEEE transactions on medical imaging* 14.4 (1995), pp. 711–718 (cit. on p. 1).

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 1).

[4] MIT Deep Learning. *MIT Deep Learning 6.S191*. URL: `http://introtodeeplearning.com/` (cit. on p. 5).

[5] Benny Prijono, Benny Prijono, Benny Prijono, Benny Prijono, Benny Prijono, Juni, Benny Prijono, Juni, and Benny Prijono. *Student Notes: Convolutional Neural Networks (CNN) Introduction*. Apr. 2018. URL: `https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/` (cit. on p. 7).

[6] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *CoRR* abs/1409.0575 (2014). arXiv: `1409.0575`. URL: `http://arxiv.org/abs/1409.0575` (cit. on p. 9).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385` (cit. on pp. 9–11).

[8] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. «Densely Connected Convolutional Networks». In: *CoRR* abs/1608.06993 (2016). arXiv: `1608.06993`. URL: `http://arxiv.org/abs/1608.06993` (cit. on pp. 10, 17).

[9] Sinno Jialin Pan and Qiang Yang. «A survey on transfer learning». In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359 (cit. on pp. 12, 14, 15).

[10] Veronika Cheplygina, Marleen de Bruijne, and Josien P. W. Pluim. «Not-so-supervised: a survey of semi-supervised, multi-instance, and transfer learning in medical image analysis». In: *CoRR* abs/1804.06353 (2018). arXiv: 1804.06353. URL: http://arxiv.org/abs/1804.06353 (cit. on p. 18).

[11] Afonso Menegola, Michel Fornaciali, Ramon Pires, Sandra Eliza Fontes de Avila, and Eduardo Valle. «Towards Automated Melanoma Screening: Exploring Transfer Learning Schemes». In: *CoRR* abs/1609.01228 (2016). arXiv: 1609.01228. URL: http://arxiv.org/abs/1609.01228 (cit. on p. 18).

[12] *Diabetic Retinopathy Detection*. URL: https://www.kaggle.com/c/diabetic-retinopathy-detection/data (cit. on p. 18).

[13] *7-POINT CRITERIA EVALUATION DATABASE*. URL: http://derm.cs.sfu.ca/Download.html (cit. on p. 18).

[14] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. «Transfusion: Understanding Transfer Learning with Applications to Medical Imaging». In: *CoRR* abs/1902.07208 (2019). arXiv: 1902.07208. URL: http://arxiv.org/abs/1902.07208 (cit. on p. 19).

[15] Jeremy Irvin et al. «CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison». In: *CoRR* abs/1901.07031 (2019). arXiv: 1901.07031. URL: http://arxiv.org/abs/1901.07031 (cit. on pp. 19, 25, 26, 52).

[16] Varun Gulshan et al. «Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs». In: *Jama* 316.22 (2016), pp. 2402–2410 (cit. on p. 19).

[17] Richard Zhang, Phillip Isola, and Alexei A. Efros. «Colorful Image Colorization». In: *CoRR* abs/1603.08511 (2016). arXiv: 1603.08511. URL: http://arxiv.org/abs/1603.08511 (cit. on pp. 20, 22).

[18] Anat Levin, Dani Lischinski, and Yair Weiss. «Colorization using optimization». In: *ACM SIGGRAPH 2004 Papers*. 2004, pp. 689–694 (cit. on pp. 20, 21).

[19] Raj Kumar Gupta, Alex Yong-Sang Chia, Deepu Rajan, Ee Sin Ng, and Huang Zhiyong. «Image colorization using similar images». In: *Proceedings of the 20th ACM international conference on Multimedia*. 2012, pp. 369–378 (cit. on pp. 20, 21).

[20] Billal BEGUERADJ et al. *Black and white image colorization with OpenCV and Deep Learning*. Feb. 2020. URL: https://www.pyimagesearch.com/2019/02/25/black-and-white-image-colorization-with-opencv-and-deep-learning/ (cit. on p. 22).

[21] Vincent Billaut, Matthieu de Rochemonteix, and Marc Thibault. «ColorUNet: A convolutional classification approach to colorization». In: *CoRR* abs/1811.03120 (2018). arXiv: 1811.03120. URL: http://arxiv.org/abs/1811.03120 (cit. on pp. 21, 23, 32).

[22] Fabio Maria Carlucci, Paolo Russo, S. M. Baharlou, and Barbara Caputo. «(DE)^2 CO: Deep Depth Colorization». In: *CoRR* abs/1703.10881 (2017). arXiv: 1703.10881. URL: http://arxiv.org/abs/1703.10881 (cit. on pp. 22, 23, 29, 30).

[23] Philip Teare, Michael Fishman, Oshra Benzaquen, Eyal Toledano, and Eldad Elnekave. «Malignancy detection on mammography using dual deep convolutional neural networks and genetically discovered false color input enhancement». In: *Journal of digital imaging* 30.4 (2017), pp. 499–505 (cit. on p. 23).

[24] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. «Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning». In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1285–1298 (cit. on p. 23).

[25] Chelsea Finn, Pieter Abbeel, and Sergey Levine. «Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks». In: *CoRR* abs/1703.03400 (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400 (cit. on p. 24).

[26] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. «Meta-Learning with Differentiable Convex Optimization». In: *CoRR* abs/1904.03758 (2019). arXiv: 1904.03758. URL: http://arxiv.org/abs/1904.03758 (cit. on p. 24).

[27] Muhammad Abdullah Jamal, Guo-Jun Qi, and Mubarak Shah. «Task-Agnostic Meta-Learning for Few-shot Learning». In: *CoRR* abs/1805.07722 (2018). arXiv: 1805.07722. URL: http://arxiv.org/abs/1805.07722 (cit. on p. 24).

[28] Davis Wertheimer and Bharath Hariharan. «Few-Shot Learning with Localization in Realistic Settings». In: *CoRR* abs/1904.08502 (2019). arXiv: 1904.08502. URL: http://arxiv.org/abs/1904.08502 (cit. on p. 24).

[29] Yann Lifchitz, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. «Dense Classification and Implanting for Few-Shot Learning». In: *CoRR* abs/1903.05050 (2019). arXiv: 1903.05050. URL: http://arxiv.org/abs/1903.05050 (cit. on p. 24).

[30] Zitian Chen, Yanwei Fu, Yu-Xiong Wang, Lin Ma, Wei Liu, and Martial Hebert. «Image Deformation Meta-Networks for One-Shot Learning». In: *CoRR* abs/1905.11641 (2019). arXiv: 1905.11641. URL: http://arxiv.org/abs/1905.11641 (cit. on p. 24).

[31] Hongguang Zhang, Jing Zhang, and Piotr Koniusz. «Few-Shot Learning via Saliency-guided Hallucination of Samples». In: *CoRR* abs/1904.03472 (2019). arXiv: 1904.03472. URL: http://arxiv.org/abs/1904.03472 (cit. on p. 24).

[32] Eli Schwartz, Leonid Karlinsky, Rogério Schmidt Feris, Raja Giryes, and Alexander M. Bronstein. «Baby steps towards few-shot learning with multiple semantics». In: *CoRR* abs/1906.01905 (2019). arXiv: 1906.01905. URL: http://arxiv.org/abs/1906.01905 (cit. on p. 24).

[33] Pranav Rajpurkar et al. «Mura: Large dataset for abnormality detection in musculoskeletal radiographs». In: *arXiv preprint arXiv:1712.06957* (2017). URL: https://arxiv.org/abs/1712.06957 (cit. on p. 26).

[34] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald Summers. «ChestX-ray14: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases». In: (Sept. 2017) (cit. on p. 27).

[35] Vdumoulin. *convarithmetic*. Apr. 2019. URL: https://github.com/vdumoulin/conv_arithmetic (cit. on p. 30).

[36] Augustus Odena, Vincent Dumoulin, and Chris Olah. «Deconvolution and Checkerboard Artifacts». In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard (cit. on p. 31).

[37] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. «Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network». In: *CoRR* abs/1609.05158 (2016). arXiv: 1609.05158. URL: http://arxiv.org/abs/1609.05158 (cit. on pp. 31, 32).

[38] *PyTorch*. URL: https://pytorch.org/ (cit. on pp. 34, 36).

[39] Leslie N. Smith and Nicholay Topin. «Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates». In: *CoRR* abs/1708.07120 (2017). arXiv: 1708.07120. URL: http://arxiv.org/abs/1708.07120 (cit. on p. 37).

[40] Andrew P. Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, and Wenzhe Shi. «Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize». In: *CoRR* abs/1707.02937 (2017). arXiv: 1707.02937. URL: http://arxiv.org/abs/1707.02937 (cit. on p. 65).

Appendix A

Chexpert Results

In this chapter are reported all the result obtained by training over the Chexpert dataset.

The modes are:

- **Mode 1**: The module, if present, and the last layer are trained.
- **Mode 2**: All layers are trained.
- **Mode 3**: only the last layer is finetuned.

A.1 ResNet18

Below are reported the result, in detail, of all experiments on the CheXpert dataset using the ResNet18 model:

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
No	1	0.744	0.714	0.823	0.870	0.798	0.790
No	1	0.750	0.700	0.800	0.863	0.790	0.781
No	1	0.754	0.700	0.793	0.870	0.794	0.782
Mean		0.749	0.705	0.805	0.868	0.794	0.784
Std		0.005	0.008	0.016	0.004	0.004	0.005

Table A.1: Results of training the ResNet18 model without the colorization module by freezing all the architecture except the last layer over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
No	2	0.857	0.928	0.912	0.836	0.940	0.895
No	2	0.824	0.859	0.942	0.920	0.930	0.895
No	2	0.854	0.844	0.937	0.926	0.931	0.898
Mean		0.845	0.877	0.930	0.894	0.934	0.896
Std		0.018	0.045	0.016	0.050	0.006	0.002

Table A.2: Results of training the ResNet18 model without the colorization module over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
DECONV	1	0.802	0.783	0.901	0.852	0.860	0.840
DECONV	1	0.834	0.767	0.864	0.863	0.860	0.838
DECONV	1	0.807	0.790	0.886	0.862	0.871	0.843
Mean		0.814	0.780	0.884	0.859	0.864	0.840
Std		0.017	0.012	0.019	0.006	0.006	0.003

Table A.3: Results of training the ResNet18 model with the DECONV module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
PixelShuffle	1	0.829	0.746	0.872	0.843	0.868	0.832
PixelShuffle	1	0.802	0.758	0.885	0.856	0.859	0.832
PixelShuffle	1	0.816	0.766	0.893	0.881	0.840	0.839
Mean		0.816	0.757	0.883	0.860	0.856	0.834
Std		0.014	0.010	0.011	0.019	0.014	0.004

Table A.4: Results of training the ResNet18 model with the PixelSuffle module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
ColorU	1	0.800	0.791	0.902	0.875	0.860	0.846
ColorU	1	0.819	0.770	0.851	0.866	0.847	0.831
ColorU	1	0.839	0.772	0.888	0.848	0.850	0.839
Mean		0.819	0.778	0.880	0.863	0.852	0.839
Std		0.020	0.012	0.026	0.014	0.007	0.008

Table A.5: Results of training the ResNet18 model with the ColorU module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
DECONV	2	0.849	0.847	0.938	0.890	0.928	0.890
DECONV	2	0.840	0.859	0.927	0.907	0.925	0.892
DECONV	2	0.834	0.854	0.920	0.904	0.918	0.886
Mean		0.841	0.853	0.928	0.900	0.924	0.889
Std		0.008	0.006	0.009	0.009	0.005	0.003

Table A.6: Results of training the ResNet18 model with the DECONV module after cleaning the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
PixelShuffle	2	0.830	0.845	0.922	0.905	0.938	0.888
PixelShuffle	2	0.822	0.839	0.947	0.914	0.927	0.890
PixelShuffle	2	0.834	0.858	0.930	0.906	0.923	0.890
Mean		0.829	0.847	0.933	0.908	0.929	0.889
Std		0.006	0.010	0.013	0.005	0.008	0.001

Table A.7: Results of training the ResNet18 model with the PixelShuffle module after cleaning the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
ColorU	2	0.832	0.861	0.912	0.915	0.932	0.890
ColorU	2	0.820	0.842	0.941	0.933	0.938	0.895
ColorU	2	0.830	0.844	0.930	0.915	0.937	0.891
Mean		0.827	0.849	0.928	0.921	0.936	0.892
Std		0.006	0.010	0.015	0.010	0.003	0.003

Table A.8: Results of training the ResNet18 model with the ColorU module after cleaning the last layer, over the CheXpert dataset.

A.2 DenseNet121

Below are reported the result, in detail, of all experiments on the CheXpert dataset using the DenseNet121 model:

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
No	1	0.776	0.721	0.780	0.832	0.822	0.786
No	1	0.781	0.727	0.769	0.830	0.819	0.785
No	1	0.782	0.720	0.782	0.836	0.819	0.788
Mean		0.780	0.723	0.777	0.833	0.820	0.786
Std		0.003	0.004	0.007	0.003	0.002	0.002

Table A.9: Results of training the DenseNet121 model without the colorization module by freezing all the architecture except the last layer over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
No	2	0.850	0.825	0.952	0.937	0.927	0.898
No	2	0.859	0.822	0.952	0.938	0.927	0.900
No	2	0.854	0.833	0.942	0.921	0.933	0.897
Mean		0.854	0.827	0.949	0.932	0.929	0.898
Std		0.005	0.006	0.006	0.010	0.003	0.002

Table A.10: Results of training the DenseNet121 model without the colorization module over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
DECONV	1	0.781	0.798	0.883	0.861	0.841	0.833
DECONV	1	0.788	0.799	0.886	0.868	0.853	0.839
DECONV	1	0.824	0.762	0.884	0.847	0.850	0.833
Mean		0.798	0.786	0.884	0.859	0.848	0.835
Std		0.023	0.021	0.002	0.011	0.006	0.003

Table A.11: Results of training the DenseNet121 model with the DECONV module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
PixelSuffle	1	0.828	0.758	0.885	0.858	0.869	0.840
PixelSuffle	1	0.816	0.751	0.874	0.882	0.866	0.838
PixelSuffle	1	0.824	0.800	0.893	0.882	0.864	0.853
Mean		0.823	0.770	0.884	0.874	0.866	0.844
Std		0.006	0.027	0.010	0.014	0.003	0.008

Table A.12: Results of training the DenseNet121 model with the PixelSuffle module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
ColorU	1	0.802	0.782	0.882	0.852	0.864	0.836
ColorU	1	0.809	0.759	0.885	0.877	0.851	0.836
ColorU	1	0.800	0.765	0.921	0.870	0.864	0.844
Mean		0.804	0.769	0.896	0.866	0.860	0.839
Std		0.005	0.012	0.022	0.013	0.008	0.005

Table A.13: Results of training the DenseNet121 model with the ColorU module by freezing the pre-trained model except for the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
DECONV	2	0.856	0.872	0.911	0.890	0.924	0.891
DECONV	2	0.835	0.881	0.927	0.905	0.924	0.894
DECONV	2	0.840	0.835	0.947	0.922	0.917	0.892
Mean		0.844	0.863	0.928	0.906	0.922	0.892
Std		0.011	0.024	0.018	0.016	0.004	0.002

Table A.14: Results of training the DenseNet121 model with the DECONV module after cleaning the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
PixelShuffle	2	0.830	0.850	0.945	0.917	0.942	0.897
PixelShuffle	2	0.863	0.833	0.940	0.913	0.938	0.897
PixelShuffle	2	0.848	0.868	0.929	0.900	0.930	0.895
Mean		0.847	0.850	0.938	0.910	0.937	0.896
Std		0.017	0.018	0.008	0.009	0.006	0.001

Table A.15: Results of training the DenseNet121 model with the PixelShuffle module after cleaning the last layer, over the CheXpert dataset.

Module	Mode	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Mean
ColorU	2	0.856	0.860	0.940	0.913	0.935	0.901
ColorU	2	0.829	0.826	0.935	0.946	0.927	0.893
ColorU	2	0.851	0.839	0.936	0.920	0.931	0.895
Mean		0.845	0.842	0.937	0.926	0.931	0.896
Std		0.014	0.017	0.003	0.017	0.004	0.004

Table A.16: Results of training the DenseNet121 model with the ColorU module after cleaning the last layer, over the CheXpert dataset.

Appendix B

Methodology

In this chapter, all parameters needed to replicate the experiments are described in detail.

B.1 Modules structure

B.1.1 DECO

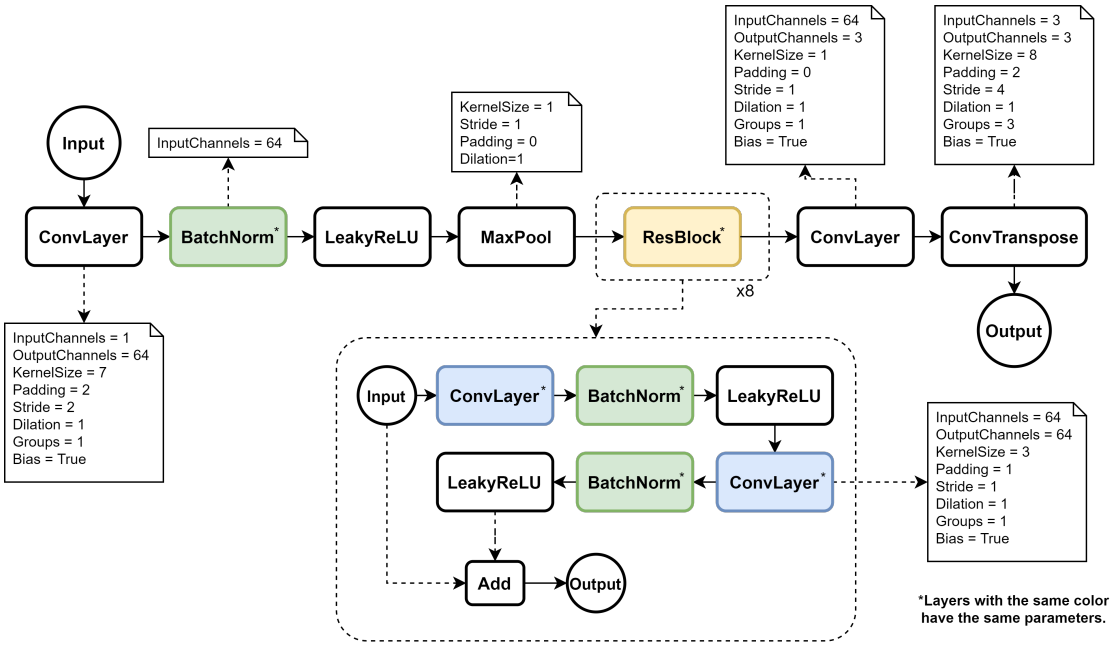


Figure B.1: DECO structure with parameters.

The initialization of the layers is:

- **ConvLayer**: Initialized by using the Xavier Uniform Initialization.
- **BatchNorm**: Weight = 1 and Bias = 0
- **Other**: Other layers are initialized with default methods of Pytorch.

B.1.2 PixelShuffle

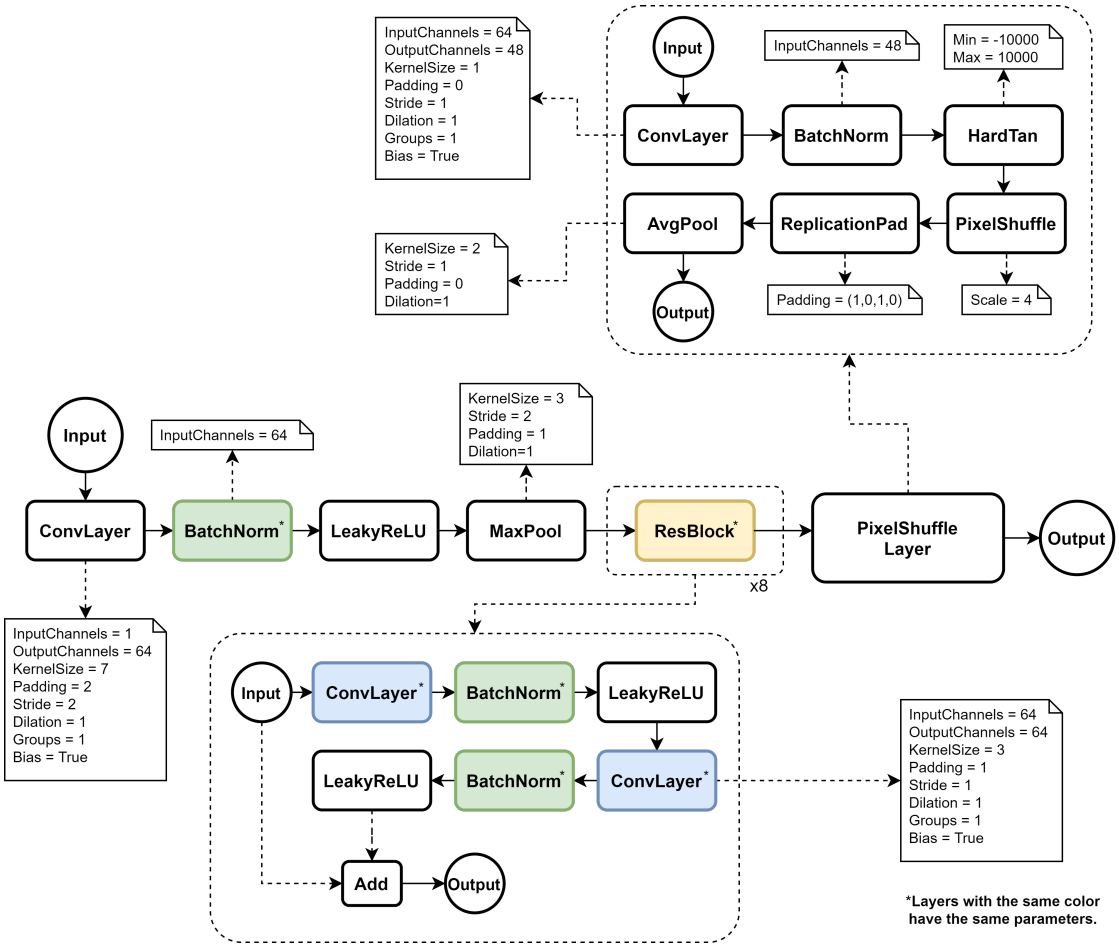


Figure B.2: PixelShuffle structure with parameters.

The initialization of the layers is:

- **ConvLayer of PixelShuffle Layer**: Initialized by using the ICNR[40].
- **BatchNorm**: Weight = 1 and Bias = 0

- **Other**: Other layers are initialized with default methods of Pytorch.

B.1.3 ColorU

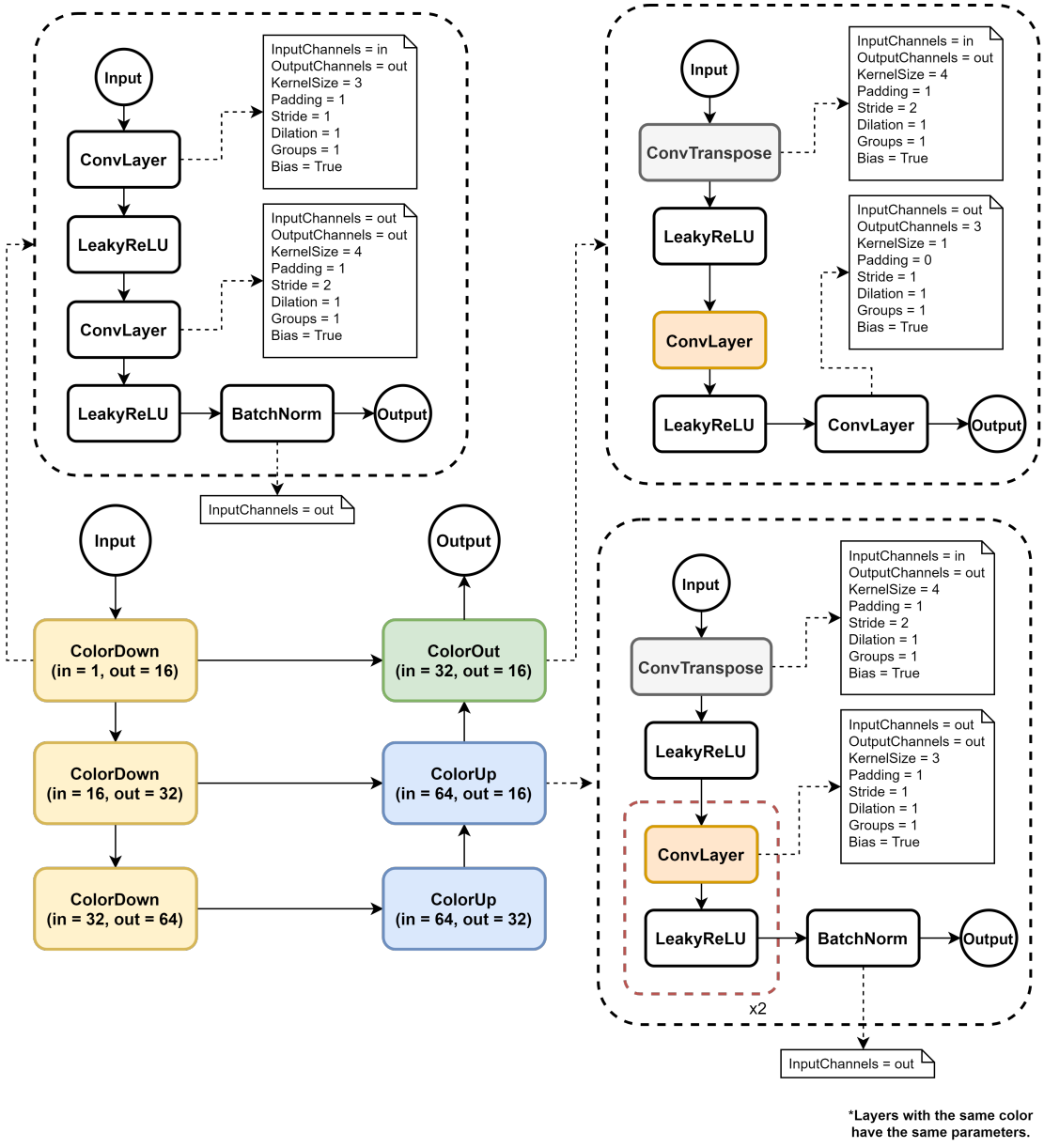


Figure B.3: ColorU structure with parameters.

The layers are initialized with default methods of Pytorch.

B.2 Experiments parameters

B.2.1 CheXpert

Baseline - Mode1:

- Model pre-trained on ImageNet (ResNet18 and DenseNet121).
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-3.
- Epochs: 12 or until convergence.
- Dataset shuffle.
- Best model saved every 4800 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

Baseline - Mode2:

- Model pre-trained on ImageNet (ResNet18 and DenseNet121).
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-2.
- Epochs: 6 or until convergence.
- Dataset shuffle.
- Best model saved every 4800 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

Tests with modules - Mode1:

- Model pre-trained on ImageNet (ResNet18 and DenseNet121).
- Models used: DECONV, PixelShuffle and ColorU.
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-3.

- Epochs: 12 or until convergence.
- Dataset shuffle.
- Best model saved every 4800 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

Tests with modules - Mode2:

- Model pre-trained on ImageNet (ResNet18 and DenseNet121).
- Models used: DECONV, PixelShuffle and ColorU.
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-2.
- Epochs: 6 or until convergence.
- Dataset shuffle.
- Best model saved every 4800 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

B.2.2 MURA

Baseline - Mode1 and Mode2:

- Model pre-trained on ImageNet (ResNet18).
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-3.
- Epochs: 48 or until convergence.
- Dataset shuffle.
- Best model saved every 2048 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.

- Pytorch 1.4.0

Tests with modules - Mode1 - Mode2 - Mode3:

- Model pre-trained on ImageNet (ResNet18).
- Models used: DECONV, PixelShuffle and ColorU (Scratch/Pre-trained on CheXpert).
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 2e-3.
- Epochs: 48 or until convergence.
- Dataset shuffle.
- Best model saved every 2048 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

B.2.3 ChestX-ray14

Baseline - Mode3:

- Model pre-trained on ImageNet (ResNet18).
- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 1e-3.
- Epochs: 48 or until convergence.
- Dataset shuffle.
- Best model saved every 43520 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0

Tests with modules - Mode3:

- Model pre-trained on ImageNet/CheXpert (ResNet18).
- Models used: PixelShuffle (Pre-trained on CheXpert).

- Learning alghoritm: SGD with OneCycle policy.
- MaxLr: 1e-3.
- Epochs: 48 or until convergence.
- Dataset shuffle.
- Best model saved every 43520 iteration.
- Data Augmentation: see Chapter 5.
- GPU: NVIDIA 1080Ti and NVIDIA K80.
- Pytorch 1.4.0