II

Experimental Part

Chapter 3°

Conducting experiments

3.1 Experiment methodology

This section will analyse the methodology followed during the experiments and present the rationale behind the comparisons made. The experiments were conducted to evaluate the performance of the three different proxy tasks (Image Rotation Prediction, Image Colorization, Image Inpainting) by transferring learning to the downstream task of image classification. Furthermore, we will explain how the testing and fine-tuning process was organized on two different datasets (CIFAR-10 and CIFAR-100), and discuss the rationale behind the comparisons made between the different training methods and parameters to ensure a fair and accurate evaluation of the results

3.1.1 Methodology and Logic of Comparisons

To ensure a fair comparison between the three proxy tasks, we chose *Image Rotation Prediction*, *Image Colorization* and *Image Inpainting*. These three tasks are different in nature: the first one is about the orientation of objects within images (rotation prediction), the second one is about the colors of images (colorization), while the third one focuses on the content, context and coherence of images (inpainting). We used ResNet-50 as the basis for all tasks. The same architecture was applied to each task, adapting it appropriately according to the requirements of each proxy task. In the next section we will discuss exactly how the adaptation of ResNet-50 was implemented for each of these tasks. This uniformity in the architecture allows for an accurate comparison of performance, without the results being affected by variations in the network architecture.

The logic of the comparisons focuses on two main fine-tuning strategies:

1. <u>Fine-tuning only of the latest level:</u> In this case, we retrained only the last fully connected layer of ResNet-50 in the downstream task of image classification. The goal was to see how the weights acquired by the proxy tasks compare to *random weights*, allowing us to evaluate what information the proxy tasks have learned and whether this information is useful in the downstream task. The comparison with random weights allows us to evaluate whether the weights learned by the network through self-supervised learning are compatible and useful for image classification.
2. <u>Fine-tuning of the entire model:</u> In this case, we retrained all layers of the model in the downstream task. The goal here is to see if training the proxy tasks is really worth it, helping the model improve its performance over initialization with random weights. We note that the same hyperparameters were used in all cases to ensure that the only difference between the experiments was the initial state of the weights.

We did not use any *data augmentation* in our experiments, as this would compromise fair comparison. Each proxy task has different requirements regarding *transforms*, which could cause problems during training. For example, in the *image rotation prediction task*, if a *random rotation* of 180° is applied to an image during preprocessed data and assigned class 2 (180° rotation), the same *random rotation* from the transformations can return it to its original position. This would make it belong to class 0 again, but we would have incorrectly given it class 2. Even if the model makes the correct prediction for class 0, the error in the labels will record it as an incorrect prediction, which will destroy the training. Similar problems arise in tasks such as *colorization*, where transform *color jitter* could corrupt the colors, making accurate training difficult. In normal circumstances, we could normally use the transformations (as long as they do not destroy the training of the indirect process) to achieve higher accuracy however as mentioned the aim was to achieve a fair comparison and not the highest possible performance. For these reasons, we limited ourselves to basic transforms such as *resize*, *normalize* and *totensor*. It is important to stress that because our goal was not to achieve the highest possible performance but to ensure a fair comparison of methods, the performance recorded will probably be only slightly below the expected standards.

The proxy tasks were trained on CIFAR-100 without using the labels of the dataset, as we used it as an *unlabeled* dataset. Then, the stored weights from the proxy tasks were transferred and tested in both CIFAR-10 and CIFAR-100 for the downstream task of *image classification*. The use of both datasets was appropriate, as we wanted to evaluate performance on both the same dataset used for training the proxy tasks (CIFAR-100) and a different but similar dataset (CIFAR-10). The two datasets were chosen due to their similarity, as they are of the same size and contain images of similar nature, which allows for a smoother comparison of results.

In addition to comparing the final performance of the proxy tasks, we also examined how quickly each proxy task achieves high performance. That is, we are interested in seeing not only which task performs best, but also which one can reach a satisfactory level of accuracy in fewer training epochs. For example, if a model trained with *SSL* can reach a satisfactory level of accuracy with fewer epochs than random weights, then this would be more cost and time efficient. The graphs illustrating the difference in performance between proxy task weights and random weights help us evaluate this dimension.

Finally, it should be noted that no extensive *hyperparameter tuning* was performed on the downstream task depending on the use of different weights by the different tasks. Our rationale was to keep the hyperparameters constant in all experiments to ensure a fair comparison. This means that we did not aim to catch the best possible performance, but to create an equivalent experimental environment for comparing methods. Therefore, the performance of the models do not compete with the best performance standards, but aim to demonstrate the efficiency of the proxy tasks relative to random weights and to highlight the most efficient process among the proxy tasks as a pre-training method of weights for the downstream task of image classification.

3.1.2 Questions and Expectations

Based on the methodology discussed in the previous section, the main research questions and expectations of our experiments are summarized as follows:

Performance using Self-Supervised Learning in relation to random weights

Our initial and key question is whether the use of proxy tasks through *self-supervised* learning can provide better results than random weights. We expect that in almost all metrics, performance will be better with the use of pre-trained weights, as they will have learned some key features from the proxy tasks, as opposed to random weights that carry no information.

Performance on different datasets (CIFAR-10 vs CIFAR-100)

We expect performance on CIFAR-100 to be lower compared to CIFAR-10, as CIFAR-100 has 100 classes, while CIFAR-10 has only 10. This means that the classification task is more complex in CIFAR-100. However, we expect that the pretrained weights trained on CIFAR- 100 will work equally well on CIFAR-10, as the two datasets contain similar images and features. Therefore, we expect that the proxy tasks will be able to transfer their knowledge to the new dataset and maintain good performance.

Fine-tuning only on the last level

In this case, the question is whether fine-tuning only the last layer of the network will yield better results with proxy tasks than with random weights. We expect that the performance of the proxy tasks will be significantly better, as the network weights are already trained on a related task. Here, proxy tasks provide pre-trained weights that contain useful information, while random weights require learning everything from scratch. Therefore, since we only train the last level, and since the random weights have not been trained except for the last level, while the pre-trained weights already contain some relevant information in most of the model, we expect significantly better performance than the proxy tasks.

Fine-tuning on the entire model

The question here is whether fine-tuning the whole model with the pre-trained weights will be better than fine-tuning the whole model with random weights. Obviously, we expect to achieve much better performance compared to fine-tuning only the last layer, since in this case all layers of the network are retrained, not just the last one. Regarding fine-tuning the whole model, we expect that the final performance with the pre-trained weights will be only slightly better than with the random weights, as the weights are only the initial
- either pre-trained or random - and then all modified during training. Therefore, we expect a slightly better convergence at the end with the pre-trained weights, but the difference will not be dramatic.

Speed of achieving satisfactory performance by fine-tuning all levels of the model

In addition to the final performance, we are also interested in the speed with which the proxy tasks will help the model to reach a satisfactory level of performance. We expect that the proxy tasks will allow the model to quickly achieve high levels of accuracy. We believe that in far fewer epochs than with random weights, the model will reach high accuracy.

Best performing indirect process

A key question we want to answer is which of the three proxy tasks (Image Rotation Prediction, Image Colorization, Image Inpainting) will prove to be more effective in enhancing the performance of the downstream task. We expect that *colorization* or *inpainting* will prove to be the most efficient, as they are more complex processes than rotation prediction, and therefore provide more information to the model.

3.2 Implementation of Indirect Processes

In this section, we will present the implementation of three indirect processes used to train the model via self-supervised learning: image rotation prediction, image colorization, and image inpainting. We will discuss how these proxy/pretext tasks were implemented, the tools used to construct them, and the hyperparameters chosen for model training in these tasks. In addition, images showing the processing of the original images and the predictions of each model will be presented.
The photographs used to illustrate model performance are not from the dataset used in the experiments, which will be described in a later section. This is because the images in this dataset are very small in size, which makes it difficult for the human eye to visually perceive the differences. I n s t e a d , images from the Caltech-256 dataset [52] were used, which includes images with sufficient dimensions, allowing for better observation and understanding of the differences between the images.

3.2.1 Implementation of Image Rotation Prediction Process

Data Preprocessing: the data preprocessing process in the image rotation prediction process involves first resizing the images to a fixed size of 224x224 pixels (the "*Resize*()" transformation) to ensure uniformity of the inputs to the model. Also the images undergo transformations, such as converting them to tensors (transformation "*ToTensor*" - the pixel values are converted from the range [0, 255] to [0,1], we are returned the tensor with dimensions [Channels, Height, Width]), as well as normalization (transformation "*Normalize(mean, std)*" - normalizes the input data so that the image has a specific mean (mean) and standard deviation (std)) which allows efficient processing by the neural network layers.

This process prepares the data to be ready for training, while the random rotation at each epoch ensures that the model is exposed to different angles of the same image, thus enhancing its ability to learn the spatial structures of the images.In the images retrieved from the envelope, random rotation is applied at angles of 0°, 90°, 180° or 270°.

The main objective is to classify the rotations, where each image belongs to one of the four categories, depending on its rotation angle. The label value is determined by the formula: *label = rotation angle / 90*, giving label values of 0, 1, 2 or 3 for the respective rotations of 0°, 90°, 180° and 270°. Thus, the 4 categories are derived from the different angles, and the model must learn to correctly identify the rotation of each image. We could say that we reduce this process to an image classification process, but with labels indicating the rotation angle instead of the image content as in the classical image classification process.

Architecture: The architecture used is the *ResNet50* model consisting of 50 layers, which include convolutional layers, pooling layers, and fully connected layers. The last layers are fully connected, with the final layer having 1000 outputs for *ImageNet* classification problems.

In the specific task of image rotation prediction, this final layer was modified to have 4 outputs, one for each possible rotation angle (0°, 90°, 180°, 270°). The rest of the network remains the same and is adjusted to extract features that help predict the correct angle.

Education: Model training for the image rotation prediction task was implemented using *CrossEntropyLoss*, which is suitable for classification problems such as this one. *CrossEntropyLoss* measures the difference between the predicted probability distribution of the model and the actual labels. Specifically for this task, the four rotation angles (0°, 90°, 180°, 270°) correspond to four categories, and *CrossEntropyLoss* is ideal for multi-category classifications.

The Adam optimizer was chosen because it provides fast convergence and dynamic adjustment of the learning rate by optimizing the model using the derivatives of the loss function to update the weights. The learning rate is set to 0.001 to maintain the stability of the model, while the weight decay at 0.0001 helps prevent overfitting by reducing the complexity of the model.

The model is trained for 50 seasons with 64 batches, taking data rotated at random angles and trying to correctly predict the angle each time.

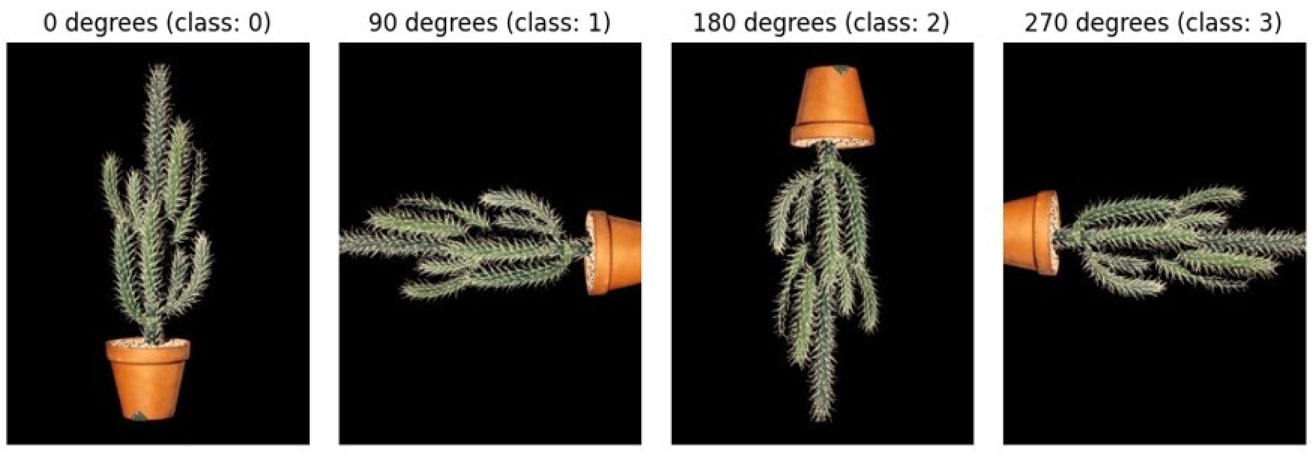


Figure 3.2.1 - 1: Image rotation examples for the Image Rotation Prediction task (Caltech-256 Dataset): the original image is rotated by 0°, 90°, 180°, and 270°, with each angle corresponding to a class (0, 1, 2, 3).

3.2.2 Implementation of Image Colorization Process

Data preprocessing: initially the same transforms are used as in the other tasks. These include resizing the images to fixed dimensions with *Resize*, normalizing the pixel values with *Normalize*, and converting the image to tensor format via *ToTensor* to make it suitable for input to the neural network.

Then, to create the appropriate input data for the model, the images are converted from colour (RGB) to black and white (*grayscale*). This conversion is necessary as the task of *image colorization* aims to train the model to "colorize" a black and white image. However, in order for this black and white image to be imported into the network, it needs to match the format of the colour images, i.e. it needs to have three channels. Therefore we repeat the *grayscale* image three times, creating a *fake RGB* format. Each of the three channels of the new image has the same information as the *grayscale* image, so that the input has the correct dimensioning for the neural network, which is designed to work with images three channels.

Finally, each returned data sample contains two elements: the black and white image used as input to the model, and the target color image, which the model is asked to reconstruct with the appropriate color information.

Architecture: The network architecture used for the image colorization task is a ResNet50 based autoencoder format. Initially, the final fully connected layers of ResNet are removed, as they are designed for classification and are not suitable for the task of image colorization. They are replaced by successive decoder layers based on transposed convolutions.

The network works in the following way: There are 5 levels of encoder and 5 levels of decoder. During encoding (encoder), the features of the image are passed through the convolutional layers of the ResNet, which extract high-level information from the image. Then, during decoding (decoder), the model uses transposed convolutions to resize the image to a normal level. More specifically, transposed convolutions, also known as deconvolutions or upsampling convolutions, is a process that reverses the operation of the normal convolutional levels. Instead of reducing the size of the image features, as normal convolutions do, transposed convolutions increase the size, restoring the dimensionality of the image to its original levels. This makes them particularly useful in decoders, such as autoencoders, where the model needs to reconstruct images from reduced representations. At each level of the decoder, BatchNorm2D and ReLU functions are applied for normalization and nonlinearity introduction, which help to stabilize training and improve predictions.

The skip connections between the encoder and decoder layers play an important role. These connections ensure that information from the initial stages of image processing is not lost during decoding. Thus, the model is able to reproduce details and textures more accurately.

The final layer of the model is responsible for generating the reconstructed image, ensuring that the result has the same size and characteristics as the original image.

Training: for the *image colorization* task, the training process focuses on teaching the model to reconstruct the entire image by adding the correct color information to the black and white data provided as input. During training, the Mean Squared Error (MSE) loss function is used, which measures the squared error between the predicted colors and the actual pixel values across the image.

At each step of the training, the model takes as input a black and white image and tries to produce the corresponding colour version. The MSE Loss function calculates the difference between the predicted pixel values and the actual color (RGB) values in the entire image. Unlike *image inpainting*, which we will see below, where the loss is calculated only for the masked portions, here MSE Loss is applied to every pixel in the image, as the model must fully colorize all areas of the image.

The choice of MSE Loss is ideal for *image colorization* because the image data are continuous values (pixel values), and MSE is suitable for minimizing the differences between the prediction and the actual image. It aims to adjust the color at each pixel of the image in such a way that the predicted values are as close as possible to the actual color values.

During training, the model goes through multiple steps, known as epochs, in which it processes images from the *training set*. The dataset is divided into *batches* of size 64, and the model performs iterative updates of its parameters using the Adam optimizer, with a learning rate of 0.001. During each epoch (50 in total), the model is trained and the loss is calculated and accumulated to monitor the overall performance.

Below are some predictions of the model on images from the Caltech-256 Dataset (to better understand the differences between the images), which illustrate the colorization process of the black and white images (some of the model predictions are successful, while others do not reproduce the color result with the same accuracy):

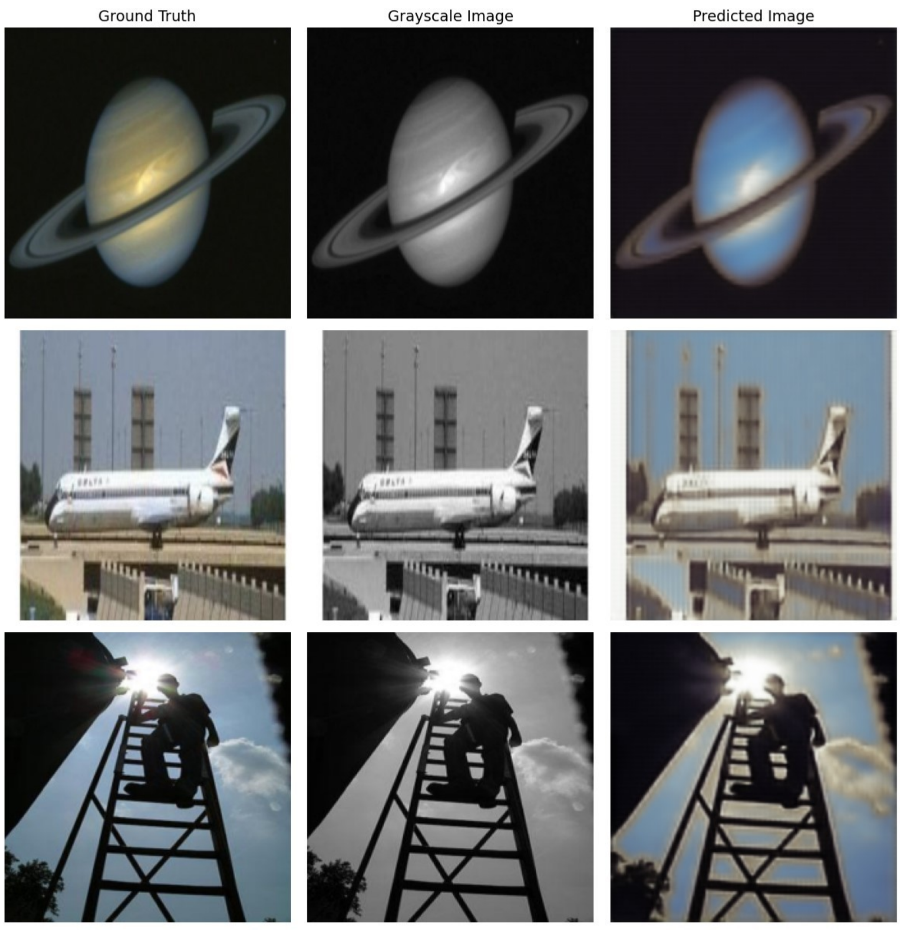


Figure 3.2.2 - 1: Examples from the image colorization task. The first column shows the original color images (Ground Truth), the second column shows the black and white versions of the images (Grayscale Image) and the third column shows the model's predictions for resetting the colors (Predicted Image).

3.2.3 Image Repair/Fill Process Implementation

Data preprocessing: in the implementation of the process for the image inpainting task, first each image is prepared by converting it to a uniform size and format that the model can process by transforming the dimensions of the image to 224x224 (Resize), normalizing it (Normalize) and converting it to a tensor (ToTensor), just as in the previous tasks. The processing involves applying a binary mask to the image, which consists of values 0 and 1, where 0 represents the regions we want to cover and fill in the model, and 1 represents the regions that remain intact.

The mask is created randomly using geometric shapes, such as lines and circles, placed at random points in the image. This results in the masking of certain areas of the image, which the model is asked to reconstruct. The next step is to multiply the image with the mask, creating the "masked" image.

At each training epoch, the mask is applied randomly, i.e. each image may receive a different mask in each iteration. This helps the model to train on many variations of the same image, improving its ability to generalize and fill in the gaps correctly.

Architecture: For the image inpainting task, the same architecture is used as described for image colorization, i.e., a ResNet50-based autoencoder. The main difference is that the model focuses on repairing the empty areas of the image as we will describe in the training below, while the same decoding layers with transposed convolutions and skip connections for detail preservation are used for the reconstruction.

Education: During the training of image inpainting, the Mean Squared Error (MSE) Loss function was used, which is ideal for this particular case, as it aims to minimize the difference between the predicted and actual pixel values only in the masked regions of the image. MSE Loss calculates the squared error between the predicted and actual pixel values. In this task, however, the loss is calculated only for the masked parts of the image, i.e. the parts that have been intentionally altered.

The rest of the image, which remains intact, is not affected by the model. The goal is for the model to learn to reconstruct only the lost parts, preserving the structures and details of the original image. Thus, by using MSE Loss for the masked parts, the model learns to produce results that approximate as closely as possible the actual data in the masked parts, without affecting the remaining regions.

The choice of MSE is appropriate as the image data is continuous information, and MSE ensures that the difference between the prediction and the actual image is minimized in a way that takes into account the pixel values and their variations.

In summary, the training process for image inpainting follows the following steps: Each image after being multiplied by a randomly generated mask, which covers certain parts of the image has a masked form. The masked image is used as input to the model, which tries to predict the masked points. The prediction is compared only with the masked parts of the original image, while the rest of the image remains unchanged. This process is run for 50 epochs with a batch size of 64, ensuring a gradual improvement in the model's ability to reconstruct the missing parts. Adam Optimizer was used for the optimization with a learning rate equal to 0.0001. In each epoch, a different random mask is applied to each image, which allows the model to view the same image with many different data losses (masks). This enhances its ability to generalize and adapt to different scenarios where parts of the image are missing.

In the following, we present some illustrative model predictions on images of the Caltech-256 Dataset (for a better understanding of the differences in the images) showing the correction/filling of the random masked regions of the images (some of the model predictions are successful, while in other cases the restoration of gaps is less expensive):

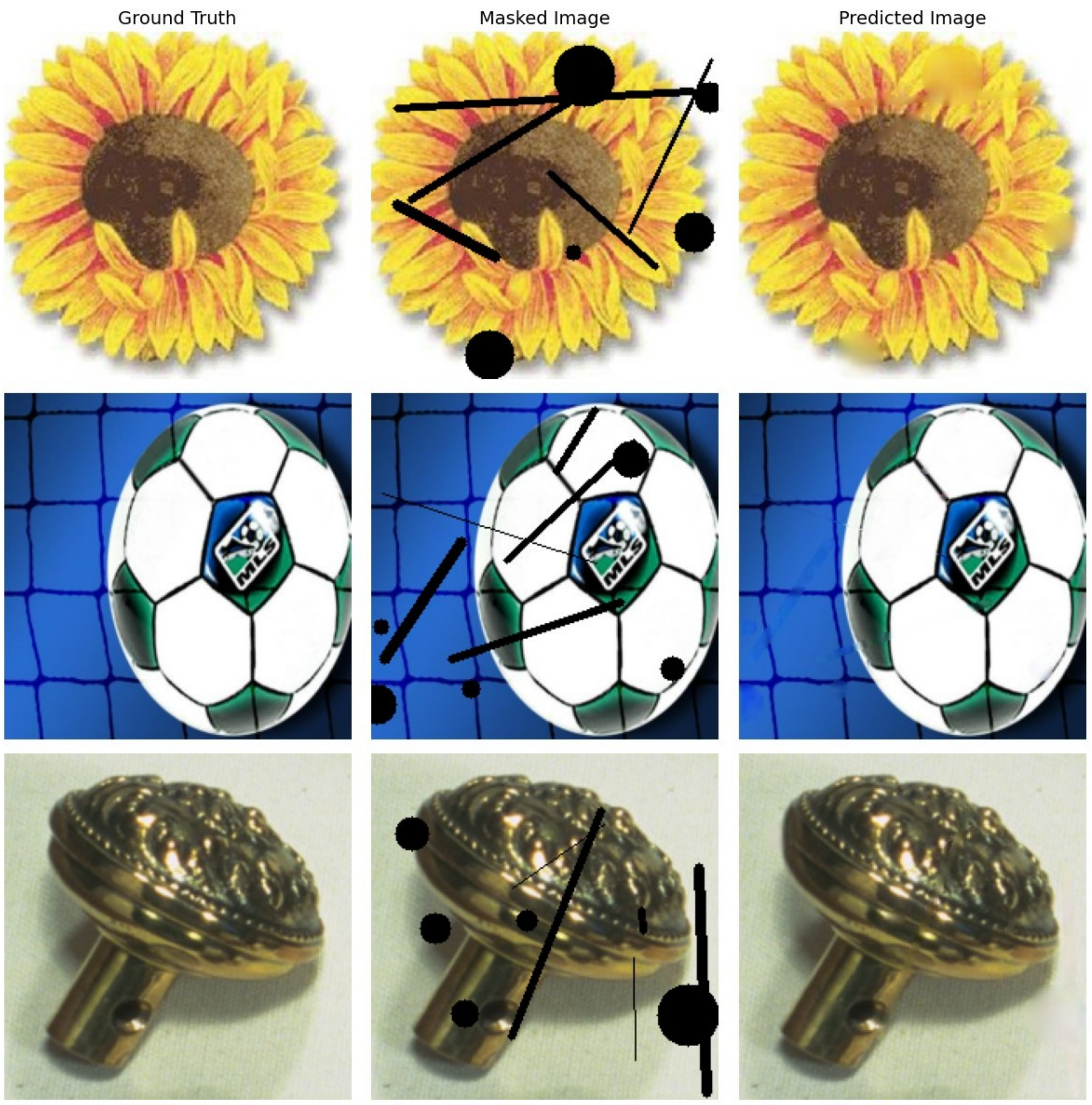


Figure 3.2.3 - 1: Examples of images from the image inpainting task. The first column shows the original image (Ground Truth), the second column
shows the image with masked parts (Masked Image), and the third column shows the image predicted by the model (Predicted Image).

3.3 Downstream Tasks and Data Sets

The downstream tasks are the next stage where the performance of the pre-trained models is evaluated in real applications, such as image classification. In this case, the model trained in proxy tasks, such as rotation prediction, colorization or image completion, is tested in the downstream task of image classification. This stage is crucial, as it highlights whether the features learned by the model from the proxy tasks can be generalized to more practical and challenging applications. Furthermore, the selection of appropriate datasets is equally important, as these directly affect the performance and generalizability of the model. In this section, both the image classification task and the datasets used will be explained.

3.3.1 Classify Images as Downstream Task

Image Classification is a classic problem in computer vision, which aims to assign an image to one of several predefined categories. This task requires the ability of the model to extract efficient features from the image and associate them with the corresponding classes. Image classification acts as a downstream task in many research experiments, allowing the generalization of features learned by the models from prior proxy tasks to be evaluated.

In this context, the model is trained to recognise high-level features, such as shapes, textures and colours, which are critical for the correct categorisation of images. This approach is based on the principles of *hierarchical learning*, where the first layers of the neural network extract simple features, such as edges, while subsequent layers combine these features to form more complex representations.

The training process includes 50 seasons and the batch size is 64. In addition, Cross-Entropy Loss was used as the loss function and for optimization we used Adam Optimizer with learning rate equal to 0.001 and weight decay equal to 0.0001 as a normalization technique. Finally, the Dropout technique was used where necessary to avoid overfitting.

The performance of the model in the image classification task is evaluated through metrics such as *accuracy*, which reflects the percentage of correct predictions relative to the image set. A high accuracy indicates that the model has learned to correctly identify classes, even when images are visually complex or contain noise. In this downstream task, the ability of the model to exploit features learned during pre-training through other indirect processes is critical to its final performance.

Neural networks used for image classification, such as ResNet, typically apply successive *convolutional* and fully *connected* layers. The final layers of the network extract a representation of the image in a high-dimensional feature space, and an activation function (e.g., *softmax*) is applied to estimate the probability that the image belongs to each category. The category with the highest probability is selected as the final model prediction.

Classifying images as a downstream task allows to evaluate how the pre-trained weights and features learned by the model from proxy tasks can help generalize and apply these features to more specific and demanding computational tasks vision.

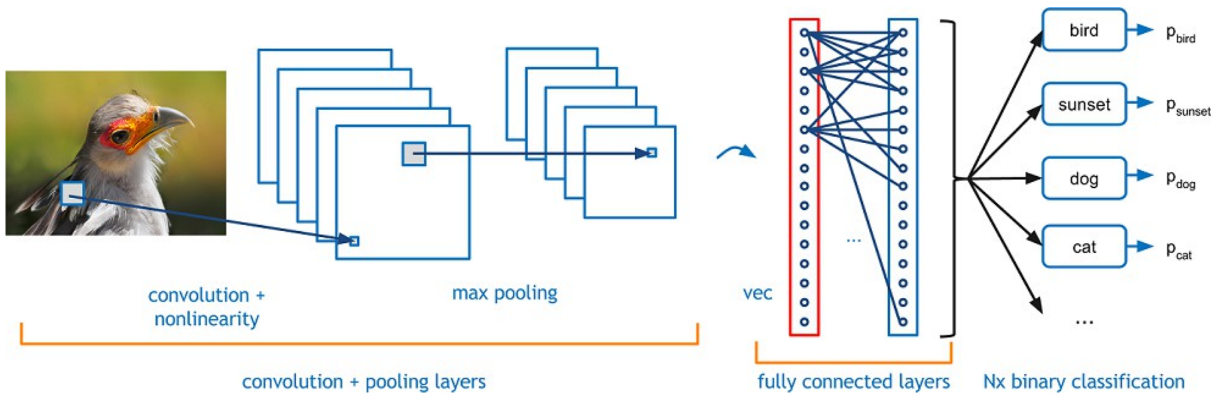


Figure 3.3.1 - 1: Process of the image classification task, where a convolutional neural network (CNN) processes an image through convolutional and max pooling layers to extract features and perform classification into predefined categories according to the generated probabilities P. (Source: TowardsDataScience)

3.3.2 Data Sets Used

The experiments used the CIFAR-10 and CIFAR-100 datasets, which are among the most widely used datasets in the field of computer vision. These datasets contain color images of small dimensions (32x32 pixels), and aim to classify images into multiple categories.

CIFAR-10 contains a total of 60,000 images, divided into 10 different classes. Each class contains 6,000 images, of which 5,000 are used for training and 1,000 for model evaluation. The classes include various objects such as cars, cats, airplanes, dogs, and other everyday objects and animals. CIFAR-10 is often used in image classification tasks, as its images are simple and widely understood, and their small dimension allows for faster training.

CIFAR-100 is similar to CIFAR-10, but contains 100 categories instead of 10, making the classification task much more demanding. Each class contains 600 images, with 500 for training and 100 for evaluation. CIFAR-100's classes are more specialized and include subcategories such as insects, fish, furniture, and other everyday objects. Despite the greater variety of categories, the CIFAR-100 images are the same size (32x32 pixels) and similar in complexity to CIFAR-10.

For the training of the proxy tasks, CIFAR-100 was used. The features learned by the model during training in the proxy tasks with this dataset were transferred to the downstream task of image classification. For classification, the pre-trained weights from CIFAR-100 were tested both on the same dataset (CIFAR-100) and on a different dataset, CIFAR-10, in order to evaluate the ability of the model to generalize to different datasets.

CIFAR-10 and CIFAR-100 are compatible with each other as they have similar characteristics. Both datasets contain images with the same resolution (32x32) and have a wide range of object categories. This compatibility between the datasets allows testing of the pre-trained weights from CIFAR-100 to CIFAR-10, providing useful information on whether the features learned by the model can be generalised to the classification task.

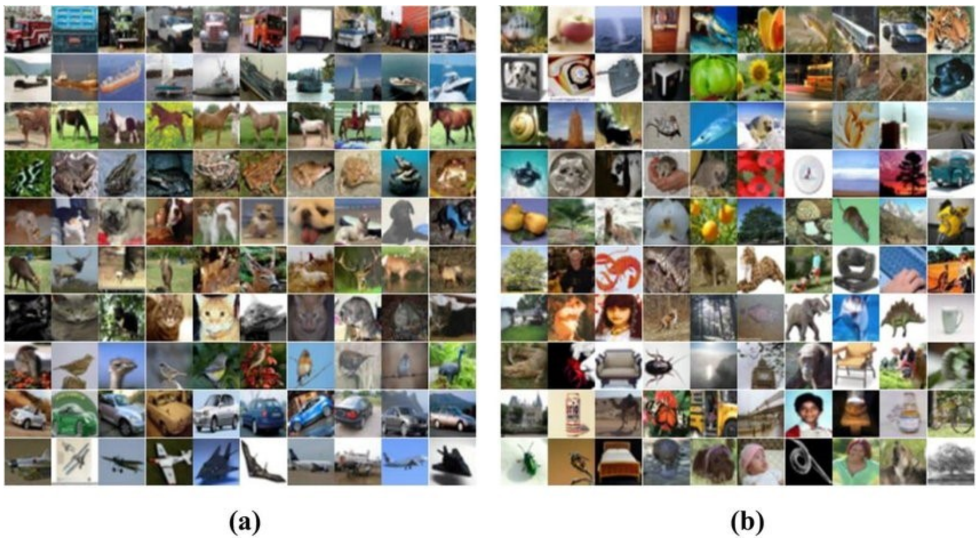


Figure 3.3.2 - 1: (a) Images from CIFAR-10 and (b) images from CIFAR-100, showing examples of categories contained in the datasets.

3.4 Transfer Learning and Fine-Tuning

Transfer learning and fine-tuning are critical techniques in modern machine learning, especially in cases where the dataset for the downstream task is
limited. The basic idea of transfer learning is to use a pre-trained model, which has learned useful features from a different task, to improve performance on a new but related task. Fine-tuning consists in further adapting the
pre-trained model weights to the new task, either by adapting only the last few layers of the network or by retraining the entire model. In this section, we review the learning transfer strategies from the proxy tasks to the downstream task of image classification, as well as the fine-tuning techniques applied to optimize the performance of the model

3.4.1 Transfer Learning from the Proxy Tasks to the Downstream Task

The process of transferring learning from the proxy tasks to the downstream task is based on the use of the optimal pretrained weights obtained during training in the three proxy tasks: rotation prediction, colorization and inpainting, which were trained on the CIFAR-100 dataset. It is important to note that CIFAR-100 was used as an unlabeled dataset for training the proxy tasks, without using the labels of the images to indicate the categories they belong to. During training on the proxy tasks, we monitored the improvement in validation accuracy and, whenever the model scored the highest performance on the validation set, we saved the model state. This ensured that we maintained the most efficient and generalizable weights necessary for transfer to the downstream task.

After training on the proxy tasks was completed, we constructed an identical model for the downstream task of image classification. To be able to use the stored weights, we loaded the pretrained weights into the new model and then modified the last fully connected layer to accommodate the number of classes required by each dataset. Specifically, for the CIFAR-10 dataset, the final layer was reconfigured to produce 10 outputs, the same number of classes in CIFAR-10, while for CIFAR-100 the final layer was modified to produce 100 outputs.

This adaptation of the last level is important, as it allows the correct matching of the feature space learned by the model with the number of classes required by the dataset in question. With this procedure, we were able to evaluate the effectiveness of the pre-trained weights acquired by CIFAR-100 not only on the same dataset but also on a different dataset, CIFAR-10. This allows us to examine whether the features learned by the model on one dataset can be generalized and improve performance on a different dataset.

3.4.2 Fine-Tuning in Downstream Task

In the fine-tuning process in the downstream task of image classification, we performed two different approaches to train the model. Specifically, we trained the model on each dataset (CIFAR-10 and CIFAR-100) in two ways:

1. Training the whole model, fine-tuning all layers (fine-tuning all layers).
2. Fine-tuning only the last layer of the network (fine-tuning only the last layer), while keeping the other layers frozen.

In the first case, full network training was performed on each task, allowing weights at all levels to be updated during the training process. In the second case, to retrain only the last layer, we freeze all layers of the network and unfreeze the last layer, leaving only the last fully connected layer free for training. In this way, we were able to evaluate how the pre-trained weights affect the performance of the model when fine-tuning only the last layer.

This procedure was applied to all proxy tasks (rotation prediction, colorization and image repair/completion). The goal was to evaluate the performance of the model on different datasets and with different fine-tuning strategies.

To preprocess the data, we used the following transformations: resize the images to 224x224 pixels, convert them to tensors and normalize the pixel values with the mean and standard deviation values corresponding to CIFAR-10 and CIFAR-100. The use of normalization ensures that the input data have fixed values, which improves model training.

The training hyperparameters were common to all tasks, whose models were trained for 50 epochs with the batch size set to 64. Also, the optimization algorithm was Adam, with a learning rate of 0.001 and a weight decay of 1e-4. Training was performed using the CrossEntropyLoss function, as the downstream task involves multi-category classification.

With this strategy, we were able to evaluate the performance of the pre-trained weights on different datasets, both with full fine-tuning and with fine-tuning only the last level. The described procedure was applied in exactly the same way for the weights of all proxy tasks to ensure a fair comparison between indirect processes.

3.5 Results

In this section we will present the results of experiments conducted to evaluate the performance of the pre-trained weights in the image classification task.

3.5.1 Results of the 1st experiment

Task: Image Classification

Fine-Tuning: Final Layer

Weights: Randomly Initialized, Image Rotation Prediction Pretraining, Image Colorization Pretraining, Image Inpainting Pretraining

Dataset: CIFAR-10

Accuracy Type: Test Accuracy

Weights	test accuracy
Randomly initialized	35.42%
Image Rotation Prediction Pretraining	40.19%
Image Colorization Pretraining	49.13%
Image Inpainting Pretraining	60.85%

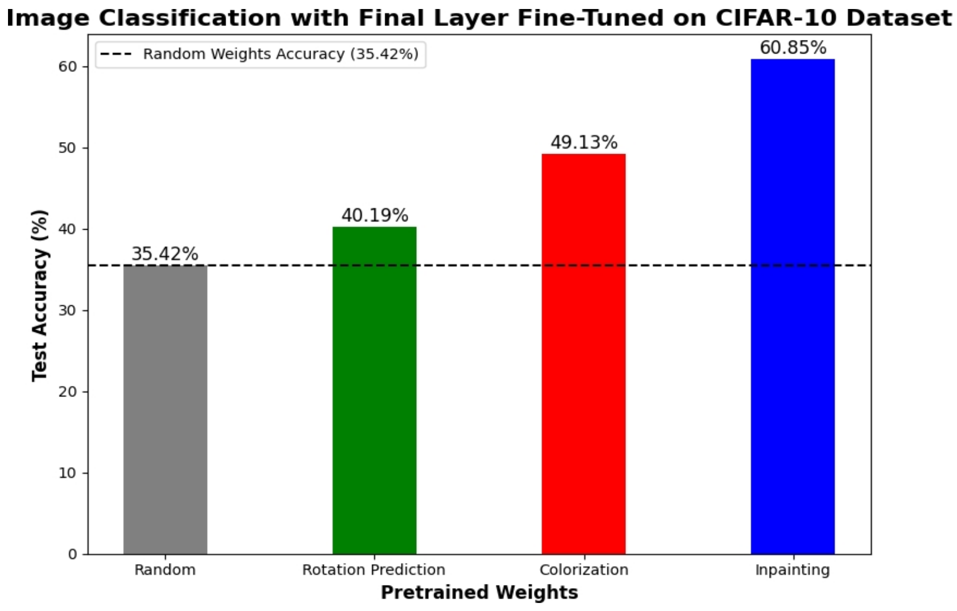
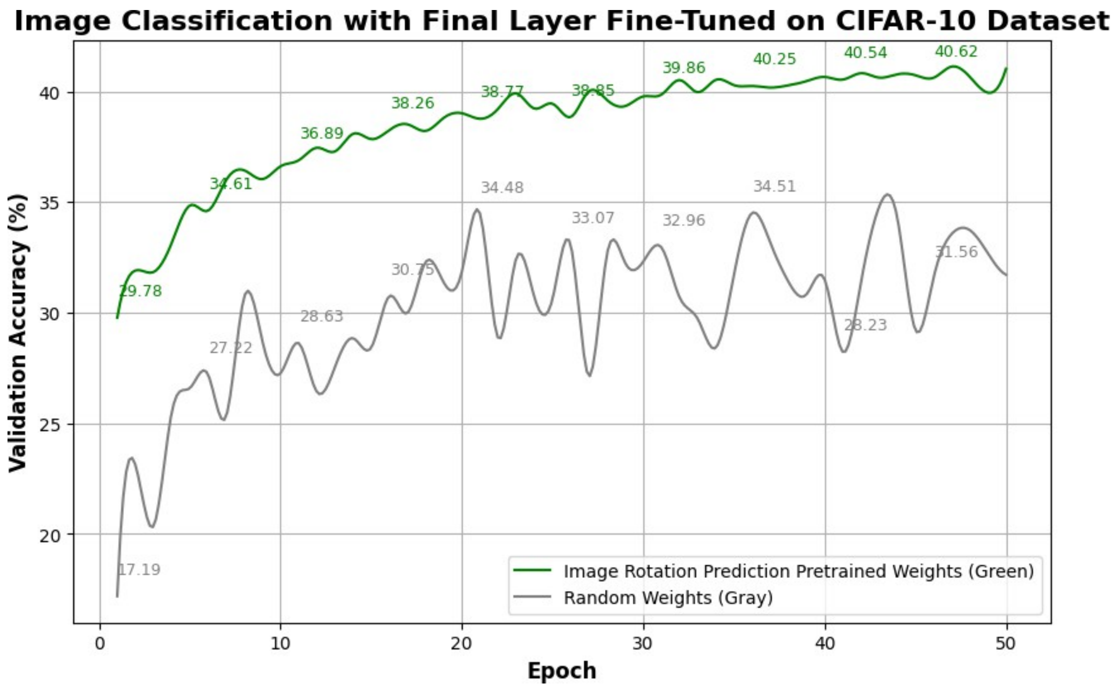
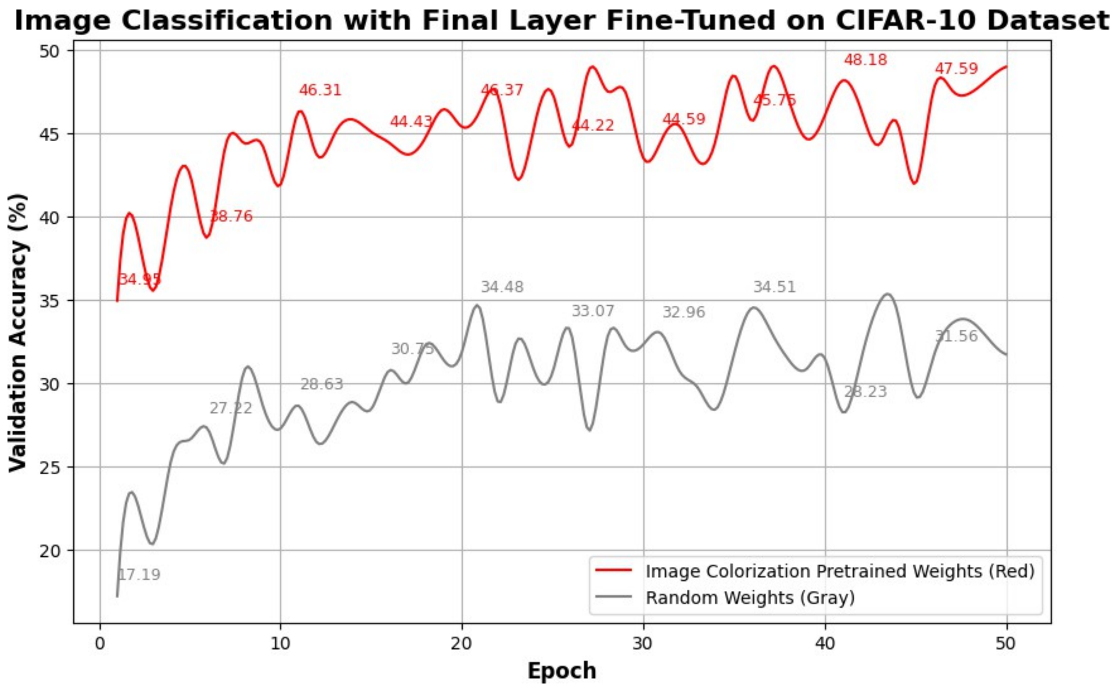
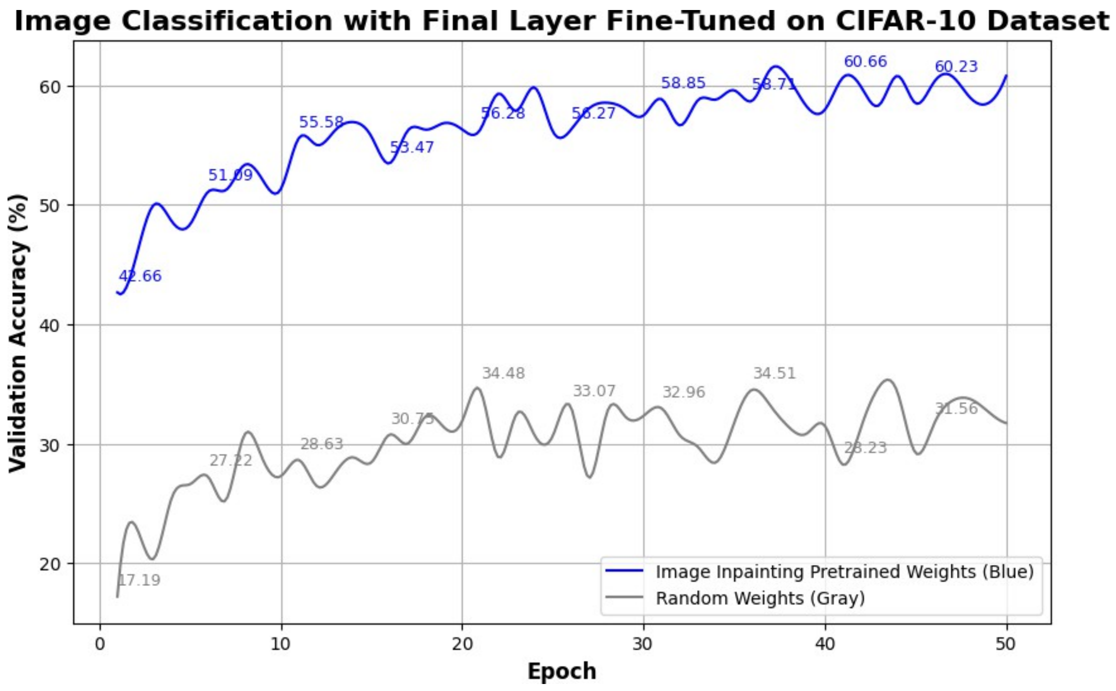


Table-Figure 3.5.1 - 1: Accuracy Test Results in image classification, retraining only the last layer of the model in this process, using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) in the CIFAR-10 dataset.

After presenting the final Test Accuracy values for the first experiment, more specific measurements for the same experiment will be presented next, breaking down the Validation Accuracy values by season. These metrics will give us a more detailed picture of the model's performance during training:







Figures 3.5.1 - 2,3,4: Accuracy results (in the validation set) by season in image classification, by retraining only the last layer of the model in the given process, using random weights as reference and the pre-trained weights from the three indirect processes: image rotation prediction, image colorization, image inpainting respectively, on the CIFAR-10 dataset.

3.5.2 Results of the 2nd experiment

Task: Image Classification

Fine-Tuning: Final Layer

Weights: Randomly Initialized, Image Rotation Prediction Pretraining, Image Colorization Pretraining, Image Inpainting Pretraining

Dataset: CIFAR-100

Accuracy Type: Test Accuracy

Weights	test accuracy
Randomly initialized	11.70%
Image Rotation Prediction Pretraining	13.16%
Image Colorization Pretraining	18.08%
Image Inpainting Pretraining	28.89%

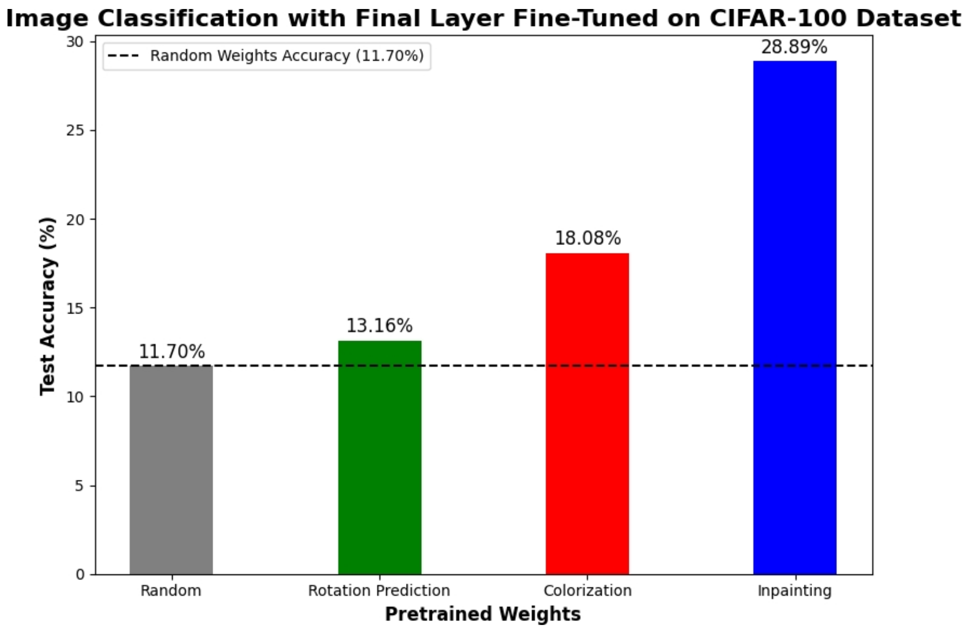
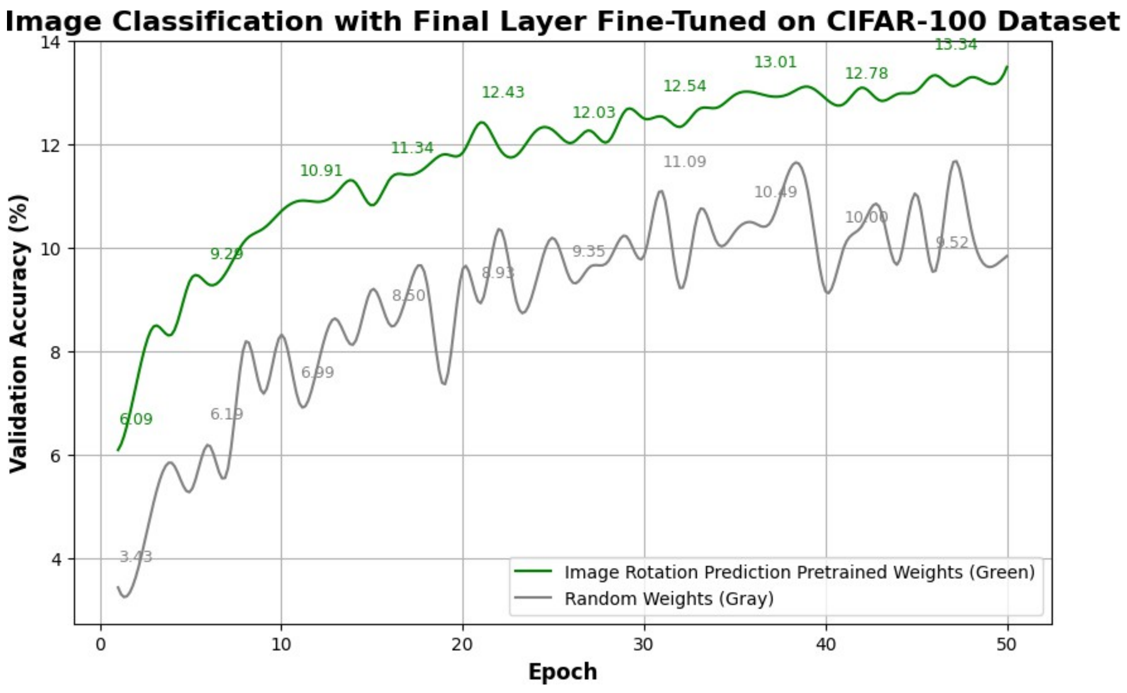
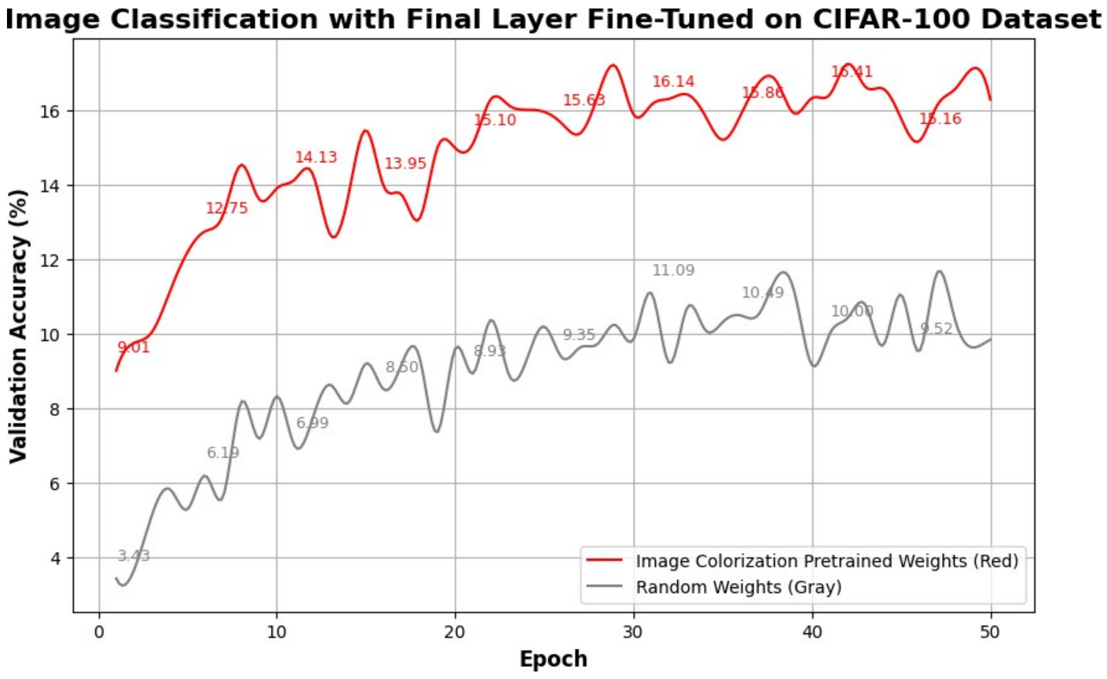
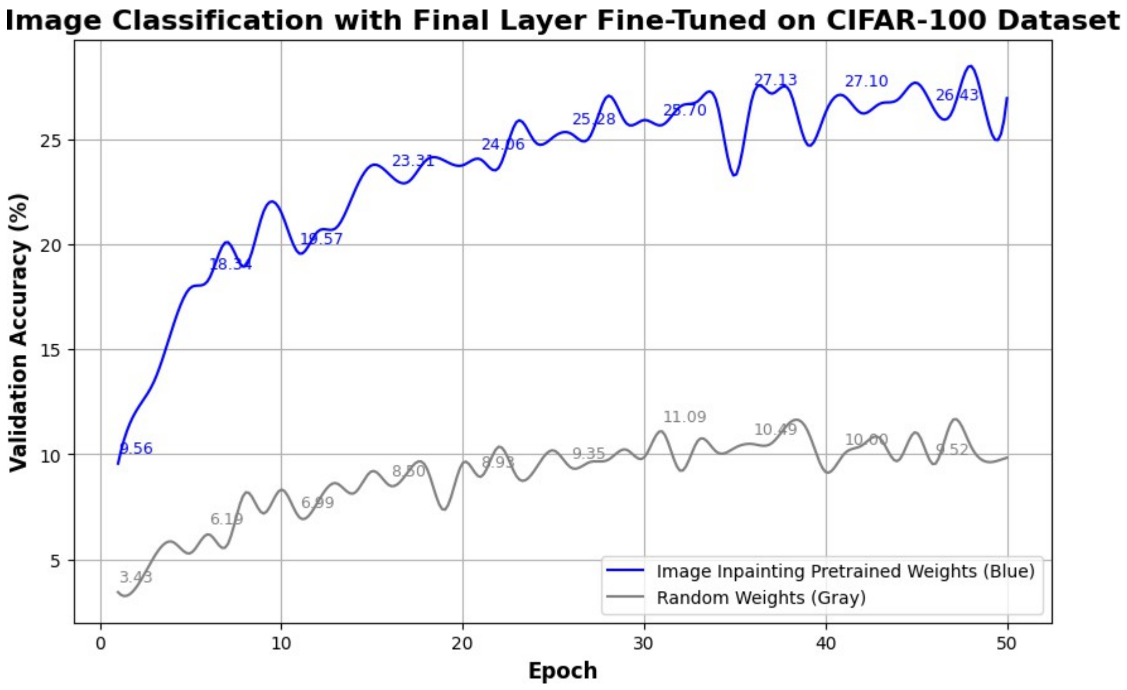


Table-Figure 3.5.2 - 1: Accuracy Test Results in image classification, retraining only the last layer of the model in this process, using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) in the CIFAR-100 dataset.

After presenting the final Test Accuracy values for the second experiment, more specific measurements for the same experiment will be presented next, breaking down the Validation Accuracy values by season. These metrics will give us a more detailed picture of the model's performance during training:







Figures 3.5.2 - 2,3,4: Accuracy results (in the validation set) by season in image classification, by retraining only the last layer of the model in this process, using random weights as a reference and the pre-trained weights from the three indirect processes Image Rotation Prediction, Image Colorization, Image Inpainting respectively, on the CIFAR-100 dataset.

3.5.3 Results of the 3rd experiment

Task: Image Classification

Fine-Tuning: All Layers

Weights: Randomly Initialized, Image Rotation Prediction Pretraining, Image Colorization Pretraining, Image Inpainting Pretraining

Dataset: CIFAR-10

Accuracy Type: Test Accuracy

Weights	test accuracy
Randomly initialized	80.42%
Image Rotation Prediction Pretraining	80.96%
Image Colorization Pretraining	82.45%
Image Inpainting Pretraining	83.14%

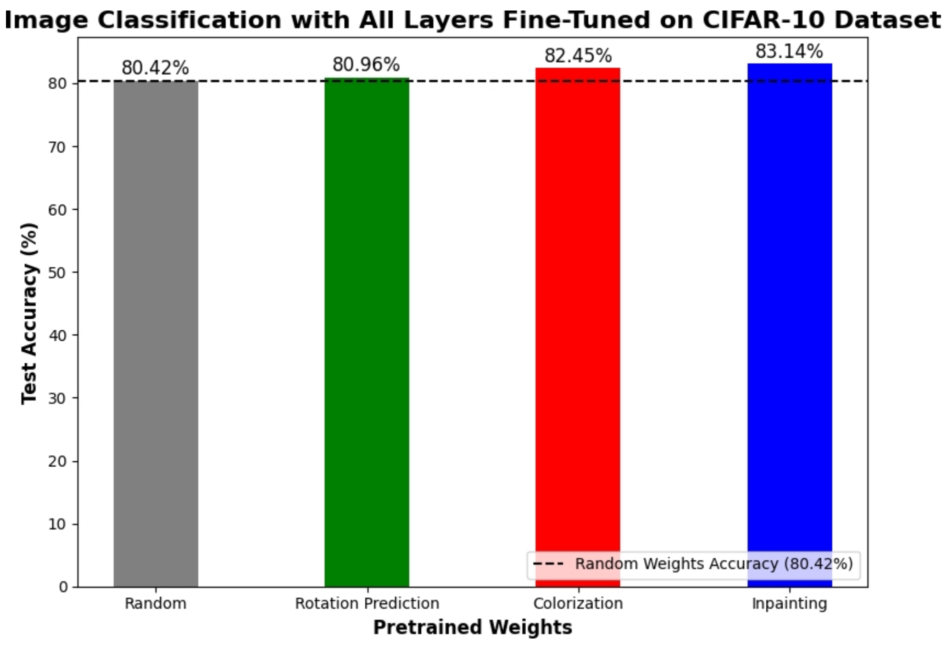
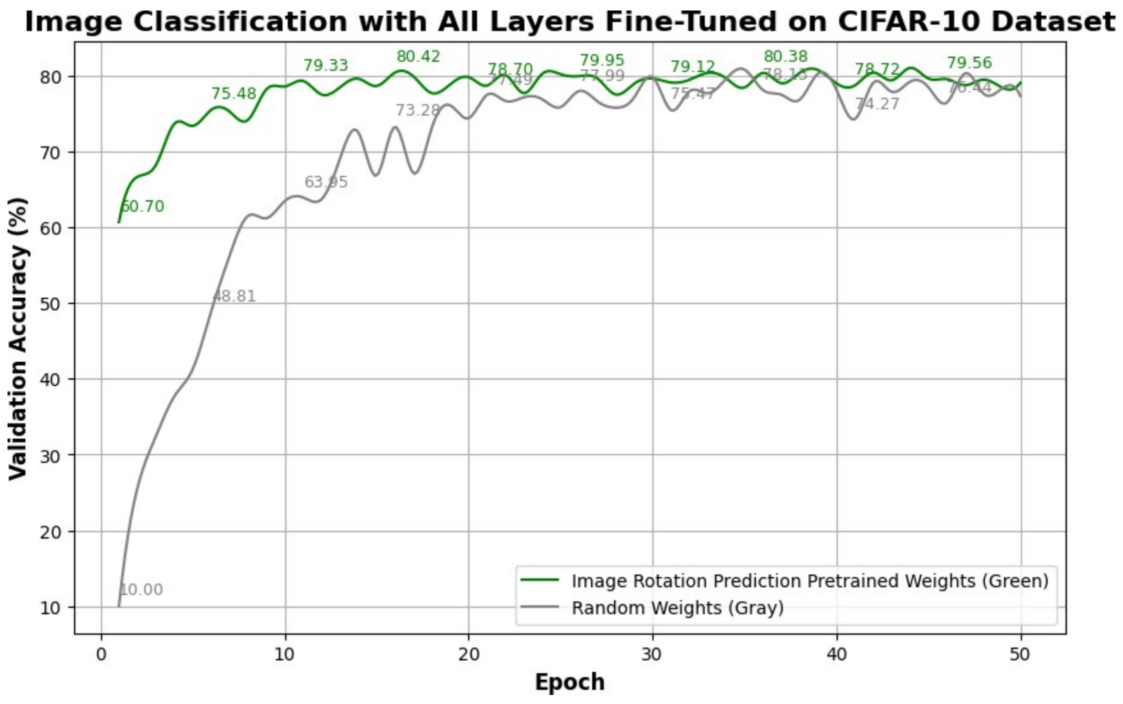
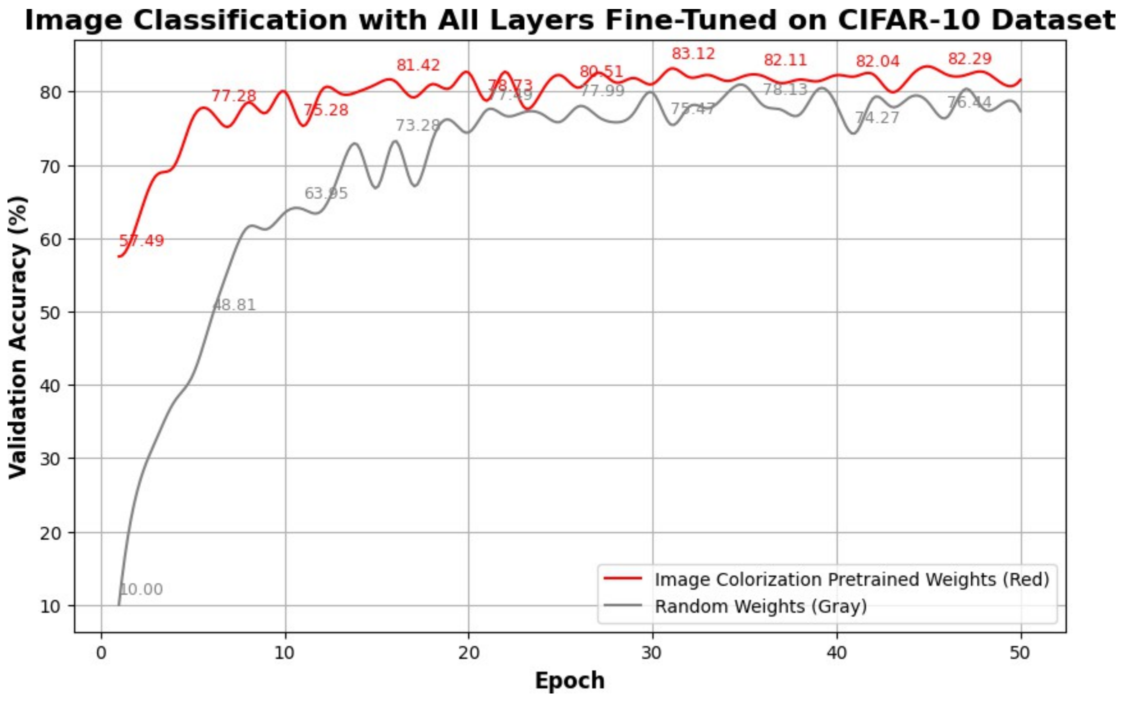
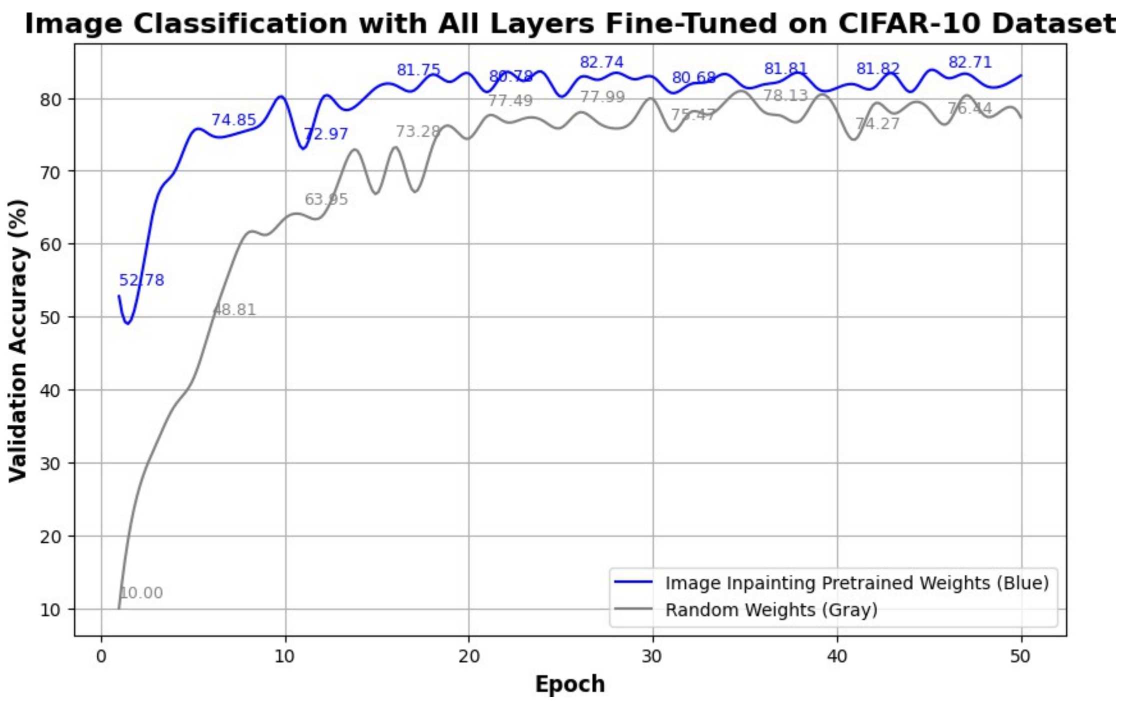


Table-Diagram 3.5.3 - 1: Accuracy Test Results (Test Accuracy) in image classification, retraining all layers of the model in the given process, using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) in the CIFAR-10 dataset.

Having provided the final Test Accuracy values for the third experiment, more specific measurements for the same experiment will then be presented, breaking down the Validation Accuracy values by season. These metrics will give us a more detailed picture of the model's performance during training:

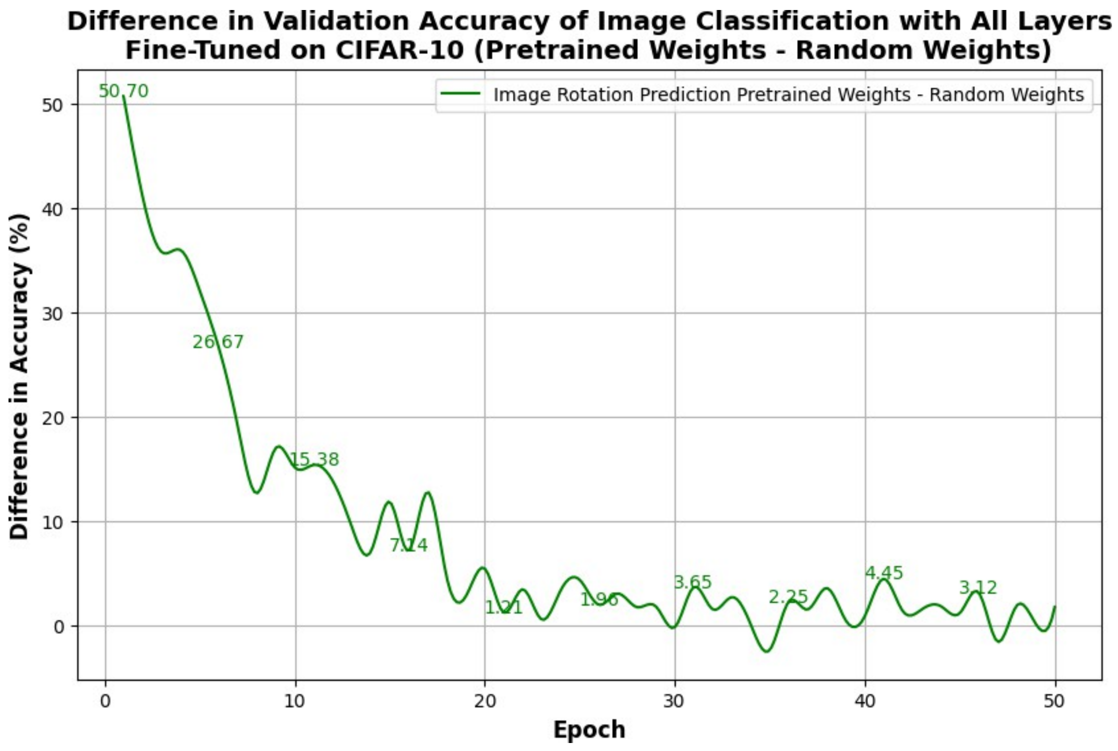


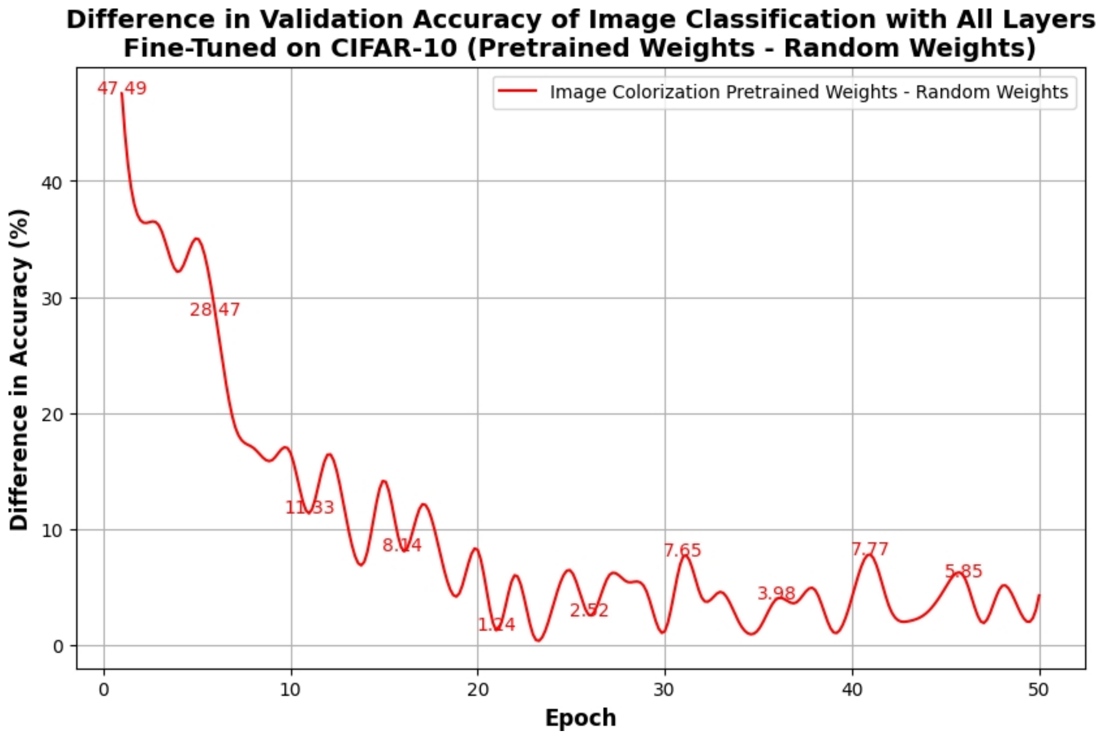


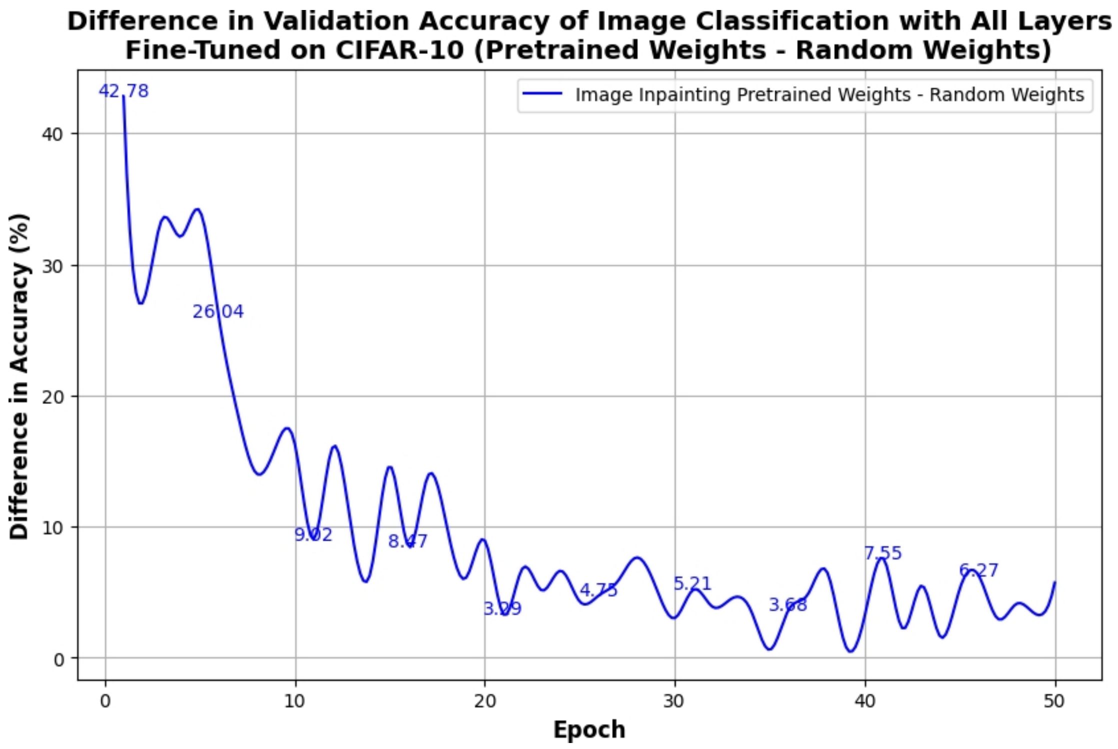


Figures 3.5.3 - 2,3,4: Accuracy results (in the validation set) by season in image classification, retraining all layers of the model in the given process, using random weights as reference and the pre-trained weights from the three indirect processes Image Rotation Prediction, Image Colorization, Image Inpainting respectively as initial weights, on the CIFAR-10 dataset.

In addition, in this experiment as in the next one, the plots showing the difference in accuracy between the values measured for the validation accuracy of the model with the pre-trained weights from each indirect process and the values measured for the validation accuracy of the model with random weights are presented. This comparison will help us to better understand the effect of the pre-trained weights versus random weights and to draw more comprehensive conclusions about the model performance over the seasons, i.e. during training:


Difference in Validation Accuracy of Image Classification with All Layers Fine-Tuned on CIFAR-10 (Pretrained Weights - Random Weights)
Image Rotation Prediction Pretrained Weights - Random Weights
Difference in Accuracy (%)
Epoch
50.70
26.67
15.38
7.14
1.21
1.96
3.65
2.25
4.45
3.12


Difference in Validation Accuracy of Image Classification with All Layers Fine-Tuned on CIFAR-10 (Pretrained Weights - Random Weights)
Image Colorization Pretrained Weights - Random Weights
Difference in Accuracy (%)
Epoch
47.49
28.47
11.33
8.14
1.24
2.52
7.65
3.98
7.77
5.85



Figures 3.5.3 - 5,6,7: Difference of Validation Accuracy in image classification, retraining all layers in the given task, between the model initialized with the pre-trained weights and the model initialized with random weights for each indirect process (Image Rotation Prediction, Image Colorization, Image Inpainting) respectively on the CIFAR-10 dataset.

3.5.4 Results of the 4th experiment

Task: Image Classification

Fine-Tuning: All Layers

Weights: Randomly Initialized, Image Rotation Prediction Pretraining, Image Colorization Pretraining, Image Inpainting Pretraining

Dataset: CIFAR-100

Accuracy Type: Test Accuracy

Weights	test accuracy
Randomly initialized	42.19%
Image Rotation Prediction Pretraining	50.92%
Image Colorization Pretraining	55.91%
Image Inpainting Pretraining	56.03%

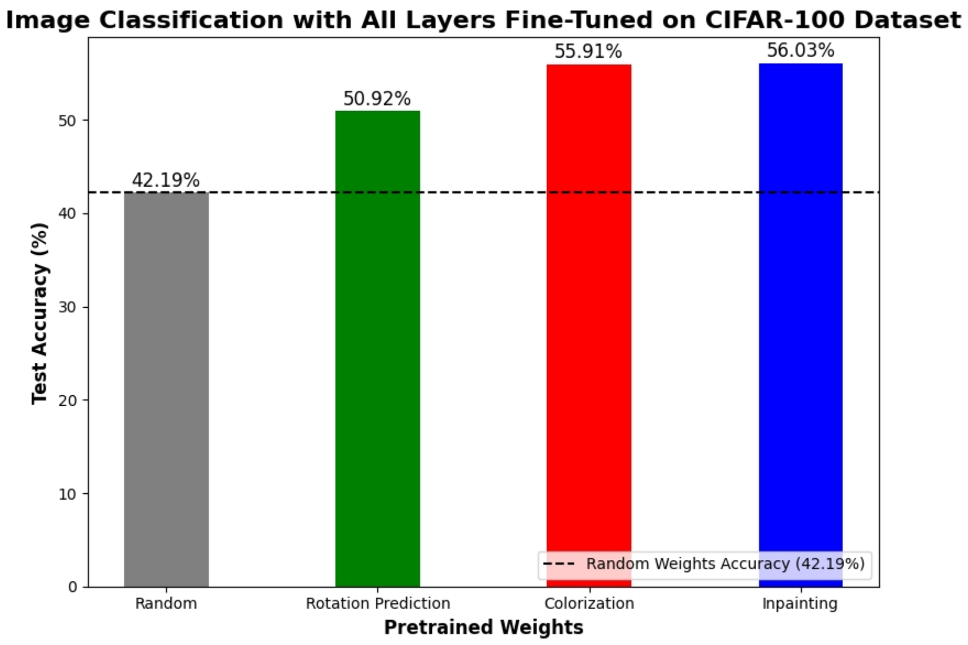
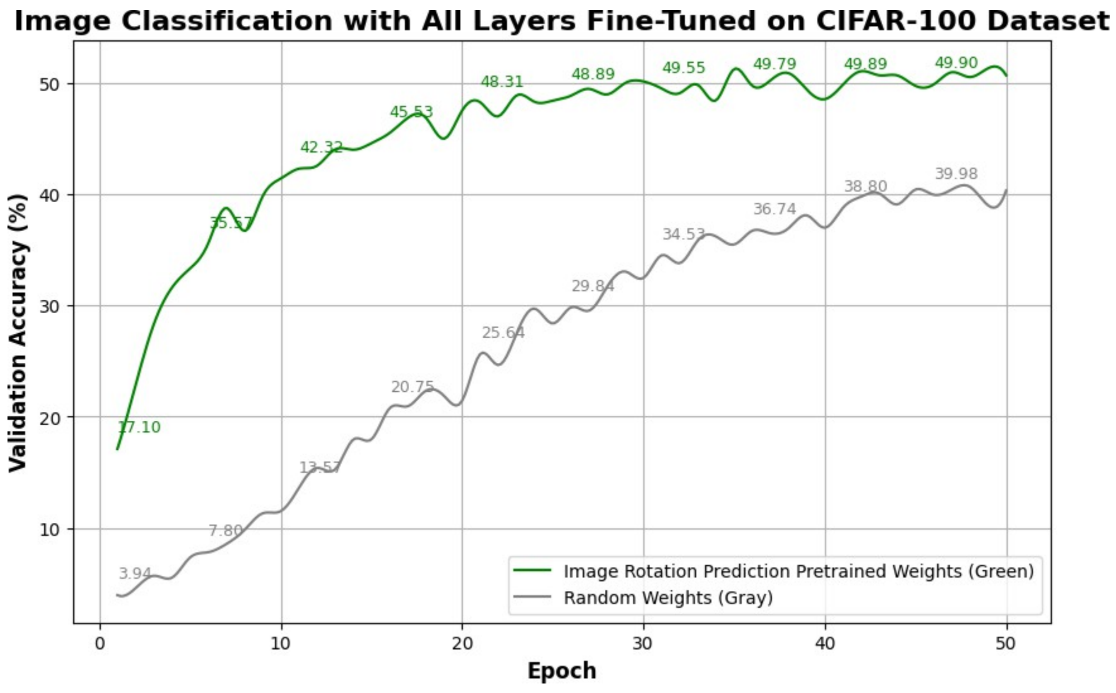
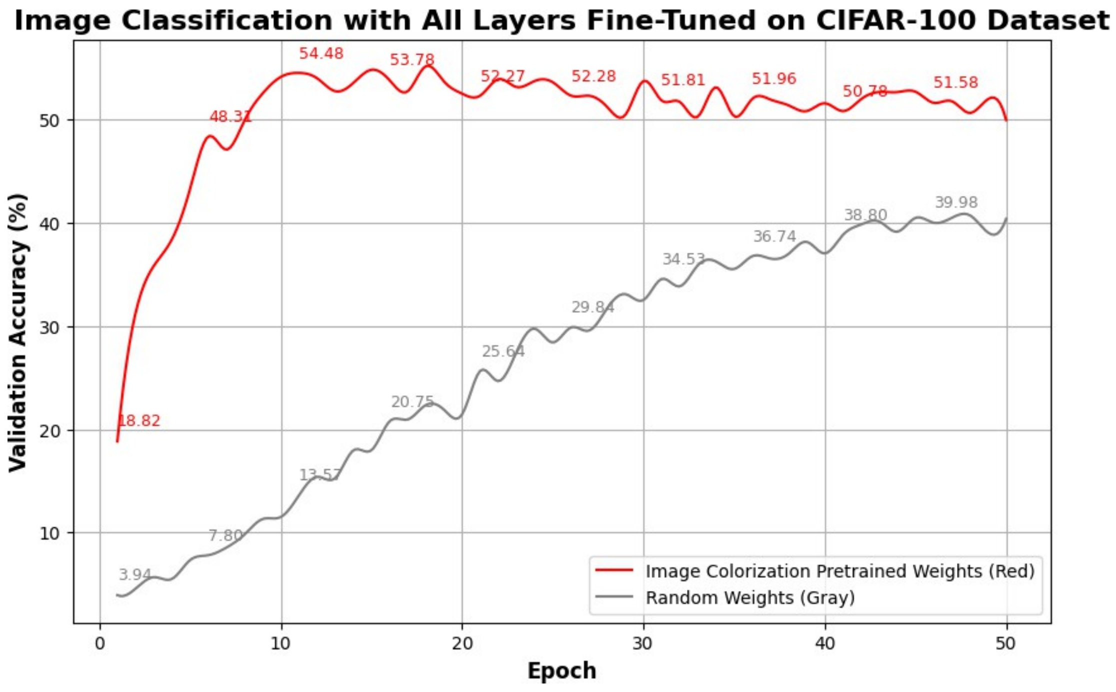
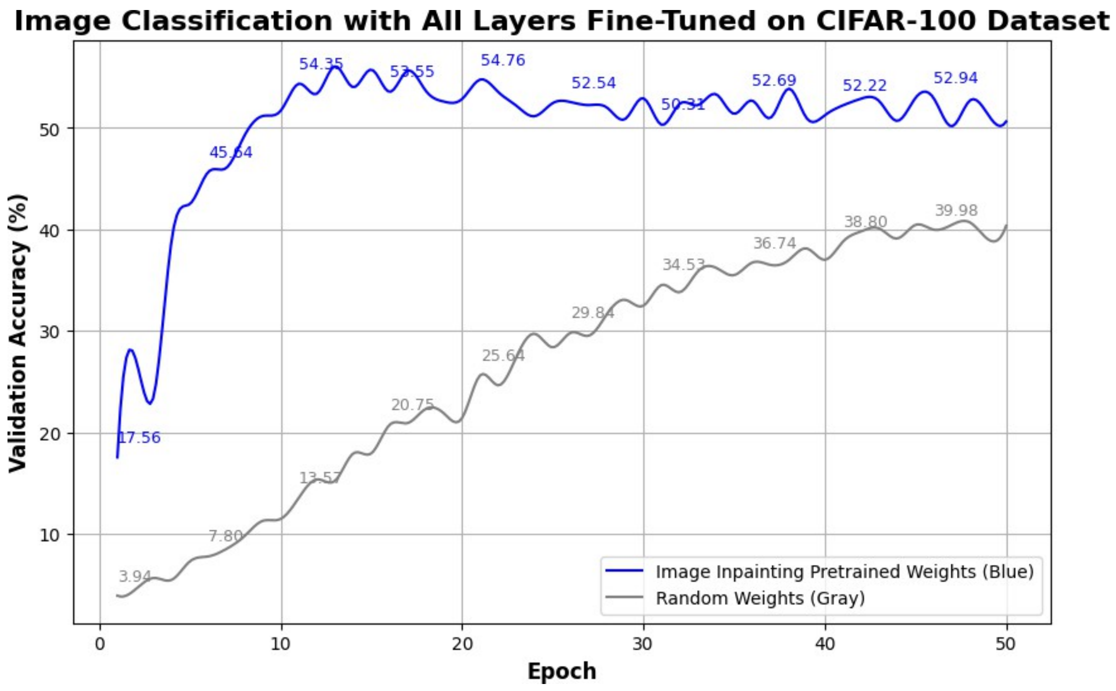


Table-Figure 3.5.4 - 1: Accuracy results (in the test set) in image classification, retraining all layers of the model in the given process, using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) in the CIFAR-100 dataset.

Having provided the final Test Accuracy values for the fourth experiment, more specific measurements for the same experiment will be presented next, breaking down the Validation Accuracy values by season. These metrics will give us a more detailed picture of the model's performance during training:

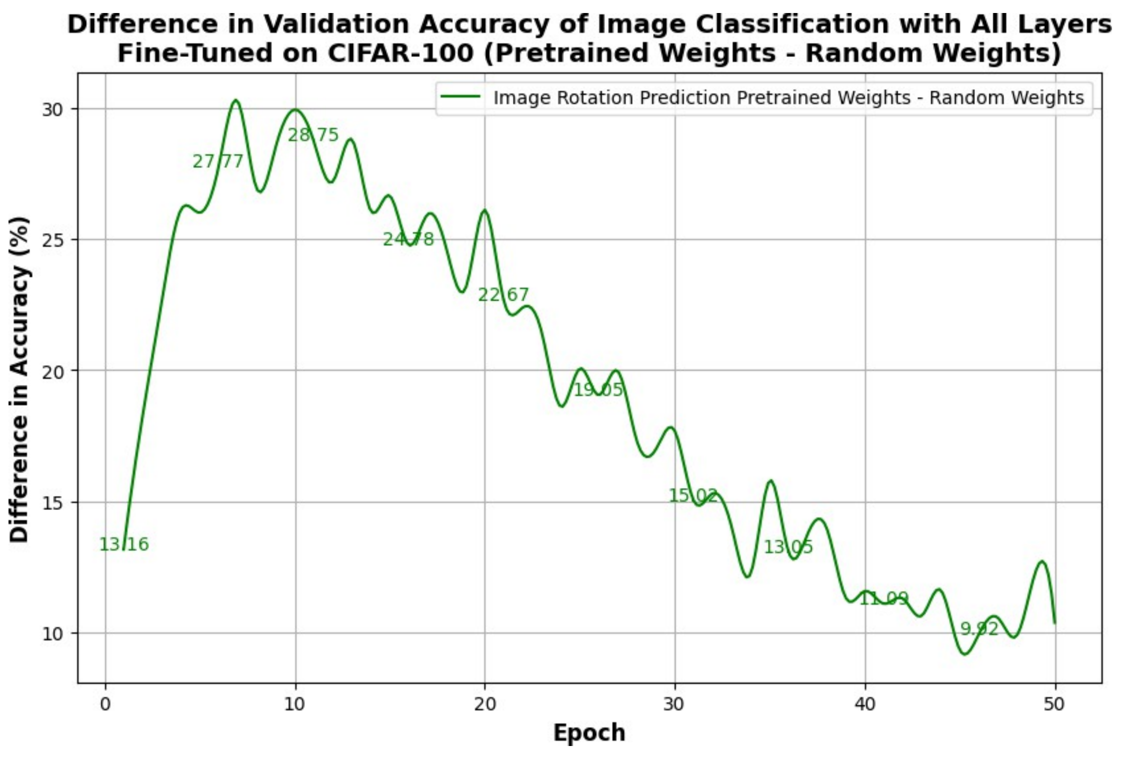


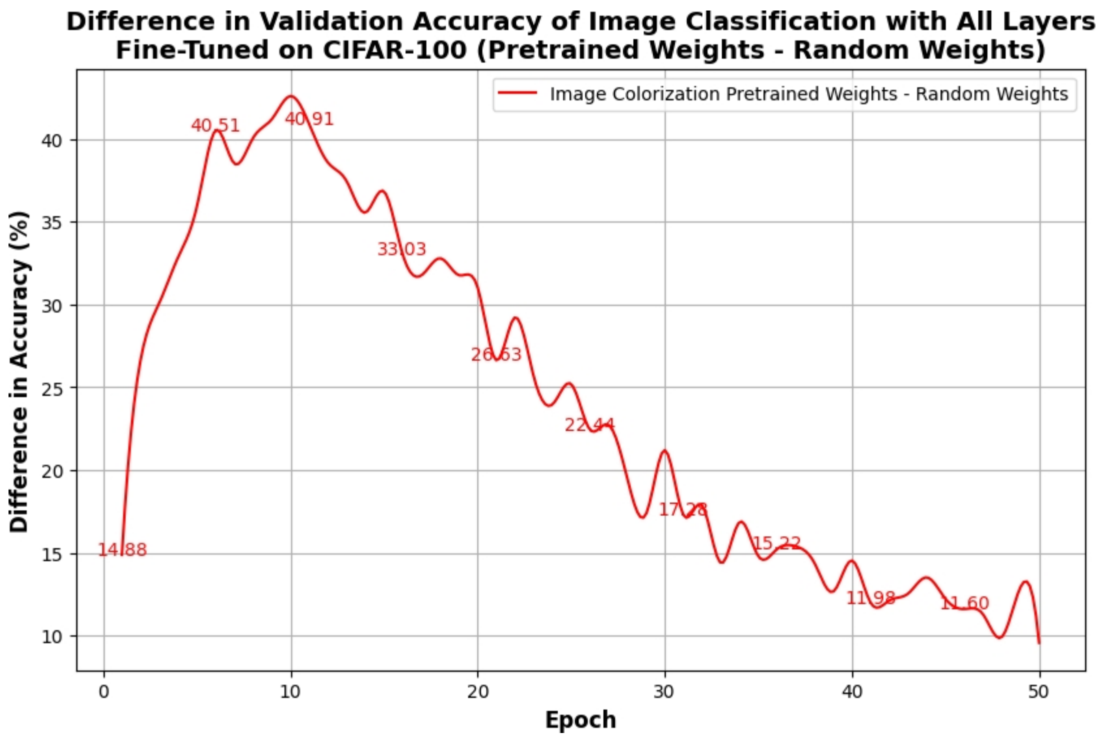


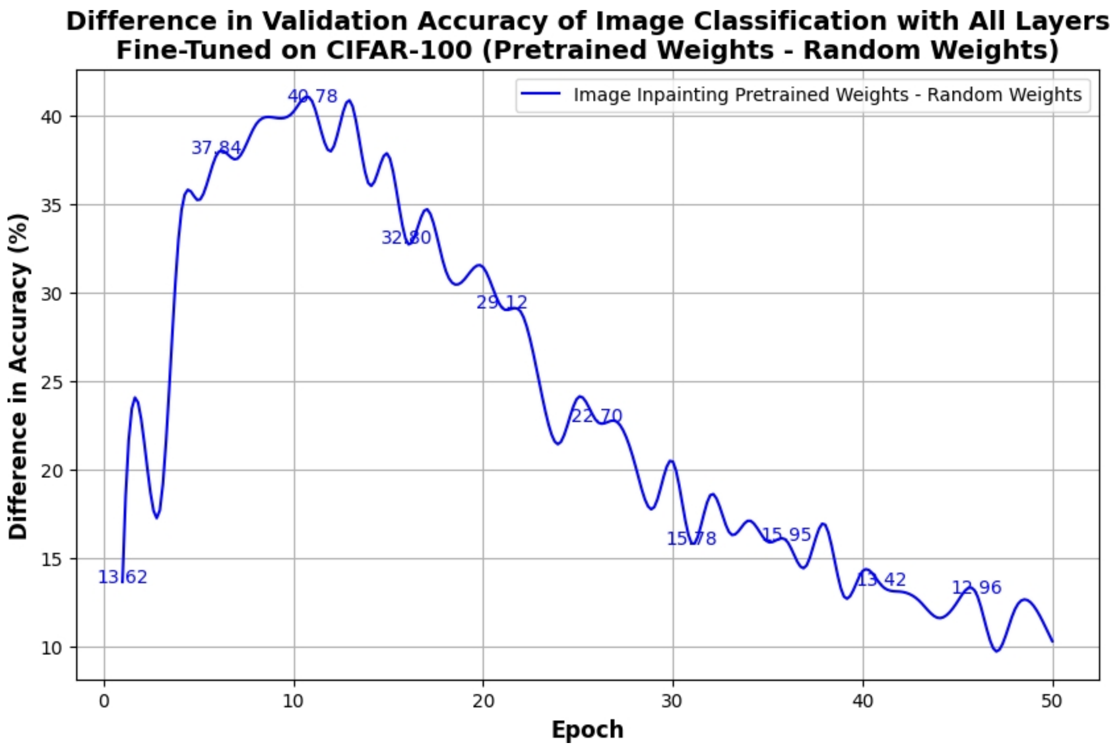


Figures 3.5.4 - 2,3,4: Accuracy results (in the validation set) by season in image classification, retraining all layers of the model in the given process, using random weights as reference and the pre-trained weights from the three indirect processes Image Rotation Prediction, Image Colorization, Image Inpainting respectively as initial weights, on the CIFAR-100 dataset.

In addition, we present plots showing the difference in accuracy between the values measured for the validation accuracy of the model with pre-trained weights from each indirect process and the values measured for the validation accuracy of the model with random weights. T h i s comparison will help us to better understand the effect of pre-trained weights versus random weights and to draw more comprehensive conclusions about the performance of the model over the seasons, i.e., during training:


Difference in Validation Accuracy of Image Classification with All Layers Fine-Tuned on CIFAR-100 (Pretrained Weights - Random Weights)
Image Rotation Prediction Pretrained Weights - Random Weights
13.16
27.77
28.75
24.78
22.67
19.05
15.02
13.05
11.09
9.82
Difference in Accuracy (%)
Epoch

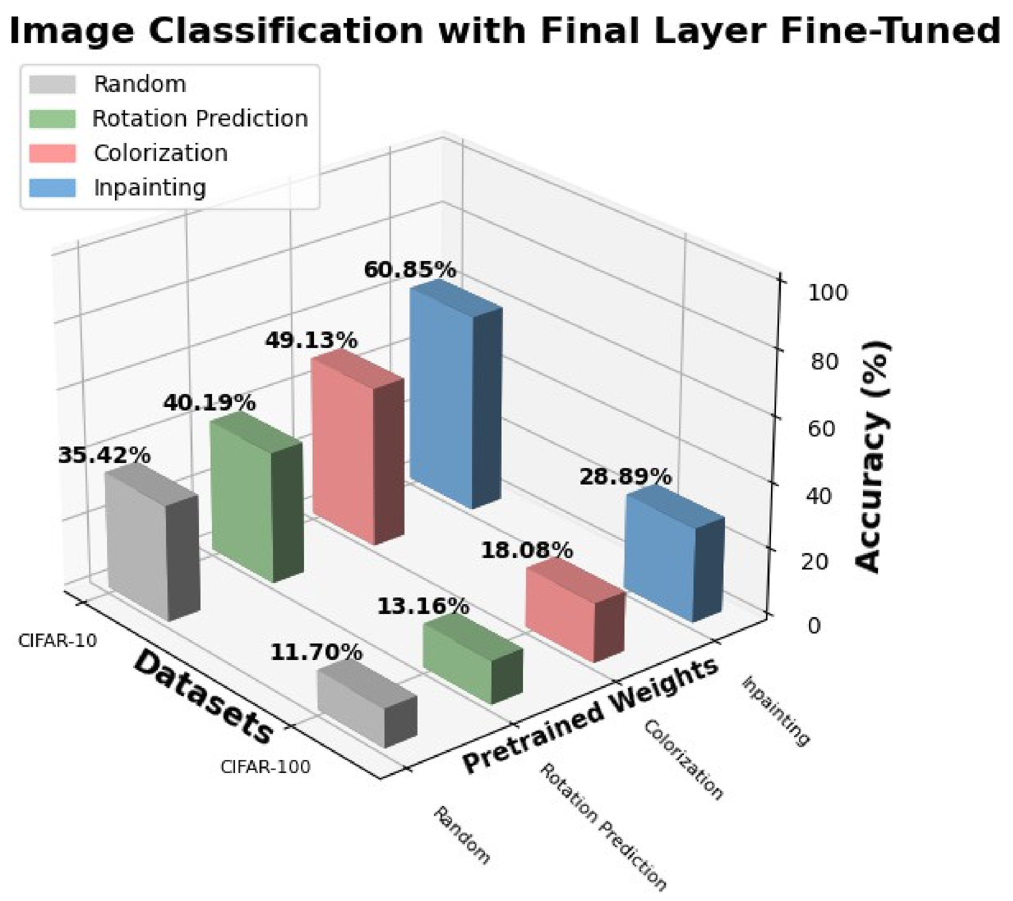

Difference in Validation Accuracy of Image Classification with All Layers Fine-Tuned on CIFAR-100 (Pretrained Weights - Random Weights)
Image Colorization Pretrained Weights - Random Weights
14.88
40.51
40.91
33.03
26.63
22.44
17.28
15.22
11.98
11.60
Difference in Accuracy (%)
Epoch



Figures 3.5.4 - 5,6,7: Difference of Validation Accuracy in image classification, retraining all layers in the given task, between the model initialized with the pre-trained weights and the model initialized with random weights for each indirect process (Image Rotation Prediction, Image Colorization, Image Inpainting) respectively on the CIFAR-100 dataset.

3.5.5 Aggregated display of results

Finally, we will summarize all the measurements in the following charts to facilitate understanding of the differences between the tasks and to provide a more complete picture of the model's performance. By aggregating the results, it will be easier to draw conclusions about the effect of the pre-trained weights of each indirect process, as well as to compare them with the random weights.



Figures 3.5.5 - 1: Test Accuracy results in image classification, retraining only the last layer of the model in the given process, using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) on the CIFAR-10 and CIFAR-100 datasets.

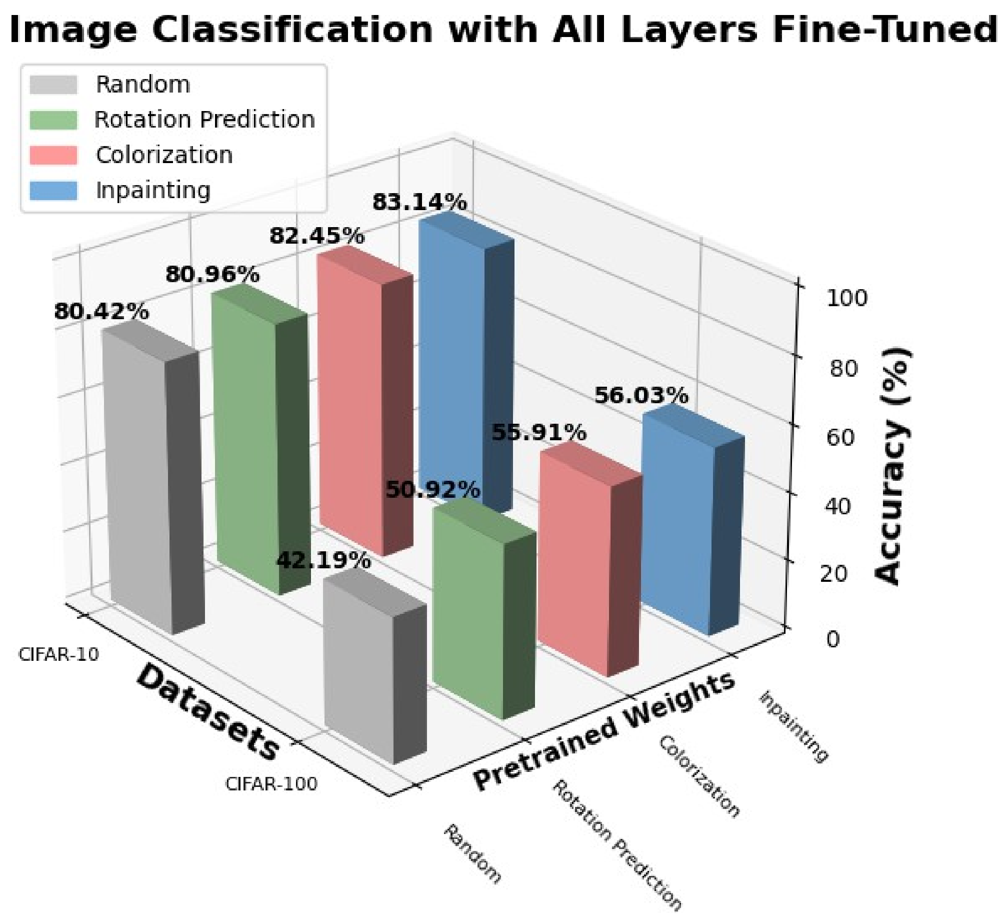
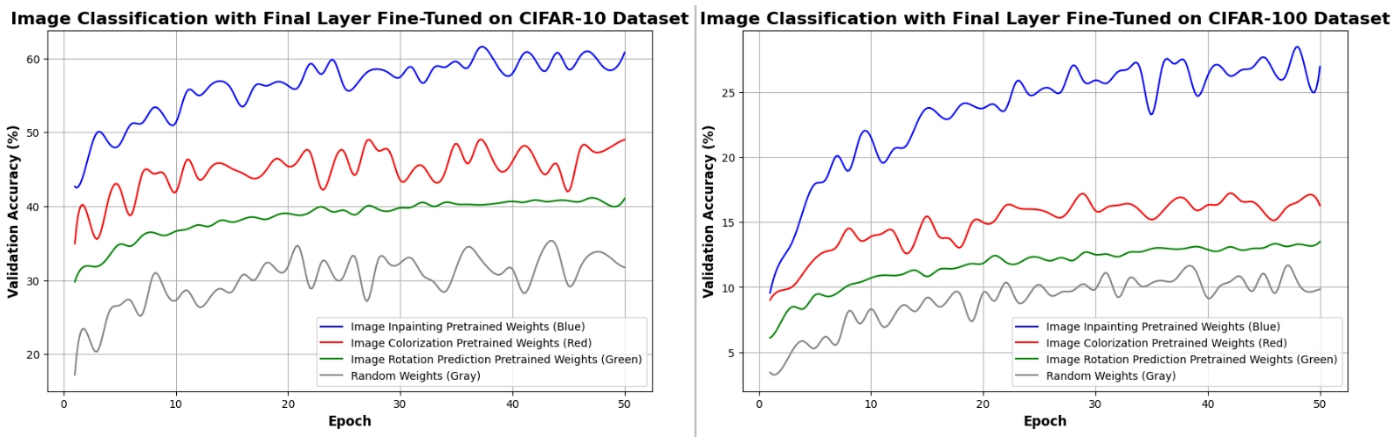
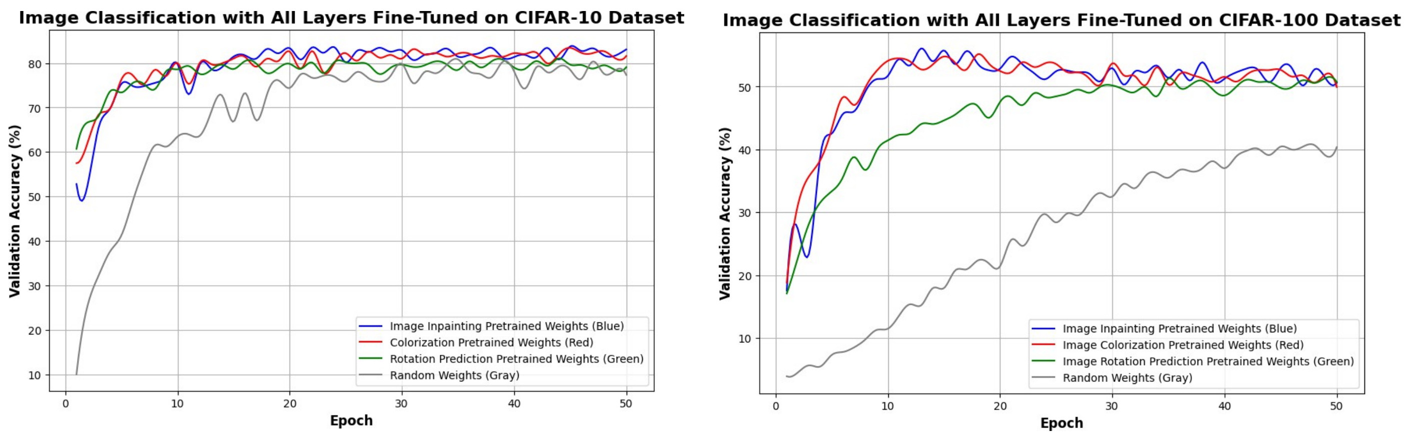


Figure 3.5.5 - 2: Test Accuracy results in image classification by retraining all layers of the model in the given process using random and pre-trained weights from the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) on the CIFAR-10 and CIFAR-100 datasets.

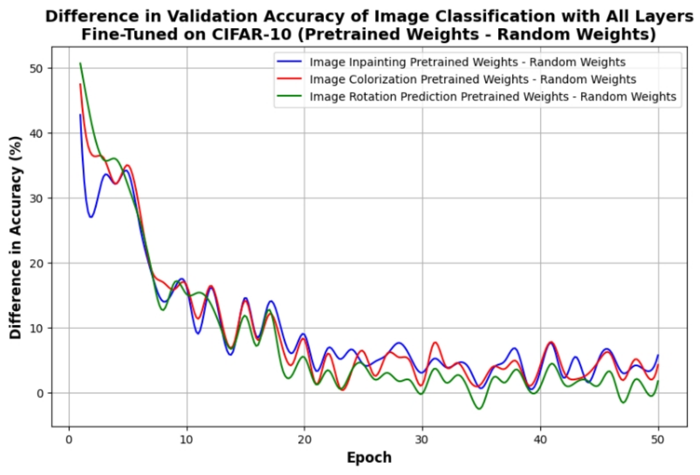


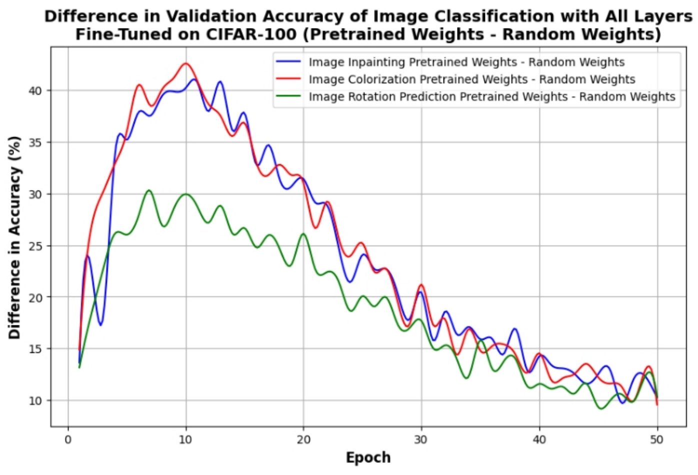
Figures 3.5.5 - 3.4: Accuracy results (in the validation set) by era in image classification, by retraining only the last layer of the model in this process, using random weights as reference and the pre-trained weights from the three indirect processes Image Rotation Prediction, Image Colorization, Image Inpainting as initial weights, on the CIFAR-10 and CIFAR-100 datasets.



Δ

Figures 3.5.5 - 5.6: Accuracy results (in the validation set) by era in image classification, retraining all layers of the model in the given process, using random weights as reference and the pre-trained weights from the three indirect processes Image Rotation Prediction, Image Colorization, Image Inpainting as initial weights, on the CIFAR-10 and CIFAR-100 datasets.





Figures 3.5.5 - 7,8: Difference of Validation Accuracy in image classification, retraining all layers in the given task, between the model initialized with the pre-trained weights and the model initialized with random weights for the three indirect processes (Image Rotation Prediction, Image Colorization, Image Inpainting) on the CIFAR-10 and CIFAR-100 datasets.

3.6 Commentary of Results - Conclusions

Performance using Indirect Processes for pre-training weights versus random weights

According to the measurements and as shown in the graphs presented in the previous section, the performance of all proxy tasks was consistently (in all experiments) better than those achieved using random weights. This confirms our initial expectation that *SSL* offers a significant advantage, as the pre-trained weights have already learned key features from the proxy tasks. In contrast, random weights carry no initial information and require more time and data to learn essential features. Therefore, the proxy tasks provided significant performance improvement on all metrics, confirming the purpose of self-initiated learning.

Performance on Different Datasets (CIFAR-10, CIFAR-100)

According to the results, the performance on CIFAR-100 was lower compared to CIFAR-10, which confirms our initial hypothesis that the classification task in CIFAR-100 is more complex due to the larger number of classes (100 vs. 10). This is evident both in the case of fine-tuning only the last level and in the case of fine-tuning the whole model, where the performance in CIFAR-100 remained lower.

However, pretrained weights trained on CIFAR-100 seem to transfer their knowledge to CIFAR-10 satisfactorily. Despite the diversity of these datasets, the pretrained weights derived from CIFAR-100 contributed to good performance on CIFAR-10, as expected. Although the proxy tasks were trained on 90 additional classes that are not present in CIFAR-10 and could be considered useless, the model was able to learn important information that was also useful in CIFAR-10, achieving satisfactory performance.

Furthermore, this shows us that the proxy tasks learn key features related to image orientation, color and coherence (for the three proxy tasks respectively) rather than based on the class of images. This allows the models to convey useful information regardless of the classes to which the images belong. This is also because we used CIFAR-100 as an unlabeled dataset for training on the indirect processes. Therefore, the claim that the model learned from the labels, even though we did not use them directly, is not valid because the results confirm that the model learns based on features such as image orientation, color and coherence, and not on the classes of the images.

In conclusion, performance was lower on CIFAR-100 due to the complexity of the task, but the proxy tasks proved that they can convey useful information in CIFAR-10, maintaining satisfactory performance.

Comparison of information between pre-trained weights from indirect processes and random weights, with fine-tuning only at the last level of the model

In CIFAR-10, we observe an increase in performance over the initial performance of the random weights:

- 13.47% for the rotation (from 35.42% to 40.19%),
- 38.71% for colorization (from 35.42% to 49.13%),
- 71.80% for inpainting (from 35.42% to 60.85%).

Similarly, in CIFAR-100 we observe an increase in performance over random weights:

- 12.48% for the rotation (from 11.70% to 13.16%),
- 54.53% for colorization (from 11.70% to 18.08%),
- 146.92% for inpainting (from 11.70% to 28.89%).

Therefore, with this direct performance comparison, we can easily see that the information learned by the weights in the proxy tasks is very useful and helps significantly in the image classification task.

Comparison of final performance between pre-trained weights from indirect processes and random weights, with fine-tuning at all levels of the model

In CIFAR-10, we observe an increase over the initial performance of the random weights:

- 0.67% for rotation (from 80.42% to 80.96%),
- 2.52% for colorization (from 80.42% to 82.45%),
- 3.38% for inpainting (from 80.42% to 83.14%). In

CIFAR-100, we observe an increase:

- 20.69% for the rotation (from 42.19% to 50.92%),
- 32.52% for colorization (from 42.19% to 55.91%),
- 32.80% for inpainting (from 42.19% to 56.03%).

According to the results, we see that with fine-tuning at all levels of the model, the final performance of the proxy tasks is only slightly better than that of the random weights. This is because the weights, whether random or pre-trained, are only used for initialization and then changed during training. Nevertheless, the pre-trained weights help the model converge slightly better.

Speed of achieving satisfactory performance with fine-tuning at all levels of the model

According to the results, it is evident from the graphs depicting the accuracy by epoch that the speed at which proxy tasks allow the model to reach high levels of performance is much higher than random weights. Specifically, in CIFAR-10, with the three proxy tasks (rotation prediction, colorization, inpainting) we achieve 75% accuracy in the first 5 epochs and with the three pretrained weights, while reaching 80% in the first 10 epochs. After there, each converges respectively to the final values mentioned above. In contrast, with the random weights, in the 5th epoch the accuracy is about 40-45% and in the 10 epochs it reaches about 61%.

Similarly, in CIFAR-100, we observe that with rotation we achieve 41% in the first 10 seasons, while with colorization and inpainting the maximum performance reaches 55% already in the first 10 seasons. Random weights, however, remain at 11% in the first 10 seasons.

The curves seen in the diagrams match perfectly with the following figure, which is presented in the paper "An analysis of transfer learning for domain mismatched text-independent speaker verification" by Chunlei Zhang, Shivesh Ranjan, John H.L. Hansen [54]. The figure shows that, in general in transfer learning, at the beginning of training we have a higher initial value (higher start), in the early epochs a higher slope is observed, and towards the end of training a higher asymptote is observed. This analysis confirms the logic of our measurements, which is clearly reflected in the graphs, showing that pretrained weights offer faster and higher performance than random weights.

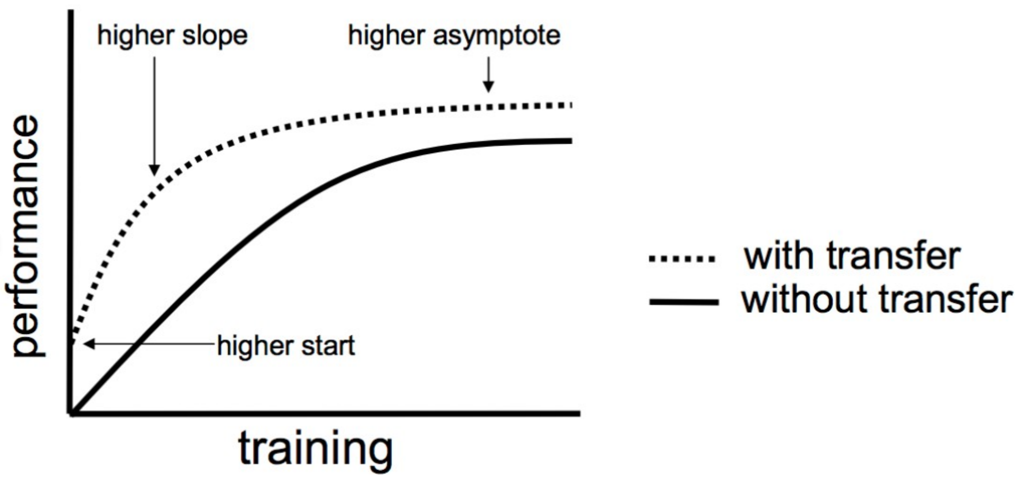


Figure 3.6 - 1: Representation of the difference between transfer learning and training from random initialization.
(Source: Zhang, C., Ranjan, S., & Hansen, J. H. (2018, June). An Analysis of Transfer Learning for Domain Mismatched Text-independent Speaker Verification. In *Odyssey* (pp. 181-186).)

This difference is even more apparent in the graphs showing the difference in accuracy between each proxy task and the random weights, highlighting the faster achievement of high performance achieved by using pretrained weights.

Best performing indirect process

All pretrained weights from the corresponding proxy tasks performed better than the random weights. The order from best to worst was 1) Image Inpainting, 2) Image Colorization, 3) Image Rotation Prediction.

This shows us that Image Inpainting was the most efficient task, probably because it is a complex process that requires the model to learn how to fill in gaps in the images while maintaining their consistency. This process forces the network to understand both the content of the image and its wider context, thus enhancing its ability to represent complex relationships and features.

On the other hand, Image Rotation Prediction turned out to be weaker, as it is a simpler process that only focuses on the orientation of objects. Although it provides some useful information to the model, it is not as feature-rich as the other two tasks, resulting in the information it carries being less useful to the downstream task, as shown by its slightly increased, but limited, performance.

3.7 Conclusion and Future Extensions

Summarizing the results of this thesis, we conclude that Self-Supervised Learning through the use of indirect processes offers significant advantages in training neural networks for image classification. In particular, pre-trained weights from proxy tasks were shown to be more efficient than random weights, both in terms of final performance and training speed. The Image Inpainting process emerged as the most efficient, followed by Image Colorization, while Image Rotation Prediction showed the lowest results, although still better than training with random weights.

This approach offers significant advantages, as it allows the exploitation of unlabelled data, reducing the need for specialised labelling and speeding up the training process. The fact that the pre-trained weights allowed the model to converge faster and achieve higher performance with fewer resources highlights the value of transferring learning to problems that do not have much labeled data.

Regarding future extensions of this work, the implementation of more complex neural network architectures or the incorporation of attention mechanisms, such as Transformer-based models, could be considered to improve performance on more demanding tasks. At the same time, the application of indirect processes to larger and more heterogeneous datasets could yield significant results on the generalizability of the features learned by the models.

Another important direction for further study would be to examine the adaptation of indirect processes to data of different nature, such as video data or even sensor data, to improve the analysis of multidimensional and sequential information. Finally, the use of more advanced fine-tuning techniques, such as progressive fine-tuning, could lead to greater efficiency and resource savings, especially in cases where the downstream task data is limited.

Research in this area shows significant promise, and this paper has provided a detailed evaluation of the potential of Self-Supervised Learning using implicit processes in improving the performance of neural networks for image classification.

1

Bibliography

Books:

- Boutalis, I. & Syracoulis, G. (2010). Computational Intelligence and applications. Krikos - Papamarkou Bros.
- Goodfellow, I. (2016) Deep learning. MIT Press Ltd.
- Hecht-Nielsen, R. (1989). *neurocomputing*. addison-Wesley Longman Publishing Co., Inc.

References:

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). imagenet classification with deep convolutional neural networks. *advances in neural information processing systems*, *25*.

[3] Simonyan, K., & Zisserman, A. (2014). very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). deep residual learning for image recognition. in *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[5] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). going deeper with convolutions. in *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

[6] Tan, M. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

[7] He, K., Fan, H., Wu, Y., Xie, S., & Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9729-9738).

[8] Doersch, C., Gupta, A., & Efros, A. A. (2015). unsupervised visual representation learning by context prediction. in *Proceedings of the IEEE international conference on computer vision* (pp. 1422-1430).

[9] Noroozi, M., & Favaro, P. (2016, September). unsupervised learning of visual representations by solving jigsaw puzzles. in *European conference on computer vision* (pp. 69-84). cham: Springer International Publishing.

[10] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.

[11] Zhang, R., Isola, P., & Efros, A. A. (2016). colorful image colorization. in *Computer Vision-ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III.*

14 (pp. 649-666). Springer International Publishing.

[12] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context encoders: feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2536-2544).

[13] Gidaris, S., Singh, P., & Komodakis, N. (2018). unsupervised representation learning by predicting image rotations*. arXiv preprint arXiv:1803.07728*.

[14] Krizhevsky, A., & Hinton, G. (2009). learning multiple layers of features from tiny images.

[15] Pan, S. J., & Yang, Q. (2009). a survey on transfer learning. *iEEE Transactions on knowledge and data engineering*, *22*(10), 1345-1359.

[16] Long, M., Cao, Y., Wang, J., & Jordan, M. (2015, June). learning transferable features with deep adaptation networks. in *International conference on machine learning* (pp. 97-105). pMLR.

[17] Vaswani, A. (2017). attention is all you need. *advances in neural information processing systems*.

[18] McCulloch, W. S., & Pitts, W. (1943). a logical calculus of the ideas immanent in nervous activity. *the bulletin of mathematical biophysics*, *5*, 115-133.

[19] Kingma, D. P. (2014). adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[20] Ioffe, S. (2015). batch normalization: accelerating deep network training by reducing internal covariate shifts. *arXiv preprint arXiv:1502.03167.*

[21] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

[22] Rosenblatt, F. (1958). the perceptron: a probabilistic model for information storage and organization in the brain. *psychological review*, *65*(6), 386.

[23] Lowe, D., & Broomhead, D. (1988). multivariable functional interpolation and adaptive networks. *complex systems*, *2*(3), 321-355.

[24] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.

[25] Watkins, C. J., & Dayan, P. (1992). q-learning. *machine learning*, *8*, 279-292.

[26] Rummery, G. A., & Niranjan, M. (1994) *On-line Q-learning using connectionist systems* (Vol. 37, p. 14) Cambridge, UK: University of Cambridge, Department of Engineering.

[27] Mnih, K. (2015). mnih V., Kavukcuoglu K., Silver D., Rusu Aa, Veness J., Bellemare MG, et al. *Human-level control through deep reinforcement learning, Nature*, *518*(7540), 529-533.

[28] Tibshirani, R. (1996). regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *58*(1), 267-288.

[29] Hoerl, A. E., & Kennard, R. W. (1970) Ridge regression: Biased estimation for nonorthogonal problems *Technometrics*, *12*(1), 55-67.

[30] Hinton, G. E., & Salakhutdinov, R. R. (2006). reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504-507.

[31] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, *19*.

[32] Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning-ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21* (pp. 52-59). Springer Berlin Heidelberg.

[33] Ng, A., & Autoencoder, S. (2011). CS294A Lecture notes. *Dosegljivo: https://web. stanford. edu/class/cs294a/sparseAutoencoder_2011new. pdf.[Dostopano 20. 7. 2016]*.

[34] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008, July) Extracting and composing robust features with denoising autoencoders, In *Proceedings of the 25th international conference on Machine learning* (pp. 1096-1103).

[35] Kingma, D. P. (2013). auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[36] Sakurada, M., & Yairi, T. (2014, December). anomaly detection using autoencoders with nonlinear dimensionality reduction. in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis* (pp. 4-11).

[37] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). imagenet classification with deep convolutional neural networks. *advances in neural information processing systems*, *25*.

[38] Redmon, J. (2016). You only look once: unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[39] Ren, S. (2015). faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

[40] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). a convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

[41] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). a survey on deep learning in medical image analysis. *medical image analysis*, *42*, 60-88.

[42] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zieba, K. (2016). end to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[43] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). generative adversarial nets. *advances in neural information processing systems*, *27*.

[44] Simonyan, K., & Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, *27*.

[45] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). pytorch: an imperative style, high-performance deep learning library. *advances in neural information processing systems*, *32*.

[46] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (pp. 265-283).

[47] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). scikit-learn: machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

[48] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: state-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).

[49] Howard, J., & Gugger, S. (2020). fastai: a layered API for deep learning. *information*, *11*(2), 108.

[50] Devlin, J. (2018). bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[51] Yenduri, G., Ramalingam, M., Selvi, G. C., Supriya, Y., Srivastava, G., Maddikunta, P. K. R., ... & Gadekallu, T. R. (2024). gpt (generative pre-trained transformer)-a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions.*IEEE Access*.

[52] Griffin, G., Holub, A., & Perona, P. (2007). *caltech-256 object category dataset* (Vol. 10). pasadena: technical report 7694, California Institute of Technology.

[53] Zhang, C., Ranjan, S., & Hansen, J. H. (2018, June). an analysis of transfer learning for domain mismatched text-independent speaker verification. in *Odyssey* (pp. 181-186).

[54] Patel, C., Shah, D., & Patel, A. (2013). automatic number plate recognition system (anpr): a survey. *international journal of computer applications*, *69*(9).

[55] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P. A., & Bottou, L. (2010). Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, *11*(12).

[56] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). a simple framework for contrastive learning of visual representations. in *International conference on machine learning* (pp. 1597- 1607). pMLR.

[57] Wu, Z., Xiong, Y., Yu, S. X., & Lin, D. (2018). Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3733-3742).

[58] Blog, A. K. (2015). The unreasonable effectiveness of recurrent neural networks. *URL: http://karpathy. github. io/2015/05/21/rnn-effectiveness/dated May*, *21*, 31.

[59] Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). Beyond short snippets: deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4694-4702).