UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Object Detection vs. Semantic Segmentation on Fashion RGB Images

---

Óscar Sementé Solà

Director: Meysam Madadi
Realitzat a: Departament de Matemàtiques i Informàtica
Barcelona, 13 de juny de 2022

Abstract

People have loved clothing fashion for thousands of years, from the early days of Egypt until nowadays. Throughout history, drawings, documents, and other archaeological finds have also revealed that people wore fashion in different moments of history.

As an example, we have had various civilizations. The Greeks wore thick woolen long dresses. The ancient Egyptians were typically dressed in light cotton clothing. The Romans became the most critical example of style and fashion because of their expansion and dominance.

In the recent past, the fashion industry has emerged as one of the crucial industries for the global economy. The trends change every second, the clothing industry has proved itself one of the most creative realms, and with the advent of the internet and handheld devices, customers can easily shop on the go.

While people keep up with fashion trends, machine learning is changing the trends in the fashion industry, and daily there are systems keeping track of every sale and the upcoming trends. This gives the companies vast knowledge about what a user is interested in.

Seeing how many opportunities there are, I want to participate and try to develop a neural network in charge of categorizing each clothing it sees. Therefore, we need to use deep learning and understand how it works.

The concept of deep learning started in 1943 when Warren McCulloch and Walter Pitts created a computer model based on the human brain's neural networks. They used a combination of mathematics and threshold logic algorithms to mimic the thought process.

Since then, deep learning has evolved steadily, with two significant developmental breaks over the years. The progress of the basics of a continuous Back Propagation Model by Henry J. Kelley in 1960, and when Stuart Dreyfus came up with a simpler version based only on the chain rule in 1962.

Now, it is an important topic. Scientists use deep learning algorithms with multiple processing layers to make better models capable of understanding large quantities of unlabelled data, such as photos with no description, voice recordings, etc.

We want to use those algorithms, especially Object Detection and Semantic Segmentation, to start a project where we want to detect different pieces of clothing in a large dataset established by Fashion RGB Images.

The purpose is to carry out a study on both implementations. We want to train a model several times with different parameters and datasets, trying to achieve the most optimal results from both.

Once we have the results, we will compare them to see which is best for our case and study why it is the best.

Index

1 Introduction

My first time working on machine learning was in my third year at university in the artificial vision subject. During that year, I discovered different ways to process images and some basic implementations to recognize objects into them using various techniques.

I remember thinking that it was instrumental in business because there were many possibilities to use it. Since then, I have always been interested and wanted to know how it worked more deeply.

Today I still think the same. Therefore, I believe that joining deep learning and fashion is a great combination. There are a lot of possibilities to take advantage of and use it to improve sales, predict new trends, or know what people want at all times. Thanks to this project, I can start to learn everything that comes with it and maybe, expand it to achieve one of those possibilities I mentioned before in the future.

Nowadays, machine learning is completely changing the trends in the fashion industry. Every brand uses machine-learning techniques to increase customers and stay ahead of the movement.

People are into fashion and want to know what looks best and how they can improve their style and elevate their personality. Using Deep learning technology and infusing it with Computer Vision techniques, one can do so by utilizing Brain-inspired Neural Networks. Training them, playing around with Unstructured Data, and investing in the transformer architecture are just some highlights that can be touched in the Fashion domain.

This project will study different Deep Learning algorithms, primarily Object Detection and Semantic Segmentation, to train a model to detect clothing items by itself on Fashion RGB Images. To make this possible, we have to do different tasks.

First, we have to research to understand the basics of deep learning and how to implement it. We will do experiments with Cifar trying multiple structures, adding data augmentation, or changing parameters to improve the results.

After that, once we have the basic knowledge, we have to search for an extensive fashion dataset with a considerable number of images as well as the data they imply, besides

developing the corresponding functions on each dataset to extract all the information that we have talked about and display it in the requested format.

Then, look for Object Detection and Semantic Segmentation implementations that fit our purpose, adapt them for proper functioning, and start to train our model.

Once we arrive here, we have to train our model multiple times because, in each training, we will play with different variables to achieve the most optimal outcome of both methods. This phase will be the longest part of the project due to the big data, considering that it has to process many images once and again to improve its skills.

Finally, once we have our trained models, we will check and compare the results to make a study and discover the most optimal way to train our fashion model.

2 Objectives

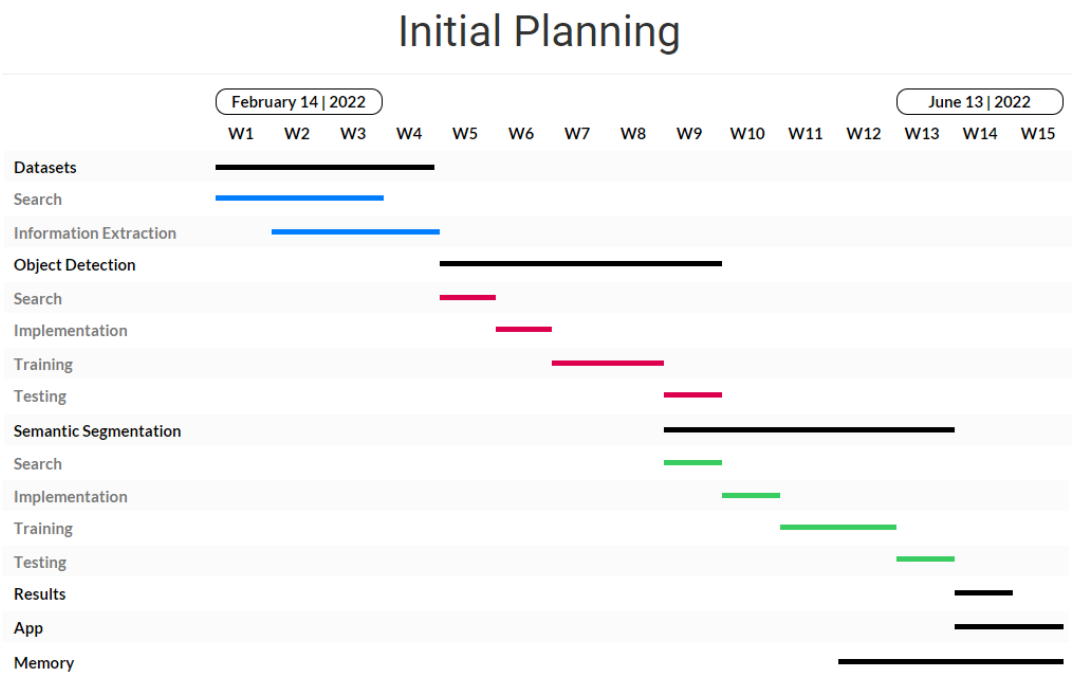
This section will define the main objectives the TFG has attempted to address since the beginning, regardless of whether the results obtained may or may not be the expected ones. You can see which they were below:

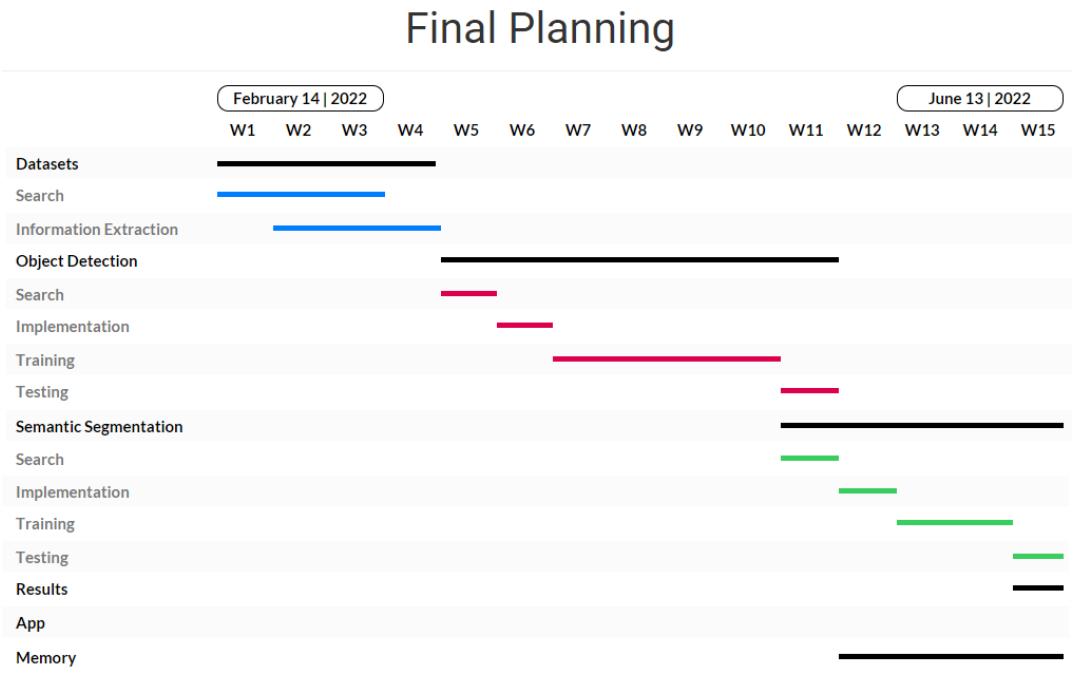
1. Search for a database that fits the requisites.
2. Extract the required information from datasets.
3. Implement the Object Detection algorithm using the dataset from task one.
4. Implement the Semantic Segmentation algorithm using the dataset from task one.
5. Train and test both implementations using multiple setups.
6. Compare and study the results from both algorithms.
7. Write a Memory to explain the process and the methodologies used in the project.

These are the main functions established initially. In addition, in case of having enough time, the idea was to develop an application that could be fed with images and returns the objects detected with the selected method.

3 Planning

Next, we have two Gantt diagrams from February 14th to June 13th. The first one displays the initial planning and the second one how it ended. They have principal and secondary tasks with the corresponding time needed to finish them. As you can see, it took more time than expected to complete some objectives, and in the end, the app failed.





4 Theoretical Background

Throughout the project, we will use general concepts, such as deep learning or convolutional neural network, which are essential to know and how they work. Therefore, we will describe and explain all those concepts to understand them better and not get lost in some parts of the project.

4.1 Deep Learning

Deep Learning is a subset of Machine Learning that uses mathematical functions to map the input to the output. These functions extract non-redundant information or patterns from the data, enabling them to form a relationship between the input and the result. This is known as learning, and the learning process is called training.

To grasp the idea of deep learning, imagine an infant and parents. The child points to objects with his finger and calls him ‘dog.’ The parents tell him ‘Yes, that is a Dog’ or ‘No, that is not a dog.’ The kid persists in pointing at objects but becomes more accurate with ‘dogs.’ The kid does not know why he can say it is a dog or not. He has just learned how to classify complex features coming up with a dog by looking at it and focusing on details such as the tails or the nose before making up his mind.

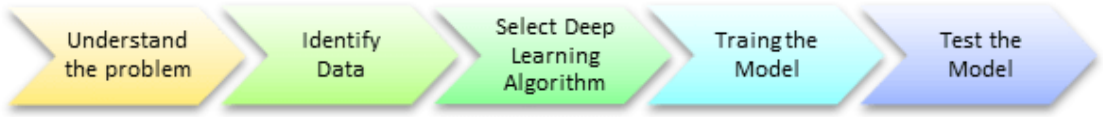


Figure 1: Deep Learning Process

Modern deep learning models use neural networks inspired by the human brain to extract information. The neural networks are the key to deep learning and are designed to mimic the working of neurons in the human brain. Next, we have an example of a neural network neuron.

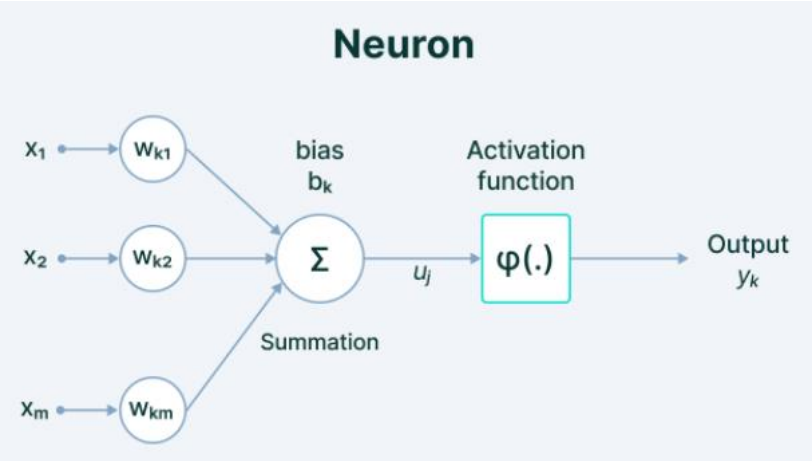


Figure 2: Example of a Neuron from a Neural Network

These analyze and cumulate insights from the data and later learn from the same. Any deep learning algorithm reiterates and repeatedly performs a task, tweaking and improving a bit every time to improve the outcome.

When we speak about neural networks, we can say there are several types: Feed Forward, Recurrent Neural Networks, Generative Adversarial Networks, etc.

4.2 Convolutional Neural Network

A Convolutional Neural Network or CNN is a Deep Learning algorithm that can take an image as an input, assign importance (learnable weights and biases) to various objects in the image, and differentiate one from the other. Also, the pre-processing required in a CNN is much lower than the others.

Four layers form a Convolutional Network:

1. Input Layer – Input Image.
2. Convolutional layer – Kernel.
3. Pooling layer.
4. Classification – Fully connected layer.

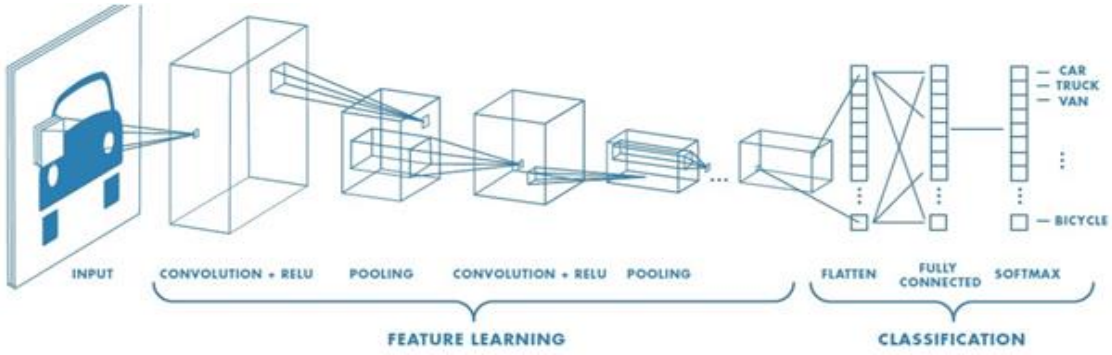


Figure 3: Process of Convolutional Neural Network

4.2.1 Input Layer

The input layer is the first layer of the CNN. It receives an RGB image with a specific shape (width, height, channels).

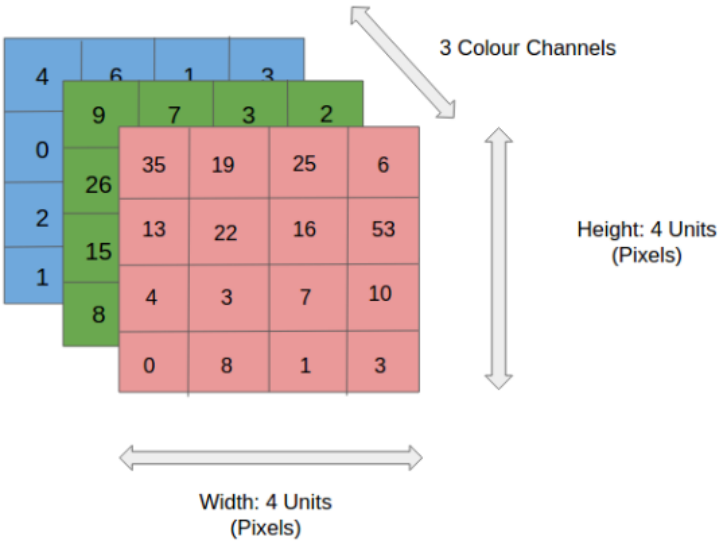


Figure 4: Example of Input Image

Depending on the image's dimensions, computationally, it can increase exponentially due to the large amount of data it can receive. For that reason, the role of the convolutional layer is to reduce the images into a form that is easier to process without losing critical features. It is crucial to take care of when we want to design an architecture that is good at learning features and scalable to massive datasets.

4.2.2 Convolutional Layer

The next layer is the convolutional layer. As its name suggests, it is in charge of carrying out a convolution operation that shifts a kernel through the input image x times depending on the stride value. The layer will make a matrix multiplication between the filter and the input portion on each shift.

For every shift, all the results will be summed with the bias to give us, in the end, a Convoluted Feature Output, also known as a Feature Map. The number of filters we use depends on the depth of the input.

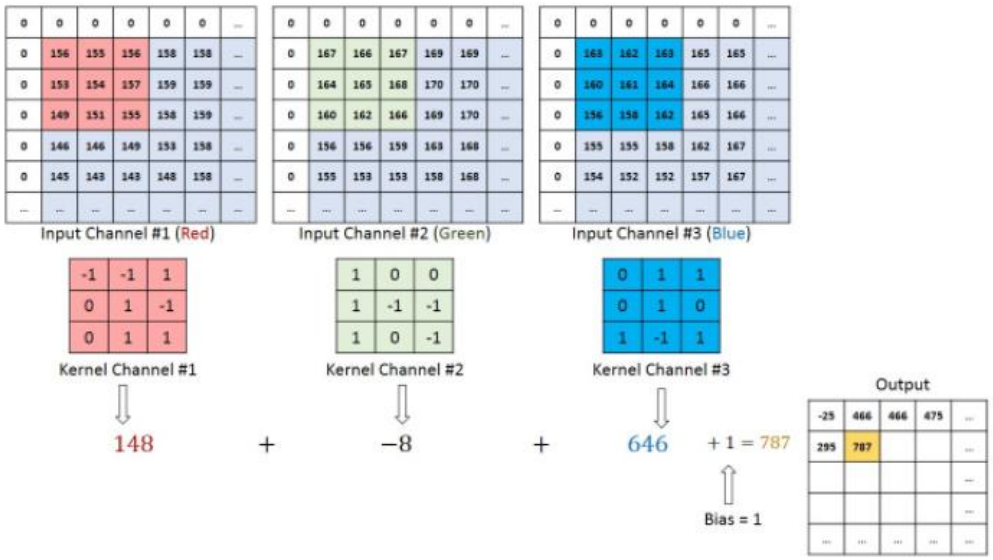


Figure 5: Convolution process

This layer aims to extract high-level features from the image. Convolutional layers are not limited to just one. Usually, the first one is responsible for capturing low-level features like edges, colors, etc. The architecture also adapts to high-level features adding layers, giving us a network capable of understanding images as humans.

Moreover, the convolution results depend on the padding used:

- Valid padding: The image is not padded, and filters always stay inside the image. There can be a loss of information, and as a result, the convolved feature is reduced in dimensionality compared to the input.

- Same Padding: The image is half padded. This technique ensures that the filter is applied to all the input elements and no information is lost. As a result, the convolved input increases or remains the same in dimensionality compared to the input.

4.2.3 Pooling layer

The pooling layer is similar to the convolutional layer because it performs downsampling to reduce the spatial dimensionality of the convolved feature and decrease the computational power required to process the data, the learning time, and the likelihood of overfitting. Additionally, it works with a filter that shifts across the input image x times based on a stride value as the convolutional layer.

There are two types of pooling layers:

- Max Pooling: Returns the maximum value from the image portion covered by the kernel. In addition, it is a noise suppressant.
- Average Pooling: Returns the average of all values from the image portion covered by the kernel.

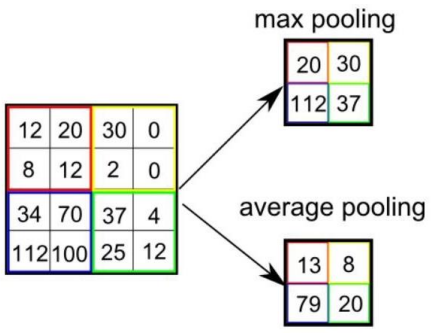


Figure 6: Examples of Max Pooling and Average Pooling

The convolutional and the pooling layers form the i-th layer of a convolutional neural network. If we want more accuracy in detecting low-level details, we can increase its number, but at the cost of more computational power.

Up to now, our model will be able to understand the features of an input image.

4.2.4 Fully connected layer

The Fully Connected Layer is the last one of our CNN. That layer enables our model to learn non-linear combinations of the high-level features represented by the output of the convolutional layer in a cheap way.

The first step is to flatten the image into a column vector. The flattened output will be fed to a feed-forward neural network, and backpropagation will be applied at every learning iteration. After some epochs, our model will be able to detect features in the image and classify them with some classification techniques.

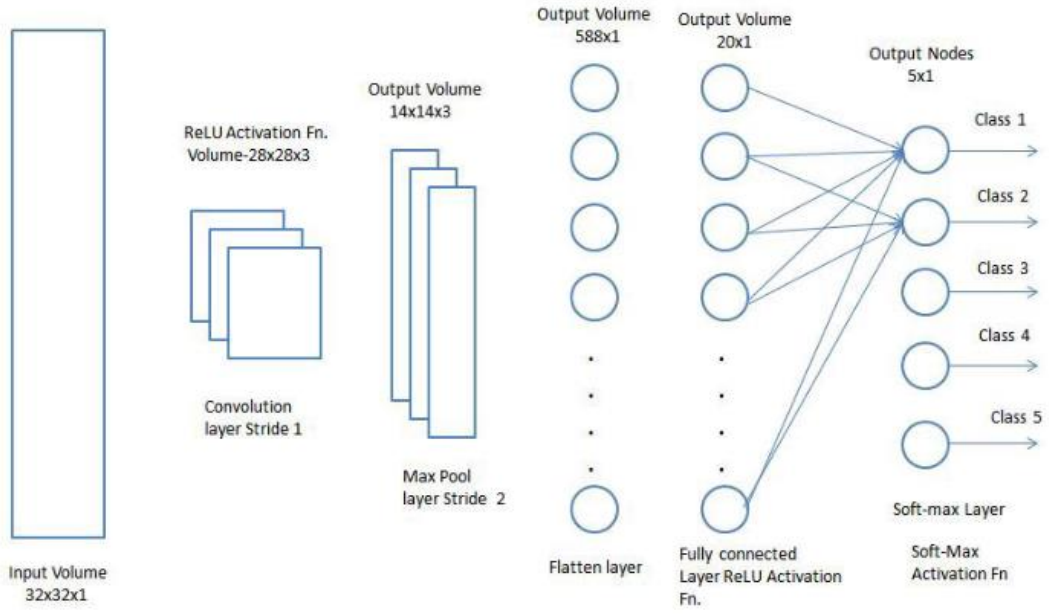


Figure 7: Example of Fully Connected Layer

4.3 Activation Function

The activation function is performed in each node of every convolutional layer of the CNN and decides whether a neuron should be activated or not. We speak of an activated neuron when the information is relevant enough to pass to the next layer. Moreover, depending on the selected activation function, the capability and performance of the neural network can be affected.

Many different activation functions are used in neural networks, although only a small number are used in practice. Here are three of the most commonly used activation functions:

- Rectified Linear Unit (ReLU).
- Logistic (Sigmoid).
- Hyperbolic Tangent (Tanh).

4.4 Loss Function

The loss function computes the distance between the current output of the algorithm to the expected outcome and allows us to evaluate how the algorithm models the data. Furthermore, we can derive the gradients from this function to upgrade the weights.

In conclusion, the loss function helps us adjust the weight values on each training epoch, finding the best parameters combination to perform good predictions.

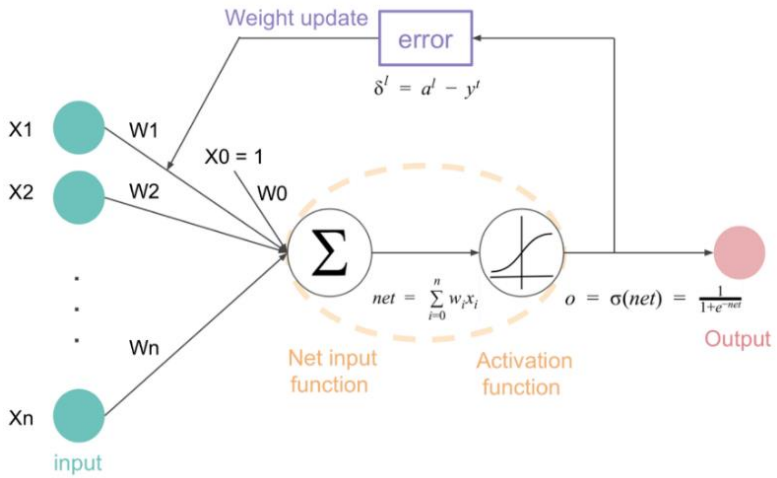


Figure 8: Example of Loss Function

Next, you can see some of the most commonly used loss functions:

- Cross-entropy.
- Log loss.
- Exponential Loss.

4.5 Overfitting and Underfitting

On the one hand, overfitting appears when our model performs well for training data but poorly for testing data. This may be due to a small training dataset, a complex model, or dirty training data containing noise. In addition, if we train the model for too long, it can learn unnecessary details and noise from the data.

On the other hand, underfitting appears when our model has not learned the patterns during the training and makes unreliable predictions. This may be due to noisy data, a high bias, a small training dataset, or a simple model.

We need to check the training and testing set performance to verify our model is well fitted. The objective is to simultaneously decrease the training and testing data error as much as possible.

To achieve this, we can adapt our model based on the problems we have observed above, for example, cleaning the data to avoid noise, increasing or decreasing the dataset, etc. Also, we can use methods that help us, such as dropout.

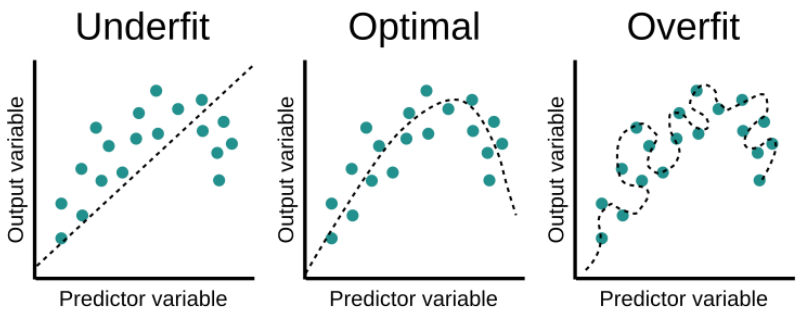


Figure 9: Model Underfitted, Well fitted, and Overfitted

4.6 Dropout

Dropout is a regularization method used to face the problem of overfitting in neural networks. It is in charge of randomly selecting neurons from a layer and ignoring them with their outputs during the training. Therefore, weight updates are not applied to the neuron on the backward pass.

The effect is that the network becomes less sensitive to the specific weights of neurons. This results in a network capable of better generalization and less likely to overfit the training data.

Next, we can see an example of a neural network with and without dropout and how the number of outputs varies in both:

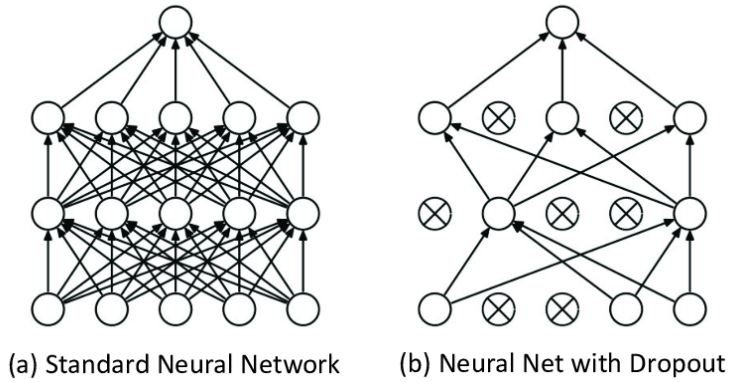


Figure 10: Neural Network with and without Dropout

4.7 Object Detection

Object detection is a general term to describe a collection of related computer vision tasks that involve identifying and locating objects in pictures or videos. Specifically, object detection draws a bounding box around these detected objects.

We have considered various options like Faster R-CNN and YOLO to carry out this task.

On the one hand, YOLO only looks at the image once to detect what objects are present and where they are. Hence the name You Only Look Once.

It can do this because it divides the image into an SxS grid cell. Each grid cell predicts x bounding boxes and their confidence scores. The confidence score indicates how sure the model is that the bounding box contains an object and how accurate it thinks the box is that predicts.

On the other hand, Faster R-CNN is a faster extension of Fast R-CNN due to the region proposal network or RPN. This region proposal network is a fully convolutional network that generates proposals with various scales and aspect ratios.

The process of Faster R-CNN is to generate region proposals with RPN. After that, from all the region proposals in the image, a fixed-length feature vector is extracted to be classified using the Fast R-CNN. Then, the class scores of the detected objects and their bounding boxes are returned.

Finally, we decided to use YOLO in our project because we found other works comparing these two implementations. The results showed that YOLO was slightly more accurate and faster.

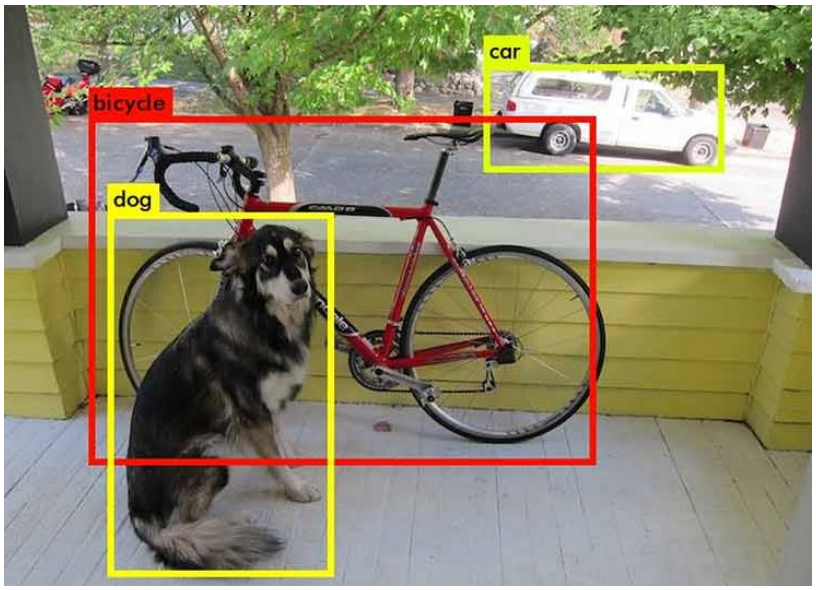


Figure 11: Example of Object Detection

4.8 Semantic Segmentation

Semantic Segmentation is a process of image segmentation that classifies each pixel in an image belonging to a specific class. In addition, another technique related to image segmentation is instance segmentation, where all the instances of an object are classified with a unique label.

We have considered different options such as Mask-RCNN and Unet to carry out this task.

On the one hand, Mask RCNN is an extension from the Faster R-CNN that adds a third branch that outputs the object mask. There are two stages of Mask RCNN. First, it generates region proposals where an object might be based on the input image. Second, it predicts the object's class, refines the bounding box, and generates a mask at the pixel level of the item based on the first stage proposal.

On the other hand, Unet handles the problem of segmenting objects at multiple scales with an encoder-decoder network. The encoder reduces the feature map to capture high semantic information, and the decoder slowly recovers the spatial data.

In the end, we decided to use Unet.

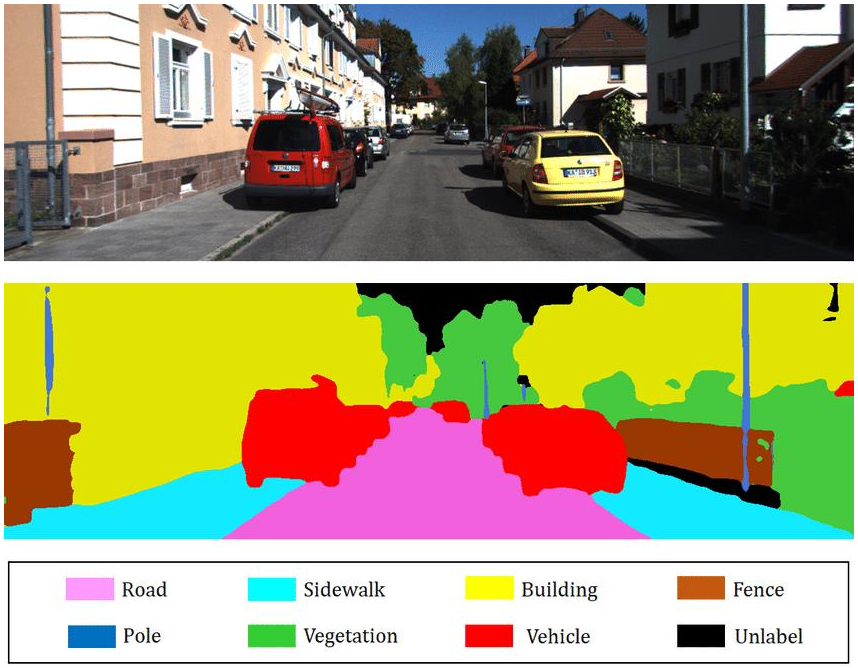


Figure 12: Example of Semantic Segmentation

5 Experimental Background

Before starting the project, we experimented with the Cifar database to do some implementations ourselves and have a first contact with the subject. This section will explain all these experiments and what we have learned from them that we will use later in the project.

5.1 Cifar

Cifar is one of the most widely used datasets to implement object detection and stands for Canadian Institute for Advanced Research. There are two variations from this dataset, CIFAR-10 and CIFAR-100. They are labeled subsets from the 80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Next, we will describe the characteristics of CIFAR-10:

- It consists of 60.000 images.
- Each image is 32x32x3.
- There are 50000 training images and 10000 test images.
- There are ten classes, and 6000 images represent each class.
- The classes are mutually exclusive (The same photo cannot belong to more than one class).

Below, we can see the dataset and the ten classes that make it up:

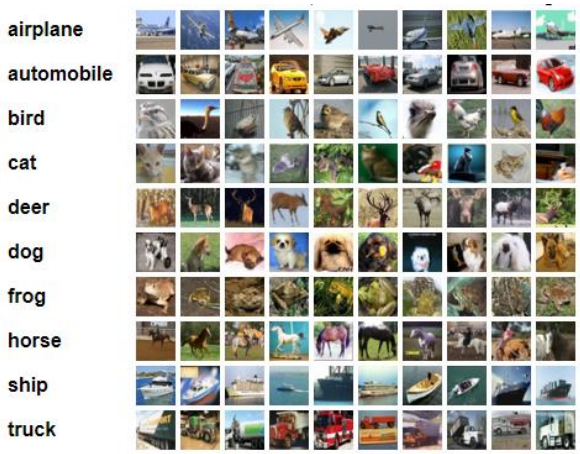


Figure 13: Cifar-10 Dataset

In addition to CIFAR-10, we also have CIFAR-100, with the same characteristics but much more challenging:

- It consists of 100 classes.
- Each class has 600 images.
- Twenty superclasses group all the classes.

Some examples of superclasses are flowers and insects, and its subclasses are orchids, poppies, roses, sunflowers, tulips for flowers, and bee, beetle, butterfly, caterpillar, and cockroach for insects.

These two datasets have been of great importance at the beginning of the project. We have used them to do experiments and learn how deep learning works.

5.2 Experiments

The first experiment was to try different ways of structuring the model. The idea was to add or remove layers from the model to see the impact on the results. Also, thanks to that, we learned the purpose of each layer.

Another experiment was about data augmentation. I had to develop different functions to create new images based on the originals but with a few modifications and, as a result, increase our dataset. Some of the augmentations we used are the next ones:

- Horizontal or vertical flip.
- Contrast.
- Rotation.
- Scale.
- Crop.
- Translation.

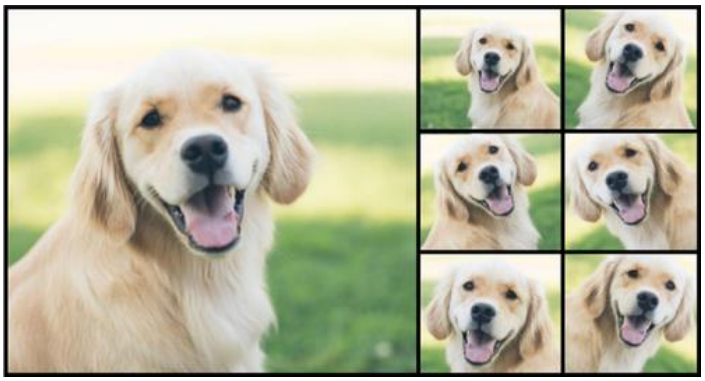


Figure 14: Examples of Data Augmentation.

The idea of this experiment was to train the model with no standard images because in real life, not all the pictures are centered, or we can see perfectly all its components. Therefore, we wanted to apply these modifications to prepare the model for all the possible situations.

Finally, the last experiment was about trying default models, such as ResNet or Inception, and seeing their performance.

On the one hand, ResNet stands for Residual Network and solves the problem of a maximum threshold for depth in CNN using residual blocks. The residual blocks have a direct connection, called skip connection, that skips some of the layers of the CNN.

Thanks to this skip connection, ResNet solves the vanishing gradient problem in deep neural networks and allows the model to learn the identity functions, ensuring that the higher layer will perform at least as well as the lower layer.

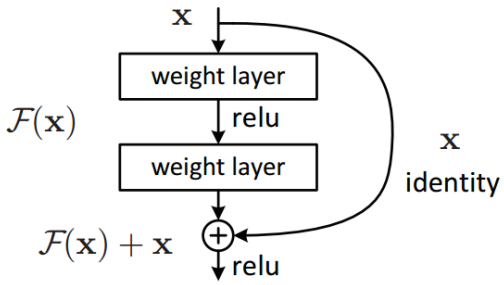


Figure 15: Residual Block

On the other hand, Inception is a deep neural network with an architectural design that consists of repeating components referred to as Inception modules.

There are different types of inception modules. An example is naïve, which performs convolution with three different sizes of filters and max pooling on the same level. The outputs are concatenated and sent to the next layer.

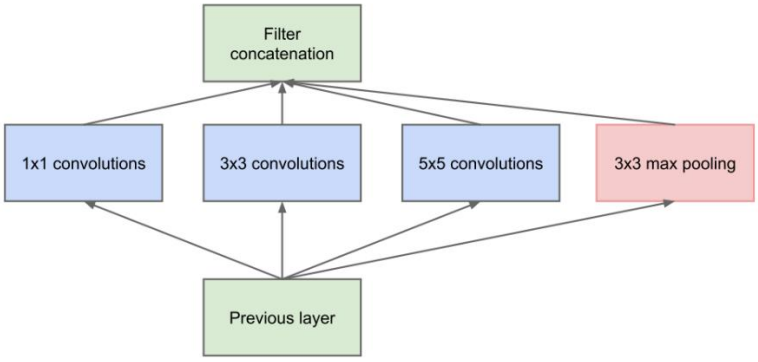


Figure 16: Inception Module

6 Methodology

This section will explain the algorithms we have used to implement object detection and semantic segmentation. These methods are Yolo and Unet, and we will show their network architecture and how they work step by step.

6.1 Yolo

Yolo is an object detection algorithm that applies a single forward pass neural network to the whole image and predicts the bounding boxes and their class probabilities. This technique makes YOLO a super-fast real-time object detection algorithm.

To aim this, it uses a 53 layers neural network called Darknet-53.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 17: Architecture of Darknet-53

6.1.1 Process

Yolo divides every input image into an SxS grid of cells, and each cell predicts B bounding boxes and C class probabilities of the objects whose centers fall inside the grid cells. The B refers to the number of using anchors, and each bounding box

has 5+C attributes. The first five attributes are from the bounding box: center coordinates, height, width, and confidence score. The last one, C, is the number of classes of the dataset.

After passing a cell into a forward pass convolution network, we will obtain a 3-D tensor that looks like [S, S, B*(5+C)]. Next, you can see an example of the process using the COCO dataset:

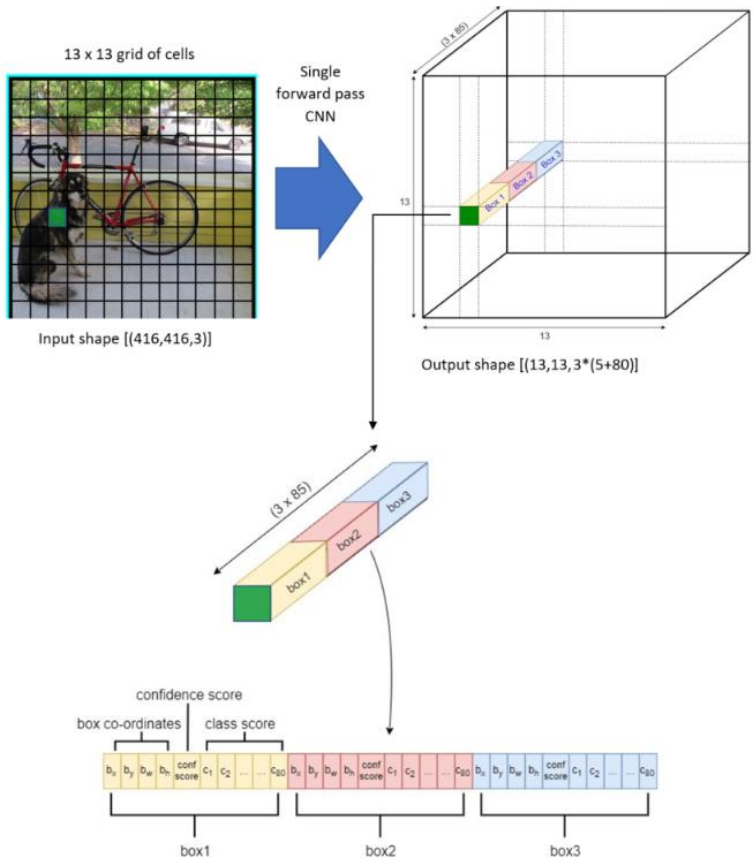


Figure 18: Yolo Process

6.1.2 Anchors

Yolo can only detect one object per grid cell, so it uses three different anchor boxes to see overlapping items on every detection scale, which means there are nine anchor boxes in total.

Each anchor is a set of pre-defined bounding boxes of a certain height and width used to capture the scale and different aspect ratios of specific object classes we want to detect.

These aspect ratios are defined beforehand. For this reason, to create these anchor boxes, it has been used the K-means clustering method over the entire dataset's bounding box shapes.

6.1.3 Scales

Yolo detects items in three scales to accommodate various object sizes using strides of 32, 16, and 8.

On the first scale, Yolo downsamples the image with a stride of 32 and predicts at layer 82.

After that, Yolo takes the feature map from layer 79 and applies a convolutional layer before upsampling it by a factor of two. Then, this upsampled feature map is concatenated with the feature map from layer 61. This concatenated feature map will be subjected to more convolutional layers until the second prediction at layer 94.

Finally, the previous process is repeated on the third scale. A convolutional layer is added to the feature map from layer 91, is upsampled and concatenated with the feature map from layer 36, and the predictions will be performed at layer 106.

The following figure contains the Yolo network architecture with all the steps we mentioned above:

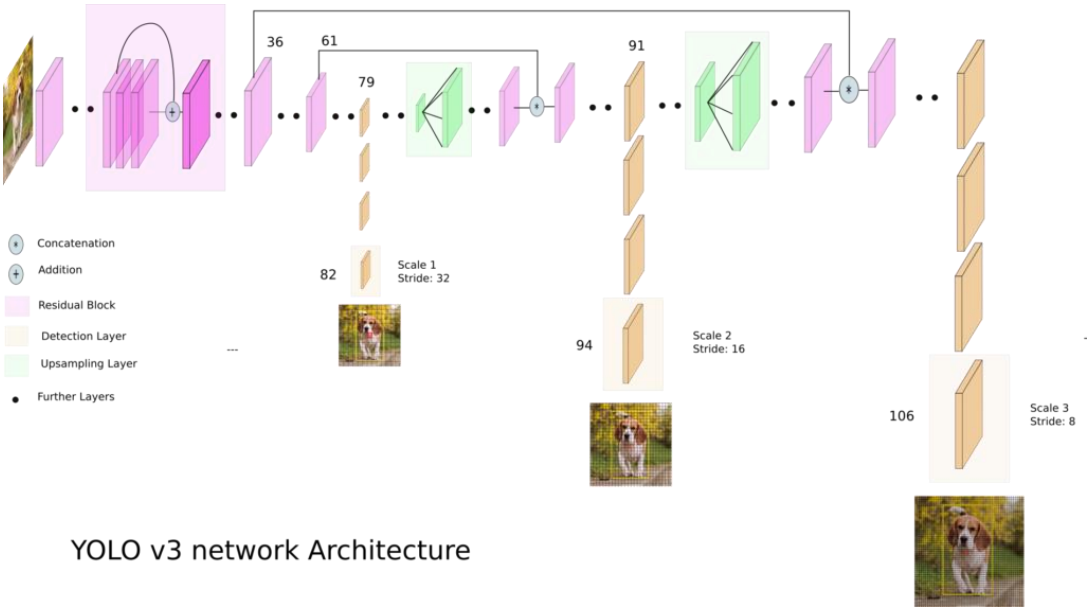


Figure 19: Yolo Network Architecture

6.1.4 Bounding Boxes Prediction

For each bounding box, Yolo predicts four parameters:

- The bounding box center coordinate relative to the grid cell whose center falls inside. (Tx, Ty)
- The width of the bounding box. (Tw)
- The height of the bounding box. (Th)

The output of the bounding box predictions is refined using the formula from figure 20, where Pw and Ph are the anchor's width and height. In addition, next to the procedure, there is an image that describes the transformation in more detail.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

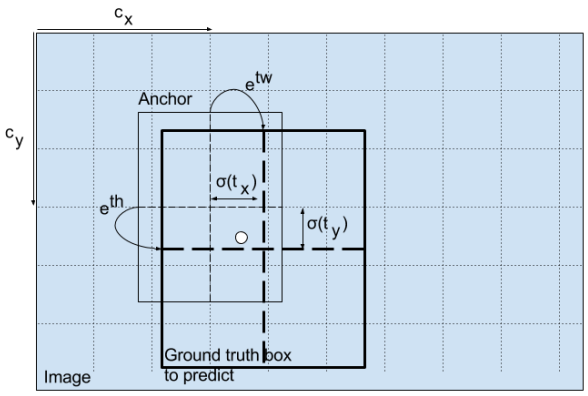


Figure 20: Example of Bounding Box Prediction

Once we have all the parameters, we use the top-left corner and the box width and height to draw the bounding box. The top-left corner is calculated based on previous parameters using the following formula:

$$x_1 = b_x - \frac{w}{2}$$

$$y_1 = b_y - \frac{h}{2}$$

Figure 21: Formula to calculate top-left corner

6.1.5 Non-Maximum Suppression

After processing the entire input image, Yolo probably will have predicted more than one bounding box for every object when we only wanted one. That is why is used the method of non-maximum suppression.

First, this method suppresses all the predicted bounding boxes with a confidence score under a specific threshold value.

After that, all the remaining confidence scores are sorted from the highest to the lowest, and the one with the best score is selected as the correct one. Then, we compute the Intersection Over Union between the chosen bounding box and the rest one by one. If the value surpasses a certain threshold, both bounding boxes are pretty oversampled, so the second one must be removed because it is pointing to the same object.

Figure 22: Example of Non-Maximum Suppression

6.2 Unet

Unet is one of the most well-recognized image segmentation algorithms. It was initially invented and first used for biomedical image segmentation, and its architecture can be broadly thought of as an encoder network followed by a decoder network.

6.2.1 Process

Unet uses an Encoder-Decoder Network. The encoder is in charge of gradually reducing the feature maps and capturing higher semantic information. The decoder, however, slowly recovers the spatial data.

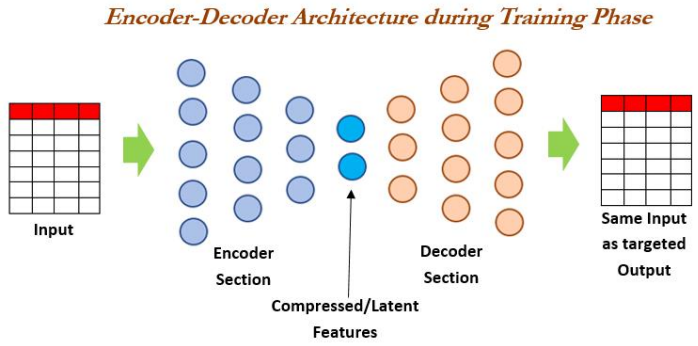


Figure 23: Encoder-Decoder Network

On the encoder side, Unet uses a pre-trained classification network that acts as a feature extractor. It is in charge of repeatedly applying two 3x3 convolution blocks, each followed by the activation function ReLU and a 2x2 max pooling operation with stride two for downsampling. At each downsampling step, the number of feature channels is duplicated. In conclusion, it encodes the input image into feature representations at multiple levels.

On the decoder side, Unet aims to semantically project the discriminative features (lower resolution) learned by the encoder onto the pixel space (higher resolution) to get a dense classification. It consists of an upsampling of the feature map followed by a 2x2 up-convolution that halves the number of feature channels, a concatenation with the corresponding cropped feature map from the encoder, and the application of two 3x3 convolutions, each followed by an activation function ReLU.

The cropping is used due to the loss of border pixels in every convolution. Finally, in the last layer, a 1x1 convolution is used to map 64-component to the desired number of classes. In total, there are 23 convolutional layers.

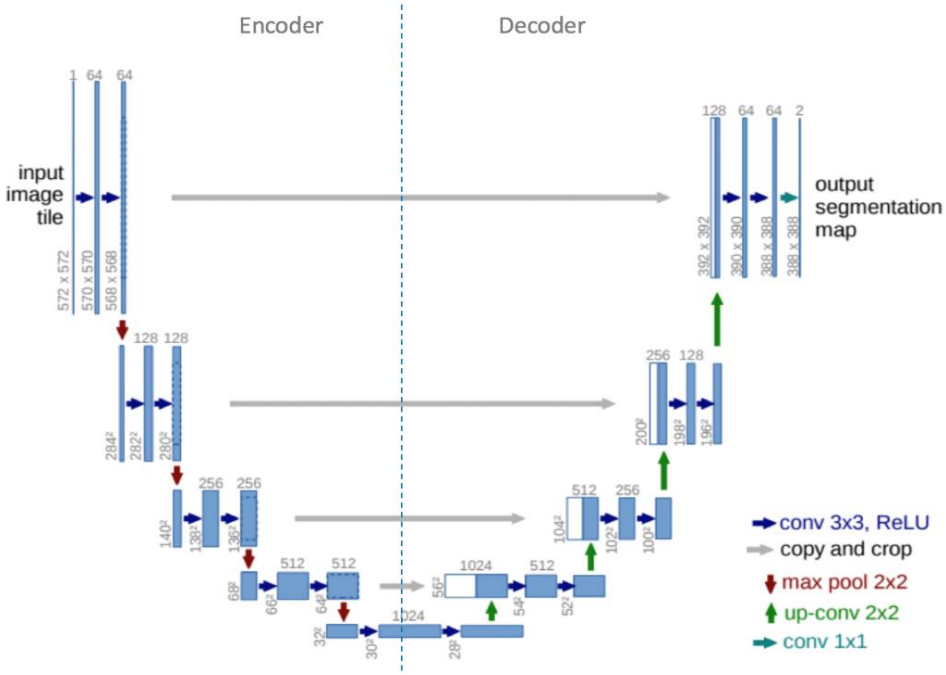
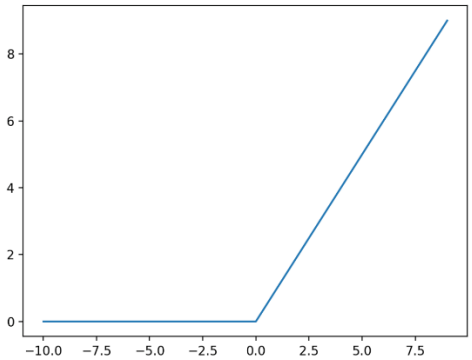


Figure 24: Unet Architecture. Blue boxes represent multi-channel feature maps, while white boxes represent copied ones.

Worth mentioning that ReLU is a type of activation function that is linear in the positive dimension but zero in the negative. In addition, the positive side's linearity prevents the non-saturation of gradients.



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

Figure 25: ReLU activation Function

7 Datasets

This is the first task we have to do to start our project, search for a dataset that fulfills all the requisites. As we mentioned previously, it is needed a dataset with fashion as the topic, a significant number of images, and as much information as possible about them, for example, the bounding boxes, the category of each one, the masks, etc. We have found three different datasets acceptable at first sight to our project:

1. Clothing Co-Parsing
2. DeepFashion
3. Fashionpedia

After selecting the dataset, we also have to develop a function to process all the information and put it in a specific format that we will explain now. Below, you can see an example of the structure to follow.

image_path x_min, y_min, x_max, y_max, class_id x_min, y_min... class_id

The first parameter, the image_path, refers to the local path where the image is located. The next four are coordinates (x, y) of a bounding box, the first two are the bottom left corner (x_min, y_min), and the last two are the top right corner (x_max, y_max). This bounding box covers an element of the image that our model should detect by itself in the future. The final parameter, the class_id, refers to the element's name that covers the bounding box expressed previously. Finally, these variables can be repeated as many times as the number of categorized features in the image.

Below, you can see an extract of the text document created:

```
Fashionpedia/train/90da28d23c2600b7845fbbcc050b852a.jpg 352,309,460,498,32
17,115,585,997,0
Fashionpedia/train/5355598ed3791edfa9c888bc42c21796.jpg 159,831,490,1021,6
```

Figure 26: Part of the Annotation document created

Next, we will describe all the three datasets independently, explaining the process we have developed for each of them to extract the information and which one we will select for the project.

7.1 Clothing Co-Parsing

Clothing Co-Parsing is a new fashion database including elaborately annotated clothing items. It is established with 2.098 high-resolution street fashion photos with 59 tags, where each image has a wide range of styles, accessories, garments, and poses. In addition, all the images have image-level annotations, and 1.000 have pixel-level annotations.

The dataset folder is structured as follows:

- Photos – Directory of original photos
- Annotations – Directory of annotations
    - Pixel-level – pixel-level annotations (1.004 files)
    - Image-level – image-level annotations (1.094 files)
- Label_list.mat – [1*59] cell array, which maps label numbers to label names
- README.md – File with information about the dataset.

Below, you can see some examples of the dataset:

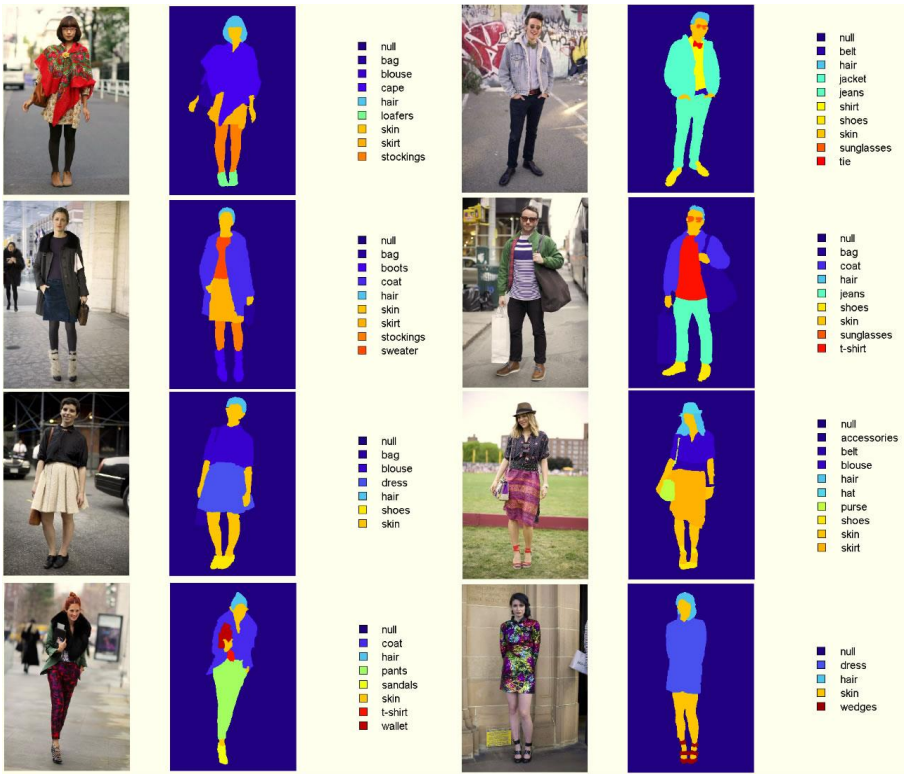


Figure 27: Clothing Co-Parsing Dataset

7.1.1 Information extraction

To extract all the corresponding information about the dataset, we have developed a process with python that we will explain now.

First, we must extract all image masks from the Pixel-level folder and the 59 labels from the file Label_list.mat. We will use the method loadmat from the package scipy that enables us to load the MatLab files.

As you can see below, a mask is an image where each component has the value of the correspondent label id. To grasp the idea, if the label id of the skirt is ten, then all the pixels into that clothing item will have that value:

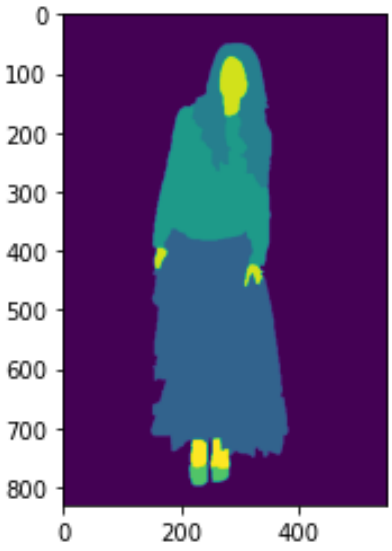


Figure 28: Example of an Image mask

After that, we have built a method to extract the categories from an input mask. The first step is to take the unique values of the input to collect the label Ids within the image. Then, we loop through all these Ids and assign them the category that coincides with the list's Ids. As a result, we will get a dictionary of all the labels of an input image. Next, you can see an example of an output:

```
{0: 'null', 2: 'bag', 3: 'belt', 5: 'blouse', 7: 'boots', 13: 'coat', 14: 'dress', 19: 'hair', 24: 'jacket', 29: 'necklace', 3
1: 'pants', 33: 'purse', 41: 'skin', 47: 'sunglasses'}
```

Figure 29: Dictionary with labels and ids of an image

Then, we have to create a text document where we will write all the information in the format we have explained at the beginning of this section. To achieve this, we have had to follow the following steps:

1. Create and open a text document.
2. Loop through all the masks.
3. Loop through all the labels obtained with the previous function for each mask.
4. For each label, make a copy of the mask and put all the pixel values within the category to 255 and the rest to 0.
5. With the method 'ConnecetedComponentsWithStats' from cv2, extract all the coordinates of the bounding boxes, the bottom left and top right corners, of all the components with a pixel value of 255.
6. Loop through all these components and write their coordinates and categories in the document we have created in the first step.
7. Close the text document once all the images and labels are processed.

Finally, we have recovered all the coordinates from the previous text document and drawn all the corresponding bounding boxes over the original images to verify that the results are correct. Below, you have an example of the result obtained:

Figure 30: Original and Recovered Image

7.1.2 Discussion

In conclusion, we can say that this database fits our requisites quite well. There are no irrelevant or incorrect labels, and the bounding boxes extracted from the masks are very accurate. There is only a problem with Clothing Co-Parsing. It does not have enough images to train the model and achieve optimal results.

For that reason, we have not selected Clothing Co-Parsing as our final database, but we have used it to do fast trains and see some results in a short time. We have done that to see which combination of parameters gives us the best results and then maybe accelerate the training process with the final database.

7.2 DeepFashion

DeepFashion is a large-scale clothes database. It contains over 800.000 diverse fashion images ranging from well-posed shop images to unconstrained consumer photos, constituting the largest visual fashion analysis database. In addition, it is annotated with information about the clothing items within the pictures. Each image in this dataset is labeled with 50 categories, 1.000 descriptive attributes, bounding boxes, and clothing landmarks.

Four benchmarks are developed using this database:

- Category and Attribute Prediction.
- Consumer-to-shop Clothes Retrieval.
- In-shop Clothes Retrieval.
- Landmark Detection.

We would use the Category and Attribute prediction benchmark from these four options. It is a large subset of DeepFashion that contains:

- 289.222 clothes images.
- 50 clothing categories and 1.000 clothing attributes.
- Each image is annotated with the bounding box and clothing type.

Figure 31: DeepFashion Dataset

7.2.1 Discussion

Contrary to Clothing-Co-Parsing, DeepFashion has a correct number of images to train the model. It would be a perfect database to start our training if it were not because we found that classes were mutually exclusive during the extraction process, meaning that we only have a bounding box per image.

Because of that, we stopped the development of the information extraction and concluded that DeepFashion was not the best database for our model. We prefer a database where we can train the model to detect multiple objects on a single image.

7.3 Fashionpedia

Fashionpedia is a dataset provided with many images of people wearing various clothing types in multiple poses. It contains over 45.000 images to train and 3.200 to test and validate.

This database project consists of two parts: an ontology built by fashion experts containing 27 main apparel categories, 19 apparel parts, 294 fine-grained attributes, and their relationships; and the second part is a dataset with fashion images annotated with segmentation masks and their associated per-mask fine-grained features, built upon the Fashionpedia ontology.

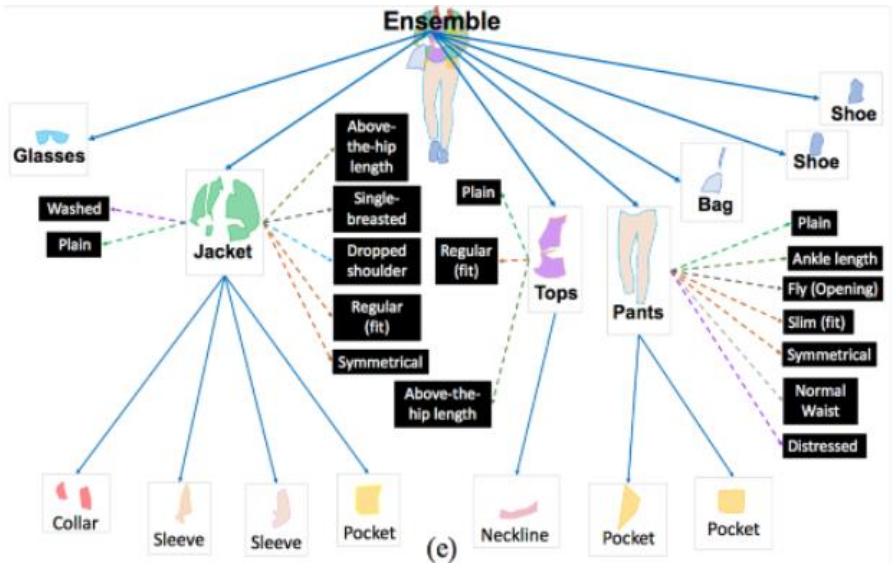


Figure 32: Fashionpedia Dataset

The dataset follows the annotation format of the COCO Dataset with additional fields, and the annotations are stored in the JSON format. Below, you can see some examples of how they are organized:

```
image{
  "id" : int,
  "width" : int,
  "height" : int,
  "file_name" : str,
  "license" : int,
  "time_captured": string,
  "original_url": string,
  "isstatic": int, 0: the original_url is not a static url,
  "kaggle_id": str,
}
```

```
annotation{
  "id" : int,
  "image_id" : int,
  "category_id" : int,
  "attribute_ids": [int],
  "segmentation" : [polygon] or [rle]
  "bbox" : [x,y,width,height], # int
  "area" : int
  "iscrowd": int (1 or 0)
}
```

Figure 33: Examples of Fashionpedia JSON format

7.3.1 Information extraction

To extract all the corresponding information about the dataset, we have developed a process with python that we will explain now.

First, we must consider that this dataset has subcategories such as sleeves, pockets, neckline, etc. These subcategories can confuse our model during the training, so we will not include the corresponding annotations that contain these labels in the text document.

Considering this, the first step is to extract the JSON file information. We will use the method loads from the python package JSON and put all the information we need into a pandas dataframe.

As you can see below, we have an example of the resulting dataframe with the information we need. The image id, the coordinates of the bounding box, the category id, and the path where the image is located.

	Id	x1	y1	x2	y2	Category	Path
0	16305	454.0	390.0	46.0	109.0	32	Fashionpedia/train/2f10b4d61e445fe5d0777723630...
1	16305	271.0	133.0	98.0	223.0	31	Fashionpedia/train/2f10b4d61e445fe5d0777723630...

Figure 34: Result dataframe from JSON file

Once we have extracted all the data, we can create the text document by following these steps:

1. Create and open a text document.
2. Create an empty list where we will add each image processed to avoid duplications.
3. Create a list of category Ids to refuse.
4. Loop through all the rows of the dataframe.
5. For each image id not in the list of processed images, create a sub-dataframe that only contains its annotations.
6. Loop through the annotations of each image id.
7. Add the annotation information to the text document only if its category is not in the list of categories to refuse.
8. Close and save the document once all the images and annotations have been processed.

Finally, we have recovered all the coordinates from the previous text document and drawn all the corresponding bounding boxes over the original images to verify that the results are correct.

Below, you have an example of the results obtained:

Figure 35: Original and Recovered Image

7.3.2 Discussion

After processing all the images and information from the dataset, we can say that this database fits perfectly to train our model. It contains many pictures with various poses and styles for training and testing. In addition, many image data are related to object detection and image segmentation.

As we can see in the previous images, the data is accurate, so our model will make reasonably accurate predictions. In addition, thanks to the COCO Dataset format on the annotations, we can easily access the information.

The only disadvantage is that it contains some subcategories that can affect the model eventually. Still, there is no problem because we can control this issue by excluding these subcategories while extracting the information.

Therefore, Fashionpedia will be the dataset selected to train and test the model during this project.

8 Tests, evaluation, and results

This section will show all the tests we have done throughout the project and the obtained results on both implementations. First, we will describe all the modifications we have made to optimize our model on its predictions, and after that, we will display some of the results to comment on them.

8.1 Object detection

This implementation aims to predict where the objects are located and their label. To get these predictions, we have to recover the checkpoint with the minor test loss saved during the training and convert it into a pb file using the function freeze_ckpt_to_pb provided in the code.

We must remember that we have trained our model with a variation of the Fashionpedia database where all the irrelevant subcategories, such as sleeves, pockets, and others, are ignored to avoid possible confusion by the model.

Finally, we have trained and tested our model in multiple ways, which we will explain in the following sections with their corresponding setup.

8.1.1 Training Setup

Next, we will explain the different variables we have modified during the project to obtain the actual models. We will start with the training variables.

First, The GPU is our Graphics Processing Unit, and thanks to it, we have accelerated the training and testing process because it allowed us to process multiple images at once. With a GPU worse than the one used, we probably would not have finished the project because of time.

The selected network in all the models is Darknet-53. It is a CNN that is 53 layers deep and acts as a backbone for the YOLOv3 object detection approach.

The training sessions have been organized in epochs. We process all the images from the corresponding dataset, training or testing, for each period. In addition, we have the batch

size, which is the number of samples that will be propagated through the network at one on every step of the epoch.

The IoU Loss Threshold is a parameter that teaches our model when a predicted bounding box is correct or not. If the IoU Loss is less than the value specified, the bounding box will be marked as wrong; otherwise, the bounding box will be marked as valid. The IoU Loss is an evaluation metric that computes how overlapped are the original bounding box and the predicted one. The higher the values, the better results and vice versa.

The learning rate is a parameter of the neural networks between 0.0 and 1.0 that controls how quickly the model is adapted to a problem. For higher values, we will need fewer epochs and vice versa. The weights' amount is updated during training is referred to as the step size or the learning rate.

The Data Augmentation variable is a Boolean. If true, the dataset will be enlarged by augmentations functions; otherwise, it will stay normal.

The anchor variable is the path where the document with the anchors we want to use is located.

Finally, the checkpoint variable is the path where the checkpoint we want to use is located. If there are not any checkpoints, it means that we are working from scratch. Otherwise, we will use pre-trained weights from our model, which needs more training, or other people's models. The use of this variable is also called finetuning.

8.1.2 Testing Setup

The following parameters we are going to explain are from the test configuration.

The IoU threshold is used for non-maximum suppression. It is a technique that selects the bounding box with the best confidence and suppresses the others when the IoU Loss is greater than the threshold established. This avoids having more than one bounding box recovering the same object.

Finally, the Score threshold is used to only accept bounding boxes with a confidence equal to or higher than the specified value.

8.1.3 Model I

First, we will summarize all the configurations we have used to train and test the model.

TRAIN CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Network	Darknet-53
Nº epochs	50
Batch size	2
IOU Loss Threshold	0.5
Initial Learning Rate	1e-4
Data Augmentation	True
Anchors	Own anchors
Checkpoint	None

TEST CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Batch size	1
IOU Threshold	0.3
Score Threshold	0.6

With this configuration, the best test loss obtained is 7,37%. Next, you can see a graph with the evolution of the test loss over the epochs:

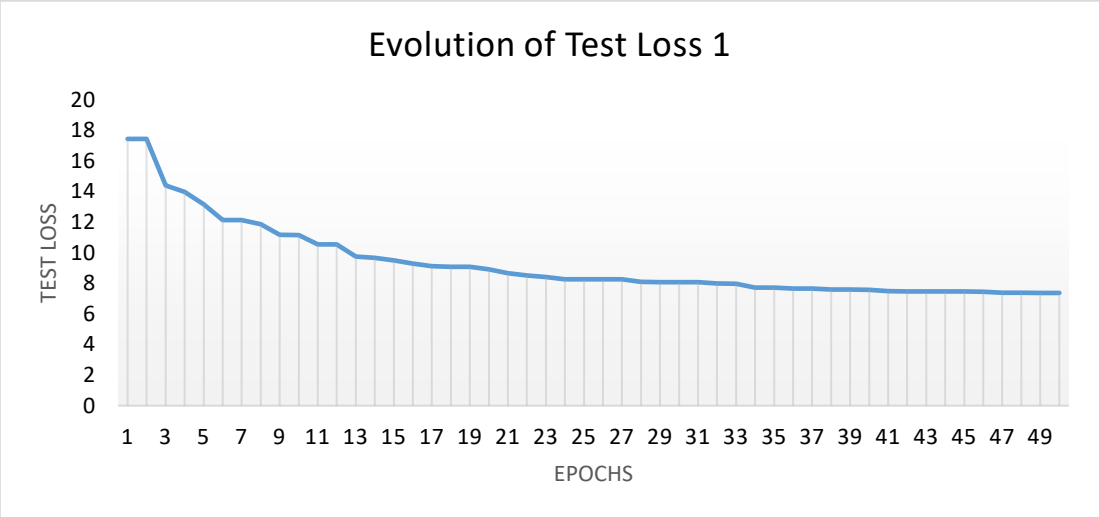
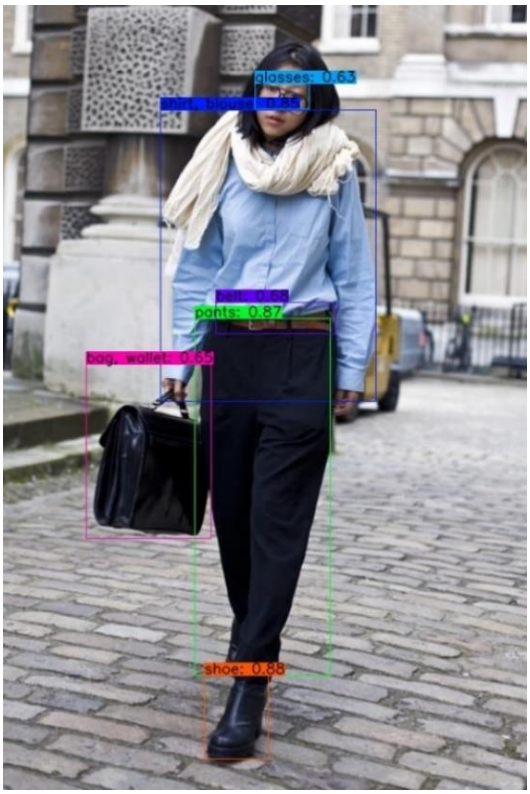


Figure 36: Evolution of the Model I Test Loss

We can see that the test loss decreases exponentially during the first steps, but it becomes more difficult to reduce in the end.

Below, you can see some of the results obtained from Model I:



As we can see, the results are incredible. The model can detect almost all the clothing items from the image with precise bounding boxes and high confidence. We can say that this model is qualified to do its job with quite reliable predictions.

8.1.4 Model II

As before, we will summarize all the configurations we have used to train and test the model.

TRAIN CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Network	Darknet-53
Nº epochs	50
Batch size	2
IOU Loss Threshold	0.5
Initial Learning Rate	1e-4
Data Augmentation	True
Anchors	Default anchors
Checkpoint	None

TEST CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Batch size	1
IOU Threshold	0.3
Score Threshold	0.6

With this configuration, the best test loss we have obtained is 10,44%. The only change we have made in this model is the anchors' file. In this case, we have used the anchors from the COCO Dataset provided by the code.

The first change we can observe is the test loss. Compared with Model I, the test loss obtained in Model II is 3.09 points higher. Also, as shown in figure 37, these 3-4 points of difference seem to be maintained during all training.

This loss in accuracy is expected because we have used anchors with predefined bounding boxes from another model where the classes and boxes' ratios are not the same as our database. So probably, the predictions will get worse on Model II.

Next, you can see a graph with the evolution of the test loss from both presented models over the epochs:

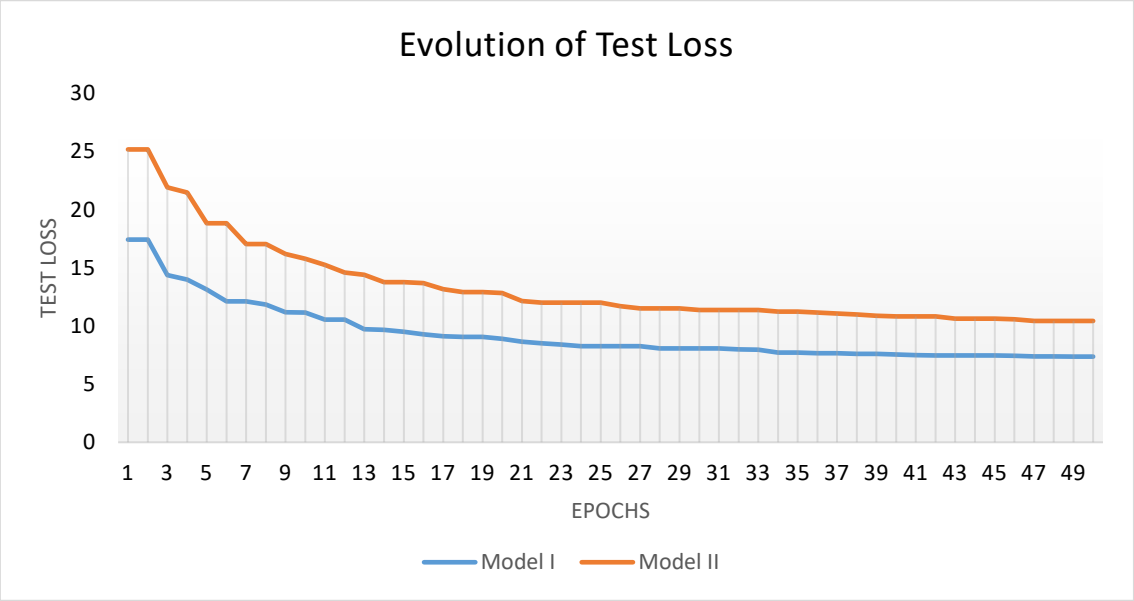
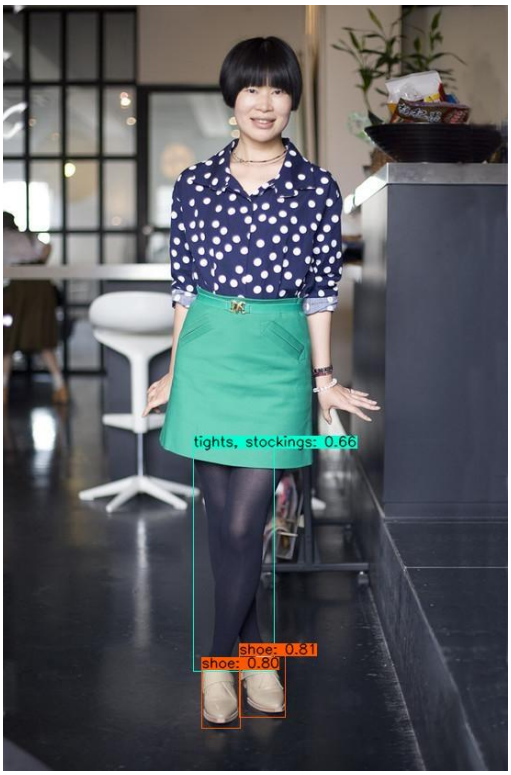


Figure 37: Evolution of the Model I and II Test Loss

Below, you can see some of the results obtained from Model II:



As we can see, the results are worse than Model I but are acceptable. Some clothing items are not detected, but the bounding boxes of the detected objects are generally precise and accurate.

8.1.5 Model III

As always, we will summarize all the configurations we have used to train and test the model.

TRAIN CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Network	Darknet-53
Nº epochs	50
Batch size	2
IOU Loss Threshold	0.5
Initial Learning Rate	1e-4
Data Augmentation	False
Anchors	Own anchors
Checkpoint	None

TEST CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Batch size	1
IOU Threshold	0.3
Score Threshold	0.6

With this configuration, the best test loss we have obtained is 8,07%. In this case, we have trained the model without data augmentation, which means we have trained with the default dataset size.

The first change we can observe is the test loss. The result is between both previous models, 2.36 points under Model II and 0.70 points over Model I. Also, as you will see in figure 38, the test loss at the beginning is even better than Model I, but eventually, it gets static while Model I keeps reducing it little by little. Still, it is better than Model II during all the training sessions.

Next, you can see a graph with the evolution of the test loss from all the presented models over the epochs:

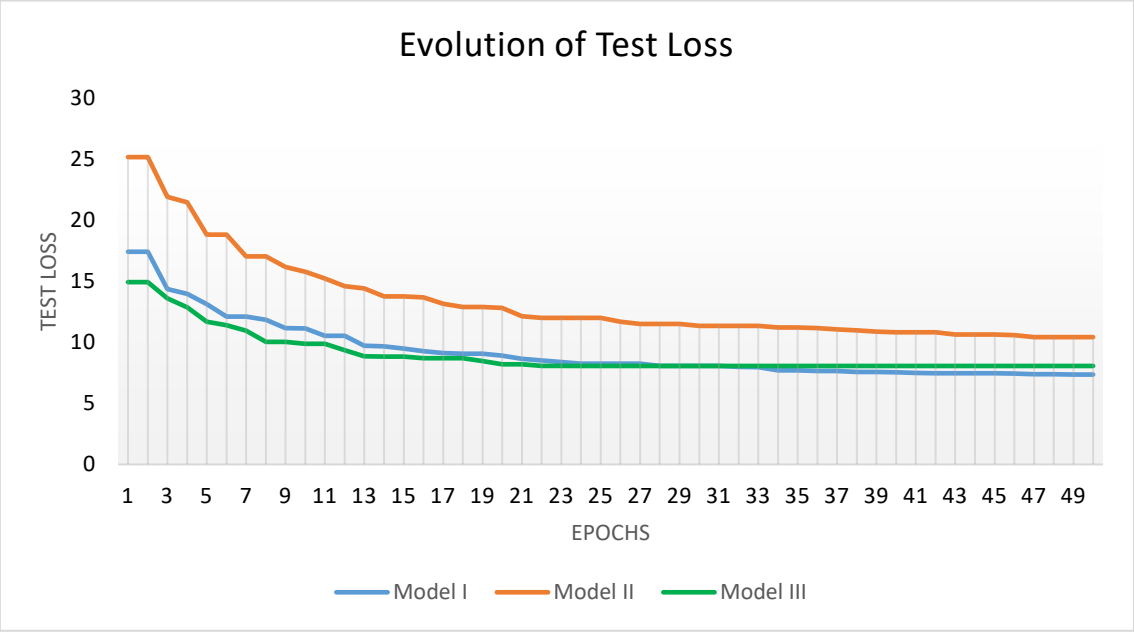


Figure 38: Evolution of all the Models Test Loss

Below, we will display some of the results obtained with Model III:

As we can see, the results are similar to Model I, so we can say that Modell III is also qualified to detect images in our project.

8.1.6 Discussion

So far, we have seen all the models and their results, and the question we want to solve is which one of them is better.

In the first instance, we can discard Model II because it is the one with less accuracy in making predictions and is not as reliable as the others are, as we can observe in the previous results.

Then the choice is between Models I and III. As we have seen, both models have a shallow test loss and can perform good predictions in all the dataset images. The only difference between them is that Model I use data augmentation, and the other does not.

We have concluded that Model I is the best because of data augmentation. This extra training that gives data augmentation to the model allows it to be prepared for more situations than Model III. This decision has been taken thinking of the future, where the model could be tested with images not as clean and clear as the ones from the dataset.

If we focus only on the tests from the dataset, both models are equivalent. Maybe, Model I has higher confidence in the bounding box predictions, but that is all.

8.2 Semantic Segmentation

This implementation aims to predict where the objects are located and what label they are, like in object detection but adding a mask, where the image is segmented depending on the items from it.

We must remember that we have trained our model with a variation of the Fashionpedia database where all the irrelevant subcategories, such as sleeves, pockets, and others, are ignored to avoid possible confusion by the model.

Finally, we have trained and tested our model in multiple ways, which we will explain in the following sections with their corresponding setups.

8.2.1 Training Setup

Next, we will explain the different variables we have modified during the project to obtain the current model.

In this implementation, we have been playing primarily with the classification networks in the part of the encoder. We have used three different types:

- Unet.
- Mobilnet.
- Renset50.

The training sessions have been organized in epochs. During each period, all the images from the respective dataset are processed. In addition, we have the batch size, which is the number of samples that will be propagated through the network at one on every step of the epoch.

Finally, the checkpoint variable is the path where the checkpoint we want to use is located. If there are not any checkpoints, it means that we are working from scratch. Otherwise, we will use pre-trained weights from our model, which needs more training, or other people's models. The use of this variable is also called finetuning.

8.2.2 Model I

Like in Object Detection, we will summarize all the configurations we have used to train and test the model.

TRAIN CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Classification Network	Resnet50
Nº epochs	50
Batch size	2
Checkpoint	None

With this configuration, we have achieved a model with a test loss of 34,81% and an accuracy of 89,44%. Next, you can see a graph with the evolution of the test loss and accuracy over the epochs:

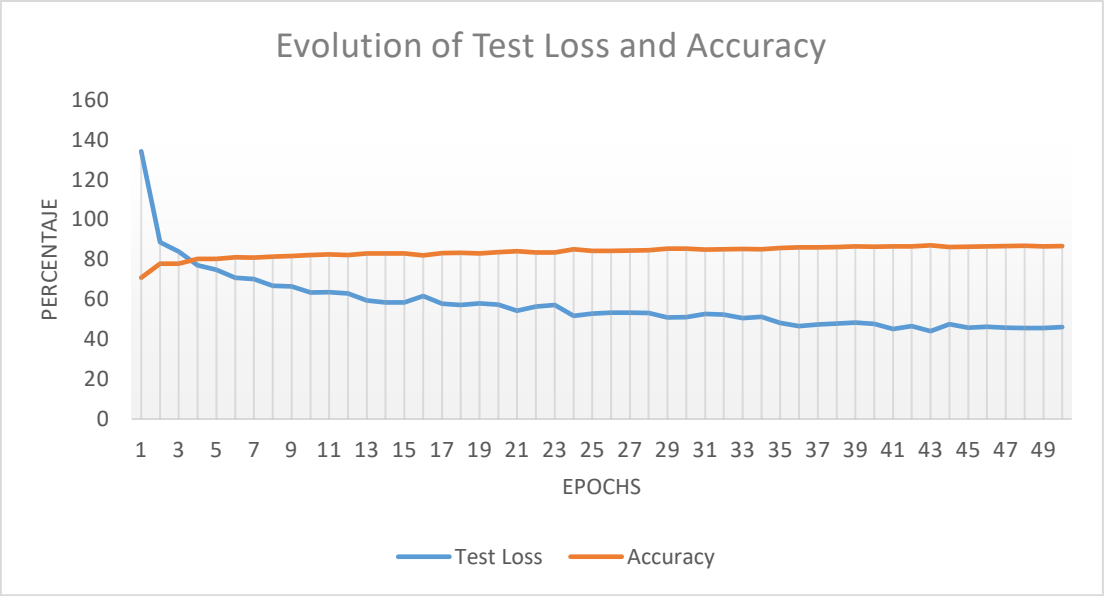
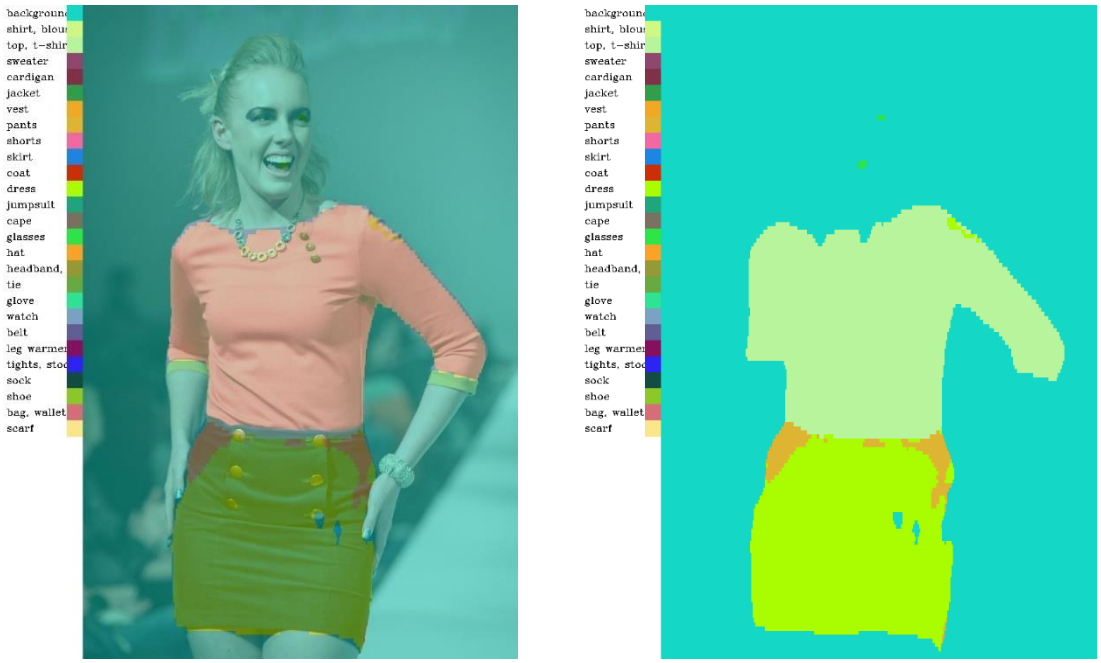


Figure 39: Evolution of Model I Test Loss and Accuracy

We can observe in the image above that test loss decrease quickly at the beginning but becomes slower in the end. Accuracy, however, increases little by little during all the training.

Below, we can see some of the results obtained from Model I:



As we can see, the results are decent as much. The model accurately detects the pixels where there are clothing items but is very unreliable at classifying them. For example, the image above detects the pixels where there is the skirt but says it is a dress, which is incorrect.

8.2.3 Model II

As before, we will summarize all the configurations we have used to train and test the model.

TRAIN CONFIGURATION	
GPU	NVIDIA GeForce RTX 2070 SUPER
Classification Network	Mobilnet
Nº epochs	50
Batch size	2
Checkpoint	None

With this configuration, we have achieved a model with a test loss of 34,52% and an accuracy of 89,84%. We have improved some tenths to both metrics, specifically, 0.29 points less in test loss and 0.40 points more in accuracy. Next, you can see a graph with the evolution of the test loss and accuracy over the epochs:

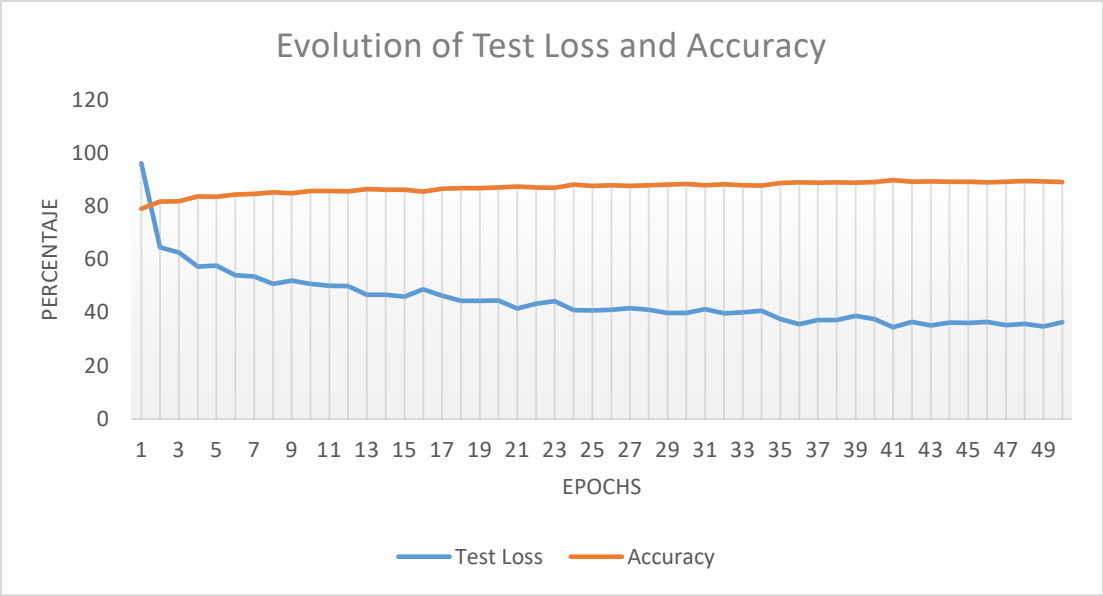
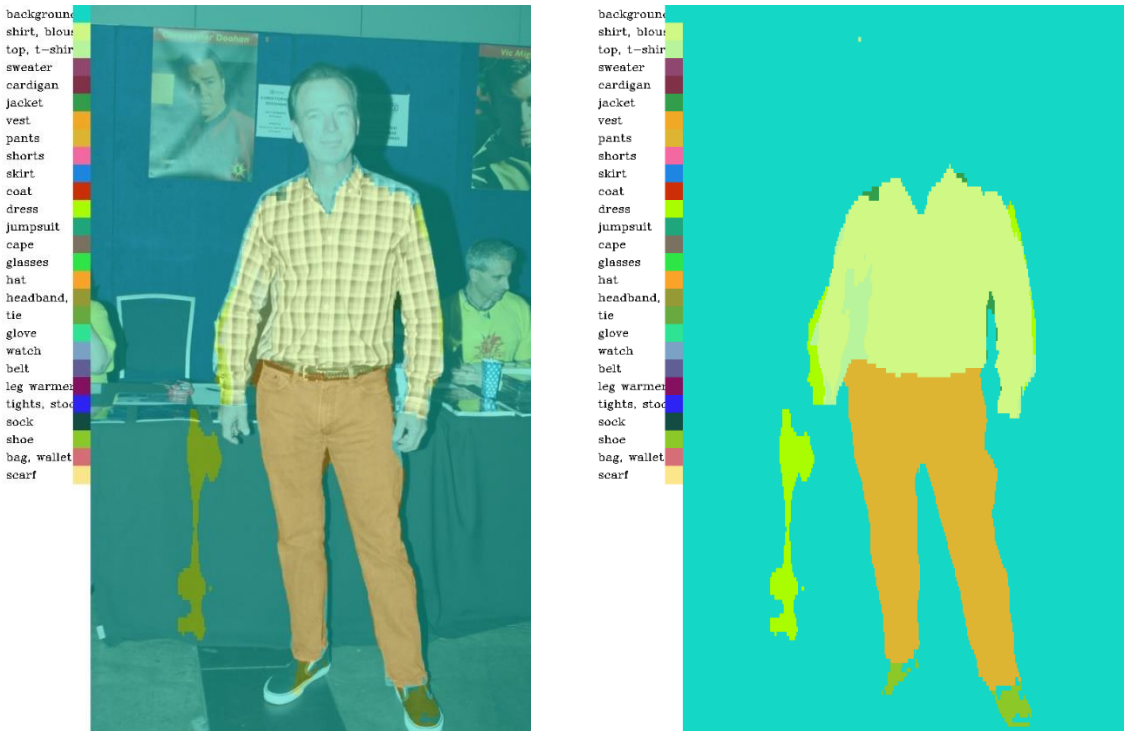
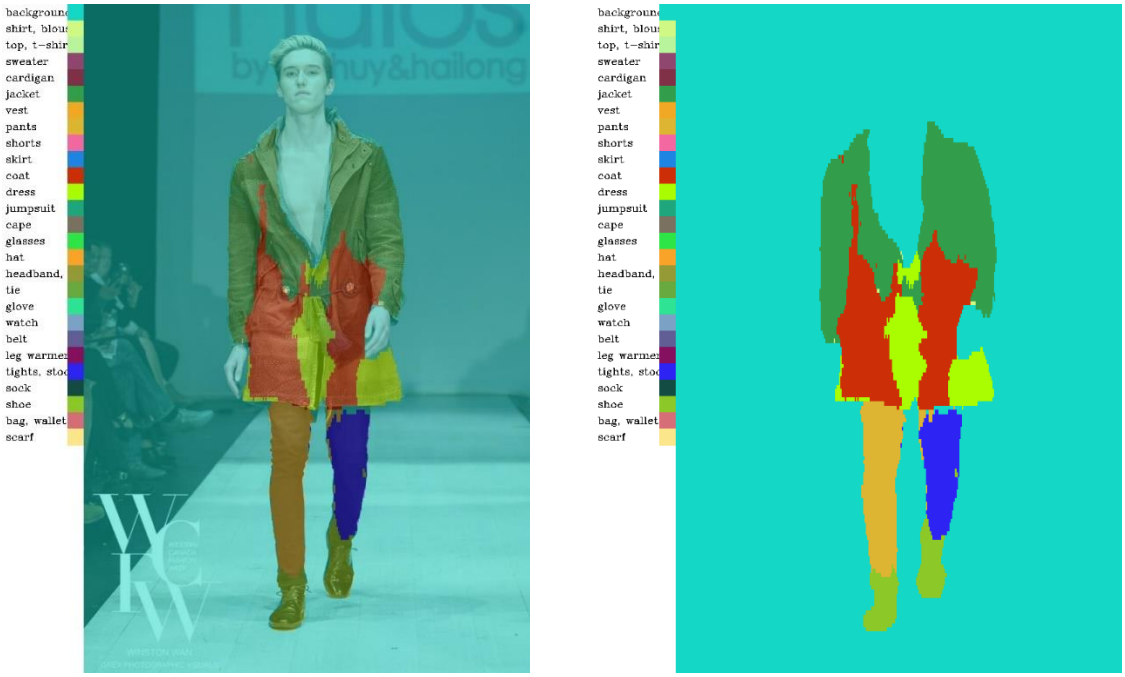


Figure 40: Evolution of Model II Test Loss and Accuracy

We can observe that both models have a similar evolution and values on the metrics. The test loss may be more variable during the training in Model II, and Model I has a more linear development.

Below, you can see some of the results obtained from Model II:





As we can see, the results are very similar to Model I, even changing the feature extractor. The same problem appears. It is good at detecting objects but not classifying them. However, the classification in Model II improves a little bit.

8.2.4 Discussion

Having seen both models' results, we can observe that they poorly perform good predictions on the images. We could say these predictions are partially correct because pixels are correctly selected, showing a fashion item in the end but with ambiguous labels.

Therefore, we concluded that Models I and II are unsuitable for performing image segmentation, at least with fashion as the topic. However, if we had to choose one, Model II would be selected due to the extra tenths achieved in the accuracy. Furthermore, as we have seen in the results, the classifications are closer to being right in this model.

8.3 Yolo vs. Unet

This section will compare both algorithms and their results, and we will see which one performs better in detecting fashion items on images.

We have seen the results from both implementations, and the models selected for each are Model I from Object Detection and Model II from Semantic Segmentation. In addition, remember that the algorithms used in the project are Yolo and Unet.

The final decision is quite apparent due to the best results coming from Object detection Model I, so we concluded that in this project, the best method to predict fashion items in RGB images is Yolo.

On the contrary, Unet has not been the best option for developing Semantic segmentation with fashion as the topic. Which does not mean there are no other algorithms that could make better predictions than Yolo on this topic.

9 Conclusion

Finally, we will summarize all the work done, analyze the pros and cons of the project, and how it can be improved in the future.

In this project, we have implemented two deep learning algorithms, Object Detection, and Semantic Segmentation, to recognize clothing items in RGB images. We have used YOLO as an object detection function and Unet as a semantic segmentation function.

To allow them to recognize the different parts of a picture, we have made many training sessions feeding them with a bunch of photos from a dataset that we have explicitly selected. Thanks to that, the models have learned slowly to extract all the features from an image and understand it as a human.

Once we concluded the project, we saw that comparing the two developed implementations, the one that performs its job better is Yolo. However, the fact that Yolo obtained better results than Unet does not mean object detection is better than semantic segmentation.

Semantic segmentation would be an excellent option to work with, but with another algorithm like Deeplab or Pspnet. These methods could be possible options to research and put in practice in future projects. Unfortunately, Unet has not been the best development option for this project.

Concerning object detection, Yolo has been a great option to implement, and we have achieved excellent results. We could get even better results with a more extensive dataset and training.

In conclusion, we could make improvements in general to the implementations and use them in various situations. For example, suppose we could access all Instagram photos or TikTok videos. In that case, we could analyze all this information to see which combinations of clothing items are more used, liked, etc., and use them in our favor to make assumptions based on that.

10 Bibliography

ArcGIS. (n.d.). *How does U-net work?* Retrieved from https://developers.arcgis.com/python/guide/how-unet-works/

Biswal, A. (2021, November 11). *The Complete Guide on Overfitting and Underfitting in Machine Learning.* Retrieved from https://www.simplilearn.com/tutorials/machine-learning-tutorial/overfitting-and-underfitting

Brownlee, J. (2020, June 20). *Dropout Regularization in Deep Learning Models With Keras.* Retrieved from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Brownlee, J. (2020, 1 25). *Understand the Impact of Learning Rate on Neural Network Performance*. Retrieved from https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Code, P. w. (n.d.). *Rectified Linear Units*. Retrieved from https://paperswithcode.com/method/relu

Gad, A. F. (2020). *Faster R-CNN Explained for Object Detection Tasks*. Retrieved from https://blog.paperspace.com/faster-r-cnn-explained-object-detection/

Gandhi, A. (2021). *Data Augmentation | How to use Deep Learning when you have Limited Data*. Retrieved from https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

Jia, M. a. (2020). *Fashionpedia: Ontology, Segmentation, and an Attribute Localization Dataset*. Retrieved from https://fashionpedia.github.io/home/Fashionpedia_download.html

Johnson, D. (2022, May 14). *Deep Learning Tutorial for Beginners: Neural Network Basics*. Retrieved from https://www.guru99.com/deep-learning-tutorial.html

Liu, Z. a. (2016, June). *DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations*. Retrieved from https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

Mantripragada, M. (2020, 8 16). *Digging deep into YOLO V3 - A hands-on guide Part 1*. Retrieved from https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29

Meng, J. (n.d.). *YoloVx(yolov5/yolov4/yolov3/yolo_tiny)*.

Pere, C. (2020, June 17). *What are Loss Functions?* Retrieved from https://towardsdatascience.com/what-is-loss-function-1e2605aeb904#:~:text=The%20loss%20function%20is%20the,be%20categorized%20into%20two%20groups.

Raj, B. (2018, 5 29). *A Simple Guide to the Versions of the Inception Network*. Retrieved from https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Rosebrock, A. (1019, 6 3). *Fine-tuning with Keras and Deep Learning*. Retrieved from https://pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

Rosebrock, A. (2016, 11 7). *Intersection over Union (IoU) for object detection*. Retrieved from https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks*. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Sahu, B. (2019, 7 12). *The Evolution of Deeplab for Semantic Segmentation*. Retrieved from https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571

Shivaprasad, P. (2019, January 9). *A Comprehensive Guide To Object Detection Using YOLO Framework*. Retrieved from https://towardsdatascience.com/object-detection-part1-4dbe5147ad0a

Team, G. L. (2020, 9 28). *Introduction to Resnet or Residual Network*. Retrieved from https://www.mygreatlearning.com/blog/resnet/

Yang, W. a. (2013). *Clothing Co-Parsing by Joint Image Segmentation and Labeling*. Retrieved from https://github.com/bearpaw/clothing-co-parsing