POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Transformer-based Geolocation of Indoor Images with Segmentation-based Visual Explanations

Tesi di Laurea Magistrale in
Mathematical Engineering -
Ingegneria Matematica

Author: **Imanuel Rozenberg**

Student ID: 942645
Advisor: Prof. Mark James Carman
Academic Year: 2020-21

Abstract

As reported by the CDTC (Counter-Trafficking Data Collaborative), data collected since 2002 show that human trafficking crimes are steadily increasing, with up to 15% increase [7]. When computer forensic experts come in possession of encrypted photos of the victims, which are usually taken inside of hotel rooms, they face various difficulties in trying to locate the scene of the crime. This thesis aims to develop a tool to facilitate the localisation of crime scenes. The localisation of hotels is a rather difficult task because of the similarity that can exist between hotel rooms belonging to the same chain in different locations around the globe, and at the same time the dissimilarity that can exist between different rooms in the same hotel. In addition, our work can be useful for detecting fake advertisements with photos showing the interior of a house located in a different place than the one mentioned on real estate agency web-pages or Airbnb. In order to be really useful to the Police, a tool should not simply be capable of identifying the location of hotels or B&Bs (Bed and Breakfast). It should not only accurately locate the place, or at least narrow the search, but also provide explanations on which elements were relevant to arrive at a given prediction.

For these reasons, in this thesis we have decided to focus on both aspects (locating and providing explanation), using some of the most powerful neural networks and adapting numerous XAI (Explainable Artificial Intelligence) approaches to obtain meaningful visual explanations. We addressed the indoor geolocation problem as a hierarchical classification task, predicting simultaneously the city, the country and the region in the world of a desired indoor scene image. To conclude, we show the effectiveness of our model with a demo and some experiments capable of identifying highly informative elements within an image in terms of predicting its geographical location.

Keywords: indoor geolocation, hierarchical classification, explainability, artificial intelligence, deep learning.

Abstract in lingua italiana

Come riportato dal CDTC (Counter-Trafficking Data Collaborative), i dati raccolti dal 2002 mostrano che i crimini legati al traffico di esseri umani sono in costante aumento, con un incremento fino al 15% [7]. Quando gli esperti di perizie informatiche forensi entrano in possesso di foto criptate delle vittime, che di solito sono scattate all'interno di stanze d'albergo, affrontano varie difficoltà nel cercare di localizzare la scena del crimine. Questa tesi mira a sviluppare uno strumento per facilitare la localizzazione delle scene del crimine. La localizzazione degli hotel è un compito piuttosto difficile a causa della somiglianza che può esistere tra camere d'albergo appartenenti alla stessa catena in diverse località del mondo, e allo stesso tempo la dissomiglianza che può esistere tra diverse camere dello stesso hotel. Inoltre, il nostro lavoro può essere utile per individuare annunci falsi con foto che mostrano l'interno di una casa situata in un luogo diverso da quello menzionato sulle pagine web delle agenzie immobiliari o su Airbnb. Per essere davvero utile alla polizia, uno strumento non deve semplicemente essere in grado di identificare la posizione di hotel o B&Bs (Bed and Breakfast). Dovrebbe non solo localizzare accuratamente il luogo, o almeno restringere il campo di ricerca, ma anche fornire spiegazioni su quali elementi sono stati rilevanti per giungere ad una determinata predizione.

Per questi motivi, in questa tesi abbiamo deciso di concentrarci su entrambi gli aspetti (localizzare e fornire spiegazioni), utilizzando alcune delle reti neurali più potenti e adattando numerosi approcci XAI (Explainable Artificial Intelligence) per ottenere spiegazioni visive significative. Abbiamo affrontato il problema della geolocalizzazione di foto di interni come un compito di classificazione gerarchica, prevedendo simultaneamente la città, il paese e la regione del mondo in cui un'immagine di interni è stata scattata. Per concludere, mostriamo l'efficacia del nostro modello con una demo e alcuni esperimenti in grado di identificare elementi altamente informativi all'interno di un'immagine per la predizione della sua posizione geografica.

Parole chiave: geolocalizzazione di interni, classificazione gerarchica, spiegabilità, intelligenza artificiale, deep learning.

Contents

1 | Introduction

The introduction presents the problem addressed in this thesis, indoor geolocation, and illustrates our goal to develop a valid tool to face this problem.

1.1. Goal

Hotel geolocation and recognition means identifying the geographical location and the name of hotels related to images taken inside hotel rooms. The need to geolocate images of indoor scenes arises mainly from the need to help police forensic investigations with a tool capable of identifying, or at least narrowing the research when dealing with human trafficking material. Victims of human trafficking are often brought and photographed in hotel rooms, for this reason hotel recognition and geolocation is important to help track down current, and also prevent future crimes with victims who might be taken to the same hotels. Internet technologies such as P2P networks or Instant Messaging applications allow to freely exchange in an encrypted manner harmful and sensitive material. Computer forensics experts often come in possession of such material by direct retrieval from a criminal's device or by reports of the Internet Watch Foundation. Unfortunately, in many cases it is very hard to identify the location of the committed crimes in order to intervene and prosecute. One of the main challenges of recognizing hotels from interior images is that pictures taken inside of the same hotel but in different rooms might present significant different visual appearances; while at the same time pictures from different hotels, in different countries, but belonging to the same chain, might be quite similar as in Fig. 1.1.

Another case for which it might be useful to geolocate interior images is related to fake news, or also fake ads on websites as Airbnb and Booking, using images from another place than the one being referenced. With the advent of deep learning the ability to solve image classification problems improved drastically, thus allowing also to explore the capability of neural networks to perform content-based indoor geolocation. Even though, it has to be mentioned that there exist only few works dealing with interior scenes geolocation, while most of the works in literature focus on outdoor geolocation scene. The reason is that outdoor scene could present iconic monuments like the Colosseum or the Statue of Liberty,

Figure 1.1: The image on the left and the central image are taken in the same hotel, while the right one is taken in another hotel in another geographical location, but which belongs to the same chain. The image is taken from the Kaggle challenge: The 2021 Hotel-ID to Combat Human Trafficking Competition Dataset.

landscapes like the Himalayas, Niagara Falls or the Caribbean Sea, that might be easy to locate. While in case of outdoor scenes sometimes the identification of only one monument or even a few architectural patterns might be enough to point out the exact location, in case of interior scenes we need to be more cautious, since a living room of a house in Madrid could have pictures on the wall of the Tour Eiffel and some other decorations, based on personal decision which could be misleading. Even if in the globalized world in which we live, "IKEAfication" of furniture has penetrated urban and rural regions all over the globe, making extremely difficult to understand the correct location of interior scenes. Xi Liu et al. in [19], show that it is still possible to cluster patterns and regional cultural decorative behaviour.

We argue that there are many decorative and structural patterns, which when observed jointly are strictly correlated to the geographical location and that it is possible to learn these features which are helpful to distinguish one location from another. For example in Fig. 1.2 we observe a danish lamp (Louis Poulsen), a mouse pad with Hebrew characters which might be misleading, but if we look at the tablecloth, the floor and some of the pictures on the wall we might guess that this is a living room of an Italian house since there are no similar floors, combined with these types of furniture and walls of this size in Israely or Danish houses. The aim of this work is to develop a tool to facilitate geolocating indoor pictures, providing a list of predictions at different granularity levels, such as: Continent, Country and City. In addition, to make more useful and interpretable the predictions of our model, we also employed some explanatory techniques capable of highlighting discriminating information in the geolocation process.

Figure 1.2: The image is taken inside of a house located in Rome. There are many elements that may suggest that this is an Italian house, but just as many that may lead, especially for an inexperienced eye, to commit a mistake when trying to locate this image.

2 | Related Works

This chapter is meant to give a brief introduction about the Deep Learning concepts, architectures and previous works on image geolocation that allowed us to develop our entire framework. We will start by introducing some basic concepts about Convolutional Neural Networks (CNN) for image classification to then move to the most recent Vision Transformers for image classification. We will explain also different approaches for visual explanations and some of the most important works done for image geolocation, even though most of them were meant to solve the outdoor geolocation problem, they still serve as inspirations to the development of our model for indoor geolocation.

2.1. Image Classification

In our work we decided to tackle the indoor geolocation task as a classification problem. For this reason we will present a brief introduction on the ideas behind most of the state-of-the-art deep learning architectures for image classification. We can recognize two dates as "revolutionary" for the world of image classification: the first one is 2012, with the introduction of AlexNet [17] which significantly outperformed all the prior competitors on the ImageNet [8] Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. Even though the first CNN was introduced in 1998 (LeNet [18]), AlexNet is deeper, with more filters per layer, and with stacked convolutional layers, more similarly to current CNNs.
The second important date is 2020 with the revolutionary publication of the Vision Transformer (ViT) expanding the transformer adoption from Natural Language Processing (NLP) to the world of computer vision, using self-attention on image patches instead of textual tokens.

2.1.1. Convolutional Neural Networks for Image Classification

The goal of this chapter is to describe the mathematical tools that Convolutional Neural Networks exploit and then give a description of the whole architecture and functioning of

a CNN.
In deep learning, CNN is a special class of multi-layer neural networks designed to recognize visual patterns directly from pixel images and then use the extracted information in order to give some final predictions. The general scheme of a CNN is divided into two main building blocks:

- **Feature extractor**, characterized by the alternation of convolutional layers alternated by pooling operations to extract relevant information from the image.
- **Fully Connected Classifier**, the incoming features information from previous blocks are flattened into a one dimensional vector fed to a fully connected network. The fully connected network is made of dense layers with non linear activation functions, ReLU [1] or s-shaped functions, and at the end a final layer which assigns the respective probabilities to each class.

A general structure of a CNN is presented in Figure 2.1.

Feature extractor (CNN)

A convolutional layer performs convolution between an image and a kernel (filter) whose weights are learned via back-propagation. Before entering deeper into the explanation of convolutional layer, we need to understand what is the mathematical idea behind a convolution operation and how it works. A convolution is an operation, typically denoted with an asterisk, between two functions f and g as reported in the following equation:

$$h(t) = (g * f)(t) \tag{2.1}$$

The output of a convolution, h represents the way the function g influences the function f. In the deep learning framework the function f is referred to as the **input** and g as the **kernel**, while the output h is usually called **feature map**. When the functions are defined over a continuous interval, the convolution operation in one dimension is defined as follows:

$$h(t) = \int f(a) \cdot g(t-a)da \tag{2.2}$$

Since in our case the functions are defined over a discrete interval, image pixel grid, the convolution is a **discrete convolution**.

$$h(t) = \sum_{a} f(a) \cdot g(t-a) \tag{2.3}$$

Since color images (RGB) are 3-dimensional matrices, we can express the input image as: $I(i, j, k)$ and the kernel as $K(i, j, k)$ thus the convolution of K over I is given by:

$$S(i, j, k) = (I * K)(i, j, k) = \sum_m \sum_n \sum_p I(m, n, p)K(i - m, j - n, k - p) \tag{2.4}$$

The output function $S(i, j, k)$ will still be an RGB image, but its pixel will be manipulated as intended by the kernel K, see Fig.2.2.

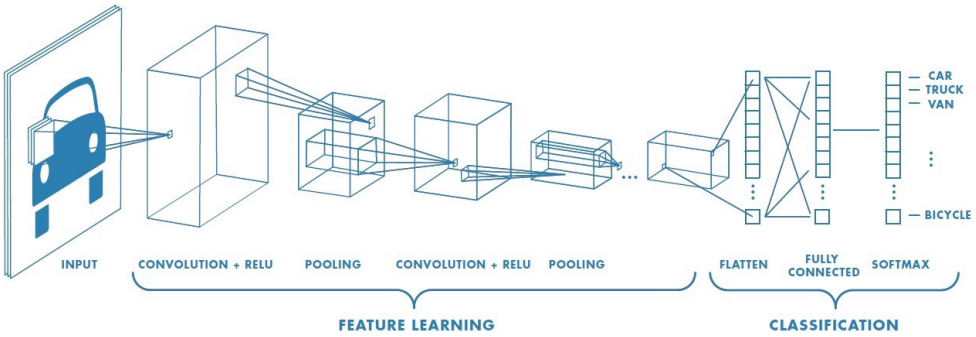


Figure 2.1: General structure of a CNN [1].

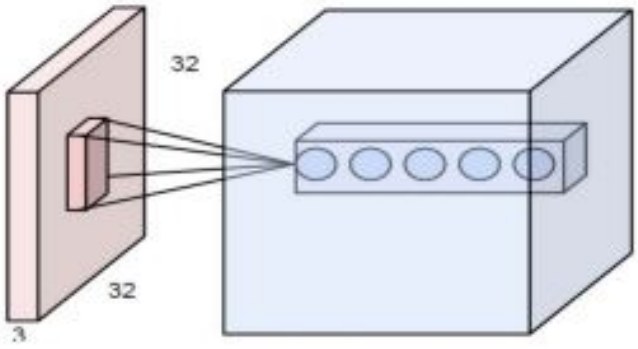


Figure 2.2: Convolution operation picture taken from [2]. On the right-hand side of the image it is possible to see how each kernel applied on the input image (left) generates a new representation of the output independently of the other kernels.

Stacking multiple convolutional layers in CNNs allows to gradually extract useful patterns to be used later for the desired classification task. The first block of filters that operate directly on the raw pixel values will learn to extract low-level features, such as edges, lines and simple patterns that are discriminative to describe data. This process continues until very deep convolutional layers extract high-level semantic features as faces, animals and so on.

[1]Source: `https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html`

In CNNs the convolution operation is applied over the input tensor (array) according to the number of kernels stored inside the convolutional layer.
All the values of the kernels are trainable, that is to say they change through time when the model is learning via back-propagation, so to maximize the performance for the final prediction task.
One of the biggest limitations of the feature maps output of convolutional layers is that they store the precise position of features in the input. Thus if small movements, like rotation, shifting or cropping, are applied on the input, the result after the same convolutional layer will produce a different feature map. To avoid this problem a rather used approach from signal theory, called down sampling, is adopted in this framework. The idea, in signal theory, is to create a lower resolution version of the input signal which still contains the important structural elements. In order to reduce the spatial size of the output volume of a convolutional layer, CNN adopts **pooling layers**. More specifically, an s-shaped differentiable activation function (e.g. ReLU [1]) is applied to the feature maps output of the convolutional layer, and just then a pooling layer is adopted to create a new set of the same number of input feature maps of a reduced size.

Pooling Layers

The Pooling layer operates independently on every depth slice of the input and resizes it spatially. Some of the most popular pooling operations are: **Max-Pooling** and **Average Pooling**. The Max pooling down-samples the input representation by applying a max filter to non overlapping regions (patches) of a specified size. Such operation is shown in Fig.2.3. Average pooling, on the other hand, consists in calculating the average for each patch of the feature map.

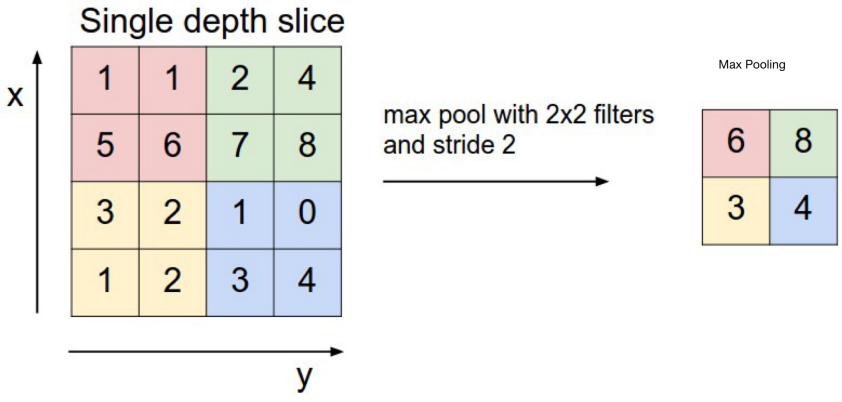


Figure 2.3: Max-Pooling operation with a 2×2 filter.

Fully Connected Classifier

The patterns learned by the feature extractor are then flattened into a one dimensional vector to be fed to a Fully Connected Neural Network (FCNN) also called Multi-layer Perceptron (MLP), see Fig.2.5, made of dense layers whose units are called Neurons, endowed with non linear s-shaped differentiable activation functions as ReLU, Leaky ReLU, Tanh or other functions as the ones presented in Fig.2.4.

The reason why activation fucntions need to be differentiable is that learning is performed via back-propagation, i.e: propagate back the error so as to pass through all the weights and correct them in order to predict the correct output. Back-propagation is based on gradient descent optimisation algorithm which requires differentiable functions.
One of the main advantages of the ReLU function with respect to Sigmoid and Tanh is that it is easy to optimize, thanks to its similarity to linear units. Moreover, its derivatives remain large whenever the unit is active (greater than zero) while it is zero in any other case, speeding up the gradient computation and, subsequently, the neural network backpropagation.
More recently, researchers have designed new variations of the ReLU functions, such as the Leaky ReLU that avoids zero values which could affect the gradient computation [11].

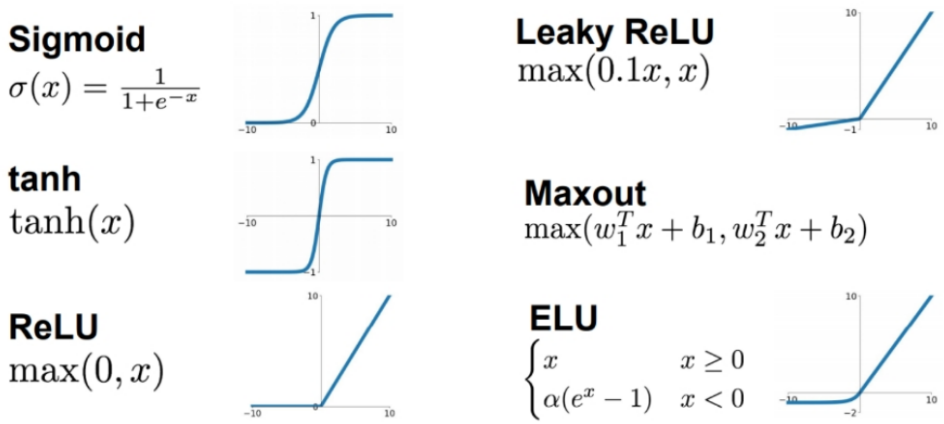


Figure 2.4: Cartesian representation of some S-shaped activation functions, taken from [2].

Neurons are grouped together inside dense layers, identified by their parameters: weights w and biases b, both trainable. Dense means that each neuron in input is connected to all the neurons in the output layer.

Given an input x, representing all the neurons in the input layer and a bias b, the output

[2]Source: `https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7`

of a neuron in the consecutive dense layer can be described as follow:

$$h(x) = g(W^t * x + b) \tag{2.5}$$

where g can represent for example the ReLU function, applied to an affine transformation of the input values.
As previously mentioned, dense layers are fully-connected, thus each unit in the input is connected to all the units in the consecutive layer, and each neuron in the output receives a weighted sum of all inputs coming from the previous set of neurons.

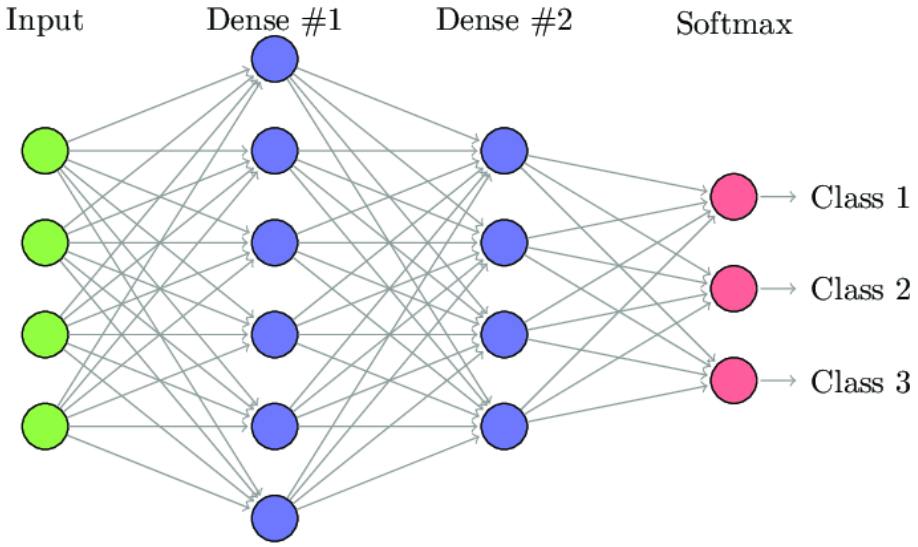


Figure 2.5: Fully-connected network structure from [26].

In a multi-class classification problem, the last layer usually adopts a Softmax function in order to output the respective probabilities for each class. A Softmax function is a differentiable function whose output could be interpreted as the probability over a discrete variable with n possible values. If we consider the output of a linear layer z, the Softmax function can be expressed as:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.6}$$

2.2. Attention Models

In 2017 a great advance in the Neural Machine Translation (NMT) was achieved with the paper "Attention is all you Need" [37] which presented a new architecture: the **Transformer**, on which most of the modern NLP models, like GPT-2 [27], and BERT [9], are based. The Transformer is the first transduction model relying solely on **self-attention** to compute representations of its input and output without using sequence-aligned Recurrent neural networks (RNNs) or convolutions. Before entering deeper into the architecture of the Transformer, we give a brief explanation of the original idea of the **attention** mechanism and some of its most popular variants like self-attention. The original idea of the attention mechanism was presented by Bahdanau et al. in the paper [3], to address the performance bottleneck of conventional encoder-decoder architectures, achieving significant improvements over the conventional approach used so far for NMT. Bahdanau attention mechanism can be summarized as follows.
Given an input sequence x of length n and a target sequence y of length m in an NMT problem:

$$\begin{aligned} x &= [x_1, x_2, ..., x_n] \\ y &= [y_1, y_2, ..., y_n] \end{aligned} \tag{2.7}$$

assume our NMT architecture is an encoder-decoder, h_i is an annotation (the encoder hidden states), with $i = 1, ..., n$, h_i contains information from all the words forming the input sentence with a strong focus on the $i-th$ word among the n words in input, call s_{t-1} the hidden decoder state at previous state. The decoder uses an attention mechanism in order to produce the target words by focusing on the most relevant information coming from the source sentence. The decoder thus takes each hidden encoder state (annotation) and feeds it to an alignment model, $a(\cdot)$, jointly with the previous hidden decoder state s_{t-1} generating an attention score:

$$e_{t,i} = a(s_{t-1}, h_i) = v_a^T \tanh(W_a[s_t; h_i]) \tag{2.8}$$

$e_{t,i}$ is an attention score that scores how well h_i and s_{t-1} match. s_{t-1} and h_i are combined by means of an additional operation, from here the name "additive attention". This can be implemented by applying a weight matrix over the concatenated vectors s_{t-1} and h_i, and v^T is a weight vector. The alignment model is parametrized as a Feedforward neural network (FFN) and trained with the remaining system components. In order to normalize the annotation values to a range between 0 and 1, then a Softmax function is performed over each attention score to obtain the corresponding weight value.

$$\alpha_{t,i} = Softmax(e_{t,i}) = \frac{\exp(a(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(a(s_{t-1}, h_i))} \tag{2.9}$$

$\alpha_{t,i}$ is a score which explains how well y_t and x_i are aligned and $e_{t,i}$ is an attention score that scores how well h_i and s_{t-1} match:

> "Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector [3]."

The final stage is the computation of the context vector as c_t where:

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{2.10}$$

Then, the context vector is fed to the decoder together with the previous hidden decoder state, s_{t-1} and the previous output, y_{t-1}, to compute the final output, $s_t = f(s_{t-1}, y_{t-1}, c_t)$. The whole procedure is repeated until the end of the sequence.

We present below some different types of attention computation originated from the original attention mechanism, which are essential to understand our work.

- **Self-Attention**: relates different positions of the same input sequence in order to compute a representation of it, differently from the original additional attention which relates the alignment between input and output positions in a sequence. This kind of attention is widely used not only in NLP problems, but also in computer vision and is the basic ingredient for the Transformer architecture.
- **Hard vs Soft Attention**: are two different types of attention mechanism introduced in [38] for an image captioning problem, in order to understand on which parts of the image to focus to produce meaningful captions. The distinction between hard and soft attention consists in whether the attention block has access to the whole image or only to a patch. In soft attention, the alignment weights are learned by taking into consideration the whole image, so that the computation is smooth and differentiable, on the other hand the main limitation is that if the input is large, it will be computationally intensive to compute the soft attention. Hard attention, on the other side, processes only one patch of the image at a time,

allowing faster computations, but requiring a more sophisticated training, due to its non differentiable nature.

- **Global vs Local Attention**: In Global attention, analogously to soft attention, the output of the attention block is computed taking into account every source state from the encoder and all decoder states prior to the current state. While in Local attention, more similarly to hard attention, only a window of positions from the encoder are used to compute the output, but the advantage with respect to hard attention is that local attention is differentiable. A visual idea of the two different mechanism is shown in Fig. 2.6 taken from the original paper by Loung et al. in [23] were these two different kinds of attention mechanism were presented.

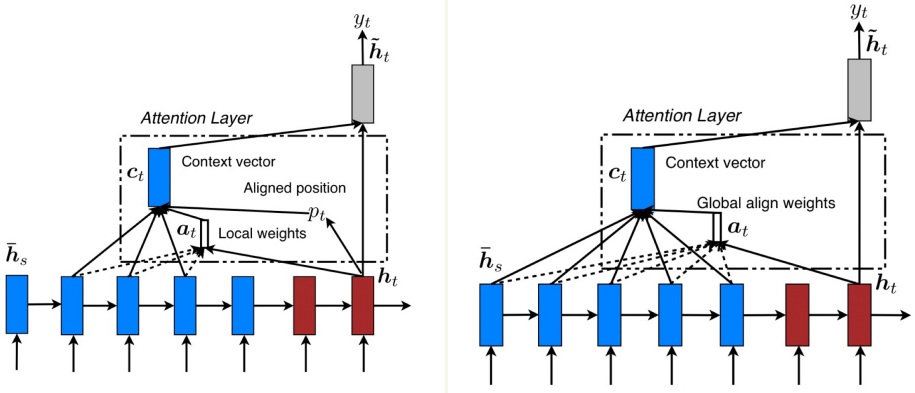


Figure 2.6: Image taken from [23]. **Local attention** model on the left: it first predicts a single aligned position p_t for the current target word, then a new window centered around the source position p_t is used to compute a context vector c_t and a weighted average of the source hidden states inside the fixed window size. The weights a_t are inferred from current target state h_t and the source states $\bar{h}_s$ which are in the window. **Global attention** model on the right: at each time-step t, the model infers a variable-length alignment weight vector a_t, based con current target state h_t and all source states $\bar{h}_s$. Finally a vector c_t is computed as the weighted average, according to a_t over all the source states.

2.2.1. Transformer

The Transformer was introduced in order to overcome the structural limitations of prior architectures like Recurrent Neural Networks (RNNs) when dealing with long sequences of text. Differently from RNNs, transformers do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence. Assuming that the input data is a natural language sentence, the transformer does not

need to process the beginning of the sentence before the end, rather, it identifies the context that confers meaning to each word in the sentence. This feature allows for more parallelization, which is precluded by the sequential nature of RNNs, and thus reduces training time, allowing to process huge corpus of text in a more efficient way. So, instead of having a recurrent encoder and decoder, the Transformer architecture is mainly built by a stack of encoders and followed by a stack of decoders. Assume the input sequence is a sequence of words, first the words are embedded by an embedding algorithm and then the embedded vectors are passed as input to the first encoder, then encoded information flows through the stack of encoders until it reaches the encoder on top of the stack, from which it is shared to all the decoders, see Fig. 2.7.

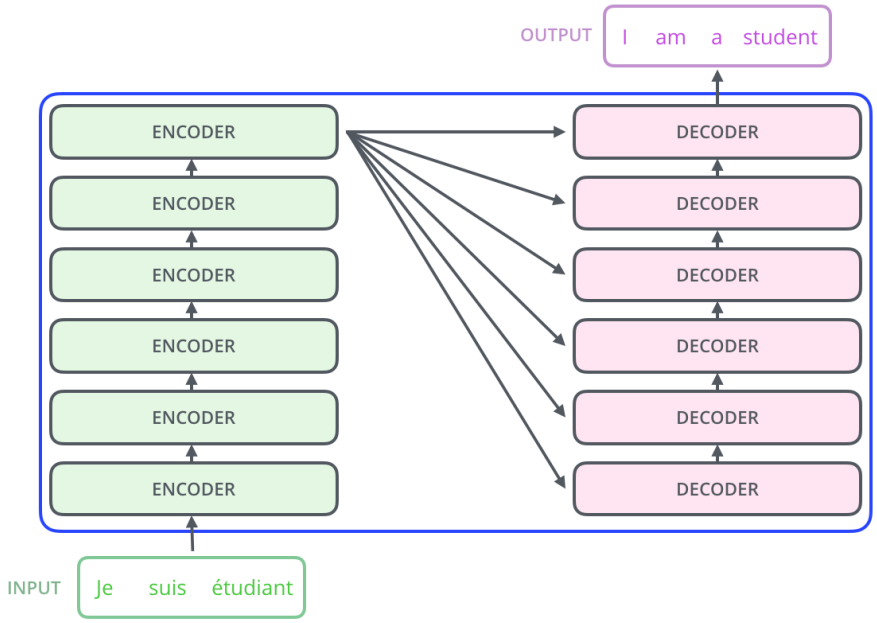


Figure 2.7: Macro-view of the Transformer architecture, the paper [37] proposed stacks of 6 encoders and decoders. The image is taken from [3].

All the encoders and decoders present the same identical structure, which is the one presented in Fig. 2.8.
The encoder's inputs first flow through a self-attention layer which helps the encoder look at other words in the input sentence when encoding each specific word. We will give a more detailed explanation of self-attention afterwards. The outputs of the self-attention layer are then fed to a feedforward neural network (FFN). The decoder has both those layers, with in addition between them an extra encoder-decoder attention layer that helps the decoder to focus on relevant parts of the input sentence (similarly to what attention does in seq2seq models). We now enter a little bit more into details about the components

[3]Source: `https://jalammar.github.io/illustrated-transformer/`

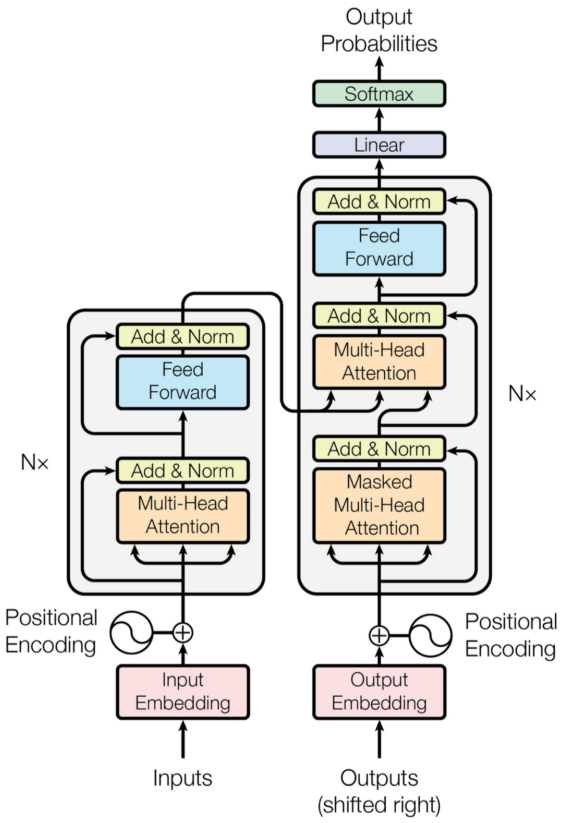


Figure 2.8: An encoder block and the respective decoder to which encoded information are passed [37].

of the encoder-decoder structure. As mentioned above, the very first step before passing the input sequence to the first encoder is to turn each input word into a vector via an embedding algorithm. In order to take into account the order of a word in a sequence, the transformer adds a vector to each input embedding, called positional encoding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The idea is that incorporating this information to the embeddings can provide meaningful distances between the embedded words. Then, in order to compute self-attention, three vectors of smaller dimension are generated from each of the encoder's input vectors (the embedding of each word): a **Query vector**, a **Key vector**, and a **Value vector**. These vectors are created by multiplying the embedding by three matrices that we trained during the training process. The actual implementation is not made of singular input vectors x but of matrices packing all the input embedded vectors X, which when multiplied by the respective weight matrices W^q, W^k, W^v give as output respectively: the Query, Key and Value matrices, see Fig. 2.9. Self attention is then computed as a scaled dot product of

the matrices, as presented in the equation:

$$Z = Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.11}$$

Where d_k is the square root of the key vectors, the division is made in order to have more stable gradients. The final Softmax is used to normalize the scores.

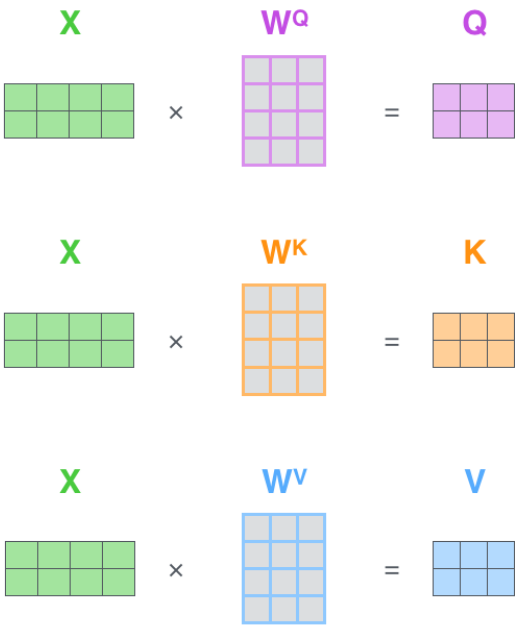


Figure 2.9: Assume we have two input words embedded into two vectors of same length and then merged in the two rows of the matrix X, we then multiply it by the weight matrices W^q, W^k, W^v that we have trained and obtain the Q, K and V vectors [3].

Moreover, the paper proposes to use a mechanism called: "**Multi-Head Attention**" which consists in performing multiple times - in parallel - the self-attention block as presented in Figure 2.10. The result is an improvement in the model ability to focus on different positions of the input sequence, and have multiple outputs which are different representation in different representation sub-spaces.

The consecutive block, which is a feed-forward layer, is not expecting different matrices for each word, it's expecting only a single matrix for each word. So, we concatenate for each word the multiple output coming from the multi-head attention into a single matrix, and then multiply by an additional weight matrix W^o in order to obtain a single output matrix Z for each word, which captures the information coming from all the attention heads. A visual description of the presented schema is presented in Fig. 2.11.

[3]Source: `https://jalammar.github.io/illustrated-transformer/`

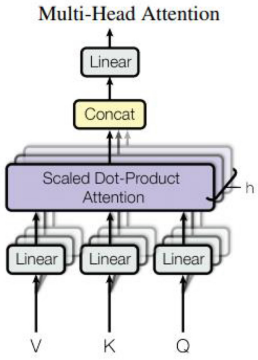


Figure 2.10: Multi-Head Self Attention. Image taken from [37].

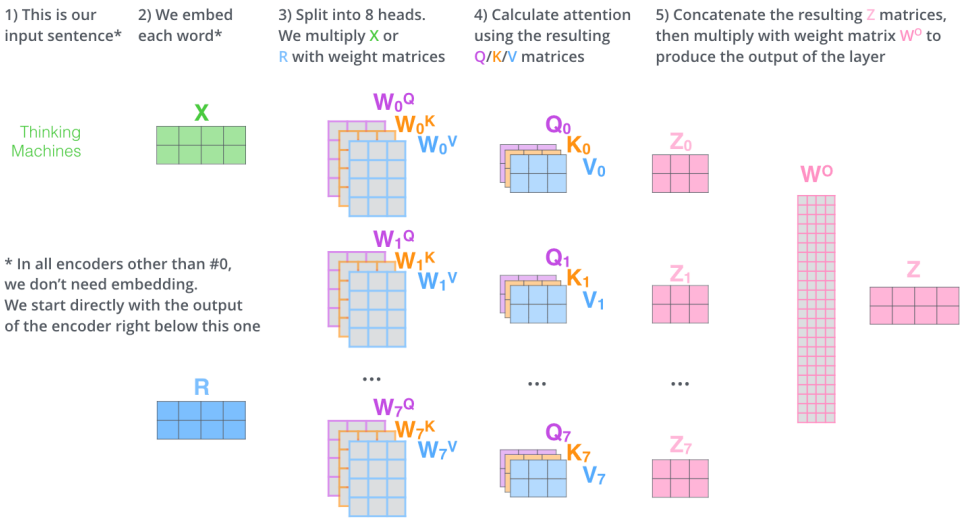


Figure 2.11: Processing of the output of the Multi-Head self attention in order to feed the FFN [3].

As it can be seen in Fig. 2.8, each sub-layer in an encoder block presents a residual connection, followed by a layer-normalization step. The same is also proposed in the Decoder blocks.

To conclude, the output of the top encoder is finally transformed into a set of attention vectors K and V which are shared with the "encoder-decoder attention" layer of the decoder, in order to focus on appropriate places in the input sequence. Each step in the decoding phase outputs an element from the output sequence. The following steps repeat the process until a special symbol is reached, indicating the transformer decoder has completed its output. Just like in the encoder inputs, also on the embedded decoder inputs we add a positional encoding to keep track of the position of each word.

[3]Source: `https://jalammar.github.io/illustrated-transformer/`

There is a small difference between the first self-attention layer in the decoder with respect to the encoder. In the decoder, in fact, future positions of the sequence are masked, before doing the Softmax calculation, so that self-attention is performed only between the word in the current position and earlier positions in the sequence.

The "encoder-decoder Attention" layer works just like multi-head self-attention, except that it creates its Queries matrix from the previous layer, while the Keys and Values matrix are provided by the output of the encoder stack.
Then, to make the final prediction, the output of the decoder is passed through a final linear layer which projects the output vector into a larger vector (of size of the vocabulary), the logit vector. Finally a Softmax computation is performed over the logit vector in order to have scores which are interpretable in terms of probability, so to pick the cell with the highest probability.

2.2.2. Vision Transformer

Inspired by the Transformer huge success in NLP, Dosovitskiy and Houlsby together with Google Research Brain Team recently proposed in "An Image is worth 16X16 words: Transformers for Image Recognition at Scale" [10] to apply a standard Transformer directly to images in order to replace CNNs for computer vision tasks. We present a short overview of the Vision Transformer (ViT) architecture, see Fig. 2.12, which mainly can be subdivided into the following fundamental parts:

1. Split the input image into fixed-size patches
2. Flatten the patches
3. Create lower -dimensional linear embeddings from these flattened image patches
4. Include positional embeddings
5. Feed the sequence as an input to a Transformer encoder identical to the one presented in [37] (whose mechanism we explained in the previous subsection).
6. Pre-train the ViT model with image labels, (fully supervised on a big dataset)
7. Finetune on the downstream dataset for image classification.

The idea is to treat the image patches as a sequence of tokens as we do with words in NLP problems. Differently from the original Transformer where there are 6 stacked encoders , in ViT the number of proposed encoder blocks varies from a minimum of 12 in the ViT-Base model, to 32 in the ViT-Huge model. There is no decoder in the ViT architecture,

only the encoder part of the Transformer are used, and on top of the stack of all the encoders an extra linear layer is added, MLP head, for the final classification.

One of the drawbacks of ViT is that in order to achieve state-of-the-art results on image recognition benchmarks as (ImageNet, CIFAR-100, etc.) it has to be pre-trained on large amounts of data, and then fine-tuned. When dealing directly with small datasets, in fact, the results are slightly worst with respect to CNNs like ResNet or EfficientNet. The reason this happens is that Transformers lack some of the inductive biases inherent to CNNs, such as translation invariance and locality, and therefore do not generalize well enough when trained on insufficient amounts of data. While when trained on large datasets ViT beats most state-of-the-art architecture on multiple image recognition benchmarks. This is due to the fact that large scale training trumps inductive bias.

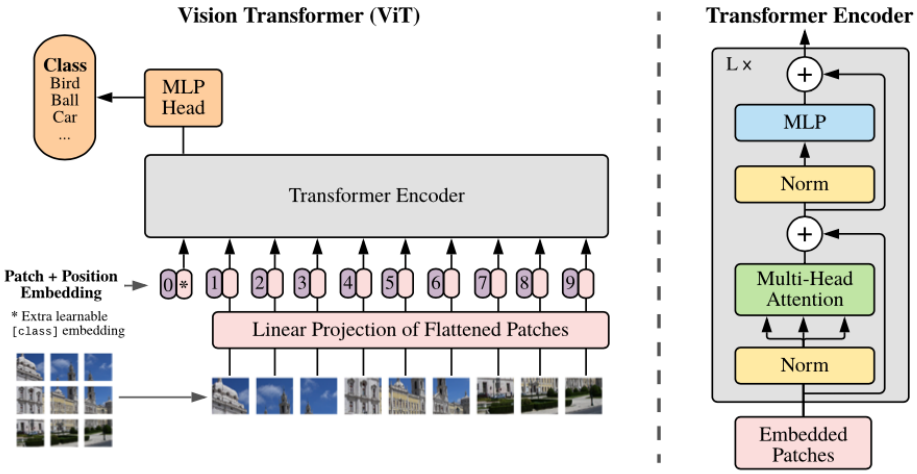


Figure 2.12: ViT overview taken from [10]. The input image is split into fixed-size patches which are linearly embedded with additional position embeddings. Then the resulting sequence of vectors is fed to a standard Transformer encoder. In order to perform classification, an extra learnable "classification token" is added to the sequence. The illustration of the Transformer encoder on the right is inspired by Vaswani et al. [37].

2.2.3. Swin Transformer Architecture

One of the biggest challenges in adapting Transformers from language to vision is due to the computational complexity of self-attention which is quadratic to image size. Models like ViT, for example, compute global self-attention on the processed images, this lead to quadratic complexity with respect to the number of tokens, making it unsuitable in case of large images or for dense prediction like segmentation problems, which requires a huge set of tokens. To overcome this problem Ze Liu et al. presented in [20] a new

vision Transformer called Swin Transformer. The key feature of the Swin Transformer is that it computes self-attention within local non overlapping windows which are shifting between consecutive layers. This results in a more efficient implementation with respect to previous vision transformers which, instead, conduct global self-attention (computing the relationship between a token and all the other tokens).

The Swin Transformer has four main components:

- **Patch Partition**: Th input RGB image is split into non-overlapping patches. Each patch is considered as a "token". The patch size is of 4×4 thus the feature dimension of each patch is $4 \times 4 \times 3 = 48$.
- **Linear Embedding**: it is applied on the raw valued feature obtained by the Patch Partitioning to project it on an arbitrary dimension C which denotes the size or capacity of our model. (We decided to use the Swin-B version where C is = 128)
- **Patch Merging**: reduce the number of tokens, as the network gets deeper, in order to have a hierarchical representation.(The first patch merging layer for example concatenates the features by groups of 2×2 neighboring patches, and then applies a linear layer on the 4C-dimensional concatenated features reducing the output dimension to 2C).
- **Swin Transformer Block**: is replacing the standard multi-head self attention (MSA) module of a Transformer block by a module based on window multi-head self attention (W-MSA) and a shifted windows multi-head self attention (SW-MSA), with other layers kept the same. As illustrated in Figure 3(b). The idea is to divide the image into windows (of $M \times M$ patches) and perform self attention between patches inside of every window. In a second consecutive block the attention windows are shifted with respect to the previous layer, like in strided convolutions, such that some patches that landed into separate windows in the previous layer and couldn't communicate, fall together in a new window on which self-attention is performed, allowing them to communicate, as it can be observed in Fig. 2.13. This is the real novelty introduced by the Swin transformer which tries to mimic the dynamic of CNNs where there is a kernel sliding along the image, differently from previous vision transformers which just computes global self-attention.

In Figure 2.14 we present an image taken from the original paper [20] of the Swin Transformer showing us the whole architecture of the Swin-T model.

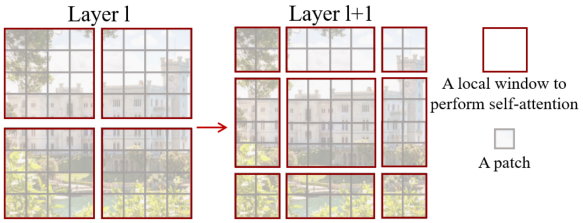


Figure 2.13: On the left the red windows within the first Swin Transformer Block performs self- attention, on the right the shifted windows in the consecutive layer on which the shifted self attention will be performed. The image was taken from [20].

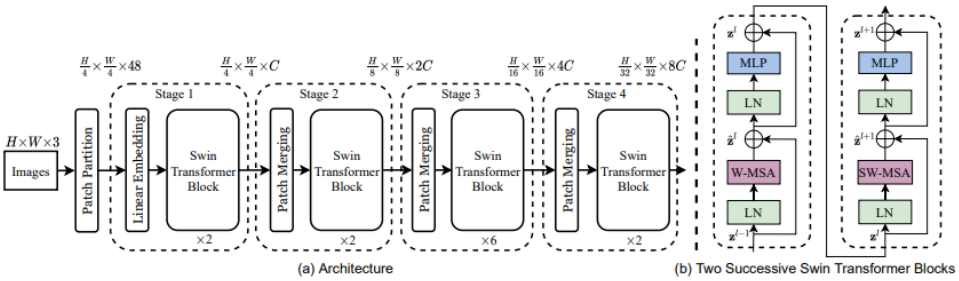


Figure 2.14: In the picture, taken from [20], it is possible to observe the whole architecture of the Swin-T model.

2.3. Semantic Segmentation

In this section we present a brief overview of semantic image segmentation and introduce some of the most commonly used architectures for this task. Image segmentation is a computer vision task in which given an image I we want to label each pixel (r, c) to a corresponding class of what is being represented. The result of segmentation is a map of labels containing in each pixel the estimated class as in Fig. 2.15. Notice that in semantic segmentation we don't separate different instances belonging to the same class, this is done in another kind of problem known as instance segmentation.

The training set for a semantic image segmentation problem is made of pairs (I, GT) where GT is a pixel-wise annotated image over the categories in the picture as in Fig. 2.16. Assume the input image is an RGB image $(h \times w \times 3)$ the goal of the prediction is to output a segmentation map where each pixel contains a class label represented as an integer $(h \times w \times 1)$.

The target values can be created by one-hot encoding the class labels, creating an output channel for each of the possible classes. Then the final prediction can be collapsed into a segmentation map by taking the argmax of each depth-wise pixel vector.

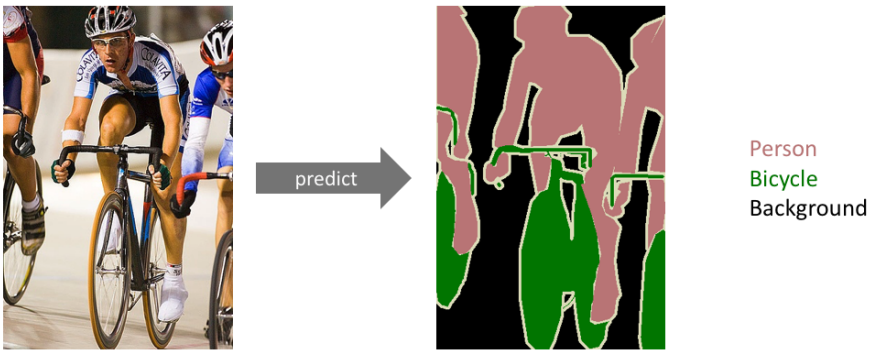


Figure 2.15: An example of semantic segmentation. The goal is to predict class labels for each pixel in the image taken from [21].

Figure 2.16: An image taken from the Microsoft COCO dataset and the corresponding annotated categories for training.

Constructing an architecture for Semantic Image Segmentation

Recall that for Deep CNNs the first convolutional layers are used to learn low-level concepts while the deeper convolutional layers extract more high-level semantic information. Thus, in order to preserve expressiveness, we need to increase the number of feature maps as we go deeper in the network, while for image classification this is not a problem since in deeper layers we augment the number of feature maps but we also downsample them through pooling operations. This is because we just care about **what** is in the image but not where. On the other side, in semantic segmentation we need to care also about **where** in order to produce a full-resolution semantic prediction. One of the most general ideas for image segmentation models is to build an architecture based on an encoder-decoder structure. In the encoder the spatial resolution of the input is reduced (downsampled) while extracting relevant high-level information in order to distinguish the different cate-

gories in the image. Instead, the Decoder part is retrieving the full resolution of the image through upsampling the feature representations provided by the encoder, see Fig. 2.17.

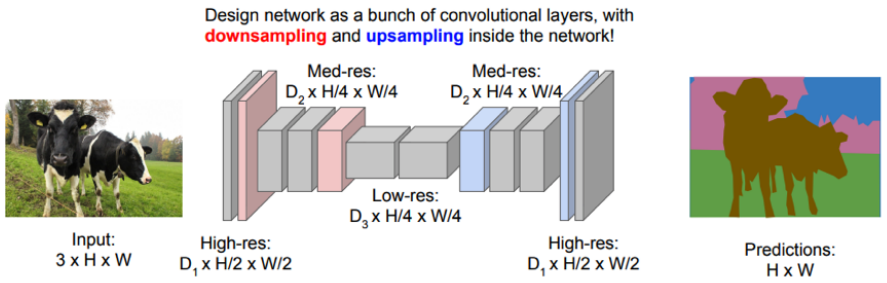


Figure 2.17: An image taken from [21]. A Fully convolutional network made of an encoder and decoder.

There are various approaches that could be used to upsample the resolution of the feature maps in the decoder, as for example: Nearest Neighbor, "Bed of Nails", Max Unpooling, Transpose convolutions etc. We will describe briefly only transpose convolution, since it is the most popular approach due the fact that it allows to learn the upsampling filter weights. In a transpose convolution we take a single value from the input and multiply all of the weights in out filter by this value, projecting these weighted values into the output feature map. In another perspective, as in Fig. 2.18, transpose convolution can be seen as a traditional convolution after having upsampled the input image.

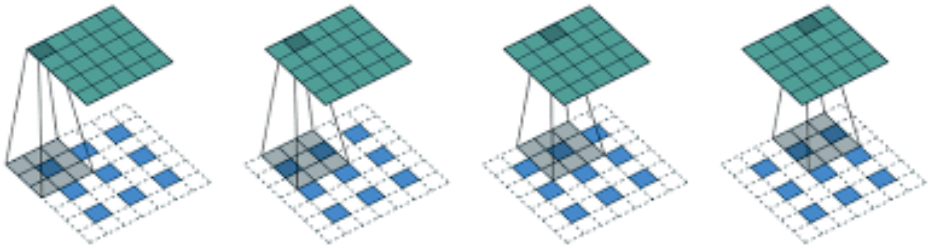


Figure 2.18: The input image (on the bottom)is upsampled in order to explain how a transpose convolution (filter in gray, output in green) works. The image was taken from [4].

The approach of using a fully convolutional network trained end-to-end, for the task of image segmentation was introduced by Long et al. in [21], adapting CNNs used for

[4]Source: `https://github.com/vdumoulin/conv_arithmetic`

classification to serve as the encoder module of the network. Then append a decoder module made of transposed convolutional layers to upsample the coarse feature maps into a full-resolution segmentation map. However, this initial proposal struggled to produce fine-grained segmentations.
As explained by Long et al. in their paper [21] : "Semantic segmentation faces an inherent tension between semantics and location: global information resolves **what** while local information resolves **where**(...). Combining fine layers and coarse layers lets the model make local predictions that respect global structure." For this reason they decided to introduce **skip connections** which connect the output of the upsampling layers in the decoder with the corresponding pooling output from the encoder and sum these two feature maps as it is presented in Fig. 2.19. The skip connections should provide a sufficient quantity of details in order to reconstruct accurate fine-grained segmentations.

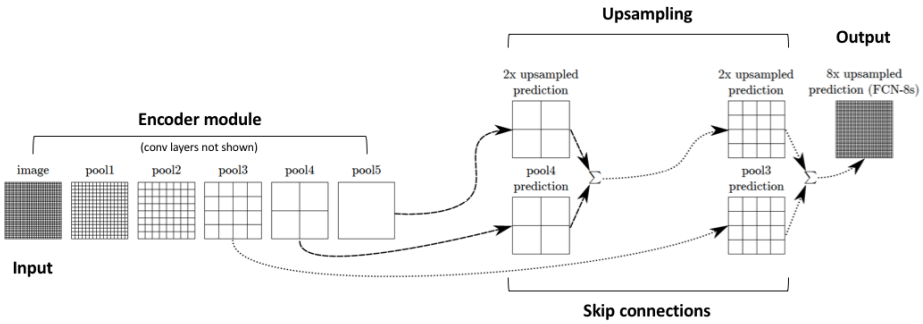


Figure 2.19: Skip connections combining coarse high layer information with fine low layer information. Image taken from [21].

In [29] Ronneberger et al. proposed a new architecture, **U-Net** expanding the capacity of the decoder module, see Fig. 2.20. U-Net doesn't have any fully connected layers, it is made of:

- A contracting path.
- An expansive path (which is symmetric to the contracting path).

Aggregation is performed through concatenation and then convolution is performed to mix different feature maps in a learnable manner.

Another important step in an image segmentation problem is to define an appropriate loss function. One of the most commonly adopted loss function for this task is a pixel-wise cross entropy loss examining each pixel individually, comparing the class predictions to the one-hot encoded target vector. This could be problematic in case of unbalanced classes in the image, since cross entropy loss is essentially asserting equal learning to each

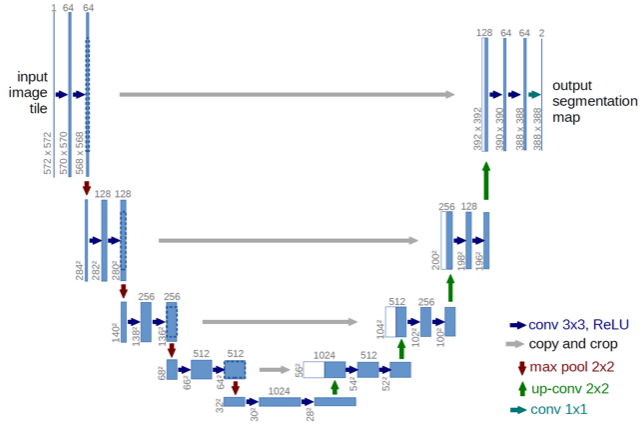


Figure 2.20: U-Net architecture taken from [29].

pixel in the image, thus training could be dominated by the most frequent class. In [21], Long et al. propose weighting this loss for each output channel in order to counteract eventual class imbalance situations.

Another interesting proposal was made by Ronneberger et al. in [29], introducing a higher weight at the border of segmented objects. This loss weighting scheme allowed U-Net to segment cells in biomedical images in a discontinuous fashion such that individual cells may be easily identified within the binary segmentation map.

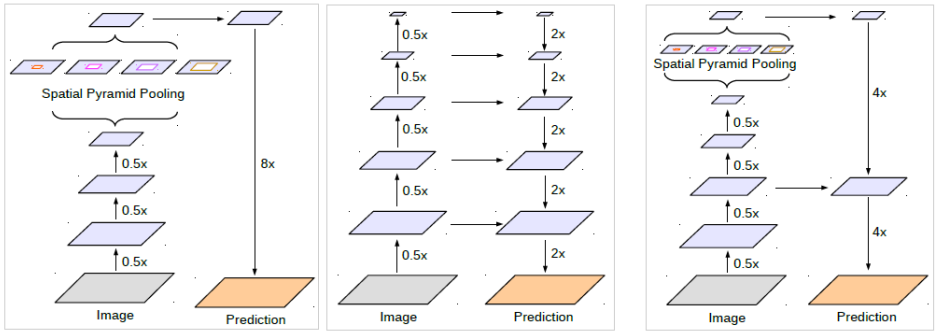
Spatial pyramid pooling module or encode-decoder structure are used in deep neural networks for semantic segmentation tasks. The former networks are able to encode multi-scale contextual information by probing the incoming features with filters or pooling operations at multiple rates and multiple effective fields-of-view, while the latter networks can capture sharper object boundaries by gradually recovering the spatial information. In the next subsection we present an architecture which combines the advantages from both methods.

2.3.1. DeepLabv3+

DeepLabv3+ [6] is a semantic segmentation architecture based on previous DeepLabv3 and adding a simple yet effective decoder module, Fig. 2.21, to enhance segmentation results.

We give a short explanation about the main components of DeepLabv3+:

- **Atrous Separable Convolution**: can be decomposed into two main concepts:



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

Figure 2.21: The architectures of DeepLabv3 is presented in picture (a), it employs a spatial pyramid pooling module, in picture (b) an encoder-decoder structure is presented. The proposed model, DeepLabv3+, makes use of (a) and (b) containing rich semantic information from the encoder module, while detailed object boundaries are recovered by the additional introduced decoder module. The encoder module allows to extract features at an arbitrary resolution via atrous convolution. The image was taken from [6].

1. **Atrous Convolution**: introduces another parameter to traditional convolutional layers called the dilation rate, which defines a spacing between the values in a kernel. E.g: a 3×3 kernel with a dilation rate of 2 will have the same field of view of a 5×5 kernel, while only using 9 parameters. The result is a wider field of view, as it can be seen in Figure 2.22 at the same computational cost.

2. **Depth separable convolution**: integrates ordinary convolution into two steps: The first step is to do convolution for each channel separately, which is the depthwise (a) in Figure 2.23. The second step is to combine the convolution results of the first step with 1×1 convolution and channel, which is the pointwise in (b) in Figure 2.23. In other words, the first step consists of performing convolution on each channel, so that the number of channels remains unchanged. The second step uses 1×1 convolution to adjust the number of channels reducing drastically the computation complexity.

To resume, Atrous Separable Convolution mentioned in DeepLab v3+ refers to changing the first step of deep separable convolution to atrous (dilation) convolution.

- **Encoder-Decoder Architecture**: can also be decomposed into two main components:

1. DeepLabv3 as Encoder. It adopts an output stride = 16 (or 8) for denser

[5] Source:https://towardsdatascience.com/review-deeplabv3-atrous-convolution-semantic-segmentation-6d818bfd1d74

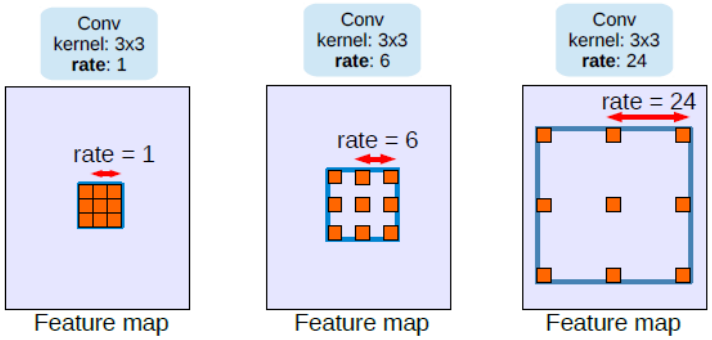


Figure 2.22: Atrous convolutions with different rates [5]

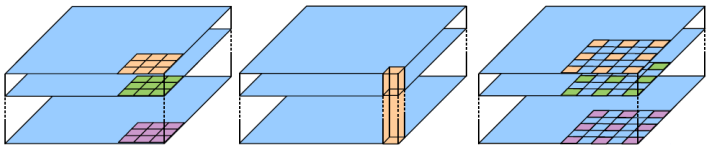


Figure 2.23: In this picture, taken from [6], a 3×3 Depthwise separable convolution decomposes a standard convolution into (a) a depthwise convolution (applying a single filter for each input channel) and (b) a pointwise convolution (combining the outputs from depthwise convolution across channels). The Atrous separable convolution where atrous convolution is adopted in the depthwise convolution is shown in (c) with rate $= 2$.

feature extraction by removing the striding in the last one (or two) block(s) and applying the atrous convolution correspondingly. Additionally, DeepLabv3 augments the Atrous Spatial Pyramid Pooling module, which probes convolutional features at multiple scales by applying atrous convolution of different rates, with the image-level features.

2. A Decoder responsible of recovering object boundaries. The encoder features are first bilinearly upsampled by a factor of 4 and then concatenated with the corresponding low-level features. A 1×1 convolution on the low-level features is performed before concatenation to reduce the number of channels, since the corresponding low-level features usually contain a large number of channels (e.g., 256 or 512) which may outweigh the importance of the rich encoder features. After the concatenation, a few 3×3 convolutions are applied to refine the features followed by another simple bilinear upsampling by a factor of 4.

- **Modified Aligned Xception**: pretrained on ImageNet1K is employed to extract dense feature maps by atorus convolution. The original Xception model was in-

troduced for Image classification, while at a later time a modified version called Aligned Xception was adopted in Deformable Convolutional Network (DCN) for object detection.

Most of the novelties introduced by Aligned Xception, in blue color in Fig. 2.24, consist of replacing some of the max pooling operations with separable convolutions in the entry flow. The number of repeating is increased from 8 to 16 in the middle flow. An additional convolutional layer is added in the exit flow.

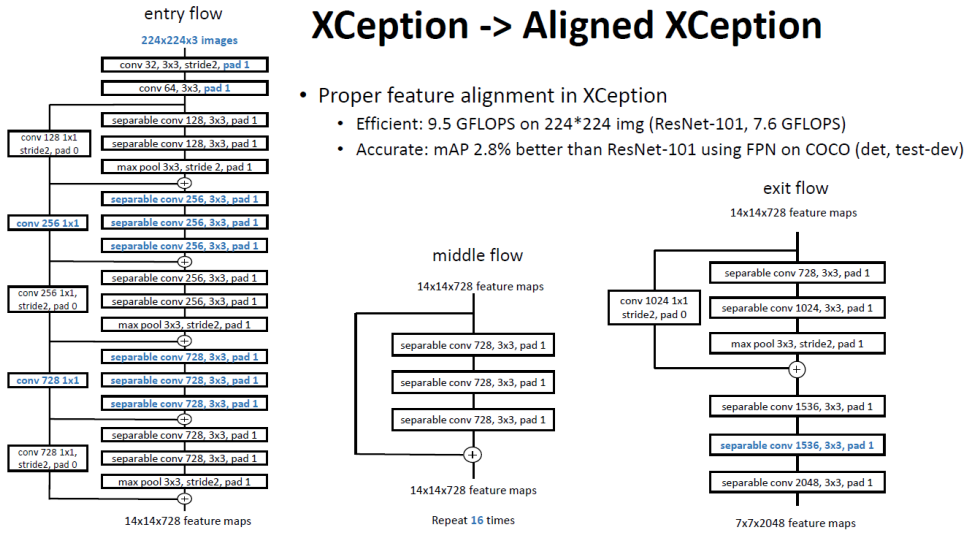


Figure 2.24: The blue text represents the modification introduced by Aligned Xception with respect to the original Xception architecture [6].

Modified Aligned Xception is a deeper version of the previous Aligned Xception where all the max pooling operations are replaced by depthwise separable atrous convolutions. Extra batch normalization and ReLU activation layer are added after each 3×3 depthwise convolution as it can be seen in Figure 2.25.

2.4. Explainable AI

The ability to correctly interpret a prediction model's output is extremely important. It engenders appropriate user trust, provides insight into how a model may be improved, and supports understanding of the process being modeled.

[6] Source:https://sh-tsang.medium.com/review-deeplabv3-atrous-separable-convolution-semantic-segmentation-a625f6e83b90

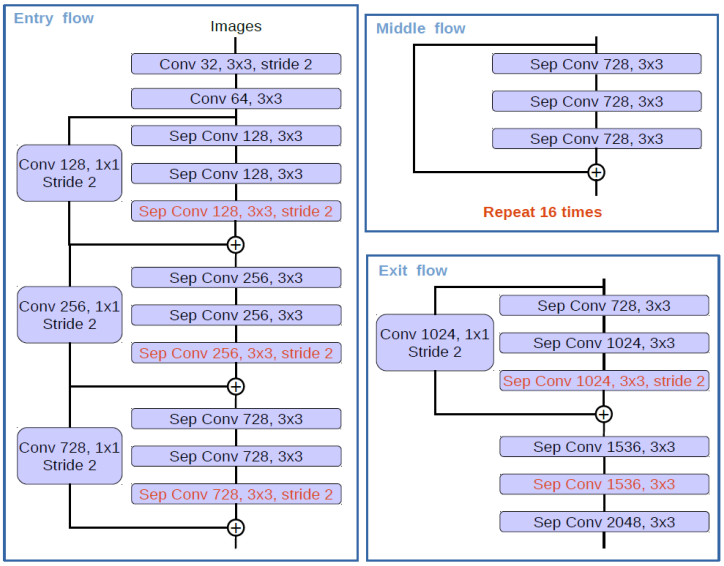


Figure 2.25: Architecture of the Modified Aligned Xception model. The image was taken from [6].

2.4.1. SHAP

Before explaining how SHAP works we will first explain the mathematical foundation it is built on. Shapley value was introduced in 1951 by Lloyd Shapley as a way of providing a fair solution to the following question: "If we have a coalition C that collaborates to produce a value V, how much did each individual member contribute to the final value?". Given a set N of n players and a function v that maps subsets of players to a real value: $v : 2^n \rightarrow \mathbb{R}$, with $v(\emptyset) = 0$; If S is a coalition of players, $v(S)$ represents the worth of the coalition, which is the total expected payout that can be obtained by the members of S in cooperation. Assuming all the players in the coalition collaborate, Shapley value is a way to distribute the total gains (payout) to the players, yielding a "fair" distribution in the sense that it is the only distribution with certain desirable properties listed below.

$$\phi_i(v) = \sum_{S \subseteq N \setminus i} \frac{|S|!(n - |S| - 1)!}{n!}(v(S \cup i) - v(S)) \tag{2.12}$$

Where the Shapley value $\phi_i(v)$ is the amount that player i gets given a coalition game (v, N), n is the total number of players and the sum is done over all the subsets S of N not containing player i. Moving from game theory to a Machine Learning framework, we can interpret the features of a data instance as players in a coalition, thus, given an input instance x, the goal of **SHAP** [22], is to explain the prediction by computing the contribution of each feature to the prediction. A player can be an individual feature value

or also a group of feature values; in case of images, for example, pixels can be grouped to superpixels and the prediction distributed among them.
SHAP represents the Shapley value as an additive feature attribution method (a linear model). For complex models, such as deep neural networks, we cannot use the original model as its own best explanation, as we could do in case of a simple linear regression. Instead, we must use a simpler surrogate explanation model, which could be any interpretable approximation of the original model. Call g the explanation model:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{2.13}$$

$z' \in \{0,1\}^M$ is the coalition vector (also called simplified features e.g. in case of images), where M is the maximum coalition size and $\phi_j \in \mathbb{R}$ is the Shapley value for a feature j. In the coalition vector, an entry of 1 means that the corresponding feature value is "present" while 0 mean that it is "absent", as when we compute Shapley values, where we simulate that only some players (feature values) are playing ("present") and some are not ("absent"). In the [22], you will find discrepancies between SHAP properties and Shapley properties (Efficiency, Symmetry, Dummy and Additivity). SHAP describes the following three desirable properties:

1. **Local accuracy**: $f = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j$, where f is the original model, thus the explanation model $g(x')$ matches the original model $f(x)$ when $x = h(x')$, where x is the original input to which the simplified feature x' is mapped via a function h.

2. **Missingness**: $x' = 0 \Rightarrow \phi_j = 0$, which means that a missing feature gets an attribution of zero.

3. **Consistency**: Let $f_x(z') = f_x(h_x(z'))$ and $z'_{\setminus j}$ indicate that $z'_j = 0$. For any teo models f and f' that satisfy:

$$f'_x(z') - f'_x(z'_{\setminus j}) \geq f_x(z') - f_x(z'_{\setminus j}) \tag{2.14}$$

for all inputs $x' \in 0, 1^M$ then:

$$\phi_j(f', x) \geq \phi_j(f, x). \tag{2.15}$$

The consistency property states that if a model changes so that the marginal contribution of a feature value increases or stays the same (regardless of other features), the Shapley value also increases or stays the same.

Young (1985) demonstrated that Shapley values are the only set of values that satisfy three axioms similar to Local accuracy, Consistency, and another property that is redundant in this setting. SHAP also satisfies these, since it computes Shapley values. Missingness is required to adapt the Shapley proofs to the class of additive feature attribution methods.

Interpreting SHAP feature importance values is quite simple: features with large absolute Shapley values are important. Since we want the global importance, we average the absolute Shapley values per feature across the data.

2.4.2. Integrated Gradients

Integrated Gradients, [36], is an axiomatic model interpretability algorithm that is used for measuring attribution of input features to the output of a neural network. It belongs to the family of Gradient based methods which are based on the gradients with respect to the input of each layer, computed during backpropagation. These methods often multiply the gradient by the input activations, as done by **Gradient*Input** method [31]. Integrated Gradients in addition takes the derivatives of the value for the predicted class with respect to the input features. This is done along a straight line from a certain baseline (which is a representation of an input that reflects the absence of signal, e.g. a black image or an zero-embedding) to the actual input. In implementations, the Riemann sum is mainly used to approximate the integral from the baseline to the actual input. We first formalize the problem of attributing the prediction of a deep neural network to its input features, followed by a more detailed explanation of Integrated Gradients.

Definition 1. *Formally, suppose we have a function* $F : \mathbb{R}^n \to [0, 1]$ *that represents a deep neural network, and an input* $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$*. An attribution of the prediction at input* x *relative to a baseline input* x' *is a vector* $A_F(x, x') = (a_1, \ldots, a_n) \in \mathbb{R}^n$ *where* a_i *is the contribution of* x_i *to the prediction* $F(x)$*.*

In case of an image classification network, an attribution method could tell us which pixels of the image were responsible for a certain label being picked.

Remark 1. *A common way for humans to perform attribution relies on counterfactual intuition. When we assign blame to a certain cause we implicitly consider the absence of the cause as a baseline for comparing outcomes. In a deep network, we model the absence using a single baseline input. For most deep networks, a natural baseline exists in the input space where the prediction is neutral. For instance, in object recognition networks, it is the black image.*

Integrated Gradients design is guided by two axioms which according to the paper [36], must be satisfied by attribution methods:

1. **Sensitivity**: An attribution method satisfies Sensitivity(a) if for every input and baseline that differ in one feature but have different predictions then the differing feature should be given a non-zero attribution. Sensitivity(b). If the function implemented by the deep network does not depend (mathematically) on some variable, then the attribution to that variable is always zero.
2. **Implementation Invariance**: Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations.

Notice that if an attribution method fails to satisfy Implementation Invariance, the attributions could be sensitive to unimportant aspects of the models.

We can now define Integrated Gradients as follows :

$$IntegratedGrads_i(x) := (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x')}{\partial x_i} \, d\alpha \tag{2.16}$$

Integrated Gradients satisfies an additional axiom which is **Completeness**, which states that the attributions sum up to the difference between the output of F at the input x and the baseline $x_{\prime}$.

This is formalized by the following proposition, which instantiates the fundamental theorem of calculus for path integrals.

Proposition 2.1. *If $F : \mathbb{R}^n \to \mathbb{R}$ is differentiable a.e. then: $\sum_{i=1}^{n} IntegratedGrads_i(x) = F(x) - F(x')$.*

In most of the cases it is possible to choose a baseline such that the prediction at the baseline is near zero. In case of image classification models, the black image baseline satisfies this property. Integrated Gradients can be visualized by aggregating them along the color channel and scaling the pixels in the actual image by them as in Fig. 2.26.

2.4.3. Grad-CAM

Grad-CAM [30] is a very popular method for producing post-hoc visual explanations on an already-trained neural network. It can be thought as a generalization of: CAM (Class Activation Maps) [43], which is another explanation method used for explaining Neural Networks with a particular architecture. CAM, in fact, requires an architecture that replaces the traditional stack of fully connected layers with a Global Average Pooling

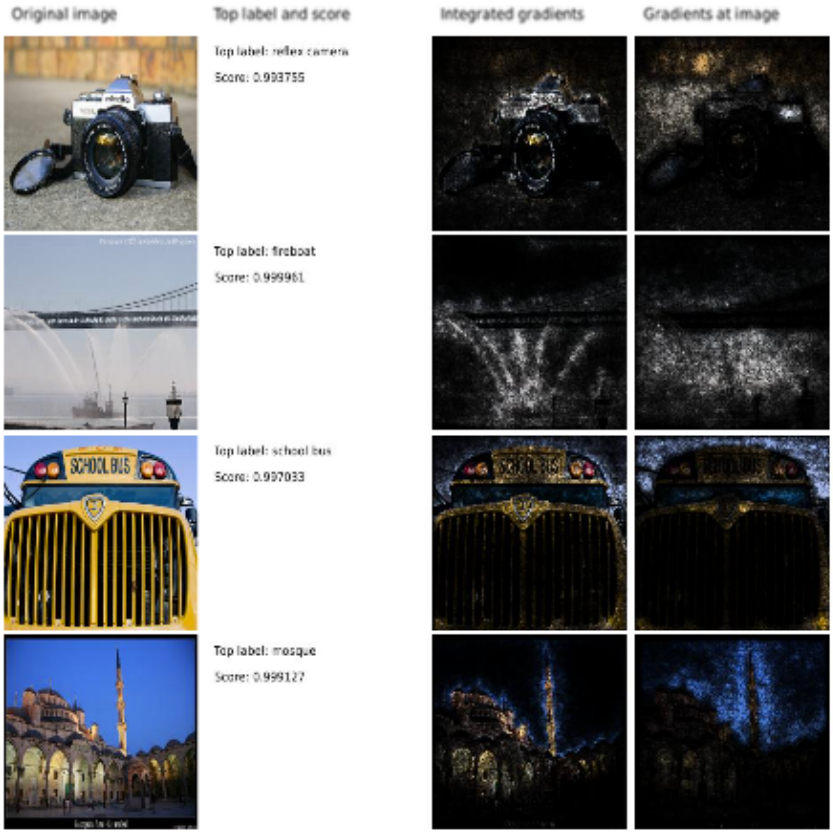


Figure 2.26: A comparison, taken from [36], of Integrated Gradients with gradients at the image. On the left the original input image, followed by label and Softmax score for the highest scoring class. On the third column Integrated Gradients are showed and, on the right of it, visualization of gradients*image. Visualization of Integrated Gradients are better at reflecting distinctive features of the image.

(GAP) which averages the activations of each feature map (suppose we have $h \times w \times d$ feature maps) and concatenates these averages and outputs them into a $1 \times 1 \times d$ vector(where d corresponds to the number of feature maps). Then, a weighted sum of this vector is fed to the final Softmax loss layer. An illustration of this architecture is presented in Fig. 2.27. This approach allows to highlight the important features of an image for the corresponding classification task by projecting back the weights of the output on the convolutional feature maps.

Grad-CAM, diversly from CAM, can be used to produce visual explanations on any arbitrary CNN architecture. The output produced by Grad-CAM is a "class-discriminative" localization map i.e. a heathmap where the hot part corresponds to the specific targeted class. Also for Grad-CAM, the basic idea is to exploit the spatial information which is preserved through convolutional layers so to identify the parts of an input image which

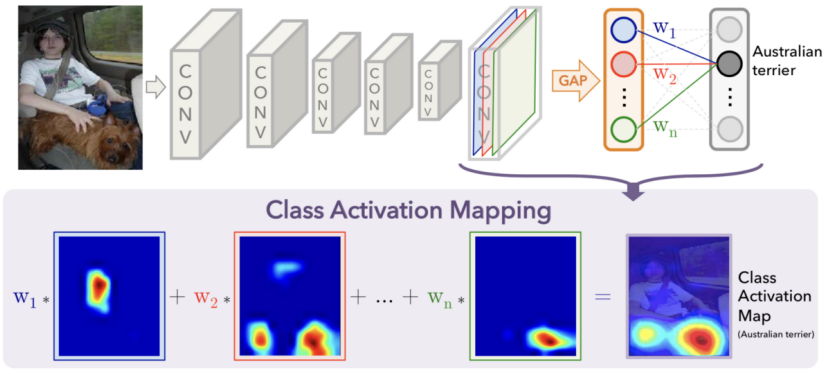


Figure 2.27: CAM architecture, taken from the original paper of Zhou et al [43].

mostly influenced the classification decision. As for CAM, Grad-CAM exploits the feature maps produced by the last convolutional layers of a CNN, which have the best compromise between high-level semantics and detailed spatial information.

Call y^c the logit for class c, before the Softmax, Grad-CAM method can be divided in three main steps:

1. Compute gradients of y^c with respect to the feature map activations A^k with $k = 0, \ldots, N$ and N = number of feature maps in the targeted layer. Assuming that each feature map has a shape $h \times w$, the gradients: $\frac{\partial y^c}{\partial A^k}$, have the same shape as the feature maps. So in case we have N feature maps in the targeted layer we will have $h \times w \times N$ gradients at the end of this step.

2. Calculate α weights by averaging the gradients. For each feature map we perform a GAP on the gradients over the width and height dimension, so to obtain the following neuron importance weights: $\alpha_k^c = \frac{1}{z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$ with $i = 0, \ldots, w$ and $j = 0, \ldots, h$. So, at the end of this step we end up with k different α coefficients, one for each feature map.

3. Calculate The final Grad-CAM heatmap. It is produced via a weighted combination of the feature map activations A^k where the weights are the α_k^c (While in CAM the weights applied to A^k are w_k of the final fully connected layer, here we use the α_k). Then we apply a ReLU function on the linear combination to emphasize only positive values and turn negatives to 0. $L^c_{Grad_CAM} = ReLU(\sum_k \alpha_k^c A^k)$.

In Fig. 2.28 we present some results of Grad-CAM taken from the paper [30] by Selvaraju et al.

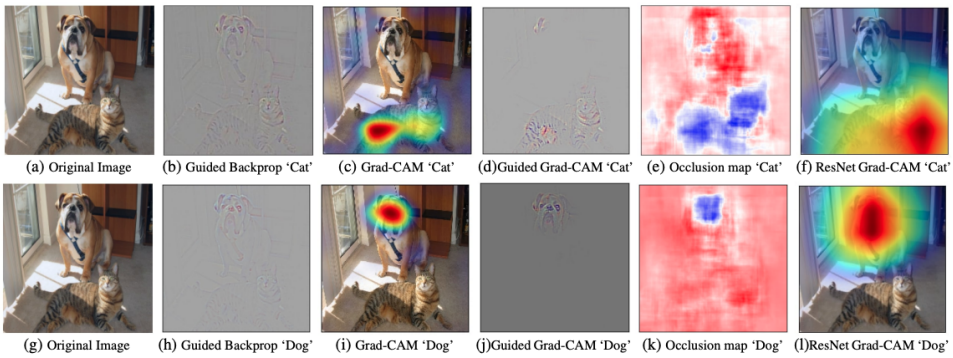


Figure 2.28: In (a) the Original image with a cat and a dog. In (b) Guided Backpropagation is shown, which highlights all contributing features. (c, f) present Grad-CAM for the targeted Cat category: localizes class-discriminative regions. In (d) a combination of both Grad-CAM and Guided Backpropagation. (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. In (e) occlusion sensitivity method is shown. (c,i) are Grad-CAM visualizations for VGG16. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Red regions in (c, f, i, l) correspond to high score while in (e, k), which uses occlusion sensitivity, blue regions correspond to evidence for the class.

2.5. Image Geolocation

The image geolocation problem is a very challenging task which began to arouse particular interest around 2000, mainly focusing on outdoor geolocation. We will present some of the most important works done in this field as IM2GPS [14] which is one of the first data-driven models based on Machine learning techniques; PlaNet [40] adopting deep learning architectures to solve the problem as a classification task, and Müller-Budack et al. [24] hierarchical model which mostly inspired our work.

2.5.1. IM2GPS and Improved IM2GPS

IM2GPS model is based on the idea of identifying correlations between specific image features and geographic locations. For this reason the authors created a huge geo-tagged database of image features extracted from more than six million images taken from significant places all over the world, to be used for comparison with a query image at inference time. More specifically, to make a prediction on a test image, features like color, Gist descriptor, texture histograms and geometric context are extracted automatically with a data-driven approach . Then the model is supposed to find the best match to the query image features with the features present in the database adopting a k-Nearest Neigh-

bour (k-NN) algorithm. With the advent of deep learning Hays et al. [39] improved the previous IM2GPS model exploiting a Convolutional Neural Network for the feature extraction part. Moreover, in this revisited IM2GPS model predictions at inference time are computed via a k-NN density estimation based on a Gaussian kernel that takes into account: the label similarity (GPS distance) and at the same time also the query image and reference image features, see Fig. 2.29.

Figure 2.29: An overview of revisited IM2GPS architecture [39]. Features are extracted with a CNN then nearby neighbors are found in the reference features database and finally GPS coordinate is estimated using either the k-NN or the density.

2.5.2. PlaNet

The "revolution" introduced by CNN architectures led some models such as PlaNet [40] to approach the geolocation problem as a classification task. The world map is split into a number of cells which corresponds to the different classes. The cell generation is done with the S2 Google's algorithm [12], setting a minimum and maximum thresholds of images per cell in order to have balanced classes. The adopted backbone for feature extraction is based on Inception. The loss function used for training is a customized function based on the euclidean distance between the centroid of the predicted cell and the real coordinates of the inspected image. Moreover Weyand T. et al. extended, in the same paper, the geolocation task to albums of photos, so to highlight the location correlation between images from the same set. The results provided by PlaNet are much better than the ones of IM2GPS: this is due mainly to the huge amount of images used for training, around 189 million images, but also to the innovative classification approach to the geolocation problem.

2.5.3. Hierarchical Model and Scene Classification

Müller-Budack et al. in [24] propose a model which also exploit Google S2 algorithm for the subdivision of the world map into cells, but instead of producing only one subdivision as PlaNet, they produce in parallel three different subdivisions (coarse, middle, fine) by varying the thresholds parameters of the algorithm. The model uses also a multi-task Network (based on ResNet [15]) trained on both geographical information and scene

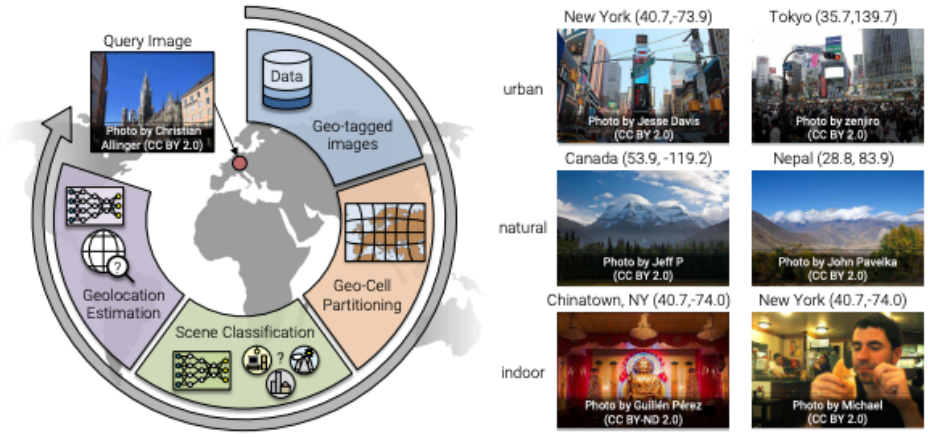


Figure 2.30: On the Left: Workflow of the geolocation estimation approach proposed by Müller-Budack et al [24]. On the Right: a few sample images from different locations for specific scene concepts.

labels (i.e., indoor, natural, or urban setting etc), see Fig. 2.30. In order to estimate the GPS coordinates from the classification output,the trained multi-task Network is applied on three evenly sampled crops of a given query image according to its orientation. Afterwards, the mean of the resulting class probabilities of each crop is calculated, see Fig. 2.31. In the paper it is shown that the geolocator benefits from the multi task approach which incorporates features extracted by the scene classifier. An explanation for the improvement due to the scene classifier is that exploiting contextual information of the environmental scenario can help in classification at different level of granularity due to the huge diversity caused by different environmental settings, which requires specific features to distinguish different locations. For example while at country level we might deal with scenery such as mountains, beaches or forests, at city level we mainly deal with urban scenes including people, buildings, traffic lights and cars.

2.5.4. A Global Hotel Recognition Dataset

Stylianou et al. where among the first to address the indoor geolocation problem. The need to solve this problem is linked to the fact that in recent years, the number of images of victims of human trafficking circulating online has drastically grown. For this reason Stylianou et al. in [35] present a new dataset formed by more than 1 million annotated hotel room images, of different quality, from 50,000 different hotels. The paper presents two different approaches to the problem; the first addresses the problem as a classification task using as backbone a pre-trained ResNet-50 [15] model pre-trained on ImageNet to

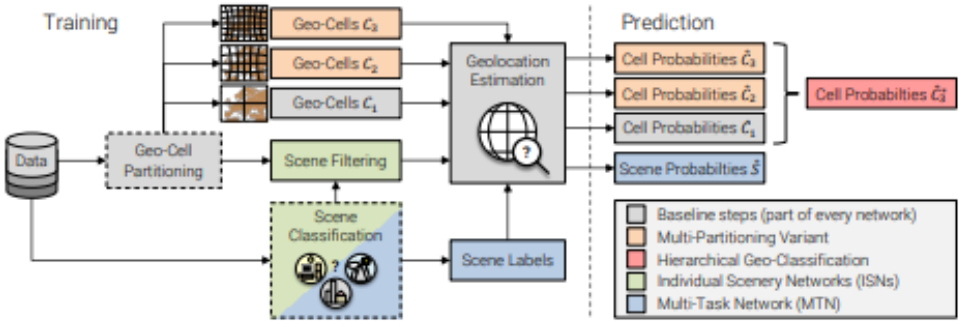


Figure 2.31: Geolocation estimation frameworks proposed by Müller-Budack et al. Image taken from [24]. In grey: Baseline steps that are part of every network. Additional steps are visualized in different colors. Dashed elements are applied to all images before the training process takes place. The blue multi task approach is the best performing one.

output a 256-dimensional features vector to be used by the classifier in order to predict the hotel instance or the chain label. The authors also performed a set of data augmentation techniques specifically tailored for this task. The second approach consists in addressing the problem as an image retrieval task by returning the most similar image in the reference set. This is done by computing the feature representations for all images in the Hotels-50K training set using the previous mentioned network. Then feature representations are also computed for each image in the test set and the database images are ranked by cosine similarity to each test image. In both approaches the authors compared their model with two "off-the-shelf" pre-trained networks, improving their performances, trained for scene and object recognition.

3 | Research Questions

This section is dedicated to the research question that we address in this thesis work. In particular:

1. Is it possible to develop a geolocation model, which could be useful in applications such as law enforcement, capable of geolocating indoor scenes at different levels of granularity with a meaningful level of accuracy?
2. Is it possible to provide meaningful and useful visual explanations for the predictions made by our model?
3. What are the most appropriate backbone and dataset to adopt for the feature extractor of an indoor geolocation model?

Indoor scene geolocation is a very challenging fine-grained visual recognition task which, in cases like the Hotel50k dataset, presents a huge number of classes (e.g. 10458 different cities) and potentially high intraclass and low interclass variation. Due to the difficulty of this task, only few works exist in literature that focuses on this problem. Stylianou et al. in [35] were the first to introduce a huge dataset for this task, and try different Deep learning models. Recently a competition called: "Hotel-ID to Combat Human Trafficking 2021", was released in Kaggle. In this contest, competitors are tasked with identifying the hotel seen in test images from a subset of the Hotel-50k dataset. Also Virginia Negri in her master thesis [25] built a model for the indoor geolocation task. In our work we remarkably improve the performances of previous models, obtaining a higher accuracy at different level of granularity (subregion, country, city and identifying the hotel chain).

Deep Learning models are very powerful tools for classification task of unstructured data, but unfortunately since they extract the features in a data driven way, we are not able to understand just by the results which elements of the input mostly influenced the predicted output. Thus, in order to answer to the second question, we adopted different XAI (Explainable Artificial Intelligence) techniques with the aim of identifying the most relevant scenes for the prediction. We applied visual explainability methods at the three different levels of granularity of our classifier, trying to understand which elements could

influence the prediction at world sub-region level, then at country level and finally at city level.

As far as the third question is concerned, we tried different architectures, pretrained on different datasets rather than only use a pretrained network on ImageNet, which is the most used dataset for pretrained Networks addressed to solve classification problems. To conclude and show the generalization capabilities of our model we also provide users with a demo that produces top ten predictions at country level and also visual explanations with SHAP [22], in order to highlight the most informative scenes in the uploaded image.

4 | Approach and Model architecture

In this section we present our multi-task hierarchical approach to the indoor geolocation problem, allowing us to simultaneously maximise geolocation performance at different levels of granularity (world sub-region, country and city).
The diagram in Fig. 4.1 represents the overall architecture of our model. The architecture that we adopted for our multi-task framework can be divided mainly into four components:

1. A Vision Transformer for feature extraction, more specifically the Swin Transformer.
2. A fully-connected network for multiple parallel classifications at different granularity levels.
3. An additional Network for accurate multi-class semantic segmentation of indoor objects.
4. A block for visual explanation exploiting the segmentation of the input image in order to give an explicit visual explanation of the most discriminating parts of the scene for each granularity level.

4.1. Image encoder

The input to the image encoder are batches of 16 RGB resized images of size: $(224, 224, 3)$. After an ablation study we choose to use as a feature extractor the Swin Transformer [20] pre-trained on ImageNet22K [8]. The key feature of the Swin Transformer is that it computes self-attention within local non overlapping windows which are shifting between consecutive layers. More specifically, the input image is divided into non-overlapping windows, so that within the same window the collection of keys used by different queries is the same. Then, in the next layer, the window shifts half a window downwards and to the right, so as to construct connections between pixels of different windows from the previous layer. This results in a more efficient implementation with respect to previous vision

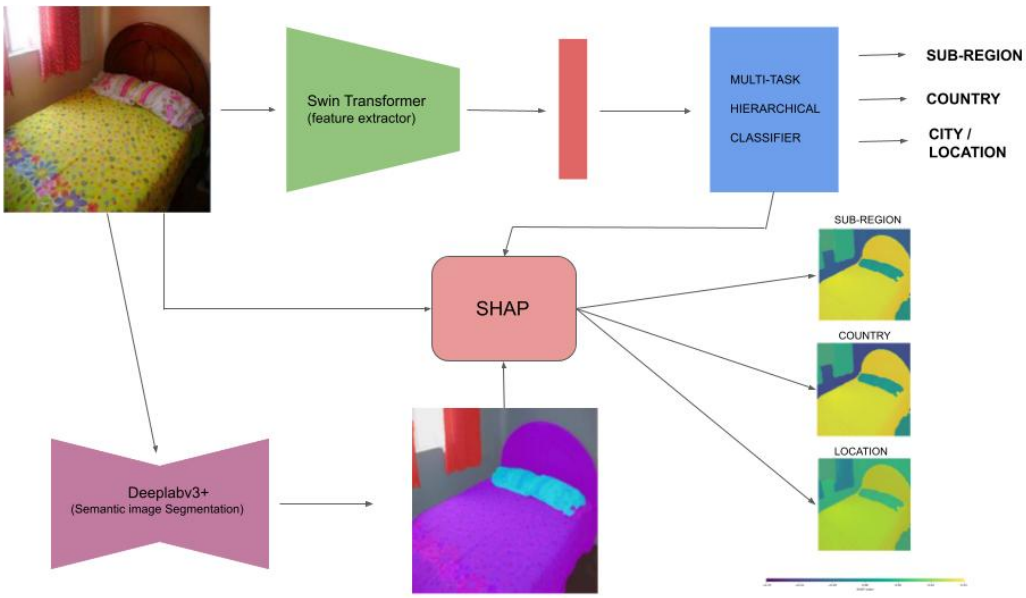


Figure 4.1: Overall architecture of our Hierarchical Indoor Geolocation model.

transformers which conduct global self-attention, computing the relationship between a token and all the other tokens. The global computation leads to quadratic complexity with respect to the number of tokens, making it unsuitable in case of large images or for dense prediction like segmentation problems, requiring an immense set of tokens. Another reason why we chose to use a vision transformer as model backbone for this type of classification problem is because of the general modeling capability of Transformers. Computer vision tasks involve the interaction of two basic elements which are: pixels and objects. So when performing computer vision tasks on an image we have to take into account the relationship between: pixel-to-pixel, object-to-pixel and object-to-object. While the first relationship is often regarded with convolution in CNNs, and the second is usually addressed with RoI Align in segmentation problem, the third relationship is often disregarded. The attention unit in Transformer has the advantage that it can be applied to the three mentioned relational modeling types at the same time.

4.2. Multi-level classifier

The output feature extracted by the feature encoder are flattened into a single vector of shape $(1, 50176)$ which is then fed to a feed-forward neural network (FNN) for classification. We tested different configurations for the FNN part, as it can be seen in the experiment Chapter 6. For the Hotel50k dataset our ultimate classifier is made up of a direct connection from the flattened layer to the city classifier, it includes an additional

head for the chain-hotel classification which, as the country and sub-region classifiers, has two additional dense layers with 500 units as output and a ReLu layer in between in order to reduce the number of units in input to the final classification layer. For the Airbnb dataset, due to the reduced number of labels of each class, we used the same FNN for all the classification tasks (location, country, sub-region), again made of two dense layers of 500 neurons and a ReLu activation layer in between. The final Result in both cases is a model which can predict concurrently the city (or location in case of Airbnb), country and sub-region to which the scene in the image belongs.

4.3. Multi-class Semantic Segmentation Network

Simultaneously we add also an extra block to our framework, which is a Network for segmenting the indoor scenes. We introduced this block to provide more explicit visual information about pieces of furniture and structural elements of interior scenes which might have a considerable importance in order to connote a specific location. More specifically, we adopted a DeepLabv3+ [6] framework to perform semantic segmentation, deploying the Xception model pretrained on ADE20k dataset [44]. The ADE20k includes annotated images covering a wide range of scene categories including some useful objects for our task as: air conditioning, bed, door, window, lamp, power plug etc. Fig. 4.2 presents an example of an image taken from the Airbnb dataset segmented by our model where all the present categories were correctly detected, providing an accurate segmentation.

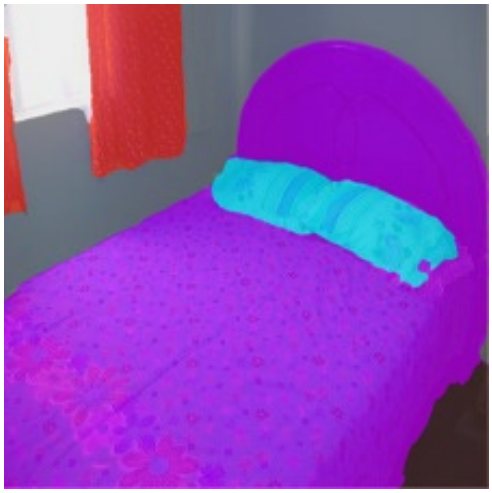


Figure 4.2: Semantic segmentation of an image taken from an Airbnb located in Rio, overlaid on the original image. There are six correctly predicted categories which are: bed, wall, curtain, pillow, windowpane and floor.

4.4. Segmentation-based visual explainer

Figure 4.3: SHAP Values associated to the segmented objects detected by the segmentation block. The image is the one presented in the previous section, it is taken from an Airbnb located in Rio. There are six correctly predicted categories which are used to partition the image, and obtain corresponding SHAP values to explain the prediction.

The segmentation block turns to be very helpful especially when exploited to partition the image to be analysed with SHAP [22], in order to produce visual explanations of the predicted outcome. One of the main drawbacks of SHAP is that, when used to produce explanations at pixel level, it is computationally expensive. For this reason a natural way to use it is by partitioning the image into a grid of super-pixels and consider coalitions of super-pixels. This approach is reducing the computational effort but doesn't produce easily interpretable results for our task. Or more correctly, interpretable results can be obtained if the image is partitioned into many super-pixels, e.g. 100, which may include also meaningful small objects for the predictions. This, however, makes the computation of SHAP algorithm rather onerous. For this reason we decided to use the semantic segmentation of the input image as a partitioning scheme on which SHAP assigns Shapley values directly to furniture and structural elements with semantic value like: window, floor, ceiling, etc. In Fig. 4.3 elements in yellow represent positive SHAP values that contributed to classifying that image as that particular class, scaling up to elements in blue and purple to which are associated negative SHAP values that contributed to not classifying that image as that particular class. A more detailed description of this method with various visual experiments is presented in Chapter 6, where we present the results of the algorithm on the top-3 most probable classes for each category: sub-region, country and city, showing which elements had the greatest influence on the prediction of each class. In this way, in case the correct prediction is in the top-3 predicted classes we are able to see which elements have most contributed to that prediction, and at the same time also to point out the ones that may have led our model to incorrect predictions.

5 | Datasets

5.1. Indoor geolocation datasets

In order to address different problems related to indoor geolocation, such as human trafficking or the presence of fake ads on websites as Airbnb, Booking or Idealista, we used two different dataset to train our model: the Hotel50k dataset presented by Stylianou et al. in [35] and the Airbnb dataset. In addition we also created a dataset, by scraping from different web sites of real estate agencies, to test the inference capabilities of our model.

Hotel50k

The Hotel50k dataset presented in [35], consists of 1.3 million geo-tagged images of interior scenes of more than 45k hotels. This dataset was collected in order to result potentially helpful for law enforcement in fighting human trafficking. The images are collected from two different websites: Expedia and TraffickCam. Expedia is a travel website showing high quality images of hotels, see Fig. 5.3. While TraffickCam is a website, developed by Exchange Initiative (an organization fighting against sex trafficking), displaying images uploaded by users, usually the images are low quality and with occasional clutter obstructing the scene. Thus the TraffickCam images, see Fig. 5.2, are more similar to real world pictures of human trafficking with which the police usually has to deal. For this reason images from TraffickCam make the training process harder but on the other side they may result very useful for testing our model capabilities. The Hotel50k dataset contains information about the latitude and longitude of the images and also the chains, as Hilton, Marriot, Wyndham, etc., to which the hotel belongs (or "no chain" in case the hotel doesn't belong to any chain). Since we decided to approach the indoor geolocation problem as a classification task with a hierarchical model, the latitude and longitude coordinates of each image have been used to extract information about the corresponding city and country with the python reverse-geocode library. Then the remaining coarser level, sub-region, was assigned according to the M49 (or Standard Country or Area Codes for Statistical Use) coding classification 5.4, which is a standard for area codes used by the

United Nations for statistical purposes. Then a unique numerical identifier was associated to each label of each category. The Hotel50k dataset contains a total of 1294289 images covering a number of: 10458 cities, 183 countries, 23 sub-regions and 94 hotel chains. To have a visual idea of the data distribution on the globe see Fig. 5.1.

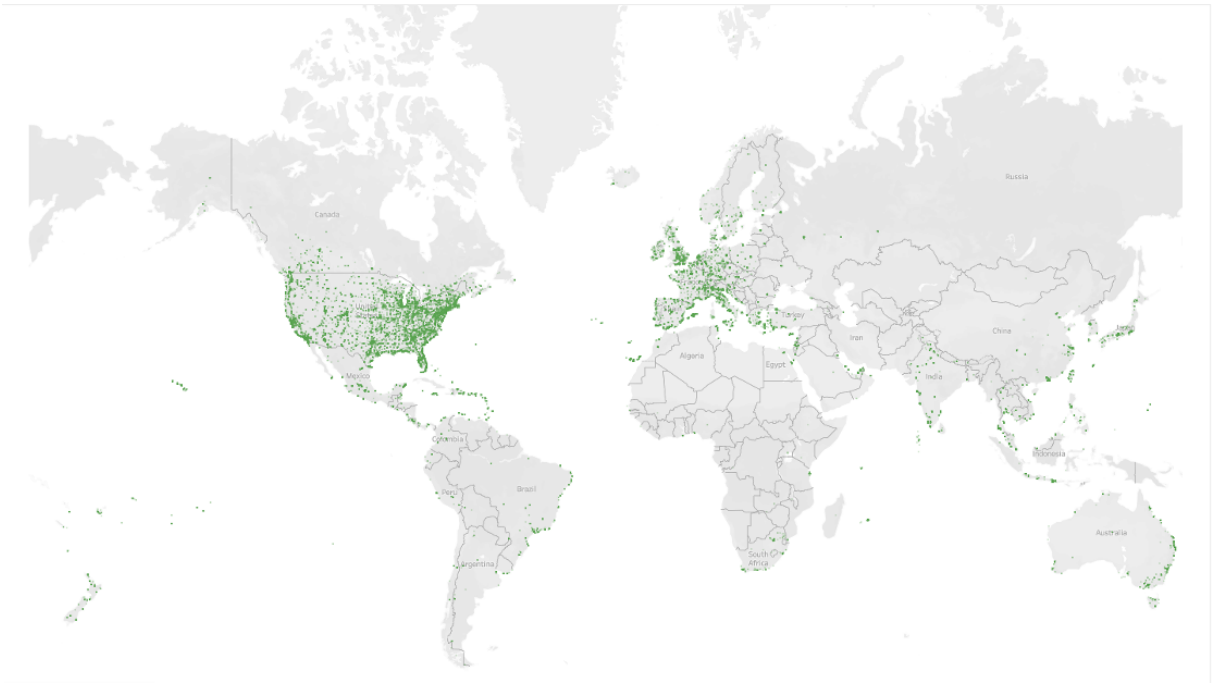


Figure 5.1: Plot of latitude and longitude coordinates of all the images present in the Hotel50k dataset. The plot was done using Tableau.

As it can be observed in tables 5.2 this dataset have imbalanced data, e.g. 42% of the images belongs to United States while only 2% are of Italian hotels. Since class imbalance could lead the model to overfit the most popular classes while not learning the features from the less populated ones (we will further discuss in Chapter 6) we adopted some oversampling techniques to avoid such a phenomenon.

Airbnb

The original Airbnb dataset is the result of a collection of datasets, made by Negri Virginia during her Master Thesis, provided by Inside Airbnb web-page, related to 100 different locations, which vary from big, densely populated cities (e.g. New York, Rome ...) to regions like (e.g. Barossa Valley, Western Australia, Sicily ...) see Fig. 5.6. Since the images from Airbnb web-page present both interior and outdoor images, a scene classifier was used to filter only interior scenes. The scene classifier is adopting a ResNet backbone pretrained on Places365 dataset [45] which collects more than 1.8 million images from 365 different scene categories. The information provided by Inside Airbnb relating to

Figure 5.2: Images taken from TraffickCam.

Figure 5.3: Images taken from Expedia.

the images also include latitude and longitude so, as in the case of the Hotel50k dataset, the geographical coordinates were translated into country and location labels (where, we remember that location may refer to a city like Amsterdam or to a region as Barossa Valley). We used the latitude and longitude coordinates to plot the data on the world map, see Fig. 5.5. Also in this case the sub-region labels were assigned according to the M49 coding classification. To conclude, a unique numerical identifier was assigned to each label of each category. As for the Hotel50k dataset we present the 10 most frequent sub-regions, country and locations in tables: 5.3 and 5.4.

Moreover we also collected some additional data, which we used only at inference time, coming from different real estate agency like Zillow, RE/MAX, Cushman and Wakefield etc and from different web pages like Idealista, Immobiliare, Badi etc. listing houses and flats for sales or rent.

Figure 5.4: M49 coding classification taken from Wikipedia.

Table with top-10 frequent Sub-regions

Sub-regions	Samples	Percentage
North America	586217	45.3%
Southern Europe	163074	12.6%
Western Europe	96476	7.5%
South-Eastern Asia	83523	6.5%
Northern Europe	74312	5.7%
Eastern Asia	61311	4.7%
Australia & New Zeland	38321	3%
Eastern Europe	33110	2.6%
South America	29315	2.3%
Central America	29058	2.2%
Other	99.572	7.6%

Table with the most frequent Countries

Country	Samples	Percentage
United States	545152	42.1%
Italy	64364	5%
France	46426	3.7%
Spain	47501	3.7%
United Kingdom	41408	3.2%
Canada	40873	3.2%
Thailand	38764	3%
Australia	38204	3%
Greece	26422	2%
Germany	26086	2%
Other	377089	29.1%

Table 5.1: Both tables show the presence of imbalanced classes.

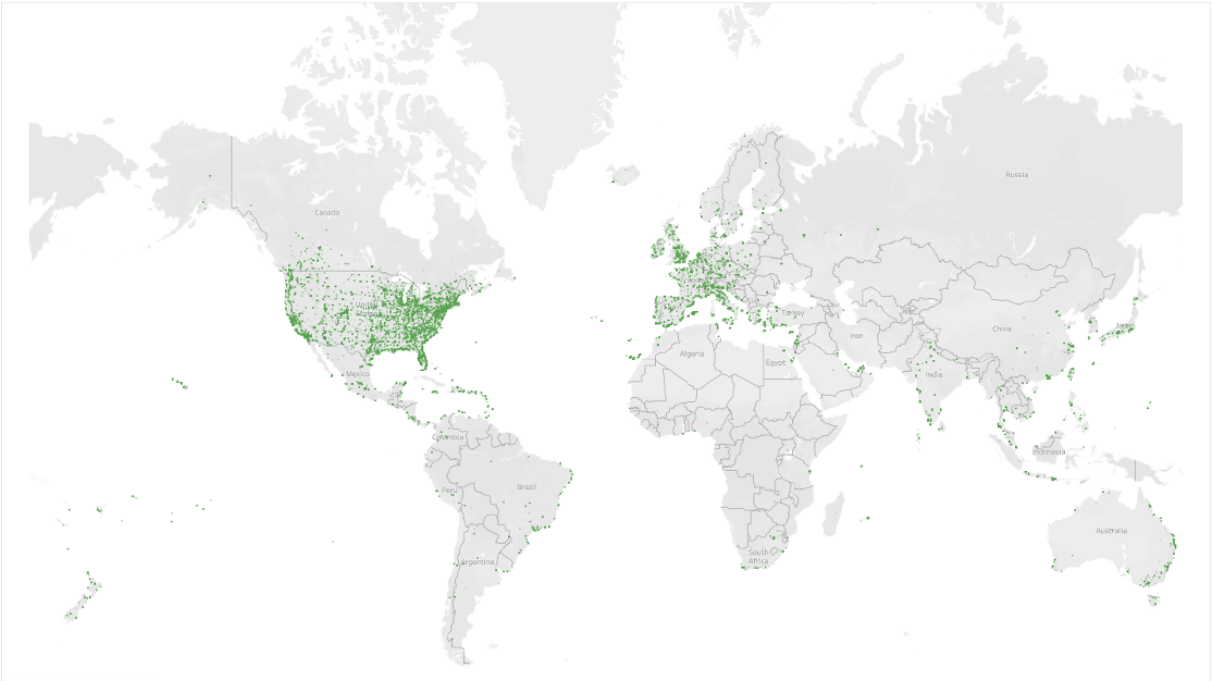


Figure 5.5: Plot of latitude and longitude coordinates of all the images present in the Airbnb dataset. The plot was made using Tableau.

Table with top-10 frequent cities

Cities	Samples	Percentage
Rome	9769	0.75%
Paris	9721	0.75%
New York	7922	0.61%
Tokyo	6675	0.52%
Florence	6549	0.50%
Vatican City	6374	0.49%
Venice	5756	0.44%
Istanbul	5665	0.43%
Ko Samui	4784	0.37%
Patong	4717	0.36%
Other	1.226.357	94.78%

Table with the most frequent Hotel chains

Hotel chains	Samples	Percentage
Unknown	709178	54.8%
Holiday Inn	64073	5%
Hampton	33993	2.6%
Best Western	27954	2.2%
Comfort Inn	22501	1.7%
Marriott	19509	1.5%
Hilton	18811	1.5%
Quality Inn	18711	1.4%
Best Western Plus	16736	1.3%
Super 8	16312	1.3%
Other	346.511	26.7%

Table 5.2: It can be observed that the phenomenon of class imbalance affects much more at country level than at city level where the distribution of the data is more homogeneous. Regarding Hotel chain class, more than half of the hotels doesn't belong to chains, for this reason in this case the Hotel chain class is labeled as Unknown.

Figure 5.6: Example images from the Airbnb dataset.

Airbnb 10 most frequent Sub-regions

Sub-regions	Samples	Percentage
Southern Europe	229414	23.8%
North America	220500	22.8%
Western Europe	146533	15.2%
Northern Europe	132358	13.7%
Eastern Asia	60353	6.3%
Australia & New Zeland	57478	6%
South America	53504	5.5%
Central America	17778	1.8%
Western Asia	17345	1.8%
Eastern Europe	12370	1.3%
Other	17702	1.68%

Airbnb 10 the most frequent Countries

Country	Samples	Percentage
United States	16304	42.1%
Italy	109227	5%
United Kingdom	91817	3.2%
France	73157	3.7%
Spain	65959	3.7%
Australia	57478	3%
Canada	47077	3.2%
China	31090	3.2%
Germany	30927	2%
Greece	27340	2%
Other	268221	29.1%

Table 5.3: Also in the Airbnb dataset we observe the presence of imbalanced classes both at sub-region and country level.

Cities	Samples	Percentage
London	73993	7.7%
Paris	55243	5.7%
New York	43108	4.5%
Beijing	31090	3.2%
Sydney	26527	2.7%
Los Angeles	25165	2.6%
Rome	24774	2.6%
Sicily	24519	2.5%
Rio de Janiero	22597	2.3%
Copenhagen	21651	2.2%
Other	616668	64%

Table 5.4: Airbnb 10 most frequent locations. It can be observed that the phenomenon of class imbalance affects more at country and sub-region level than at city level where the distribution of the data is more homogeneous.

6 | Experimental Results and Evaluations

This section is dedicated to the different experiments and evaluations that we did during the development of this Master Thesis. We provide deeper details about the architecture of our indoor geolocation model and compare performances of different configurations in different conditions. In addition, we present also different experiments aimed at finding the most suitable and informative XAI techniques for our problem.

6.1. Experimental Setup

During the whole experimental phase we used the following two Graphic Processing Units: Nvidia RTX 2080 Ti, and an Nvidia RTX 2070 that was mainly used for the inference part. The whole framework was developed mainly using Pytorch Lightning making the code simple to read, clean, and easy to reproduce.

6.1.1. Data

All the experiments were performed both on the Airbnb dataset, and on the Hotel50k dataset. Both datasets were split randomly into three separated sets: training (90%) and test (10%), then the training set was further split into 80% training and 20% validation. The models were trained with two different kind of custom **Data Generators**, which automatically generate a batch of 16 images to feed the network before each training step. In addition we also tested the use of data augmentation techniques.

Data Sampling

1. **Random Data Generator**: This is the first data generator that we decided to use on our models. At each training step a batch of 16 images is randomly selected, we investigated also batches of 32 images, but due to limited memory resources we weren't able to do it for all the models, so we decided to opt for batches of

16 images in order to compare all the models learning capability with same batch dimension. The random data generator extracts randomly the images from the training and validation set with re-sampling. This type of batch generation allows, while learning, to preserve the proportions between classes in the training set, which is supposed to be equivalent to those in the test set since they all come from the same data distribution.

2. **Country Conditioned Sampling**: as previously mentioned, both datasets exhibits a remarkable presence of class imbalance. In case of imbalanced data, a simple random generator prevents the network from learning features of locations with few examples. For this reason we decided to create a custom data generator which generates batches within which the likelihood of samples coming from different countries (Hotel50k) or subregions (Airbnb) is the same; so that for example the probability of sampling an image from United States which has 545152 samples is equal to the one of sampling from Germany which has only 26086 images. Of course this conditioned data generator can be implemented conditioning the sampling distribution at subregion level, country or city level. After some experiments we decided to condition at country level for the Hotel50k dataset and at subregion level for the Airbnb dataset.

Data Augmentation

In a first stage of our experiments we decided to adopt also Data Augmentation techniques in order to provide more training data to the model by implementing a set of transformations on our data. With a probability draw of 50%, the DataGenerator decides whether to apply one of the following transformation to a given selected image in batch:

- Rotation: rotates the image of a multiple of 90 degrees.
- Flip: randomly flips the image over the horizontal and vertical axes.
- ColorAugmentation: changes brightness, contrast and saturation of the image.

While image augmentation is almost necessary for the first generation of Vision Transformers, as for instance the ViT in order to make them competitive with the more traditional CNN, with more recent architectures like the Swin Transformer we didn't notice a remarkable improvement over our task, for this reason, after some initial comparisons, once chosen the Swin Transformer as our backbone, we have decided to omit the data augmentation.

6.1.2. Loss function

Since our problem consists of a parallel multi label classification problem, we decided to adopt **Sparse Categorical Cross Entropy** as loss function. This loss function receives as inputs the target class as an integer value (from 0 to $N_{classes} - 1$), where the integer values represent each class encoding, and a vector of floating point values, which represent the model prediction, of size equal to the number of classes.

In mathematical terms, the CategoricalCrossEntropy can be expressed as:

$$loss(x, y) = -\sum_{i=1}^{N_{classes}} y_i * log(x_i) \tag{6.1}$$

6.1.3. Optimizer

In order to solve the optimization problem with a loss function as the one mentioned above, we need to choose a stochastic gradient-based optimizer. Pytorch allows us to easily choose within a large number of pre-implemented optimizers, such as: Stochastic Gradient Descent (SGD), AdaGrad, RMSprop, and so on. After several experiments, we decided to pick the Adam optimizer [16]. Adam optimizer is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower order moments. Morover, Adam is computationally efficient and is very suitable for problems that are large in terms of data and/or parameters. It combines the advantages of the two following algorithms, while in addition to RMSProp it also uses the average of second moments of the gradients:

- **Adaptive Gradient Algorithm (AdaGrad)**: very suitable for computer vision tasks, keeping per-parameter learning rates, thus allowing to improve performances on problems with sparse gradients.
- **Root Mean Square Propagation (RMSPror)**:More effective for online and non-stationary problems. As AdaGrad maintains per-parameter learning rates which are continuously adapted according to the average of recent magnitudes of gradients for the weight.

6.1.4. Regularization functions

Due to their complexity, often Deep Learning models tend to overfit if not monitored and controlled by some regularization and stopping conditions. Overfitting is a phenomenon

that happens when the production of an analysis, in our case a deep learning model, corresponds too closely or exactly to a particular set of data, the training set, while it may fail to fit on new additional data. In other words, an overfitted model is a model which is not capable of generalizing on new data, due to the fact that it learned also from the noise in the training data. In order to avoid overfitting we decided to use some classical regularization techniques.

Early Stopping

Early stopping [5] is a regularization technique used to identify how many iterations can be run, in our case is more appropriate to talk about epochs instead of iterations, before the learner begins to overfit. The parameter which is responsible of the stopping condition is the patience which indicate after how many epochs, without an improvement on the validation loss, is necessary to stop learning. After a few experiments we decided to set the patience at 3 epochs.

Dropout

Another regularization technique that we adopted to limit the overfitting phenomenon and make the model more robust is Dropout [34] . Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. The only layer on which dropout is not usable is the output layer. Adding a dropout layer with a probability p, means that each neuron of the hidden layer on which we apply the dropout layer, at training time, will be switched off with the given probability. So the contribution of the neurons which were switched off to the activation of downstream neurons is temporally removed on the forward pass and in addition any weight updates are not applied to these neurons on the backward pass. Dropout in other words is forcing the model to learn the same task with less parameters, helping to avoid overfitting. We put a dropout layer right after the first Dense layer of our classifier. We did different experiments with different probabilities, initially we started with a probability $p = 0.5$ at the end on our final model we set the probability $p = 0.1$.

Learning Rate

Selecting a good learning rate is essential for both achieving better performance and faster convergence during the training process. Even if we already choose Adam as optimizer which is self-adjusting, the learning rate can benefit from more optimal choices. In a first moment we run many training with different learning rates and we found that the best

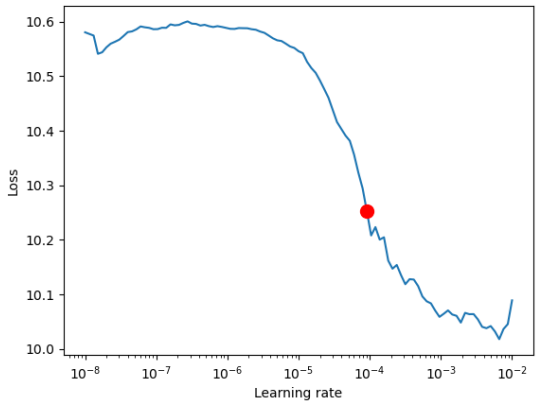


Figure 6.1: Learning rate vs loss with the optimal initial learning rate (red point) for the Swin Transformer on the Hotel50K dataset.

values were ranging between: $1e-4$ and $1e-6$. In a second moment we decided to exploit the *auto_lr_finder* of Pytorch lightning, a learning rate finder which reduces the amount of guesswork concerning choosing a good initial learning rate. Based on the paper [33], the idea is that the learning rate finder does a small run where the learning rate is increased gradually after each processed batch and the corresponding loss is logged. The result of this is a learning rate vs. loss plot that can be used as guidance for choosing an optimal initial learning rate. The red point suggests the optimal learning rate to pick; as we can see in the image below it is recommended not to pick the learning rate that achieves the lowest loss, but instead something in the middle of the sharpest downward slope.

Checkpoint callback

The purpose of this callback is to save the model checkpoints after each epoch, so to save the best model weights so far. In our framework, we decided to keep only the best weights for top two models with respect to the validation loss performance.

6.1.5. Evaluation Metrics

In order to compare and evaluate the performances of different models we took into account three different metrics: the loss function, prediction accuracy and **Improvement over Majority class** (IoMc). During training we can observe the evolution of the Training loss, SparseCategoricalCrossEntropy computed over each batch of processed images. After every epoch, we look at the validation loss, which is a computation of the SparseCategoricalCrossEntropy over the validation set, in order to have information about the learning trend of the model. For every class: city, country and sub-region we showed

respectively a top-1, top-5 and top-10 prediction accuracy on the test set. Top-N accuracy means that the correct class gets to be in the Top-N probabilities for it to count as "correct". We decided to use also an additional metric that we call: "Improvement over Majority class" which measures the improvement of our model with respect to a classifier that always assigns to any image the most frequent class in the dataset.

$$IoMc = \frac{Accuracy_{valid}}{Prob_{Majority\_class}}. \tag{6.2}$$

6.2. Experiment A: Backbone architecture

In our first experiment we compared different state-of-the art CNN models like VGG-19 and EfficientNet with some of the most recent vision Transformers as the Swin Transformer over the same multi-task classification problem. This comparison was made only on the Hotel50k dataset which is the most challenging one due to the large number of cities.

We want to understand which architecture is more appropriate to be used as backbone for our classification problem. More specific we first explored the capability of some CNN architectures, in terms of loss and accuracy, and then compared them with the Swin Transformer backbone. All the feature encoders that we used are trained on ImageNet22K [8], except of the Swin-T which is trained on ImageNet1K. We present here a table (Tab. 6.3) showing all the different Backbones that we investigated and their respective number of Parameters:

Backbone Name	Output Shape	Vector Size (Conv1D)	# Total Parameters
VGG19	(7, 7, 512)	(1, 25.088)	60.8 M
EfficientNetB0	(7, 7, 1280)	(1, 62.720)	41 M
Swin-T	(7, 7, 768)	(1, 37.632)	76.8 M
Swin-B	(7, 7, 1024)	(1, 50.176)	137.3M

Table 6.1: Feature extractor layers and number of parameters (The number of parameters are considering also the additional ones due to the dense layers of our classifier).

In order to make significant comparison, and also for memory reasons, the models were initially tested with the same basic configuration of the classifier made of two fully connected layers of 1000 neurons as output, a ReLU activation Layer in between, and three different final layers having as output the size of the corresponding classes. We kept all the feature extractor completely frozen for all the experiments. All the models were trained

for 14 epochs on the Hotel50k dataset and then we compared their performances. In a second moment where it was possible, due to the different magnitude of the number of parameters of each model, we explored different classifiers and hyperparameters for each backbone in order to find the corresponding optimal one for each feature extractor. Again the Swin Transformer turned out to be the best performing in terms of accuracy and loss.

We present below two tables, the first comparing Top-1, Top-5 and Top-10 accuracy, achieved by the different models, on the test set for each corresponding geographical level: world sub-region, country and city. The second table shows us a comparison of the loss and IoMc

Backbone	**Region Accuracy**			**Country Accuracy**			**City Accuracy**		
	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
VGG19	0.578	0.860	0.957	0.498	0.71	0.808	0.036	0.097	0.138
EfficientNetB0	0.569	0.856	0.955	0.492	0.71	0.804	0.0344	0.093	0.138
Swin-T	0.643	0.901	0.970	0.554	0.767	0.849	0.053	0.135	0.186
Swin-B	0.662	0.910	0.975	0.571	0.79	0.866	0.075	0.165	0.21

Table 6.2: Feature extractor test accuracy after 14 epochs on the Hotel50k dataset.

Backbone Name	**loss**	**IoMc Region**	**IoMc Country**	**IoMc City**
VGG19	10.758	1.28	1.18	0.39
EfficientNetB0	10.869	1.26	1.18	0.48
Swin-T	10.0507	1.41	1.315	0.71
Swin-B	9.850	1.44	1.38	1

Table 6.3: Feature extractor layers and number of parameters.

This experiment shows that the Swin Transformer is the most adapt backbone for our classification task, demonstrating the potential of Transformer-based models as vision backbones.

We can assume that the improvement obtained by the Swin Transformer compared to the others in this case is mainly due to two factors:

- One of the main advantage of such a backbone with respect to the other that we test is that self-attention eliminates the inductive bias of CNNs. (The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict

outputs of given inputs that it has not encountered). CNNs rely on two inductive biases built into the architecture itself: close pixels are related (locality), different portions of an image should be processed identically regardless of their absolute location (weight sharing). While self-attention-based vision models (like ViT, DeiT or the Swin Transformer) present minimal inductive biases since each image patch can relate to all other patches.

- the Hotel50k is a big dataset of more than 1.3 Million of images. (While CNNs' could perform very well also in case of small datasets, due to their inductive biases (high floor), in some cases these assumption could limit the learning capabilities with large quantity of data (low ceiling). On the other side most of the transformers due to their minimal inductive biases are under-performing w.r.t the state-of-the-art CNNs when trained on small dataset (low floor), but due to their flexibility they are able to outperform CNNs in case of huge datasets (high ceiling)).

Once selected the as backbone the Swin Transformer, we also build a model for the classification task on the Airbnb dataset, using the same structure of the classifier that we used for the Hotel50k dataset. The results on the Airbnb dataset are shown in table 6.4.

Region Accuracy			**Country Accuracy**			**Location Accuracy**		
Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
0.568	0.938	0.997	0.441	0.798	0.917	0.281	0.579	0.721

Table 6.4: Results on the Airbnb test dataset.

6.3. Experiment B: Pretraining on ImageNet22k vs ADE20k

Since in our problem we are geolocating indoor scenes from all over the globe, we asked ourselves if it could make sense or even improve our performances to use a Network pre-trained on a dataset for scene recognition like the ADE20K [44] instead of ImageNet22K. Unfortunately, the pretrained backbone on ADE20K has many more parameters than the one on ImageNet. Due to limited memory resources we were forced to make the comparison on the tiny version of the Swin Transformer (Swin-T) and could not do so on the larger version Swin-B. ADE20K includes 150 different scene categories (divided into objects and objects parts) from the SUN [41] and Places [45] database. An example of an interior scene image with all the annotations present in the ADE20k dataset is shown in Fig. 6.2.

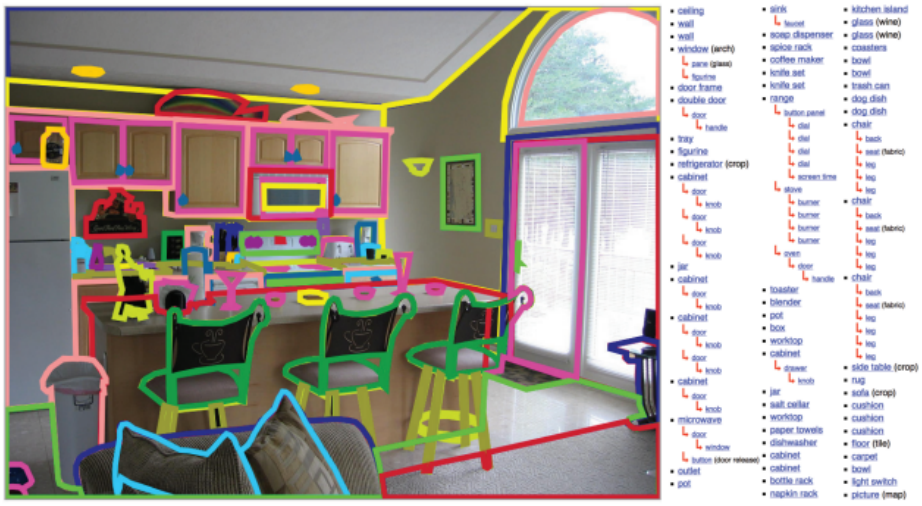


Figure 6.2: An image taken from the ADE20K paper [44] showing the annotations corresponding to the list of the objects and their associated parts in the image.

As it could be observed from table 6.5 we did not assisted to any improvement using the Swin Transformer pretrained on ADE20K, indeed the model pretrained only on ImageNet1K performs slightly better. It has to be mentioned that ImageNet1K includes one thousands classes, among which there are also many typical furniture objects such as: table, chair, lamp, etc. And in addition also some objects like lighter or laptop which are not present in the ADE20K dataset, but could be seen in an indoor scene and might help to give additional information about the location of the image. For this reason we decided to continue using the Swin-B pretrained on ImageNet22k in the next experiments.

Pretraining dataset	Region Accuracy			Country Accuracy			City Accuracy		
	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
ADE20k	0.624	0.886	0.966	0.537	0.755	0.839	0.049	0.126	0.175
ImageNet	0.643	0.901	0.970	0.554	0.767	0.849	0.053	0.135	0.186

Table 6.5: : Accuracy on the Hotel50K test set after 14 epochs of training.

6.4. Experiment C: Modifying the number of Fully Connected Layers

Since the Hotel50k dataset has much more labels for the city class than the Airbnb dataset (10458 vs 100) we found it necessary to build two different classifiers for the different datasets in order to improve the performances.

After choosing the Swin Transformer with the classifier presented in the previous experiment as our baseline model, we tested different configuration which we found more suitable for each of the two dataset. As it can be imagined, the most difficult task is to give good enough predictions at the city level (location in case of Airbnb dataset). Especially in the Hotel50k dataset this is due to the huge number of cities covered by the dataset. While we assisted to remarkable improvement from epoch to epoch in the learning at country and sub-region level, at city level the learning was quite stuck, with very small improvements. For this reason we decided to make some changes to the classifier block trained on the Hotel50k. While we kept the same configuration for the country and sub-region predictor, for the city predictor we directly connected the output of the Backbone with 50176 (37632 in case of the Swin T) units, to a fully connected layer having as output the city labels which are 10458. This led to a remarkable improvement on all the top-1, top-5 and top-10 accuracy values at city level, and to a much faster learning. After only 4 epochs we reached very similar results to the ones presented in the previous section after 14 epochs. On the other hand, this has led to a considerable increase in the number of parameters in our model, from the previous 76.8 million to 245 million of parameters, occupying all available memory on the GPU. Below we report the results obtained after 12 epochs in table 6.6:

Region Accuracy			**Country Accuracy**			**City Accuracy**		
Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
0.653	0.904	0.973	0.561	0.779	0.859	0.128	0.251	0.31

Table 6.6: Results on Hotel50k test set after 10 epochs directly flattening the output of the last Swin block for the city classifier.

In order to obtain similar results with the lighter configuration that we introduced in the previous experiment we needed to train the model for 39 epochs. In addition we found that introducing an additional classification task, which is: correctly recognize the hotel chain, led to further improvements to our model. This is probably due to the fact that trying to correctly classify the hotel chains forces the model to extract information which helps also for the other classification tasks. The final result is a hierarchical model which, given an image is predicting at the same time: sub-region, country, city and chain to which the hotel belongs (or No chain in case of hotel that don't belong to any chain). We report the achieved results in table 6.7.

Region Accuracy			Country Accuracy			City Accuracy			Chain Accuracy		
Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
0.663	0.921	0.975	0.578	0.792	0.866	0.145	0.271	0.344	0.677	0.849	0.903

Table 6.7: Results obtained after 12 epochs on the Hotel50k test set adding also the Chain classification task.

During this experimental phase we also decided to unfreeze the last two Swin Transformer Blocks leading to further improvements also on the Airbnb dataset. To conclude, we present our final results on both datasets in the following tables 6.8, 6.9:

Region Accuracy			Country Accuracy			City Accuracy			Chain Accuracy		
Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
0.691	0.922	0.979	0.597	0.814	0.89	0.213	0.364	0.423	0.696	0.864	0.922

Table 6.8: final results obtained after 37 epochs.

Region Accuracy			Country Accuracy			Location Accuracy		
Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
0.705	0.966	0.998	0.583	0.880	0.953	0.389	0.695	0.814

Table 6.9: Results on the Airbnb test dataset.

6.5. Experiment D: Comparing Integrated Gradients and Grad-CAM based Explanations.

Interpretation algorithms can give us a hint of how and why black-box models like our neural network make their decision. In our first attempt to capture visual explanations of the relationship between the model's predictions in terms of the learned features we decided to use Integrated Gradients.
Integrated Gradients computes the integral of the gradients of the output of the model

for the predicted class with respect to the input image pixels. We used Integrated Gradients since differently from other methods like Grad-CAM, which was originally thought for CNN, Integrated Gradients could be applied to any type of differentiable model (e.g. text, structured data, images). Another main reason why we decided to use Integrated Gradients is because it satisfies **Implementation Invariance**, i.e., attributions should be identical for two functionally equivalent networks, where by attribution we mean "assigning the blame (or credit)" for the output to the input features. While other explainability methods like DeepLift [32] and LRP [4] break the Implementation Invariance assumption. We used Integrated Gradients separately on each classifier in order to highlight which are the most relevant features for the sub-region, country and city classification task. We tested different integral approximation like: left, right and middle Riemann sums, Trapezoidal rule and Gauss-Legendre quadrature rule. The latter method approximates the fastest. In addition we tested different number of approximation steps, in order to find a trade off between approximation speed and the accuracy we decided to use number of steps = 50. We show now some images on which Integrated Gradients method was used, all the images present the original image on the left, followed by the image with the attributions assigned respectively at sub-region level, country level and city level.

The image in Fig. 6.3 is taken from the Expedia pictures belonging to the Hotel50 dataset, it shows a room of an hotel located in Italy, more specifically in Trapani and which doesn't belong to any chain.

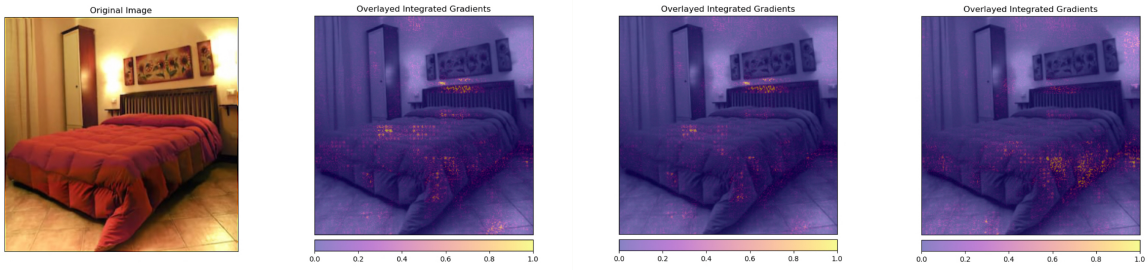


Figure 6.3: An image taken from Expedia website presenting a bedroom in a Hotel located in Trapani on the left, followed by the attributions given at sub-region, country and city level by Integrated Gradients method.

Fig. 6.4 shows an image taken in a room belonging to a Mercure hotel, located in Proserpine (Australia). It seems that while at sub-region and country level information from the window view and the wall are also exploited, at city level the model only use the sheets on the bed as main feature to predict the corresponding city. In Fig. 6.5 we present an image belonging to the Traffickam database included in the Hotel50k dataset, it was shot

inside of an hotel belonging to the Fairfield Inn in Comstock Northwest (Michigan).

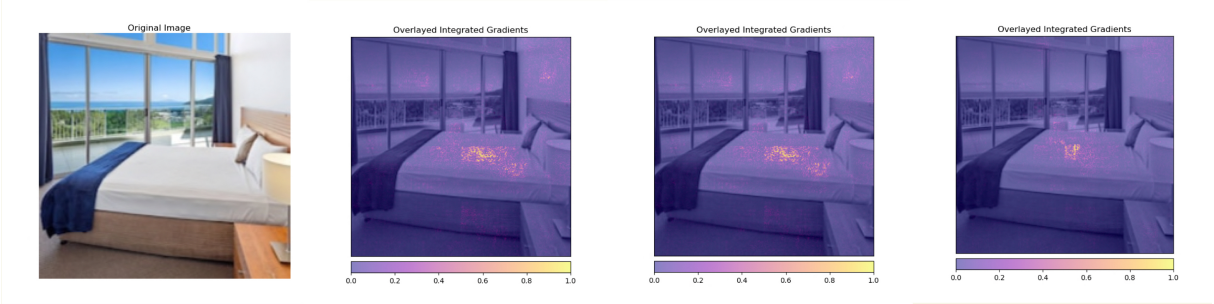


Figure 6.4: The picture on the left is taken in a bedroom of an hotel located in Proserpine (Australia) belonging to the Mercure chain, on its right the attributions given respectively at subregion, country and city level by Integrated Gradients method.

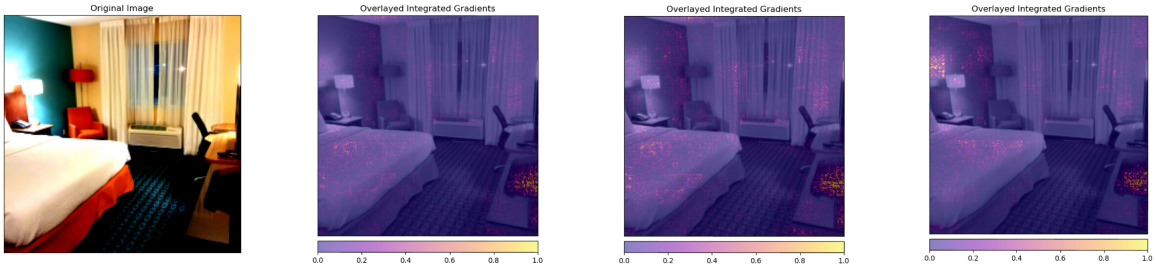


Figure 6.5: An image taken from the Traffickam subset of the Hotel50k dataset, it shows a room of a hotel in Comstock Northwest, the attributions seems to mainly highlight the moquette and the bed at subregion level, at country level additional positive attributions are assigned also to the curtains and the wall and the ceiling, while at city level the region adjacent to the lamp and the lamp itself also receive very high attribution.

To conclude, we also analyse an image taken from the Airbnb dataset, showing a living room located in Porto, Fig. 6.6. The windows are typical of the northern part of the country, but also of Galicia neighboring region of Spain.

Even if the attribution can give us a general idea of the most informative features, since Integrated Gradients assign an attribution to each pixel, as it can be observed from the reported examples the results are quite noisy and not so easy to interpret. For this reason we decided to make some additional analysis with different XAI techniques, like Grad-CAM, trying to give more interpretable explanations.

Grad-CAM is one of the most commonly used methods for visualizing where a convolutional neural network model is looking when making a prediction. While Integrated Gradients evaluates the contribution of each pixel of the input image to the output of a

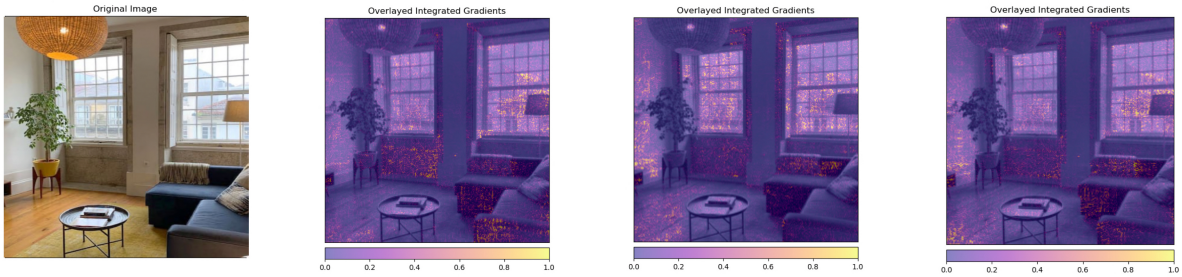


Figure 6.6: An image taken from an Airbnb in Porto on the left, followed by the attributions given at subregion, country and city level by Integrated Gradients method. In the space between the two windows we can notice the presence of a power socket on the bottom, which receives a positive attribution, confirming that this feature is one of the most critical for the model to predict the corresponding sub-region class (we checked that our model predicts Southern Europe as most probable sub-region).

model, Grad-CAM evaluates the contribution of each neuron in a given selected layer to the output of the model. Another difference between the two methods is that while with Integrated Gradients similar outputs are obtained regardless of the class used to compute the gradient that is back-propagated, Grad-CAM is a class-specific method which computes the gradients of the target output with respect to the given layer thus producing different heatmaps according to the targeted class. Since Grad-CAM was originally thought for CNN some adaptations are required in order to use it on a Vision Transformer and producing meaningful heathmaps. The idea is to treat the output of the last attention layer (before the classification head) as the designated feature map. More specific, in our case, the output of the last layer of Swin Transformer in the Swin-b version is of this type: $batch\_size \times 49 \times 1024$, where the last 49 elements corresponds to the 7×7 window size. of the image with 1024 channels. Thus we reshape the activations and gradients to 2D 7×7 spatial images and bring the 1024 channels to the first dimension, like in CNNs.

In Fig. 6.7 we show the heatmaps produced by Grad-CAM on an image taken from the Airbnb dataset, belonging to a living room located in Porto, that we previously analysed in Fig. 6.6 with Integrated Gradients. It can be observed that the areas of pixels which received a positive attribution by Integrated Gradients are approximately covered also by the heatmaps produced by Grad-CAM. In Fig. 6.8 we show the results obtained on an image from the Hotel50k dataset, showing the kitchen of an hotel located in Laguna Beach (U.S). We also investigated the potential benefits of overlaying the semantic segmentation of the interior scenes, see Fig. 6.9, in order to have a clearer explanation of the regions and objects highlighted by the Grad-CAM visualization. As it appears

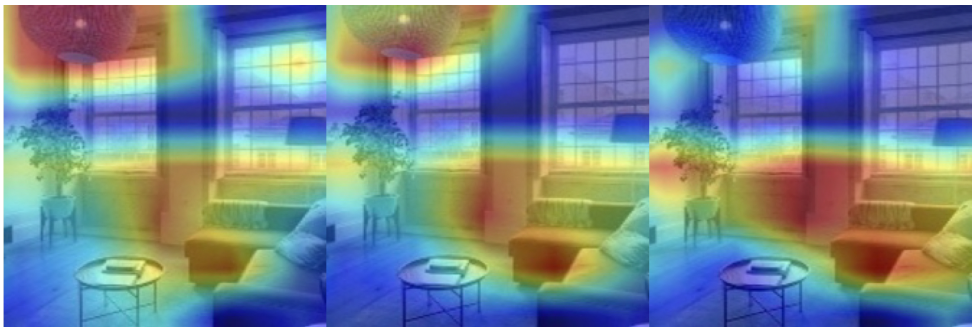


Figure 6.7: An image taken from an Airbnb in Porto, previously analysed in Fig. 6.6. The heatmaps produced by Grad-CAM are highlighting, approximately, the same areas which receive a high attribution by Integrated gradients. The heatmpas are presented in order for sub-region, country and city level.

from the reported examples the results provided by Grad-CAM are more interpretable with respect to Integrated Gradients, even though we would like to be able to blame multiple features separately, while we have proven that in most of the cases Grad-CAM visualizations cover a relatively large area including different objects. For this reason we carried out an ulterior experiment with the aim of obtaining even more interpretable and useful visual explanations.

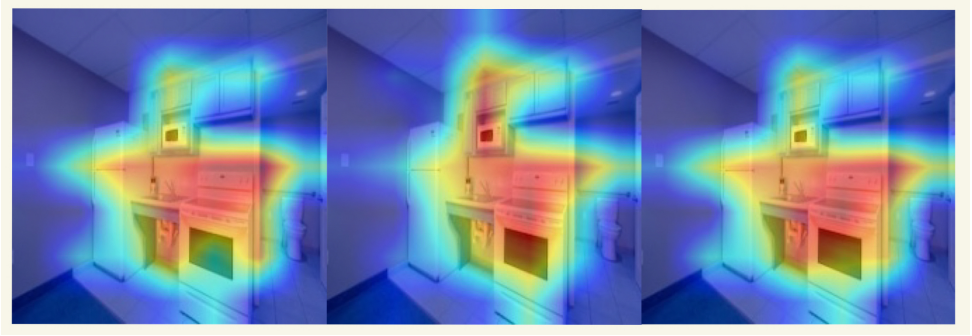


Figure 6.8: An image from Expedia web-page of a kitchen of a hotel in Laguna Beach. The visualization produced by Grad-CAM are corresponding to the prediction at sub-region level on the left, in the center at country level and on the right at city level.

6.6. Experiment E: Comparing Segmentation Methods for SHAP-based Explanations.

Methods like Integrated Gradients, which are able to provide visual explanations regarding the features in the image that are more important than others for the prediction, but

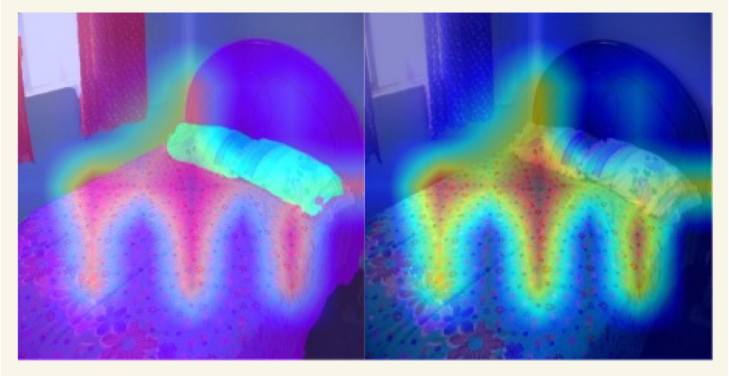


Figure 6.9: An image taken from the Airbnb dataset showing a bedroom in Rio. On the right: the Grad-CAM visualization superimposed on the original image, while on the left: the superimposition of the Grad-CAM visualization map on the corresponding segmented image.

being class agnostic they are not able to point out which parts of the image are more or less useful for the prediction of the different classes. Another drawback of Integrated Gradients is that it is computationally expansive as we already mentioned in the previous experiment. Grad-CAM on the other side is a class-specific method which can generate different maps according to the target class. Unfortunately the heatmaps produced, as it can be seen in the previous experiment, in most of the cases are coarse, including areas with many objects thus this method is not very convenient for interpretation purposes in our case. We would like to blame single objects inside of the indoor pictures instead of coarse areas. For this reason we thought that we could exploit the semantic segmentation of the indoor images produced by our model and combine it with SHAP, instead of using a generic partitioning scheme of the image.

SHAP is an algorithm capable of extracting two different types of information:

- Intrinsic content of the image (some parts of the image are more important than others).
- Discriminative information (some parts of the image are useful for distinguishing the classes).

SHAP values combine both types of importance, fairly distributing the prediction among the features. SHAP belongs to the methods based on Shapley-value, which have a solid theoretical justification although they are computationally expensive and in general their accuracy is not higher then other methods like Grad-CAM. Usually SHAP is used to explain predictions of image classification problems by dividing the input image into super-pixels, see Fig. 6.10 and then run the algorithm on the super-pixels instead of

single pixels in order to be more efficient. Thus giving explanation according to segments and not to every pixel. We show that instead of using superpixel we could exploit the segmentation provided by our network and treat each segmented object as a superpixel. This approach led to a remarkable improvement in two directions:

1. In terms of explainability, shapley values are assigned to more interpretable features in the image which are segmented scenes and objects with a semantic meaning instead of just super-pixels.

2. The algorithm is improving in terms of time efficiency. In general when dividing the image into superpixel we saw that in order to have explanations capable of highlighting single objects we need to split the image into at least 50-100 super-pixels. While in most of the pictures we analysed there are around 10-15 semantic segmented elements.

To highlight the improvements obtained by our approach we present a comparison, using Kernel SHAP, a method which computes the importance of each feature combining a special weighted linear regression with Shapley values, on an image partitioned into 20 super-pixels Fig. 6.11, and the same image segmented into 14 classes Fig. 6.12. The latter was segmented into the following classes: wall, bed, floor, curtain, painting, windowpane, table, lamp, pillow, ceiling, chair, cabinet, box, radiator, desk. It has to be noticed that not only the results are more interpretable, but also in terms of efficiency we gain more than 9 seconds, (17.05/s vs 26.25/s), with respect to an execution on the image partitioned into 20 super-pixels.

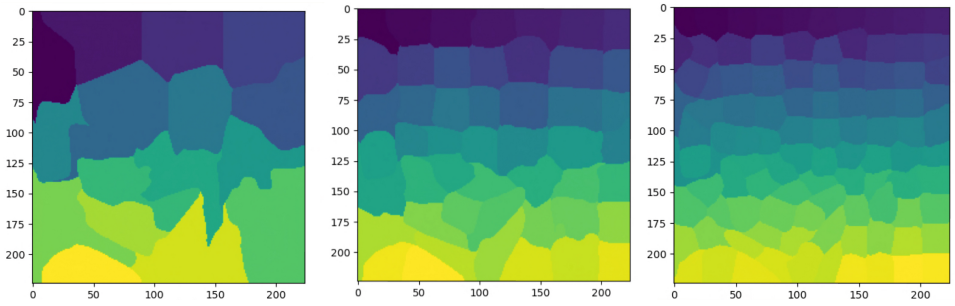


Figure 6.10: An image taken from an Airbnb partitioned respectively into 20, 50 and 100 super-pixels.

Another advantage of our approach is that it allows us to visualize which decorative and structural elements influence the predictions at each level of granularity i.e: sub-region, country and city level. Looking at Fig. 6.13, we see that most of the importance for the top-1 prediction at sub-region level is assigned to the windows, the chairs and the ceiling. While in the case of the top second and third sub-region predictions the window had less

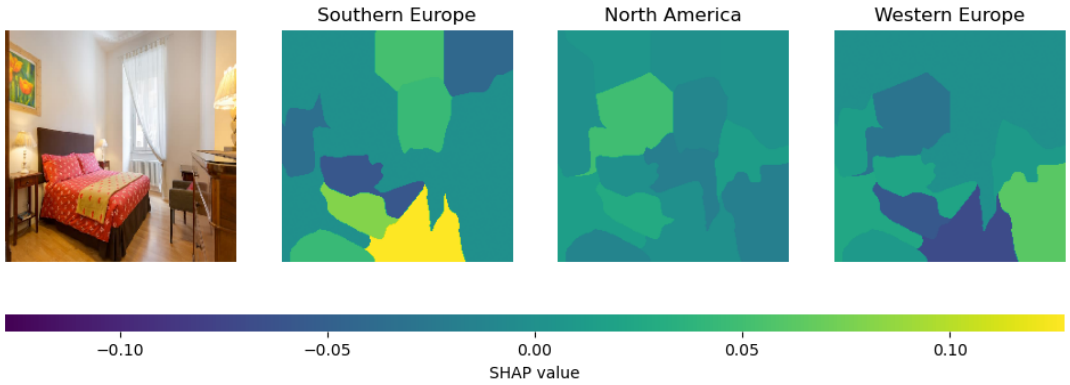


Figure 6.11: An image showing SHAP values on the top-3 predictions at sub-region level of an Airbnb located in Rome, using an image partitioned into 20 super-pixels.

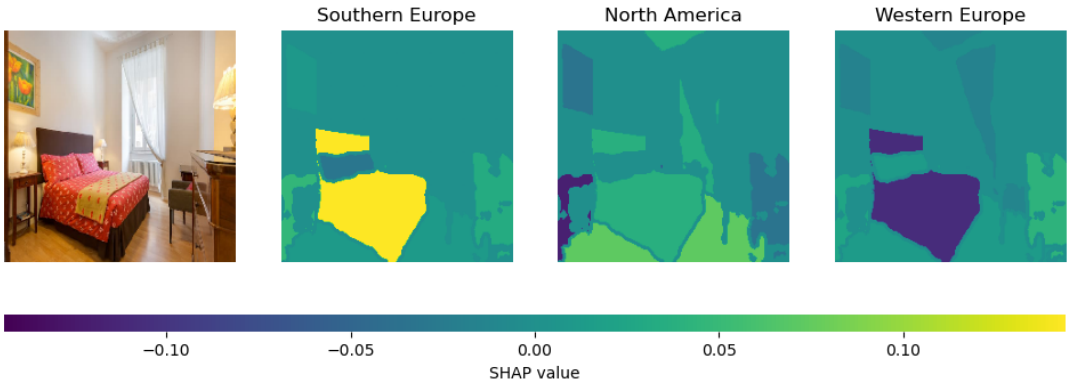


Figure 6.12: An image showing SHAP values on the top-3 predictions at sub-region level of an Airbnb located in Rome, using the segmentation provided by our model instead of super-pixels.

influence on the prediction. At country level, see Fig. 6.14, most of the focus is on the ceiling for the correctly top-1 predicted country, which is Italy. After a deeper inspection of the Airbnb dataset, we found that this attribution is coherent to the fact that there are many Airbnb images from: ancient Roman and Venice houses and some Sicilian rural houses which present a wooden beam ceiling. This also partially explains the distribution of the Shapley values in Fig. 6.15 which highlights that the ceiling was one of the most important feature for the corresponding top three predicted countries.

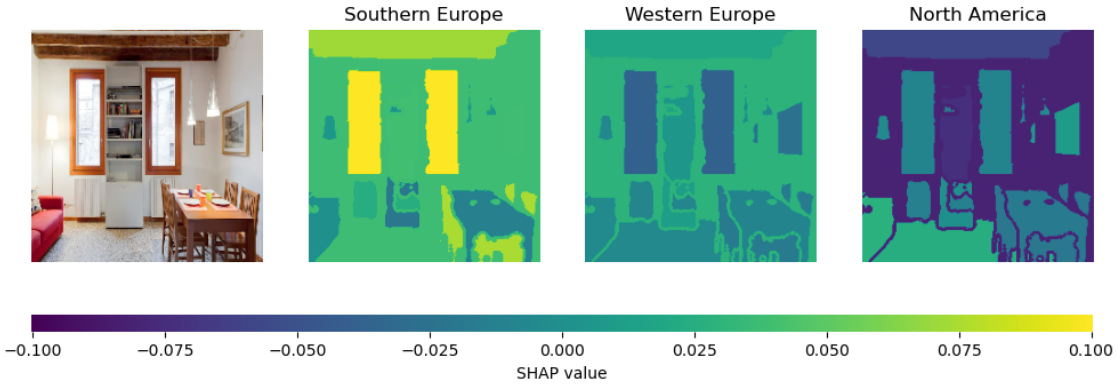


Figure 6.13: On the left an image from an Airbnb located in Venice, followed by the Top-3 predicted sub-regions, with the corresponding SHAP values assigned to the segmented features in the image.

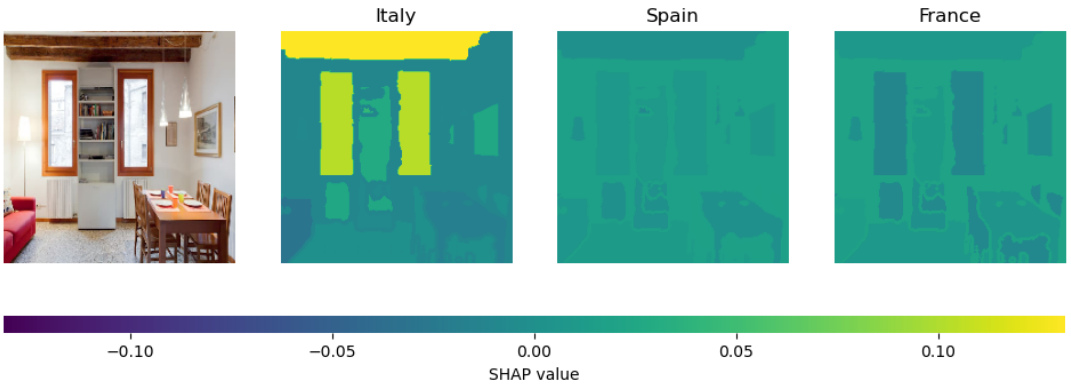


Figure 6.14: On the left an image from an Airbnb located in Venice, followed by the Top-3 predicted countries, with the corresponding SHAP values assigned to the segmented features in the image.

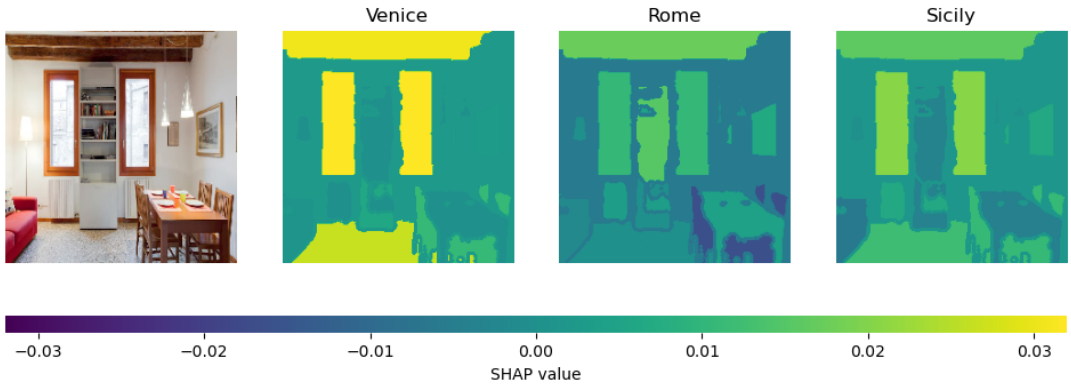


Figure 6.15: On the left an image from an Airbnb located in Venice, followed by the Top-3 predicted cities, with the corresponding SHAP values assigned to the segmented features in the image.

6.7. Analysis A: Model Calibration

This analysis was conducted exclusively on the model trained on the Airbnb dataset, to verify its adequacy before using it, as we will see in Chapter 7, for a web demonstration.

Interpreting the probability scores of a deep learning model for multi-class classification as the probability that the corresponding class was detected is not correct, unless the model is calibrated. **Confidence calibration** is the problem of predicting probability estimates which are representative of the true correctness likelihood. For this reason in order to interpret the prediction scores of our model as probabilities of class detection we need to check that our model is calibrated. The techniques that we investigated for visualizing the calibration of the model were introduced by Guo Chuan et al. in [13]. We introduce some definitions and formalise the calibration problem in our framework (supervised multi-class classification with neural networks).

Given two random variables: an input $X \in \mathcal{X}$ and label $Y \in \mathcal{Y} = 1, \ldots, K$, with a joint distribution $\pi(X, Y) = \pi(Y|X)\pi(X)$. Let h be our neural network with $h(X) = (\hat{Y}, \hat{P})$ where $\hat{Y}$ is the predicted label and $\hat{P}$ is it's associated confidence i.e. the probability of correctness. We would like $\hat{P}$ to represents a true probability, namely that the confidence estimate $\hat{P}$ is calibrated. When a model is calibrated, the confidence score should equal the accuracy. To understand this better, let's take an example from [13]: "given 100 predictions, each with confidence of 0.8, we expect that 80 should be correctly classified".

Definition 2. *We define* ***perfect calibration*** *as:* $\mathbb{P}(\hat{Y} = Y|\hat{P} = p) = p, \forall p \in [0, 1]$

Perfect calibration is practically impossible and since $\hat{P}$ is a continuous random variable the probability in the above definition cannot be computed using finitely many samples. For this reason empirical approximations like **Reliability Diagrams**, see Fig. 6.16, are used to give a visual description of model's calibration.

To estimate the expected accuracy from finite samples, the model's predictions are divided into M bins based on the confidence score of the winning class. For each bin B_m we calculate average accuracy $acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \chi_{(\hat{y}_i = y_i)}$ and the average confidence within the bin as: $conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i$. Hence a perfectly calibrated model will have $acc(B_m) = conf(B_m)$ for all $m \in 1, \ldots, M$. The reliability diagrams at the top show, for each classification task, the average confidence for each bin, as well as the accuracy of the examples within each bin. In general the average confidence for a bin lies on or close to the diagonal. For each bin the difference between the accuracy and the confidence is plotted. Ideally, the accuracy and confidence should coincide. In that case, we can affirm

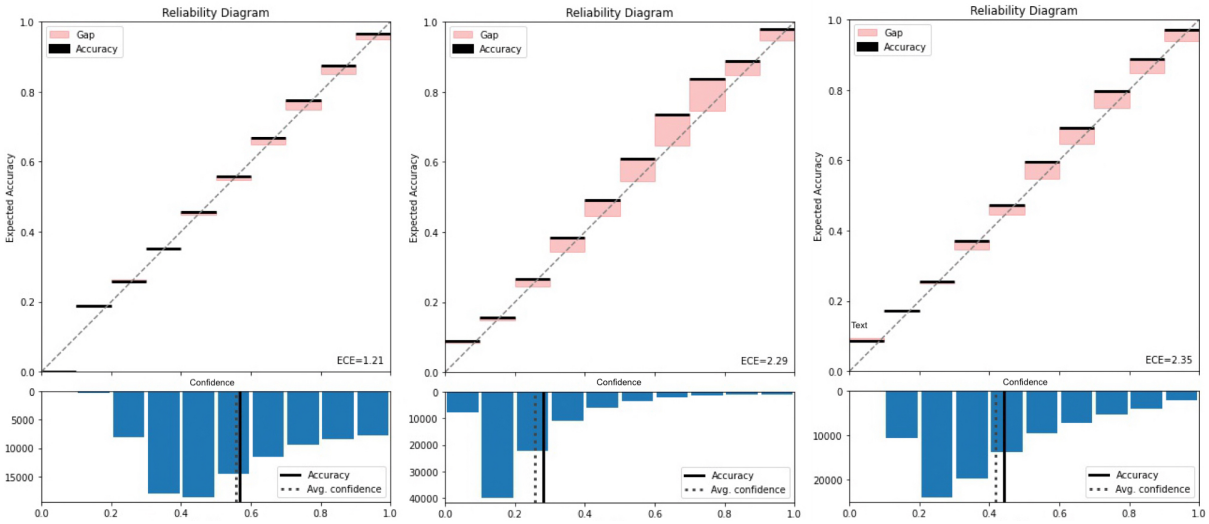


Figure 6.16: On top reliability diagrams respectively for sub-region, country and location classifiers, showing the average confidence for each bin, as well as the accuracy of the examples within each bin. We used 10 bins. At the bottom the confidence histograms, showing how many test samples are in each bin. The two vertical lines indicate the average confidence and overall accuracy.

that the model is calibrated and we can interpret the confidence score as a probability. While, when the model is not calibrated, there is a gap (red bars) between accuracy and confidence. The diagonal is the ideal accuracy for each confidence level. If the red bar goes above the diagonal, it means that accuracy is larger than the confidence, so the model is not confident enough. If the red bar goes below the diagonal, it means the confidence is larger than the accuracy and the model is overconfident in its predictions. Black lines in the reliability diagrams indicate the average accuracy for the examples within that bin: If the black line is on top of a red bar, the model is not confident enough in its predictions, while if the black line is at the bottom of a red bar, the model is over-confident for the samples in that bin.Techniques for calibration are aimed at bringing these two elements more in line with one another by fixing the confidence scores. On the bottom of the reliability diagrams we also show the **Expected Calibration Error** (ECE). ECE is a summary statistic that approximates the difference in expectation between confidence and accuracy, it is defined as follows: $\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)|$ where n is the total number of samples in our test set. The Lower the ECE the better it is.

It has to be noticed that reliability diagrams do not display the proportion of samples in a given bin, and thus cannot be used to estimate how many samples are calibrated. For this reason we added the confidence histograms at the bottom that show how many test examples are in each bin. We used 10 bins. The two vertical lines indicate the average

confidence and overall accuracy. The closer these two lines are together, the better the calibration of the model. To conclude, looking at Fig. 6.16, we can affirm that our model is rather well calibrated.

Before deploying our model in the web demonstration we also computed the confusion matrix, see Fig. 6.17, for the three classification tasks (sub-region, country and location) in order to check if the model is biased toward a specific sub-region, country or location. We can observe that at sub-region level there are some elements on the diagonal on which

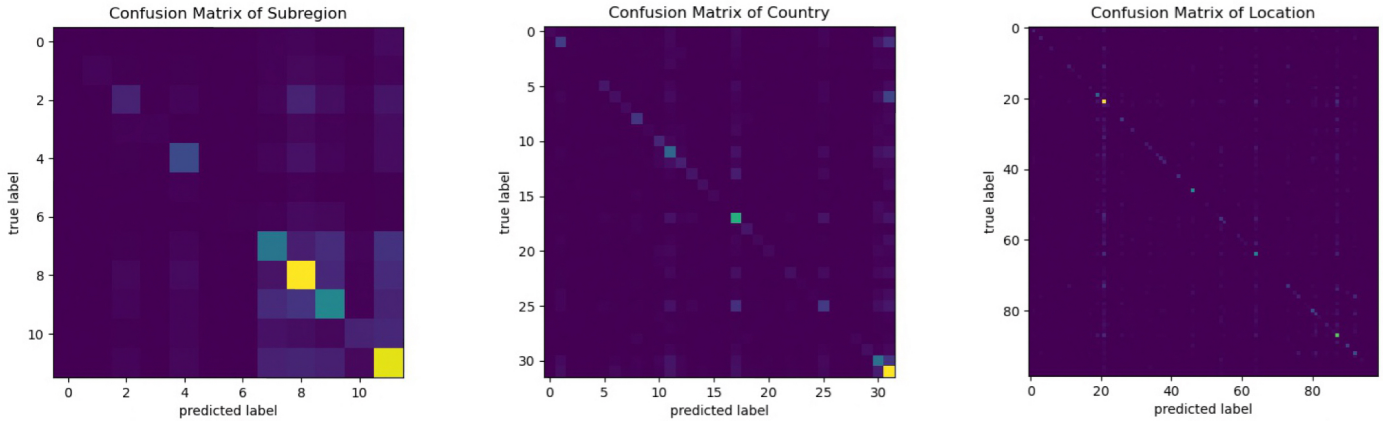


Figure 6.17: The lighter colors on the diagonal indicates that the model is predicting well.

the model is not predicting so well. These sub-regions are for example: Southern Africa, Western Asia, South Eastern Asia and Eastern Europe. One of the reasons of the bad performances on these regions is due to class unbalance, in fact we have few samples for these regions. We observed that we have approximately only 24000 observations for Southern Africa, all belonging to just one country (South Africa), same for Eastern Europe where we have only images from Prague (around 14000) and South Eastern Asia (7000 images from Singapore). While for sub-regions like Southern Europe or Northern America,(which have lighter colors on the diagonal of the confusion matrix) we have many samples from different countries covered by these areas.

7 | Qualitative Analysis with a Demonstration Application.

In this section we discuss a demo that we developed using the Gradio framework for Image Classification with Vision Transformers. The framework is very easy to use, on the left there is a button to clear data in order to upload new pictures and close to it a submit button in order to run the prediction on the uploaded image. On the right, just below the predictions there are three buttons:

- Interpret: which provides interpretation, based on SHAP [22], explaining prediction output.
- Screenshot: which allows to take a screenshot of the predictions and explanations done by the model.
- Flag: which may be very useful in case of unexpected behaviour, e.g. imagine that the model is tested on an image of a danish living room, and produce a top-10 predictions without Denmark, putting on top only African cities. It would be very helpful to mark this image and send it back to us in order to understand the anomalous behaviour. FLAG button allows the user or tester to send back to us the flagged data, in order to analyse it.

This tool could be used to perform a user study to obtain an additional qualitative evaluation of our model. Even if it goes beyond the scope of this thesis, we believe that also a subjective analysis of different interpretability techniques deployed in the experimental phase, Chapter 6, would be very useful, since people from different part of the world may have some insight visual information to compare with our XAI explanations. We leave the incorporation of the different XAI methods into a web demo for future work.

In our demonstration website we decided to deploy our model for indoor geolocation trained on the Airbnb dataset, displaying the top-10 country predictions. We will comment now on some successful predictions made by our model when tested on our demo.

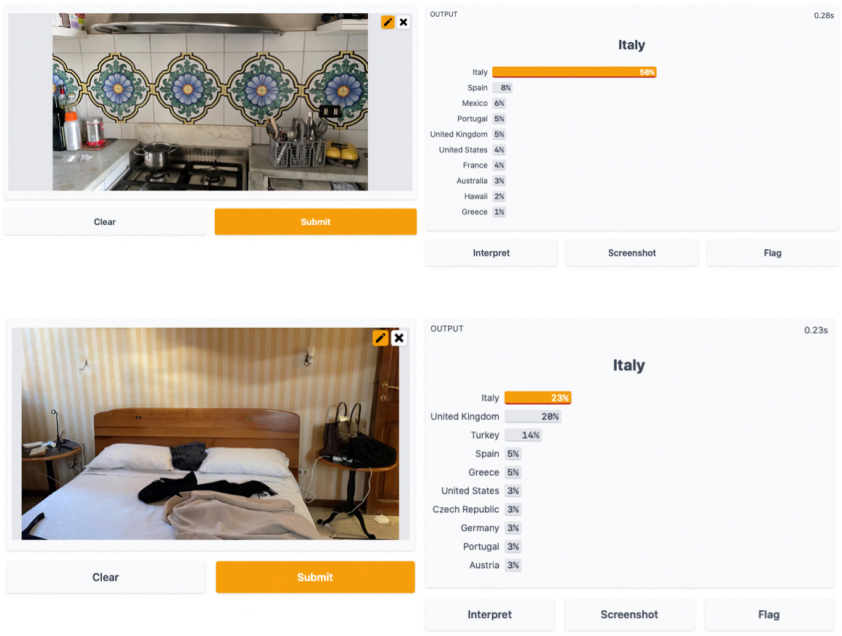


Figure 7.1: Top-10 predicted countries on a two photos taken inside of a Roman house.

We first analyse some interior pictures taken by some colleagues in their own house to collaborate in this project. The photo in Fig. 7.1 was taken by a colleague inside of his apartment in Rome, both where correctly geolocated at country level. After a deeper inspection of the Airbnb dataset we noticed that the countries predicted as most probable after Italy, which are: Spain, Mexico and Portugal, presents many pictures with "azulejos" which are painted tin-glazed ceramic tile-work, originally from Portugal and Spain, and later introduced also in former Portuguese and Spanish colonies. One of the elements which in this case gives additional information to the ceramics is the type L power socket which is only present in Italy among these countries.

The tested image in 7.2 was taken by a friend in his room in London, even if there are not so many patterns which could result useful to understand the location of the picture to an un-trained italian eye (like ours), the Network is still capable of predicting the current location. In the lower right part of Fig. 7.2 we present also an image showing the most important features for the predicted output according to the SHAP algorithm.

We had the opportunity to test our model also on some images taken by a french colleague living in Paris (a few of them are shown in Fig. 7.3) and by a friend living in Berlin, see Fig. 7.4. In both cases the results where surprisingly accurate. In the first case, we tested the model on 10 different pictures from different perspective of the same house in Paris, in 8 pictures out of 10 French was predicted as top-1, while in the remaining two pictures

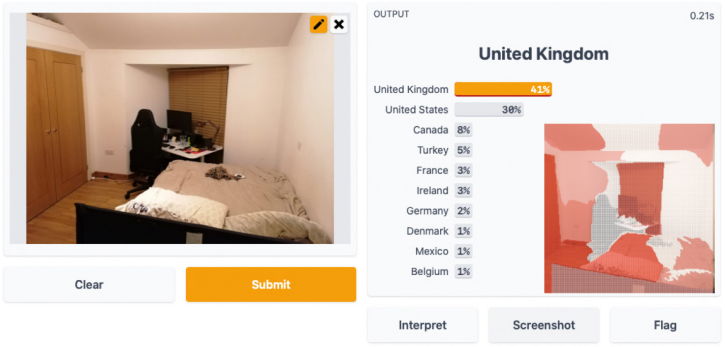


Figure 7.2: Top-10 predicted countries on a photo taken inside of a room in a house in London.

it was as top-3, right after Denmark and UK. In the second case we tested our model on 6 different pictures taken inside the same flat in Berlin, the model correctly predicted the country in 5 cases out of 6, only in one case it assigned a higher probability to Denmark and the Netherlands right before Germany.

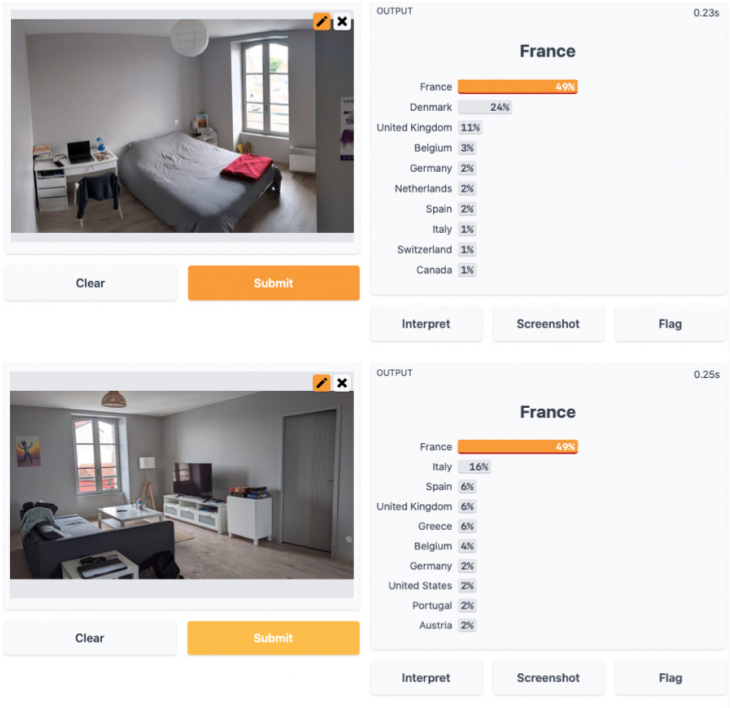


Figure 7.3: Top-10 predicted countries on a photo taken inside of a bedroom and a living room in a house in Paris.

If we consider that our model was trained exclusively on images coming from Airbnb, it is quite remarkable the capability to generalize on data coming from interior of student,

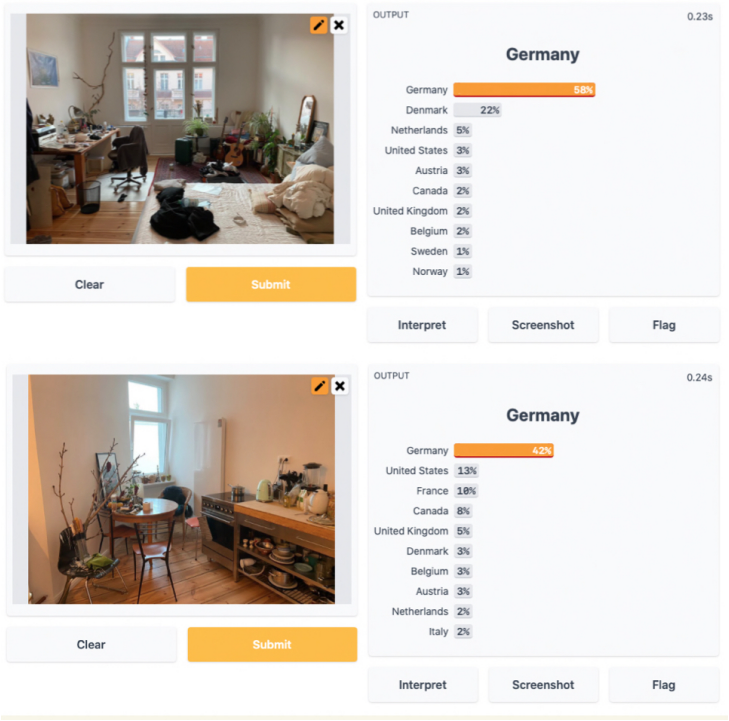


Figure 7.4: Top-10 predicted countries on a photo taken inside of a bedroom and a kitchen in Berlin.

workers and family houses. In fact, usually Airbnb images tend to present different elements and characteristics than house for long term living.

As previously mentioned in the Chapter 5, we also collected more than 2000 pictures, from different real estate web pages, which are available to the user to test our model. The image in Fig. 7.5 is another correctly predicted image taken from this dataset. We made a couple of experiments on this photo, cropping it many times so to exclude some informative elements as the power sockets on the right side of the cooker while keeping the sink still in the image: as a result the model still predicted United Kingdom as first country option but the probability decreased from 59% to 36%. While after an ulterior cropping, both on top cutting most of the kitchen hood piping and the sink on the left, the model predicted United States as most probable country with 38%, moving United Kingdom to the second most post probable country with a 29% probability.

The image in Fig. 7.6 is taken from the Agoda web-page which is an online travel agency but also a metasearch engine for hotels and vacation rentals. It shows a house located on the Bosphorus; also in this case there is a variety of elements which could provide discriminative information like the carpet, the curtains and of course the view. We also checked the inference capability of our model on images from an Airbnb in Lisbon, see

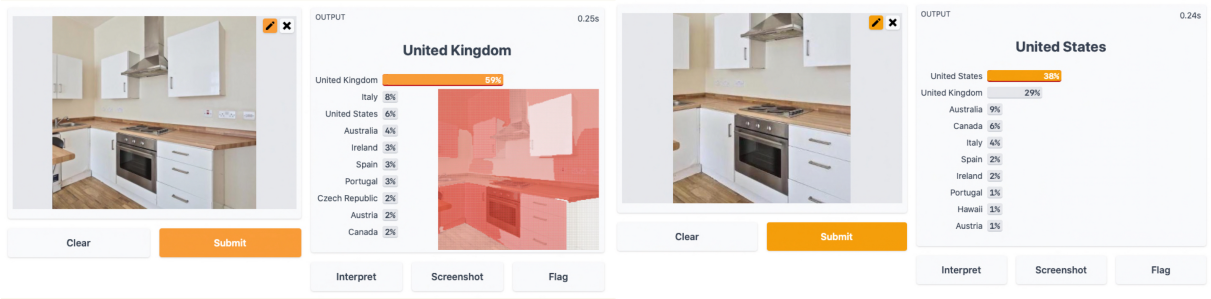


Figure 7.5: On the left the original image of a kitchen in London, with the associated SHAP interpretation, the more intense the red, the more positive the Shapley values are. On the right the same image after multiple crops, leading U.S to overtake U.K as the most probable country.

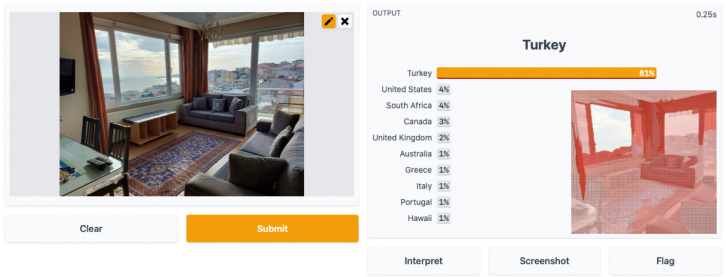


Figure 7.6: Top-10 predicted countries on a photo taken from Agoda web-page, showing a living room of a house in Turkey.

Fig. 7.7 where a colleague is currently working, of course before making the prediction we carefully checked that the mentioned Airbnb is not present in our dataset. Also in this case the model correctly predicted Portugal as the most probable country.

To conclude, I also took some pictures in my kitchen in Rome which were correctly classified. I present here one of them as a demonstrative example, Fig. 7.8. Even if there are many elements which could result easy to locate this kitchen for an Italian trained eye, probably for a tourist coming from Canada or China it could be harder to understand if this is an Italian or Spanish kitchen. Also in this case the union of many interior features as the tiled floor, the gas cooker, the counter on top, the window size and frame, the disposition of the kitchen and the curtains provide sufficient information to our model to accurately predict the location of the picture at country level.

Thanks to this demo we can highlight the fact that decorative styles, structural elements, materials, objects with functional purposes differ across geographic spaces, regions and countries, and that they can all be used together to predict the location of a given indoor image, at different level of granularity.

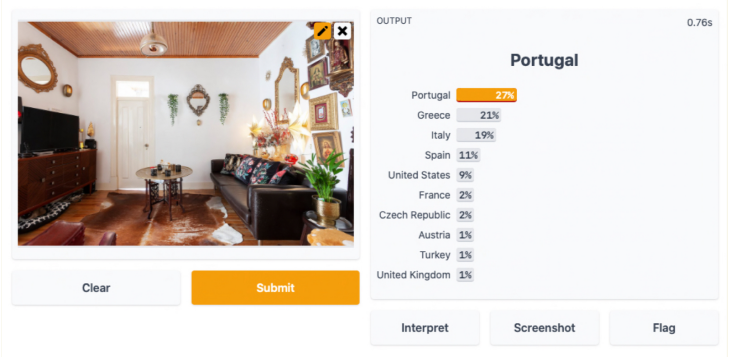


Figure 7.7: Top-10 predicted countries on a photo taken inside of a living room of an Airbnb in Lisbon.

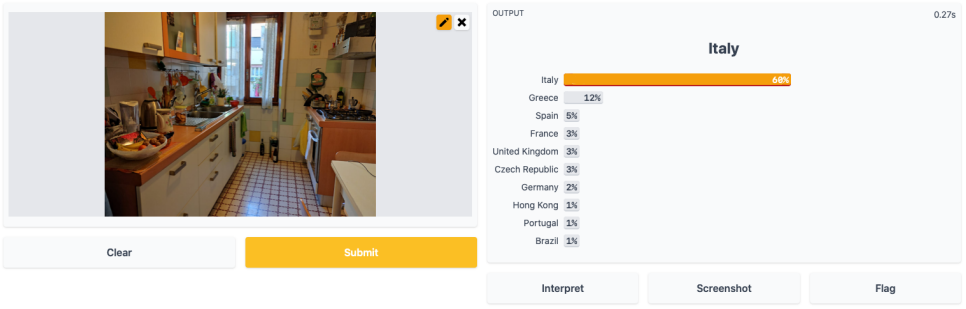


Figure 7.8: Top-10 predicted countries on a photo taken inside of kitchen in a Roman house.

8 | Conclusions and Future Research Directions

In this section we analyse the questions we asked ourselves in Chapter 3, comment the results we have achieved and what conclusions we have reached. Moreover, we discuss and propose some possible future research directions in this field.

8.1. Answering the Research Questions

The results obtained by our model on both Hotel50k and Airbnb dataset are quite surprising, reaching a rather high level of accuracy not only at sub-region and country level but also at the city level. Obviously, as the number of cities increases the task becomes more complicated and the accuracy of the predictions decreases as can be seen by comparing the Hotel50k dataset with 10458 cities where the best top-1 accuracy we reached is 20% while in the case of the Airbnb dataset with only 100 cities, it rises to 40%. Looking at some of the predictions made in Chapter 7, we are quite happy to note the generalisation capabilities of our model, which is able to make correct predictions even on photos taken inside private homes. We can therefore state that our model is in most cases capable of making more accurate predictions than the ones of an inexperienced, and on some occasions even experienced, human eye. The third question we asked ourselves was concerning which backbones and datasets are most suitable for a feature extractor to best extract information from interior scene images. In this respect, we can say that the latest vision transformers pretrained on Imagenet22k are certainly an effective solution. We argue, however, that datasets such as the SUN database or the ADE20k for scene categorisation might be very appropriate to pretrain a network for our task. Unfortunately, these datasets contain far fewer images than Imagenet22k and also fewer categories, for these reasons the best performance was obtained with a network pretrained on Imagenet22k, although the improvement is rather slight with respect to the results obtained with ADE20k (which has only 150 categories).

Regarding the second question we asked, we believe that it is possible to obtain useful visual explanations to understand the predictions of our network and to add informative elements to our knowledge for the indoor geolocation task. We have noticed that an approach that we consider valid can consist of segmenting all the objects and elements present in an indoor image and then using methods such as SHAP [22] or LIME [28] to understand which attributes mainly affect the prediction of our model. When repeating multiple times these XAI techniques on images from same scene in same location we could notice how there are some elements which characterize the geographic location of an image. E.g. we have noticed that ceiling with wooden beams, combined with floors with hydraulic tiles are very popular in Catalonia but also in ancient Roman houses while in Venice houses it is quite more common to see "Terrazzo" floors, which are made of chips of marble, quartz, granite, glass or other suitable materials, poured with cementitious binder, polymeric binder or a combination of both.

8.2. Future Research Directions

In this section we present ideas for some potential future extensions that we realised, during the development of our framework, they could improve the performance of the tool we created. In concrete terms we present four directions toward which our model could be extended.

1. **Dataset extension**: Collect data from new countries, for the Airbnb dataset, trying to extend the prediction capabilities of our model as far as possible. For this purpose, we argue that it is very important to also collect a large number of quality photos in which elements that can help to geolocate the images are visibly recognisable. We have noticed that in some cases the presence of power sockets, or writing in languages that can be highly informative, did not help our model to arrive at the predictions made. This is because in these cases there were not enough images with visibly recognisable power sockets present. We therefore argue that in addition to automatic scraping, which is helpful to download images in large quantities, in order to extend the learning capabilities of our model, it is also necessary to provide it with relevant quantities of images including semantically highly informative elements in order to extend its information extraction capability. We are currently working on extending the Airbnb dataset by adding photos of new countries and cities.

2. **Text Explanations**: In our framework we only dealt with visual explanations, trying to exploit the 150 categories present in the ADE20K dataset [44]. We argue that it could be useful to introduce a language Transformer model for image caption-

ing, not only to describe the objects in a scene, but also their spatial of functional relationship which may be key for a better understanding of the geolocation task. Moreover it has to be noticed that the introduction of vision transformer models, like the one we used, has brought the NLP and computer vision fields closer to each other. We argue therefore that our framework would benefit from a text explanation model and that the use of our vision transformer would result helpful to connect visual and NLP modeling, linking visual tasks to all semantics involved in language.

3. **Web Platform and Crowd Sourcing**: Create a more sophisticated demo which will include our visual explainability methods like GradCAM and SHAP on segmented images, allowing users to give us feedback on the usefulness of these methods and whether they think there are elements, that have not been highlighted, that could have helped to identify the location. This is of fundamental importance since, while the predicted location can be validate in a quantitative manner, judging the usefulness of the visual explanations for better understanding the model or see if there were discriminating elements which weren't used can only be carried out on a qualitative and subjective basis. Especially taking into account that people from different places have internal information that can enable them to distinguish houses from their countries more quickly and effectively than a person from another place. In addition, we believe it can be of great help to open a section within the web-page dedicated to uploading new geo-labelled images and useful annotations to extend the capabilities of our model.

4. **Segmentation Improvements**: The human eye approaches the visual world on several levels: categorising and contextualising different scenes by recognising objects made up of different components and at the same time identifying different types of textures and surfaces. We claim that a multi-task approach as the one presented by Xiao et al. in [42], which requires the machine vision systems to recognize simultaneously: scenes, objects, parts of objects, materials and textures, could give us additional meaningful visual information. For example we could use this approach to gather relevant statistics showing which objects, materials and textures are the most common within each scene (bedroom, kitchen, bathroom, living room) for each country.

Bibliography

[1] A. F. Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. URL `http://arxiv.org/abs/1803.08375`.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] A. Binder, G. Montavon, S. Bach, K. Müller, and W. Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. *CoRR*, abs/1604.00825, 2016. URL `http://arxiv.org/abs/1604.00825`.

[5] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 381–387, Cambridge, MA, USA, 2000. MIT Press.

[6] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. URL `http://arxiv.org/abs/1802.02611`.

[7] C.-T. D. Collaborative. The global dataset, 2020. URL `https://www.ctdatacollaborative.org/`.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner,

M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL `https://arxiv.org/abs/2010.11929`.

[11] I. J. Goodfellow, Y. Bengio, and A. Courville. Deep learning. `http://www.deeplearningbook.org`, 2016.

[12] Google. Google s2 geometry: spherical geometry library for manipulating geographic data. `https://github.com/google/s2geometry`, 2021.

[13] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. *CoRR*, abs/1706.04599, 2017. URL `http://arxiv.org/abs/1706.04599`.

[14] J. Hays and A. A. Efros. im2gps: estimating geographic information from a single image. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665`.

[19] X. Liu, S. Rahimi, and Z. Huang. Inside 50,000 living rooms: an assessment of global residential ornamentation using transfer learning. *EPJ Data Science*, 2019. URL `https://www.researchgate.net/publication/330979774_Inside_50000_living_rooms_an_assessment_of_global_residential_ornamentation_using_transfer_learning`.

[20] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021. URL `https://arxiv.org/abs/2103.14030`.

[21] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic

segmentation. *CoRR*, abs/1411.4038, 2014. URL http://arxiv.org/abs/1411.4038.

[22] S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017. URL http://arxiv.org/abs/1705.07874.

[23] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

[24] E. Müller-Budack, K. Pustu-Iren, and R. Ewerth. Geolocation estimation of photos using a hierarchical model and scene classification. In *ECCV (12)*, volume 11216 of *Lecture Notes in Computer Science*, pages 575–592. Springer, Sept. 2018. doi: https://doi.org/10.1007/978-3-030-01258-8_35. URL http://openaccess.thecvf.com/content_ECCV_2018/html/Eric_Muller-Budack_Geolocation_Estimation_of_ECCV_2018_paper.html.

[25] V. Negri. A hierarchical geolocation of indoor scenes with visual and text explanations using deep learning. Master's thesis, Politecnico di Milano, 2019.

[26] C. Pelletier, G. Webb, and F. Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11:523, 03 2019. doi: 10.3390/rs11050523.

[27] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[28] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL http://arxiv.org/abs/1602.04938.

[29] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[30] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

[31] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *CoRR*, abs/1605.01713, 2016. URL http://arxiv.org/abs/1605.01713.

[32] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. *CoRR*, abs/1704.02685, 2017. URL `http://arxiv.org/abs/1704.02685`.

[33] L. N. Smith. Cyclical learning rates for training neural networks, 2017.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435.

[35] A. Stylianou, H. Xuan, M. Shende, J. Brandt, R. Souvenir, and R. Pless. Hotels-50k: A global hotel recognition dataset. *CoRR*, abs/1901.11397, 2019. URL `http://arxiv.org/abs/1901.11397`.

[36] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017. URL `http://arxiv.org/abs/1703.01365`.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[38] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014. URL `http://arxiv.org/abs/1411.4555`.

[39] N. N. Vo, N. Jacobs, and J. Hays. Revisiting IM2GPS in the deep learning era. *CoRR*, abs/1705.04838, 2017. URL `http://arxiv.org/abs/1705.04838`.

[40] T. Weyand, I. Kostrikov, and J. Philbin. Planet - photo geolocation with convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016.

[41] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. pages 3485–3492, 2010. doi: 10.1109/CVPR.2010.5539970.

[42] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun. Unified perceptual parsing for scene understanding. *CoRR*, abs/1807.10221, 2018. URL `http://arxiv.org/abs/1807.10221`.

[43] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015. URL `http://arxiv.org/abs/1512.04150`.

[44] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ADE20K dataset. *CoRR*, abs/1608.05442, 2016. URL `http://arxiv.org/abs/1608.05442`.

[45] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

List of Figures

List of Tables

List of Symbols

Acronym	Description
$CDTC$ Counter-Trafficking Data Collaborative	
$P2P$	Peer-to-peer
$ILSVRC$	ImageNet Large Scale Visual Recognition Challenge
AI	Artificial Intelligence
NLP	Natural Language Processing
NMT	Neural Machine Translation
ML	Machine Learning
DL	Deep Learning
RNN	Recursive Neural Network
CNN	Convolutional Neural Network
DCN	Deformable Convolutional Network
MLP	Multi-layer Perceptron
ViT	Vision Transformer
$LSTM$	Long-Short Term Memory
$FCNN$	Fully-Connected Neural Network
$ReLU$	Rectified Linear Unit
RGB	Red-Green-Blue
GBC	Geolocation by Classification
IBL	Image-Based Localization
IG	Integrated Gradients.
GPS	Global Positioning System
$VLAD$	Vector of Locally Aggregated Descriptor
CAM	Class Activation Mapping
$Grad-CAM$	Gradient-weighted Class Activation Mapping
CCE	Categorical Cross Entropy
SGD	Stocastic Gradient Descent
LR	Learning Rate

Acronym	Description
ES	Early Stopping
$SHAP$	SHapley Additive exPlanations
$IoMc$	Improvement over Majority class
MC	Multi-Class
ECE	Expected Calibration Error
$FLOPS$	FLoating-point Operations per Second
$MBConv$	MobileNet convolution
$BERT$	Bidirectional Encoder Representations from Transformers

Acknowledgements

Now that my academic journey is coming to an end, it is time to thank all the people who have supported and helped me along the way. First, I would like to thank my supervisor: Professor Mark James Carman, for giving me the opportunity to develop a thesis on topics which are very interesting to me and which have the potential to be socially useful.

I would also like to thank most of all my parents, Lory and Ely, for accompanying me during this long journey that started well before the Politecnico. They have always supported me in difficult times and motivated me to be ambitious.

Special thanks go to Riccardo Vitali, a fellow student since our bachelor degree in pure mathematics at Sapienza University. Riccardo encouraged me to join him at the Polimi, undertaking a path of which I am very proud and satisfied that has led me to this day. (Although I confess that I hated him a little while I was preparing for my Arf exam, but I know that to be forgiven he will invite me to go sailing on his boat this summer).

I want to dedicate a few lines to thank some colleagues as Beniamino, Leonardo, Aurelio and Ariel who have been a source of inspiration and constant learning for me. A sincere thank also go to Marika and Isabella who were among the first people to welcome me at the polytechnic and with whom a great friendship continues. A more than special thank go to Marco Biazzo, a dear colleague who during the pandemic proved to be not only an excellent mathematician but also a good friend.

I also want to apologize and thank Helena, Pietro and Matej for all the times that they had to endure my pre-exams anxiety and for always supporting and helping me in times of need.

Finally I want to thank Sergio, Ludovica and all my friends from Rome who I left to start this adventure at the polytechnic but who have always been close to me nonetheless.