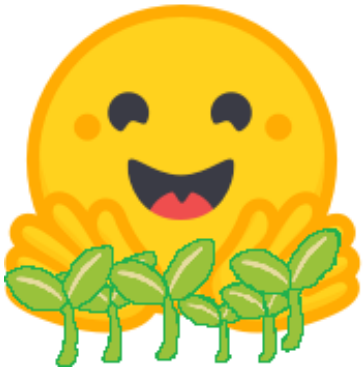
TÉCNICO LISBOA



Deep Learning Methods for Processing Digitized Herbarium Specimens

Filipe Lucas Borges Seleiro Martins

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics engineering

Supervisors: Prof. Jacinto Paulo Simões Estima
Prof. Bruno Emanuel Da Graça Martins

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Jacinto Paulo Simões Estima
Member of the Committee: Prof. David Manuel Martins de Matos

June 2022

I want to dedicate this to every friend and colleagues that I meet during my academic journey. The knowledge sharing, the brainstorming sessions we have done in order to overcome the challenges presented to us. Also the perseverance and quest of the pursuit of the know how's we so enjoyed. Also to my close friends that helped me to keep sane and relaxing and fun times, while keep been understand full of the stress and the sacrifices made. Lastly to my close relatives that helped me though out this journey, without them this would not be possible, specially to my brother that helped me financially on the acquaintance of the GPU that i used along on this thesis, for them i would probably be here, I'm grateful for everything.

Acknowledgments

I want to dedicate this section to the teachers that have guided this thesis and helped me keep in focus setting the goals clear to accomplish, also by allowing me to change a bit the original thesis proposition by changing the model to use and allowing to experiment with more modern architectures, that lead to this case study of model design. Also to the teacher of U.C. Computational Intelligence for the Internet of Things taught by João Paulo Carvalho that sparked the curiosity for this field.

Resumo

Com centenas de colecções de herbários, actualmente em Museus de História Natural e outras instituições semelhantes, acumulou-se um valioso património. Recentes iniciativas iniciaram ambiciosos planos de preservação para digitalizar esta informação e disponibilizá-la aos botânicos e ao público em geral através de portais web. Esta informação é crucial para o estudo da diversidade vegetal, ecologia, evolução e genética. Um Herbário é uma colecção de espécimes de plantas preservadas e meta-dados utilizados para o estudo científico. O método de digitalização e catalogação utilizando a visão computacional, bem como as abordagens machine learning aplicadas às folhas de herbário, podem ambos ser considerados promissores, métodos recentes baseados em redes neurais profundas ainda não estão bem estudados para a resolução deste problema em comparação com outras áreas. Passaremos a projectar um modelo que pode ser utilizado para alcançar a próxima geração de precisão para a catalogação de Herbários. Para atingir este objectivo, exploraremos aplicar ao caso de estudo modelos e técnicas mais avançadas. Utilizaremos dois modelos para extrair informação útil para a catalogar as espécies, o modelo YOLOv4 que terá a tarefa de identificar as etiquetas presentes na folha em conjunto com um modelo Transformer que ira extrair os dados uteis para catalogação utilizando uma técnica de geração de texto condicionado em imagens. Os resultados obtidos foram pouco conclusivos, devido ao tipo de rede neuronal desenvolvida ser bastante recente. Em conclusão foi que o modelo e bom para dados estandardizado, mas falha por completo em dados do mundo real por serem demasiado aleatorios.

Palavras-chave: Transformers, Yolov4, herbarios, reconhecimento óptico de caracteres, recognhecimento de campos texto, Geração de texto condicionado em imagens

Abstract

Hundreds of herbarium collections, currently in Natural History Museums and other similar institutions, have accumulated a valuable heritage and knowledge of plants over several centuries. Recent initiatives started ambitious preservation plans to digitize this information and make it available to botanists and the general public through web portals. Such information is crucial for the study of plant diversity, ecology, evolution, and genetics. The method of digitization and cataloging using computer vision, as well as the machine learning approaches applied to herbarium sheets, can both be considered promising, recent methods based on deep neural network are still not well studied in this problem domain in comparison to other areas. We will go over a model that can be used to achieve next generation precision and utilities for this field of cataloging Herbarium. To achieve this goal, we will explore and try to apply state of the art techniques, models and architectures to the study case. We will use two models to extract useful information for cataloging the species, the YOLOv4 model that will have the task of extracting the labels present on the sheet together with the Transformer model that will extract useful data for cataloging using a technique of text generation conditioned on images. The results obtained were inconclusive, because the type of neural network developed was quite recent, more tests would have to be done. Concluding the model is good for standardized data but fails completely on real world data that is not very standardized.

Keywords:Transformers, Yolov4, VIT, GPT2, TOCR, Herbarium, Image Recognition and Classification, Optical Character Recognition, Specimens, Text generation conditioned on images

Contents

List of Tables

List of Figures

Nomenclature

AI Artificial Intelligence

CAD Computer-Aided Design

CAM Channel Attention Module

$CBAM$ Convolutional Block Attention Module

CNN Convolution Neural Network

$dVAE$ discrete Variational AutoEncoder

FPN Feature Pyramid Networks

$LMhead$ Language Modeling head

$LSTM$ Long Short Term Memory (network)

MIM Masked Image Modeling

MLP Multi-layer Perceptron

MSA Multi-head Self Attention

NER Name Entity Recognition

NLP Natural Language Processing

OCR Optical Character Recognition

$R-CNN$ Recurrent Convolution Neural Network

$ResNet$ Residual Neural Network

SGD Stochastic Gradient Descent

SPP Optical Character Recognition

$tOCR$ Transformer Optical Character Recognition

VGG Visual Geometry Group

VIA VGG Image Anotator

VIT Vision Transformers

WER Word Error Rate

$YOLO$ You Only Look Once

Chapter 1

Introduction

With the age of a digital world, and subsequently of the internet, knowledge became more accessible, written and updated from almost anywhere in the globe. Creating a need to digitize some physical collections to expand their accessibility. The herbarium collections are no exception to these problems, these are mostly physically present in public or private institutions. With the goal of this theses helping to alleviate on the problem of cataloging Herbarium Colection our solution aims to automatize the digitization of these collection using deep learning methods.

By developing a state of the art model architecture that has yet to be tested in this field of cataloging herbarium collections. Our solution has the implementation of a non end-to-end model with the use of two different types and architectures. The first model was created with the conventional convoluted neural networks and a top charting model called YOLOv4 for the task of Image Detection, Recognition, and Classification. This model will identify of the text labels presented on the herbarium sheet that are a required input for our second model. The second model been a Transformer network that we developed for this use case, the task of this model is to identifying and extracting useful data for cataloging, using Text Generation Conditioned by Images. Theses models that make use of multiple encoders and decoder this technique are not fully explored in the AI community. We will be implementing using as a base the Transformers library by Huggingface [1]. The challenge of our implementation was to add support on the Huggingface library [1] for multiple Vision encoders to multiple decoders models. By such combination of transformer stacks: a Vision Transformers as a encoders and a NLP(Natural Language Processing) Transformers models as a decoder we could generate text conditioned by images. We thought of this architecture primarily to expand the functionality of the original task of Text Generation but to exploit the cross attention mechanism to allow each decoder to focus on identifying and extracting different data from the same input in hopes to function as a NER(name entity recognition). Our main challenge will be presented on the herbarium sheets themselves, where most of them are not fully standardized, combined with multiple other factors that will be a considerable when analysing the performance of our models for cataloging such collections.

(a) (b) (c) (d)

Figure 1.1: Herbarium Exemplaries

1.1 Motivation

The preservation of our knowledge is a very important task. Making it to be available by digital media simultaneously increases its accessibility and contribute for the preservation of it. Now a days this preservation task is done mostly by institutions with manual labour in the field of Herbarium collections thus requiring dedicated persons and time. We aim to automate the cataloging and archivation processes,by implementing a state of the art Ai model.

1.2 Objectives and Deliverables

The AI field is in constant evolution either by the creation of new architectures and models or new techniques being created to solve different challenges or optimizing existent models, but there is a little focus on the field of conservation herbarium collections. This lead us to implement and test newer models to tackle this problem.

As of yet most institutions catalogue this by hand, setting us on the path of developing a state of the art model using the latest advancements on the field. Our development since the beginning was very time bound, yet we achieved a fully working model with the minimal features using torch and hugging face library. Hugging Face library is an open source library, and by further developing using it will hopefully allow the general public to do more research over the subject, and on this type model. Another advancement we tried to achieve was the development of pre-processing, allowing the possibility of multiplying and creating multiple images of a single image using hand made annotations and multiple computer generated labels. This pre-processing can be yet be improved and should make use of new methodologies of blending and text writing to generate examples that are more closely resemble a real image. by this brief explanation, we set as objectives for this work to:

- Creation of a pre-processing module that generates a Dataset on which to train our models by filtering a subset of the original database of cataloged herbarium sheets, and consequently generate

multiple images from a single hand annotated and filtered image thus allowing to generate even bigger Datasets for training.

- Creation of a model using the latest techniques and architectures for cataloging herbarium collections and explaining the benefits and the shortcomings of our implementation.

1.3 Thesis Outline

This dissertation is organized as follows: The motivation and challenge that our dissertation will address were introduced in Chapter 1. We also set our dissertation's objectives in this chapter. Chapter 2 will present the logic and the thought process for why we chose to develop this specific model by presenting the work done by the models we got inspired on and their functionality. Chapter 3 will introduce all the implementation required for our thesis starting by presenting the original database used to create the Dataset and potentially problems that our model might need to overcome. Following the preprocessing module that we used to create the Datasets for our models. As well our implementation of our developed model, as well as, presenting some problems found for implementing on Huggingface library. Chapter 4 will discuss some Results obtain and give some closure to the document. Finally, Chapter 5 is the final conclusions and give possible directions for future work or other applications where these architecture can be used.

Chapter 2

Models Background and related work

In this chapter, we will review the multiple models that paved the path for our model creation, by reviewing each implementation and how we can use such techniques to achieve our goal.

2.1 Theoretical Overview

Since our solution doesn't follow a end to end architecture, it is composed of multiple models that work in conjunction to accomplish a single task. In our case we will try to accomplish our goal using two models. One model for Object Recognition and Region Segmentation called YOLOv4, that we will go in depth on what techniques it improves on and the logic behind it; The second model was developed based on the unpublished paper of Microsoft TrOCR[2] and similar papers making use of vision encoders to text decoders. We will be experimenting with model like ViT[3], Beit[4] and Deit[5] as our vision encoder Transformer stacks with GPT2[6], Bert[7], Roberta[8] and other general purpose NLP transformer stacks as our decoders. Both types of models were chosen carefully based on performance on multiple established benchmarks [9] for specific tasks and must be for generic purpose.

2.2 Convolutional Neural Networks for Computer Vision

2.2.1 CNN Applied to Computer Vision

The concept on Convolutional Neural Networks (CNN) was specially developed for imaging processing. They were developed based on the working of the neurons of an animal visual cortex, individual cortical neurons are only connected to a certain region of the visual field known as receptive field. The receptive fields of different neurons can partially overlap to ensure they cover the entire visual field. For simplification we will use examples applied to computer vision. Starting with how a basic architecture is usually constructed for these type of problem, then we'll go in depth into each layer and what their functions are, as well as some concerns to be on the look out. For a CNN to make any decision over an image, we have to achieve good results in two major steps: feature extraction and classification. For that to happen, the user must be concise on the feature selection relevant to the problem, together with

choosing a proper method for classification. The feature extraction process is everything related with the preparation of the input image; the name of the modified image is known as feature maps. Classification is usually done by dense network (MLP) that makes the decision over the previous feature maps. Since

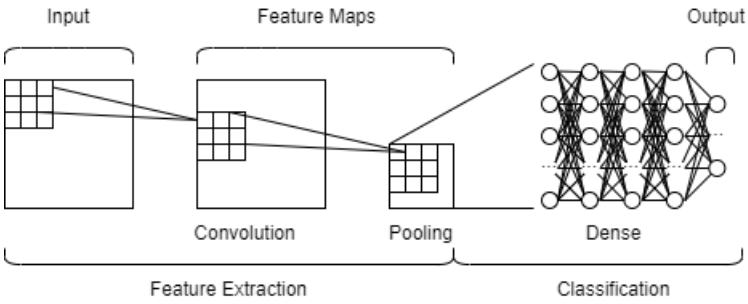


Figure 2.1: Basic CNN architecture.

the CNN is based on animal vision, the input follows a NHWC, making each letter correspond to a video-like component where N is the number of images in the batch, H the height of the image, W the width of the image and C the number of channels of the image (ex: 3 for RGB, 1 for grayscale).

Before talking about the operation convolution, we need to understand what are the existent types of convolution matrix and how they work. The utility of the convolution matrix (kernel) serves to do feature selection by removing unnecessary information from our image. The Kernel represents a matrix of coefficients usually with the common size of 3x3 or 5x5, that is applied to a patch of pixels in the process called convolution. By doing this we can extract some features of the image that are relevant to our model, like edges detecting or applying some sharpening, blur and other pre-processing that seems relevant to our problem.

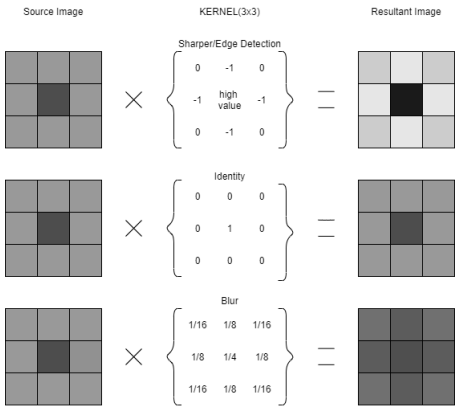


Figure 2.2: Kernel visualization.

The Convolution is a process of applying our kernel filter to our image for the purpose of filtering the relevant data for our model. By definition, we can represent the convolution by expression:

$$\mathrm{g}(x,y) = k * \mathrm{f}(x,y) = \sum_{dx=-i}^{i} \sum_{dy=-j}^{j} w(dx,dy).f(x+dx,y+dy) \tag{2.1}$$

Where g(x,y) resultant image, k is the kernel and f(x,y) our original image. This generalization is only

applied when the resultant image is the same size as the original, due to padding or striding values the resulting feature map may not have the same size when compared to the original.

The Dilated Convolutions is a special case of the conventional convolution whose inspiration of its development is to increase the receptive field of the operation. By increasing the receptive field, it allows for aggregation of multi-scale contextual information on the rest of the image allowing these techniques to happen without losing resolution [10]. To increase the receptive field with the Conventional convolution, we need to increase the kernel size thus losing border information (described in depth on the next paragraph) to prevent having to add padding.

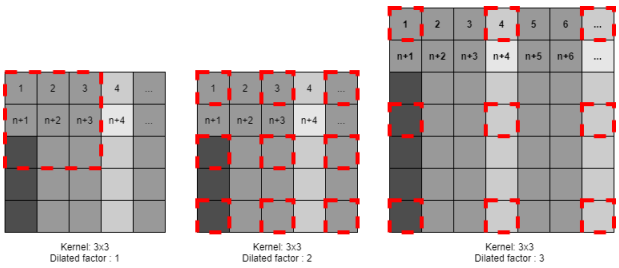


Figure 2.3: Dilated Convolutions Visualization.

Padding is a technique that consists on filling the boundaries of the image with a value (usually zero), in order to prevent loss of border information when applying convolution. Since the kernel matrix is usually bigger than 1x1, the outer edge pixels do not count for the center of the cross correlation, resulting in some loss of information. This technique also serves to keep the same image size in order to simplify the code. This case is exemplified by the figure 2.4 below.

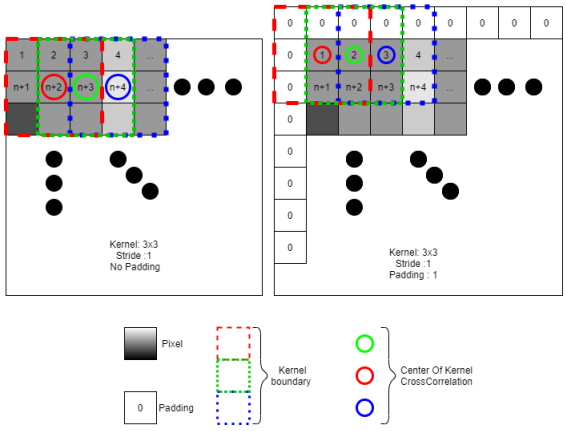


Figure 2.4: Padding Visualization.

Stride is a value that represents the number on indexes the kernel jumps for the next convolution. It can also be responsible for some resolution loss. This value should be very carefully chosen in conjunction with padding and the kernel size due to some impossible combinations that result in part of the convolution occurring outside of the picture.

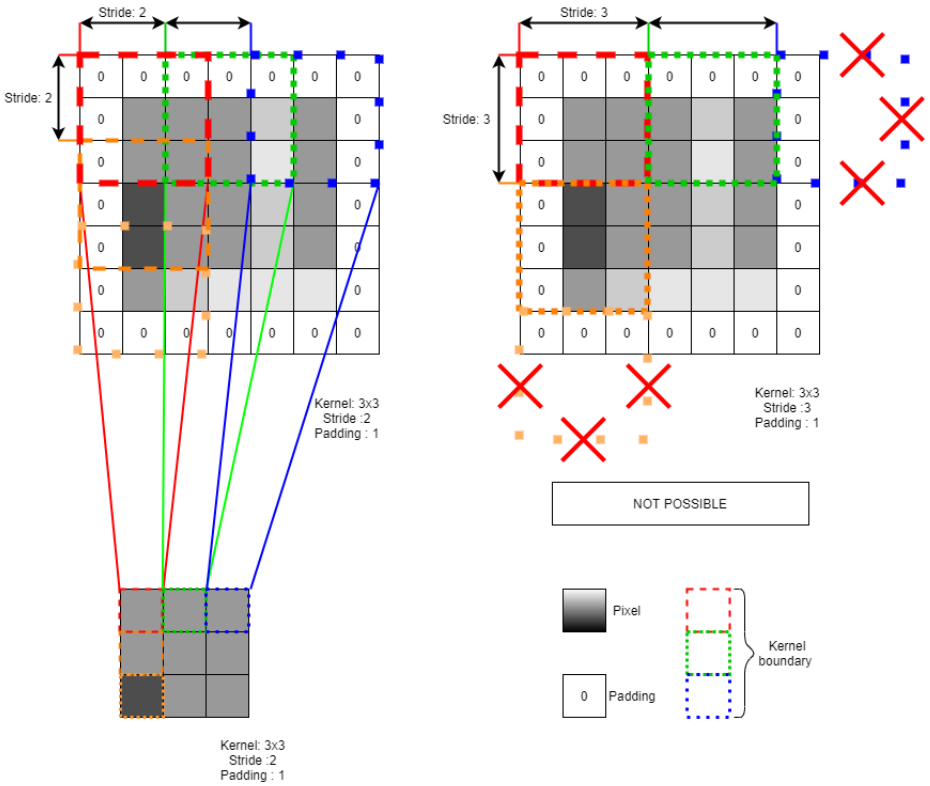


Figure 2.5: Stride visualization.

To calculate the size in one of the dimensions of the resulting feature map, we need to take into account some of the parameters explained earlier. When this formulation is applied, the result must be integer, otherwise it's impossible for the chosen values of Stride, Padding and Kernel size to be compatible, thus resulting in the convolution happening outside our image boundaries.

$$\text{FeatureMap}_w = \left(\frac{W - K + 2P}{S}\right) + 1 \tag{2.2}$$

W represents the input size in one dimension, K is the Kernel size, P is the padding and S is the stride.

Pooling operation is used to reduce the spatial dimension, resulting in a down sampling of the feature map. This operation affects the weight and height, but not the depth. To apply this logic of down sampling there are multiple methods to convert a patch of pixels into a single output, the most common being: maximizing pooling, where the max value of the patch is the only value kept; and the average, where it's kept the context of the whole patch.

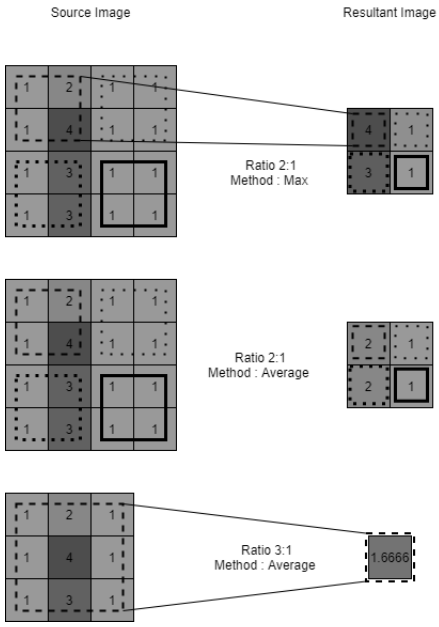


Figure 2.6: Pooling visualization.

There are two main implementations that are used for image classification: one is the Linear Classifier; another is the Dense Neural Network (DNN) Classifier. There are advantages and disadvantages for both implementations; Linear Classifier is usually much faster and can be accurate when talking to simple problems that can be easily linear and separable in data format; while the DNN is more complex and computationally expensive, it is more suitable for general purpose, for the ability of better distinguishing scattered data sets. Linear Classifier achieves its predicaments by applying a linear combination of the feature set applied. Thus this model can only distinguish multiple classes that can be separated by a hyperplane. This method is extremely light to compute. DNN Classifier implements a classical version of MLP as we described before. The architecture is as a Feed forward Neural Network as is a common practice since is the most general purpose Classifier due to the flexibility of the MLP to separate disperse clusters of classes.

2.2.2 YoloV4 - Object Recognition and Region Segmentation

To go in depth on the YOLO (You Only Look Once)v4 [11] model that we used for Object Recognition and Region Segmentation to identify regions of written data on herbarium sheets, we will explain the model architecture and techniques used to compose this top charting model. The importance of this region is where most of our meta data resides like Name of the herbaria, locality of capture and date. This Model was choose based on performance and accuracy over the Object Detection on COCO dataset[12] as of the 2021 according the leaderboard available at paperswithcode [9], as of today this model performance still pretty good but recently losing top position by newer transformer models.

The SOTA CNN architecture in the field of object detection Networks using CNN architectures can be separated into two main categories: The One-Stage detector, where we will explain the implementation of YOLO; Two-Stage detector since its out of the scope of this thesis, this can be simply be explained each regions are propositions and they will pass-through a specialized CNN layers to further refine. Before explaining the proposed implementations of the models YOLOV4, we need to clarify some methods and the basic structure for this type of purpose.

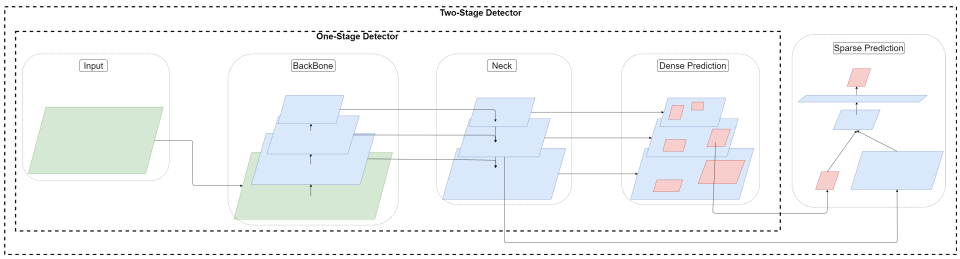


Figure 2.7: State of the Art typical CNN Architecture for object detection.

Each block of the architecture has a function: the Backbone where the feature-extraction occurs, the Neck block which purpose is to add extra layers and some refinement between the backbone dense prediction block, and the Head that can be single stage and two stage and is usually where the classification and the attribution of bounding boxes occur.
Before explaining the YOLOV4 architecture we will go over some newer techniques implemented.

The backbone used on YOLOV4 is the CSPDarknet53 that is based on the previous backbone Darknet53 2.8??, as the change of name implies the addition of a technique of Cross Stage Partial Network (CSPNet) [13] to the Darknet53. The goal of technique is to lighten the weight of the computational need and memory costs by achieving richer gradients. This concept works because having richer gradients meaning there is a reduction of needed larger digits representation than our native processor can achieve (e.g Less bits alleviates memory requirements so naturally achieving faster calculations, also Double Float Precision calculations need more Processor Clocks to calculate translating on slower calculations). The way this technique achieves richer gradients is by partitioning feature map of the base layer into two parts and then merging them through a proposed cross-stage hierarchy.

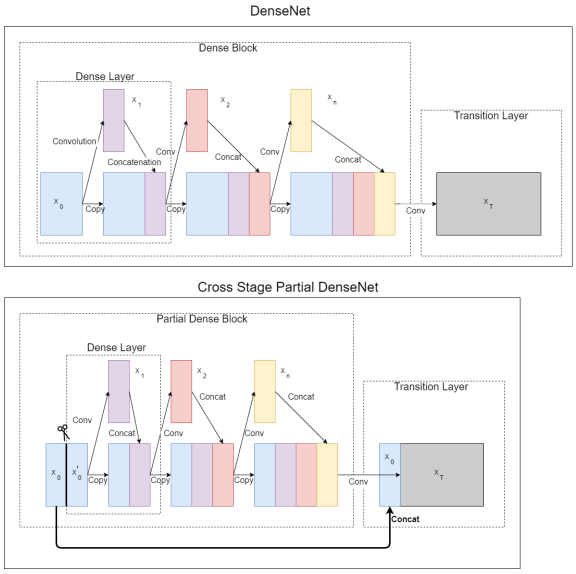


Figure 2.8: Cross Stage Partial Network applied to DenseNET[11].

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

Table 2.1: Darknet53 composed with 53 layers of DenseNET[14].

Another concept that is used besides the CSPNet in CSPDarknet53 for YOLOV4 is that the original concept of Darknet53 [14] actually is a hybrid network with CNN and residual neural network (ResNet) [15]. The principle is similar to the CSP, where the information can skip layers and still be processed together with features like relation $F(x) + X$ in order to prevent vanishing gradients. It also serves to mitigate a problem of accuracy degradation. This happens when there is a excess of layers which then leads the deep neural network to produce higher training errors. The hypothesis created is that its easier to optimize residual mapping than to optimize an unreferenced mapping.

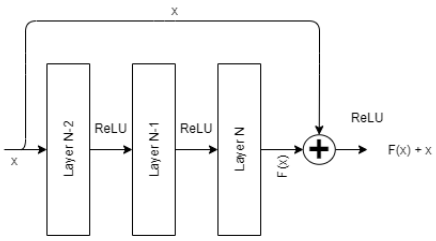


Figure 2.9: ResNet exemplification [15].

The YOLO and R-CNN networks has at is disposal multiple techniques that enhance the receptive field, one of the used techniques is the use of Spatial Pyramid Pooling (SPP)[16]. This technique was created to eliminate a technical limitation of the network and requires fixed image sizes, plus the SPP ability to generate a fixed length output regardless of the input, therefore avoiding cropping or distorting techniques where we can modify or lose context to our convolutions. The SPP layer pools the features and aggregates into a single input the information.

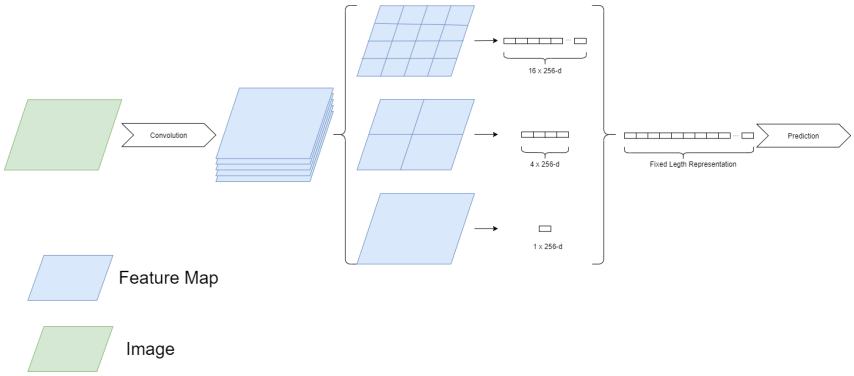


Figure 2.10: SPP visualization [16].

One of these techniques is the use of a concept known as Feature Pyramid Networks (FPN) [17]. This method aims to explore object detection on multiple feature map on different scales thus constructing a pyramid-like scheme of images or feature maps. The concept behind this is to take advantage of featurization on each level of an image pyramid. It produces a multi-scale feature representation where all levels are semantically strong. This method is proven to give accurate results while the process of each map prediction can be done in parallel so the inference time doesn't have much time penalty.

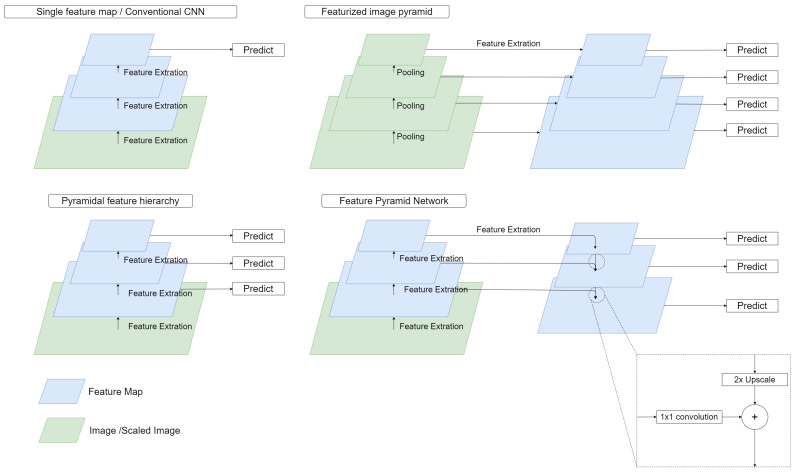


Figure 2.11: CNN and Feature Pyramid Networks.

Where the FPN was a technique used on the Yolo v3 [14] with great results. The concept of Path Aggregation Network PANet [18] is present, although modified, in the neck of the YOLOV4 model replacing the previous layer of FPN for this version. It's mainly incorporated in the model to enhance the process of instance segmentation by preserving spatial information. This modification suggested and implemented version of PANet is a simple modification that instead of the usual addition, it was replaced with concatenation.

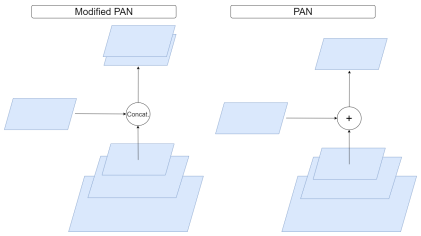


Figure 2.12: Modified PAN as implemented on the YOLOv4 model.

The implementation of a Spatial Attention Module (SAM) serves to keep spatial attention on convolutions. The original concept was proposed on Convolutional Block Attention Module (CBAM) [19], this concept uses two modules combining the channel attention module (CAM) and then SAM. It's well known that attention plays an important role in human perception; So the development of this plugin module that can add some attention mechanisms to networks was initially the goal of the original concept. This is done by generating a single feature map by concatenating, both a max pooling and average pooling feature maps, in depth to generate a single feature map that is going to be interpreted by a convolution and normalized by sigmoid function. The authors of YOLO v4 tried to improve on this model, and were able to simplify the concept by removing the pooling operations, applying the convolutions to the input, and normalizing with a sigmoid that generates values between 0 and 1, which is later multiplied by the original input in a process similar to masking.

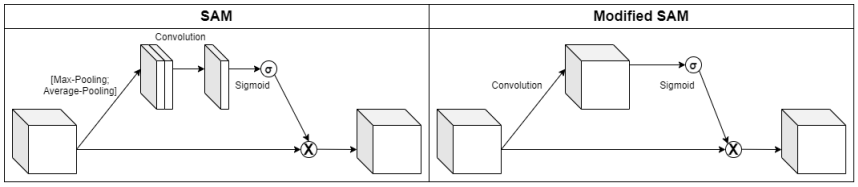


Figure 2.13: Modified Sam as implemented on the YOLOv4 model.

Now that we got the grasp of essential techniques and concepts used in this type of process, let's start with YOLO v4 where the goal of this paper was to deliver in 3 main aspects: increase the efficiency and accuracy so that could be run and trained on accessible Graphics Processing Units (GPU). To achieve that, the authors modified some state of the art methods as previously explained by trying to make them more efficient and suitable for single GPU training.

YOLOV4 model consists of: CSPDarknet53 as referred for backbone; The neck which is composed with SPP, PAN; And for detection we got the YOLOv3 head that makes use of FPN to improve compatibility.

The first step of the detector is to predict where to place the bounding box. For this to happen the network calculates the coordinates for each of the 4 corners, the feature map from the neck will be separated onto 3 parallel streams where the spatial dimension between them is different; while the detector is simply a dense network stacked on top of convolutions that reshape the feature maps. The next step is just a simple fully connected network and the model allows that each prediction box may contain multiple multi-label classifications.

Since the original Yolov4 implementation was on Darknet, we used a repository made by Tianxiaomo[20]that had made the accurate translation and implementation of the model on to python pytorch. This translation in conjunction of a module that converts the original Darknet checkpoint trained in MS COCO dataset[12] into pytorch checkpoint[21]. Since this model was not created by us we will give full credits to the original creator Tianxiaomo on Github [20]. We fine tuned the YOLOv4 model using the pretrained checkpoint in the MS COCO dataset[12] leading us to a shorter training period and great results.

2.3 The Transformer Architecture - Self Attention advancements and the vision transformers

Despite the multiple advances on this field of computer vision, there is a new emerging architecture that was developed specially to be used with Natural Language processing(NLP) [22], with this types of networks came some important advancements of attention mechanism ever made, these networks called transformers networks were crafted with a self attention mechanism in mind. This Transformer networks have been developed by Google, and multiple models based on this architecture have been created with a huge success like Bert[7] achieving the best results for the NLP [23]. With these advancements there was a need to experiment and expand the usability of this architecture on other fields. Leading to the experimentation on the field of computer vision, the creation of ViT [3] lead to some great results already paving the path to be promising model of Vision Transformers. Another main architectural advantage that has over tradition CNN is that the Transformer networks are highly parallel computational by design.

Our main goal is still to predict accurate results based on the context of the input. There are two main transformer stacks on this type of architecture an encoder and a decoder. The encoder where the input sequence $x = (x_1, x_2, ..., x_n)$ is going to get encoded (conversion to tensor) to a continuous representation on our space with vector of $z = (z_1, z_2, ..., z_n)$, for a given z after it's used as a input for the decoder that generates a vector of our output $y = (y_1, y_2, ..., y_n)$ thus this architecture are know as tensor to tensor or sequence to sequence.

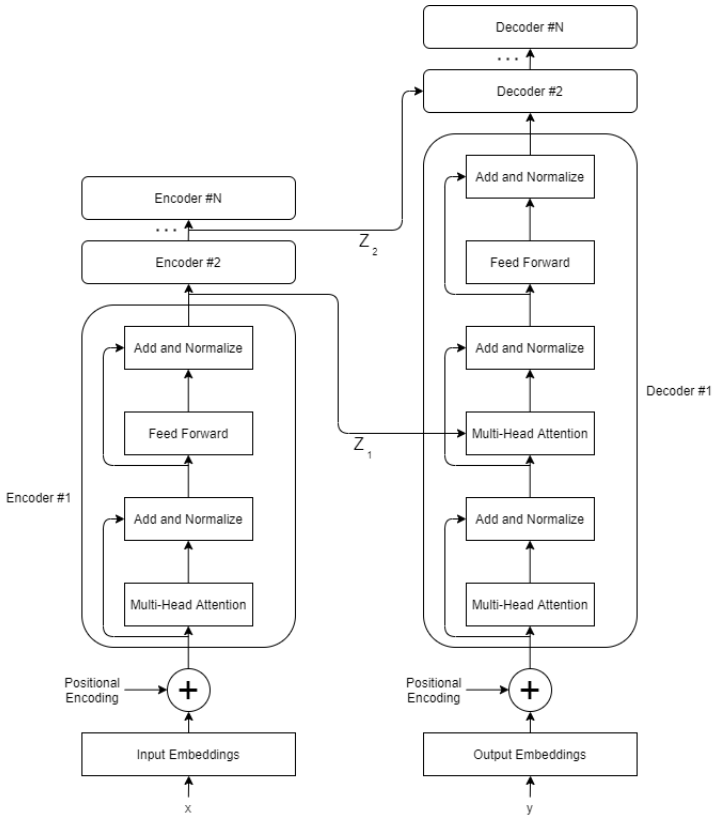


Figure 2.14: The Transformer neural network architecture.

Before talking about our encoder stack, there is some pre-processing required for our transformer to work as intended. This process begins with the embeddings, which consists encoding our input to a vector of size d_{model}. This size is static for any input and it is going to have a representation in our vector space, optimally tokens with similar meaning should be closer to one another in our dimension. Since this architecture doesn't contain any type of recurrence, it is necessary to add positional context for each token, this is done through with positional encoder. Taking the size of our token and modulates through a sinusoidal function with different frequencies according to the size.

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{/textmodel}}}\right) \tag{2.3}$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{/textmodel}}}\right) \tag{2.4}$$

Where pos is the current position and i is the total dimension.

An encoder module is constructed mainly with two sub-layers. The first one is a multi-head attention mechanism that focuses on how surrounding positions affect the current one, it achieves this by doing a all to all comparison and calculating a attention value. The second one is a simple position-wise fully connected feed-forward network. Decoder structure is similar to the encoder blocks while the difference is the repetition of two sub-layers on each decoder block. The additional sub-layers that composed the decoder block are a normalizing layer that regulates the input to an additional multi-head attention layer. This additional layer does a process called masking with the purpose to impose that the prediction over the position i are only imposed and related by less than i positions.

Where this network improves is the new mechanism imposed on the multi-head self attention layers allowing to make a correlation between the inputs and an output set based on the best probability of our inference. The way these functions achieves this is by calculating a attention value for each input that can be described Scaled dot product of multiple values. Those values that compose the aforementioned attention function are defined by mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors.

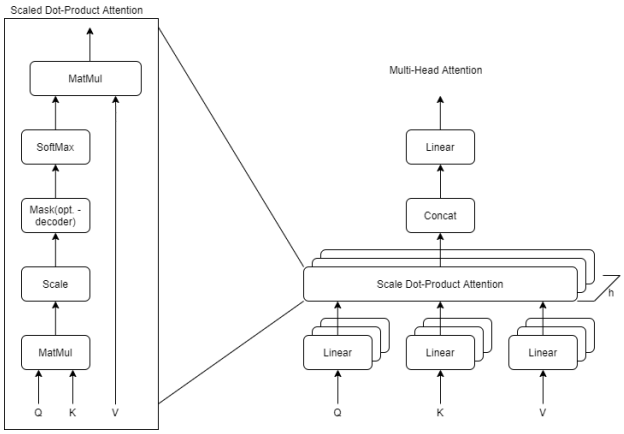


Figure 2.15: Scaled-Dot product and Multi head-Attention.

The scaled dot-product attention can be described 2.15 by the calculation for each embedding where each will generate a query vector and a set of key-value vectors. Those vectors are a representation of useful abstractions relevant to our problem. The vector are generated by multiplying our input vector and a weight of matrices that was previously trained both Query(Q) and Key(K) vector have the same dimension d_k and d_v for the Value vector. Since in practice a set of queries are computed simultaneously, we packed them together into a matrix ($Q \in \mathbb{R}^{w \times d_k}$), that consists a matrix with the queries vectors multiplied by the weight matrix ($W^Q \in \mathbb{R}^{model \times d_k}$). The same process is applied to keys vectors and values vectors where each is multiplied by a two weight matrices ($W^K \in \mathbb{R}^{model \times d_k}$ and $W^V \in \mathbb{R}^{model \times d_v}$) and also packed together into matrices: ($K \in \mathbb{R}^{w \times d_k}$) and ($V \in \mathbb{R}^{w \times d_v}$). Our attention value will be represented as:

$$\text{Attention}\,(Q, K, V) = softmax\left(\frac{QK^t}{\sqrt{d_k}}\right)V \tag{2.5}$$

The scale factor was needed to counteract where large values produced by the dot-product could push the Softmax function to regions where gradient is small 1 / $\sqrt{d_k}$. So when applying scaling, our large values number will be pushed closer to zero where the Softmax function has more variance thus improving on this limitation. The multi-head attention mechanism allows that our model can access to multiple representations in different sub-spaced by concatenating the information from multiple attention layers, also known as heads.

This can be expressed as:

$$\text{MultiHead}\,(Q, K, V) = Concat\,(head_1, head_2, ..., head_h)\,W^O \tag{2.6}$$

$$\text{Where } head_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

The training of this network it's done by Back-propagation of the error and therefore optimizing weights.

2.3.1 ViT - The Image Captioning model

The ViT model[3] is a particular transformer since it doesn't use a decoder stack. This model is composed with a encoder stack with a MLP head, which allow us to exploit this to pair up with a chosen decoder, in our case will be paired with a NLP decoder to be used for image-to-text transcription and to generate the meta data of the herbaria for cataloging reasons. Introducing the Transformer networks for image recognition, starting with the first implementations that follow the traditional Transformer, and the network the Vision Transformer (ViT) [3]. Although this network is for image captioning and not for object recognition this can be useful to do paragraph recognition. Although ViT doesn't use encoder-decoder methodologies, it just borrows the self-attention layers of the encoder. For this model to be compatible with the token like input, we need to reshape the image. This reshaping process is represented by $x \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $x_{patches} \in \mathbb{R}^{N \times (P \times P \times C)}$ where H is height and W the weight of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = \frac{H \times W}{P^2}$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. This 2D representation of our image goes through a Linear Projection to be flattened to a D size vector $X_p \in \mathbb{R}^{N \times (P^2 \times C) \times D}$.

This model also borrows the concept introduced on BERT's transformer networks [7]. The use of a learnable embedding in form of a CLASS Token embedding$z_{00} = x_{class}$, which consists on a learnable embedding, that gathers information from all the patches when attending to Multi-head Self Attention (MSA) layers. The classification head simply implements a MLP with one hidden layer. The classification head only uses this hidden output from this CLASS token. The authors decided to use the relative position of the patches for the embedding, to encode the spatial information as instead of their absolute position. They achieve this by using $1-$dimensional for the Relative Attention.

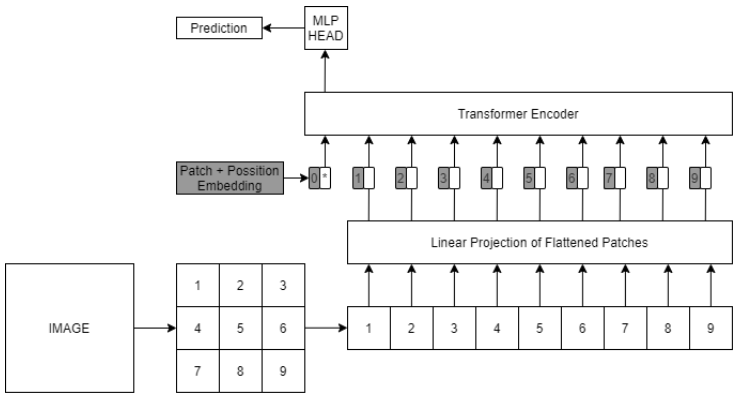


Figure 2.16: ViT model Architecture

2.3.2 GPT/GPT2 -The general purpose NLP Transformers

The OpenAI GPT model was proposed in Improving Language Understanding by Generative Pre-Training [24]. This model was developed with the goal of creating a general purpose language modeling model. This model was trained with a methodology of a Unsupervised learning method, this allowed the model to look for patterns instead of labels matching that is usually present on Supervised learning. The concept of unsupervised learning applied was proposed by Andrew M. Dai [25] on the paper Semi-supervised Sequence Learning that was proposed for RNN and LSTM RNN networks and applied to transformers in this case. This algorithm consists of abstracting from the words to labels by implementing a sequence to sequence learning patterns for the relation of tokens to other tokens. The GPT2 [6] is an improvement over the previous model with the same goal, it has some changes and methodology of training that addresses the ability of generalization instead of memorization moving towards a better predictor and natural language generation. To achieve this they use a combination of pre-training and supervised fine tuning. The GPT2 mostly follows the general language modeling architecture but each normalizing layer was moved to the input of each sub-block, similar to a pre-activation residual network and an additional layer normalization was added after the final self-attention block.

2.3.3 BEiT and DEiT - A different approach on Vision Transformer

BEiT and DEiT will be reference as possible alternatives to VIT encoders. The model DEiT [5] and BEiT [4] visual transformers models will just be referenced since our implementation on huggingface allows to switch encoder models easily. The Model BEiT Bidirectional Encoder representation from Image Transformers, was developed by Microsoft [4] that experiments with a concept called masked image modeling (MIM) while keeping the backbone same as ViT. They also changed the way feature extraction works which is been based on discrete variational autoencoder (dVAE). For the task of image classification tasks the DEiT: Data-Efficient Image Transformer is a type of Vision Transformer. The model is trained utilizing a transformer-specific teacher-student technique. It is dependent on a special token called distillation token, which ensures that the learner only learns from the teacher through attention. These are the principal features of each models. We tested them since these didn't meet our acceptance we will only refer them to discuss results.

2.4 Related Work

In this section, we'll go over some related papers of solution that could be used solve our problem. Previous attempts of the task to cataloging Herbarium collections, usually focus on either one of the two elements presented on the herbarium Sheets: Some papers develop a model that aims to only identify specimen and the family using only the specimen characteristics and another that only focus on cataloging using OCR techniques to extract information from the text label usually present.

Figure 2.17: Elements usually present on the herbaria sheets

One set of papers focused on reading directly the information from the text labels using Optical Character Recognition (OCR) plus a Name entity recognition (NER) for field identification and the other will look directly at the specimen present in order to identify it by characteristics. We will focus on the paper published in 2020 "Towards a scientific workflow featuring Natural Language Processing for the digitisation of natural history collections" [26] that uses one of such methods. This paper model is design to function on a wider specimens collections including animals, insects, herbarium and other scientific collections, in order to be this generic the model is going to be retrieving information at semi standardized text labels across the multiple types of collections, hence the abilities of this model to function in a wider datasets for multiple types of collections. For comparison our solution will allow our model to access the full information available included the herbarium specimen as well the text labels, for possible achieving

better results but also specializing our model to only work with herbarium collections.

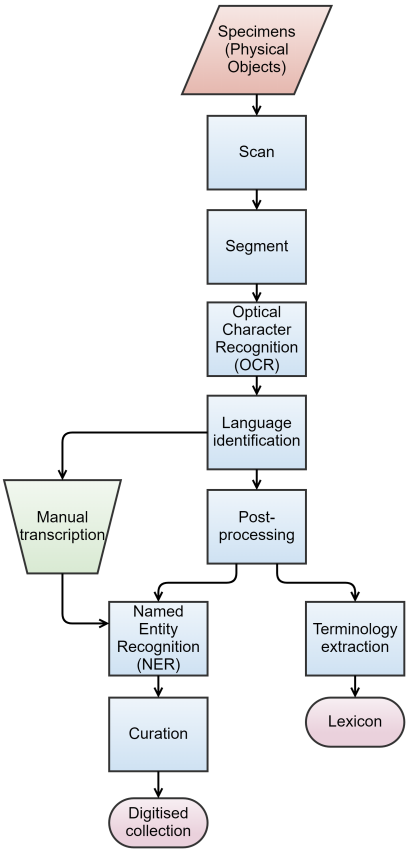


Figure 2.18: Model architecture proposed in the paper Towards a scientific workflow featuring Natural Language Processing for the digitisation of natural history collections" [26]

This paper proposes a process that can be fully automated and partially automated depending on the desired speed and accuracy of the data extraction. This decision was made due to a observation during research, since this model is reliant in OCR and other NLP based processes the variance present on the written data with possible combinations of hand written, machine typing, languages, state of preservation, coloration of the sheet and so on... it can be detrimental to the performance of the model. This challenge of data variance is also present on our dataset so we will draw our own conclusions while keeping in mind the conclusions drawn by the author of this paper.

The process starts with a scan and segmentation; this is a process where the original paper becomes digitized for segmentation. The process of segmenting the digital image into the small elements where the OCR will generate metadata the data the model is looking is described as 8 elements: look for a Title of the Organisation that owns the specimen; A Barcode of the specimen's machine readable identifier, Scientific or common name of the species; The person who identified the specimen and the date of identification; The geographical location where the specimen was collected; Habitat and altitude; Additional notes written by the collector, Collector name, The name of the person who collected the specimen, the identifier that they used to record and manage specimens, and the date that the specimen was collected.

After the data been transcribed, this is where the process can vary depending on the method selected between the automatic or semi-automatic. The semi-automatic will allow the user to manually select the language to the NER model identify correctly the fields, while the automatic will allow the model to auto select the language and possibly make the right decision.

Chapter 3

Implementation

3.1 The Dataset

Before we explain the implementation of our model, we have to explain our process for data preparation. We will go over our dataset statistics following our thoughts about the problematic to solve. We will go in detail over our Pre-Processing module, explaining some processes we apply to our data like: Filtration; Transformation; And image generation with the purpose of increasing the size of our usable dataset without manually annotate more images.

3.1.1 Datasets - dataset reviewing and manual reviewing our dataset

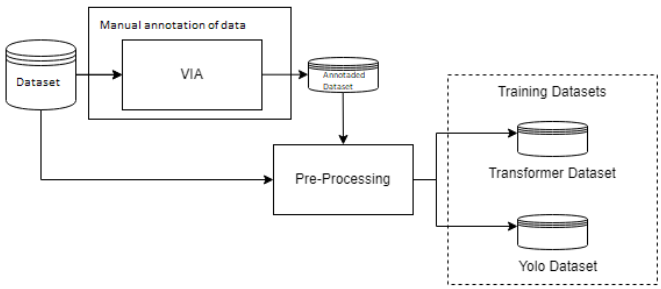


Figure 3.1: OverSimplified Pre Processing logic

The dataset used for this thesis is an already established and compiled collection with over 1,800 herbarium specimens [27] for the purpose of cataloging. This collection has entries that date as late as 1970, so the state of preservation of the sheets and the legibility could prove a challenge for our model. Besides this challenge, this collection has present multiple languages that the text labels could be written, with more than eleven possible languages: English, French, Latin, Estonian, German, Dutch, Portuguese, Spanish, Swedish, Russian, Finnish, Italian and Other unidentified languages, with all these languages provide us with a good distribution around the globe and variety of the data presented on the labels. Reference Appendices [B] to graphical distributions of the dataset. This dataset provides a JSON object

that is matched to each image, that contain some hand filled information relevant, in relation of the specimen name and family, date of capture, location of capture, and many others.

With all this complexity and scattered information provided by multiple institutions, it is normal that the data could have some mistakes and or variances on the time of filling this information, hence it is essential that our model learns from data that is correct and partially normalized to a standard. Our first step to achieve standardisation was a tool that serves to manually annotate our images from this dataset. The VGG [28] - Image Annotator (VIA) [29], is a tool that allows attributes to be associated with the image file to be annotated and/or each annotation box, in our case the annotations made are to identify the regions of each text label presented on the image. The Attributes we decided to use were:

- "file:"
    - "Preservation_Quality"
        - Bad or Good
- "region:"
    - "Label"
        - Handwritten, Artificial or Both
    - "Label_Quality"
        - Good, Bad or Average
    - "Label_Complete"
        - Yes or No
    - "Name/Family"
        - Yes or No
    - "Location"
        - Yes or No
    - "Date"
        - Yes or No

As shown above the the tree of attributes used with the possible values of each field.

All this fields where created in mind our models training and the use-fullness for the creation of the training dataset. Starting with the single attribute associated with the image file is the state of preservation of the entire sheet, specimen included, and making it fall into two categories: Good(state) or Bad(state) of preservation. The labels been more complex in nature since is where most information

resides and it will be extracted there was a need to associate more attributes in case of finer we need to filter some cases. This lead to the creation the 6 attributes presented below:

- ”Label” - associated with the type of writing present on the label.
- ”Label_Quality” - associated with the human legibility.(this can be influenced by the type of letter also the preservation of the label itself.)
- ”Label_Complete” -relative to the label if is complete and matching to the JSON attributes (We consider complete when the data of the Name and family of the specimen, location of capture, and date since this are the field we aim to extract)(when miss matched we consider not present)
- ”Name/Family” - associated with the partial data of name and family of the specimen is present on the label.(when miss matched we consider not present)
- ”Location” - associated with the partial data Location of the specimen is present on the label.(when miss matched we consider not present)
- ”Date” - associated with the partial data of date is present on the label.(when miss matched we consider not present)

The creation of overlapped fields is not to rule out the cases that the sheet could have multiple labels that complement each other information we only remove the cases that information is not present at all or fully miss matched from the JSON file provided.

3.1.2 Pre Processing module - Automatization of data manipulation and creation

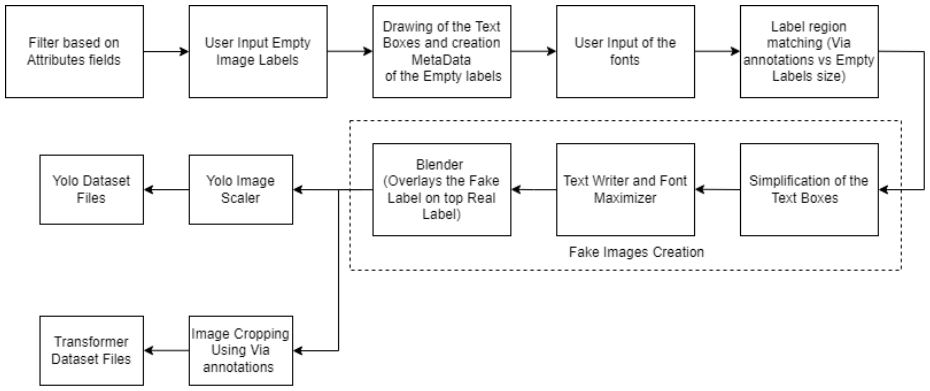


Figure 3.2: Pre-Processing Module Functions.

The Pre-Processing module has multiple available functions that work in conjunction to manipulate our data in order to reach some data standardization that will be used to train our model. We will explain the logic behind the processes and the utilities implemented as we will make use of them. The way we implemented this module it requires as input four Paths: The first path is a pointer to the VIA file resulting from the manually reviewed portion of the process from the previous task; The second path point to the folder where the dataset pictures are stored; The third the path for the folder where the JSON images attributes are stored; Lastly is the path where the dataset creation will occur.

Taken this inputs, the module first step is to filter the data based on the JSON attributes looking for missing or miss-formatted fields and excluding them from our usable examples. As example of miss-formatted fields is the Date field that can have a lot of variance as in multiple representations due to multiple standards around the globe. In this case we sampled based on the success of a date parser function that normalizes and return a formatted date when successful (dd-mm-yyyy).

The second step begins the process of computer generated images.This step starts by the user to provide, images files of empty herbarium labels. Then the user will be prompt to draw with rectangles shapes on top of each provided file, that serve to indicate the useful space in the image of the multiple text fields: Specimen, Family, Specimen/Family, Date, Location, Location_detailed, Random Name, ”Random int, RandomCharInt and Other. Some this fields are noise in order the model to ignore like in real cases there can be presented with more information that we don’t aim to extract. Theses rectangles eventually simplified and turned into line like text-box where our computer generated text will be placed in. This process will generate a Json file per Empty Image Label, that will be placed onto the same folder as the labels images. This file will contain the information of the raw boxes associated per attributes with some useful data of our label, like height and width.

Like the last step the third step will require the user to place some font files in a specific path. The pre-process module will make use of the provided fonts to write the computer generated text on to the empty labels. Our selection of Fonts follow what we think what should be the optimal cases, these Font should be similar to handwriting and Typewriters since its what is most common on the dataset labels. On the time of writing the module will be choosing at random the fonts per label.

Before starting generating computer generated labels or ”fake” labels, we decided to do a resolution matching between the Via annotation of the labels sizes with the empty labels resolution ratio, this is done to cause the minimal deformation on the labels on the act of overlaying the fake labels on top of the original labels.

Finally with all the information generated and compiled we proceed to the creation of our fake labels. The first process is based on the previous step that matched Via label regions with the empty image label. We start by reading the empty label image JSON atributes and simplify the raw rectangles boxes that the user has previously drawn, converting them into text-box like. For this to happen we will just explain briefly the code used. The code used is heavily recursive that test multiple relations box to box in order to combine and or associate them based on the type and relative position to one another. The output can be see on the 3.3.

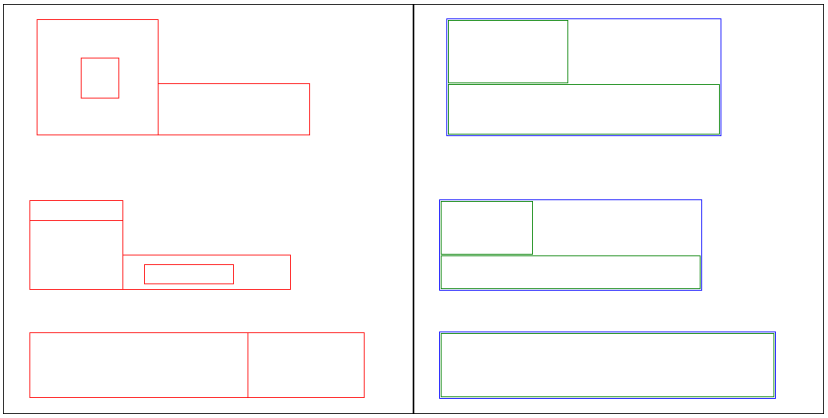


Figure 3.3: Boxes Simplification provided with our demo script

The image on the left is the user input raw boxes while the right is the simplified output. The Blue Boxes represent where the text will be split, thus multiple blue boxes means the text will be repeated and split differently according to the useful space (the green boxes) on each.

After this process, we will run an algorithm that maximizes the text font size by optimizing the text splits per usable space, and allowing to further subdivide the boxes vertically to create extra text-lines when necessary (This algorithm does a brute force approach and can be improved on). The output can be seen 3.4 applied to three sentences: ”The quick brown fox jumps over the lazy dog”; ”The quick brown fox jumps over the lazy dogThe quick brown fox jumps over the lazy dog”; And ”The quick brown fox jumps over the lazy dogThe quick brown fox jumps over the lazy dogThe quick brown fox jumps over the lazy dog”.

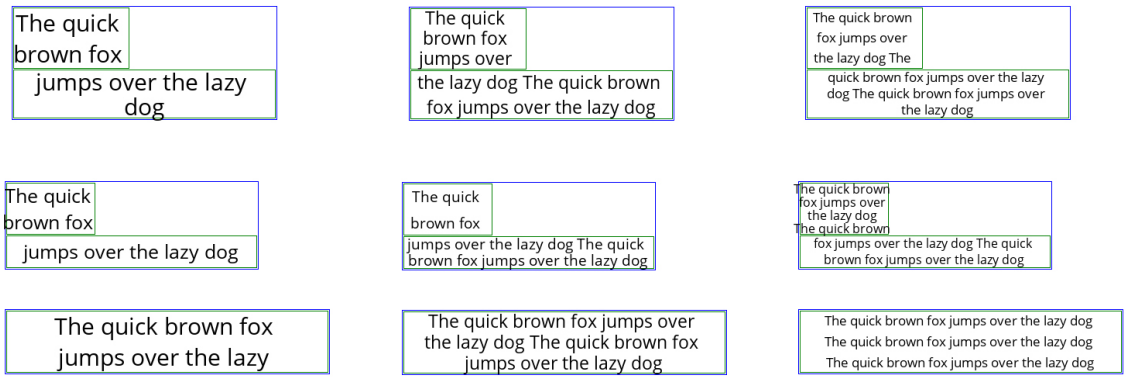


Figure 3.4: Text Algorithm applied to the Boxes Simplification example

As for methodologies for blending, we tested multiple combinations of techniques to keep the outcome as ”real looking” as possible, from the methodologies we tested one combination stood out for to be the most convincing to be closest to the originals. The methodologies we tested while developing, we decided to keep all in code allowing the user choose from multiple blending techniques. To explain the method we did the blending we will go over each steps one by one: firstly we start generating a flat colour patch to be blended to cover the original label with the most common colour of the original label; Then we colour transfer from the original label to the ”fake” generated label; Lastly we Laplacian Pyramids blend the images. The fifth step is a function that splits the text into lines of text in order to maximize the font size per usable space. The sixth step blends the complete fake label onto the original using Laplacian Pyramid Blend technique with colour matching. After the image multiplication, we will scale down and crop and save each label to a file in order to generate the inputs for our models. The YOLOv4 model requires a square image of resolution that is part of the series of (320 + 96 * N) x (320 + 96 * N) n ∈Z to train and the transformer requires the pre-cropped labels as the input. The last two steps are just creating the files required to each training module.

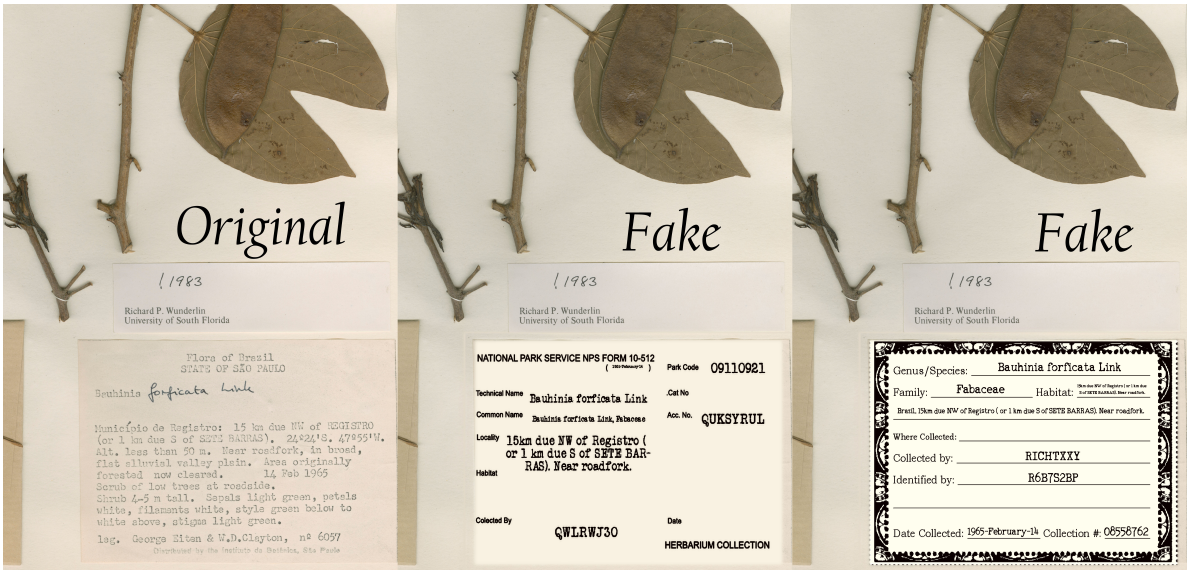


Figure 3.5: Results of generated fake labels blend onto a real image

3.1.3 Dataset Structure for Transformer module

In order to train our model, we had to set a structure for the file that is used on the supervised learning, this file contains the images path references and the expected prediction text, labels. We use a single structure for our dataset files that is used over our modules. This datasets is required for our supervised training and optional on inference module. When the dataset file is not passed as parameter for the inference module the metrics will not be able to be calculated. The Transformer training dataset Json dictionary file is structured as the following:

- "YData:"
  - "Number":[int] | eg. [3, ...]
  - "Name/Family":[[str]] | eg. [["Abelia R.Br.", "Caprifoliaceae"], ...]
  - "Location":[[str]] | eg. [["Danzha Cun. In the vicinity of Zhaobitan forest farm, ca. 26.5 direct km NNW of Houqiao (Guyong).", "China"], ...]
  - "Date":[[str]] | eg. [["29-5-2006"], ...]
- "XPath:"
  - "FullImage":[str] | eg. ["./Transformer_DataSet/Images_Filtered/Image_3.jpg, ...]
  - "Crop": [[str]] | eg. [["./Transformer_DataSet/Images_Filtered_Cropped/Image_3_crop_1.jpg","./Transformer_DataSet/Images_Filtered_Cropped/Image_3_crop_2.jpg"], ...]

Every Field on the YData is a reference to the image metadata, these are the labels we want the model to predict. While the XPath fields are referencing the model required inputs, the image text labels and full image.

3.2 Transformer model - The Hugging Face Implementation

This section will explain our implementation path and the limitations, we had to overcome for our implementation. We will skip the YOLOv4 implementation since we only use it for data extraction and was not developed by us. To keep the focus on our development. We will directly go in a depth explanation about our transformer architecture design and implementation in the hugging face library.

Our model was developed with some concepts and ideas that were yet to be fully researched and tested, in this case study we have implemented a multi encoder to multi decoder architectural modeling in expectation that each transformer stack would function as a specialized worker in our process.

Starting by our custom encoder modeling 3.6 this torch module is comprised by two encoder stacks of the same model. Each encoder transformer stack is tasked to encode a different type of images. In expecting that would allow the attention mechanism to focus on the relevant data of each image type presented. One encoder stack called the encoder_image and is responsible of encoding the full sheet image where the specimen is presented while the second encoder called encoder_label is responsible to encoder of the image labels that are identified by our YOLOv4.

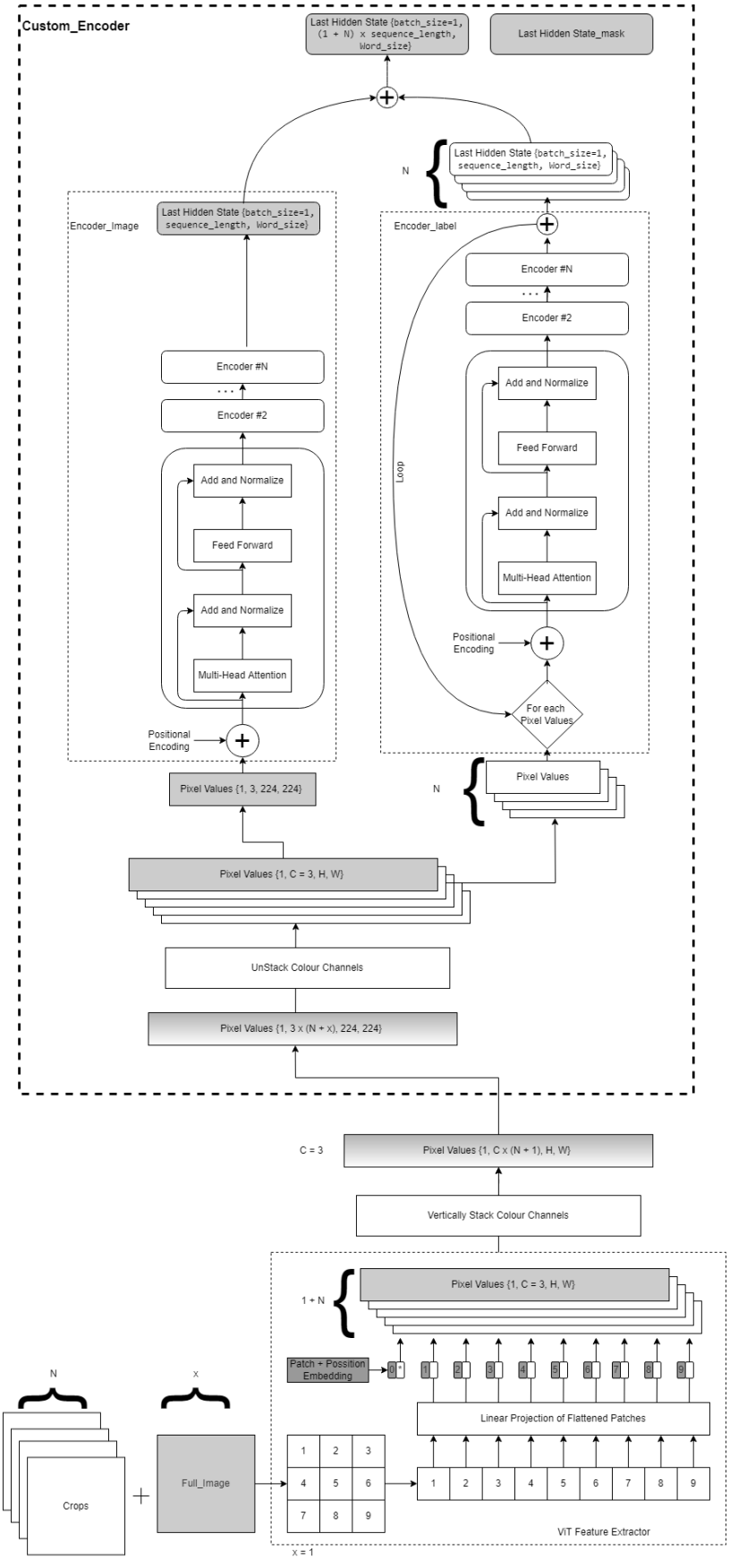


Figure 3.6: Encoder Modeling using ViT Encoders as example

With the same present logic that the attentions mechanism can be exploited to do different tasks. Our implementation of the custom decoder module 4.2 is composed by three decoder stacks of the same type, since each attention mechanism present on each decoder stack is required and will work to identifying and extract different data, we hope that this model architecture will prove an hypothesis that we can exploit attention mechanism to function as a NER. Each of the three decoders stacks in our custom decoder aims to extract different type of data: one is tasked to identifying the Specimen name and/or family; Another the Location of capture; Lastly one decoder for the Date of capture. While the data for the decoder that extract Location and Date can only be found on text on the multiple labels per sheet the Specimen/Family is expected not only to work as a tOCR like the other, to extract data from the written text label but to attend to the full sheet in hope of helping on the identification of the specimen.

If this works this multiple decoders architecture, this could lead on creating a opportunity on other fields to exploit attention mechanism to perform other tasks.

The module we modified and reverse engineered was the hugging face modeling_vision_encoder_decoder.py [1], our modifications were made to be addictive by not compromising any features in place and or the usability of our custom module. An advantage of decision of implementing this model architecture in the hugging-face is that by modifying just modeling block instead of implementing the model in PyTorch, we could test multiple vision encoder stack model in combination of any transformer stack as decoder, that are currently available in the hugging face without requiring any code modification. This will give us the possibility to test multiple model combinations all ready available on hugging face and hugging face community library and possibly allow more user to experiment with this modeling block.

The first challenge, we were required to surpass is the way we input our images to the model. The hugging face has a protocol that is required to follow since the model need information for the beam search inference. In Order to understand what the input tensor represents we recurred to the developing tools pairing with inspecting the output from ViT_feature_extraction [1].We found what each dimension from the input tensor has four dimensions comprised of:

$$\text{Pixel\_Values} = (Batch, Number\_of\_Channels, Width, Height) \tag{3.1}$$

With this challenge in mind we started looking for a solution. Our solution came by reversing engineering the ViT modeling module. This decision not only allowed us to achieve our solution but also a better understanding of what is required to make our module to function. Later after some tests, we found that the implementation of vision encoder stacks allows the input tensor to have unlimited Number of Colour Channels. This find was the optimal solution, since any other dimensions are checked an utilized in some way to the model to function. By keeping intact the batch size intact and the input sizes for the encoder model this means our first challenge was surpassed.

We still in need to define a protocol in order to stack and un-stack the Pixel_Values using our findings. This protocol is required to keep the identification of the two different image types: the single full image of the herbaria sheet; And the multiple labels crops images from our YOLOv4 model. The first step on our protocol was to normalize all the the input images to three colour channels. Then we starting to

define and order to vertically stack our images colour channels. By establishing an order to follow when stacking this would allow to keep the information of each image type. The order establishes was based on logic since there is only a single full image sheet we decided that this was the first image to stack leaving the rest of the colour channels are for the crop label images.

In sum we reserve the first three colour channels for the Full sheet image and the rest N multiple of three colour channels are for the labels crop images. Since our YOLOv4 can provide more than one label crops image we were required to implemented a loop inside the forward function of our Custom_Encoder a for each crop Pixel_Values to be encoded on the encoder_label. As seen 3.6. The only check we were required to implement was this model needs a minimum six colour channels to be passed as the input. This been: one image for each encoder stack. Resulting in a vector:

$$\begin{aligned} Pixel\_Values_{(1sheet\ \&\ 1crop)} &= (Batch, 2*3, Width, Height) \\ Pixel\_Values_{(1sheet\ \&\ 2crop)} &= (Batch, 3*3, Width, Height) \\ Pixel\_Values_{(1sheet\ \&\ Ncrop)} &= (Batch, (1+N)*3, Width, Height) \end{aligned} \tag{3.2}$$

After the image embedding have been processed by each encoders, we simply concatenate all the words generated in order the same order as the input. Since the numbers of words may not match from run to run due to the number labels present from sheet to sheet can vary from image. We utilize padding with zeros to the max batch size N_Words for this tensor in order to indicate the decoder only the use full values we provide a mask that is natively supported on this models.

$$Encoder\_hidden\_States\_mask = (Batch, N\_Words, hidden\_size) \tag{3.3}$$

We have to overcome the problem of input sizes variation per prediction, due to the variation of the number of label crops per example may vary. Our solution is to add a parameter called encoder_batch_lengths that specifies the number of images per batch, this value is only checked when the batch size of the Pixel_Values is higher than one. This decision was to solve this problem with this variable was that when creating the Encoder_hidden_States_mask we could do value count until we found zeros from padding but this is memory expensive and time consuming we decided to simply pass this parameter as a input. provide an example of the Collate function for the dataset [30] library by huggingface for Dataloader that generates this value.

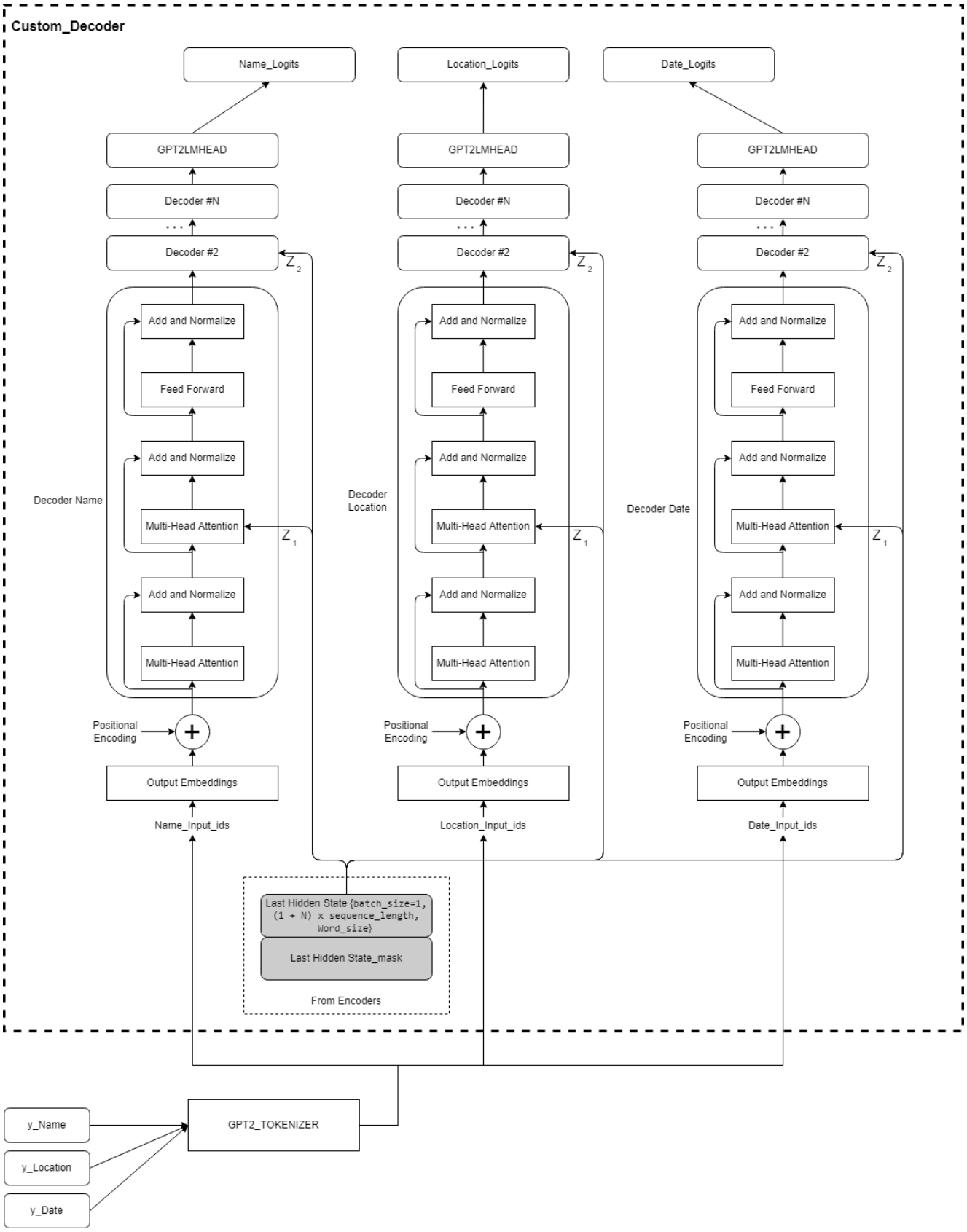


Figure 3.7: Decoder Modeling using GPT2 decoders with LM head as example

Another problem that we needed to surpass found while testing our model using the generate function. The generate method on the hugging face transformers is the methods that allows the model inference. The problem encountered is that the generate function is not prepared to handle multiple decoder stacks at once. Since this function is a core function for all models we decided to not modify it, due to complexity and importance, any mistake made could be detrimental outcome of our model. So our solution needs to circumvent this limitation any other way. The solution was to declare a single decoder stack like the default model and by modifying the checkpoint for the purpose of extracting the weight maps for each decoder stack on our custom decoder, we could load them individually and run multiple inferences with the same input and different weights map.The inputs used for the decoder can be resumed as the sum of the encoder generated 'words'. This solution is computer costly since this runs inference N times equal to N the decoder stacks declared.

With this in mind we implemented a Training flag that is used during initialization when this flag is set to true this initiates our custom multiple decoder stack when is false this initiates a single decoder model. For this to work transparent for the user we implemented a class called generator_adaptor. Upon initialization this class requires only the checkpoint path when its called, this simply initialize the model using the configuration file found on the checkpoint folder created by our training module and saves the path on a local variable where the checkpoint is location. By running the inference method called generate to keep the signature the same the user is only prompted to pass the images while the module takes care of splitting the checkpoint; Loading; And running inference. This modules simply outputs the result of the three inference runs from each the decoder weights.

3.2.1 Training module for the Transformer model

Like referenced before our training module is fully implemented in using PyTorch, this didn't oppose any problems since the Huggingface models are in essence PyTorch modules. We will firstly start by explain some features implemented that are relevant for out model training. The main feature implemented is that this module allows the user to load two distinct datasets: a dataset and a fakedataset. The dataset is used as normally in almost every epochs, while the fakedataset will be used as noise generation when training this will only used on some specific epochs. To define this specific epochs there is a value named noise_epochs that defines the periodicity of the use of the fakedataset. This module also allows the user to define parameter that are more common when training model: batch size, number of epochs, define a optimizer, define a optimizer scheduler, checkpoint save location and defining a periodicity of the savings.

3.2.2 Inference module implementation

This subsection will explain the logic of the Inference module. Firstly we need to explain what are the inputs required to make this model work and then we will present the logic of the data inside of this module. This module to work requires some information:

- "image_path_or_image_dir": Path to the image or images folder path. (required)
- "YOLOWeightFilePath": Path of the YOLOv4 model checkpoint. (required)
- "TransformerCheckpointFilePath": Path of the Transformer model checkpoint. (required)
- "image_to_label_file_path": Path of the dataset file that contains the labels of the images loaded.(optional: when not present no metrics will be initiated and evaluated)

With the knowledge of what information this module as access to, we will explain the logic and the data transformations required to this non-end-to-end model to function. to explain we will divide into steps and explaining the logic behind. Since our first model in the process is the YOLOv4 model for

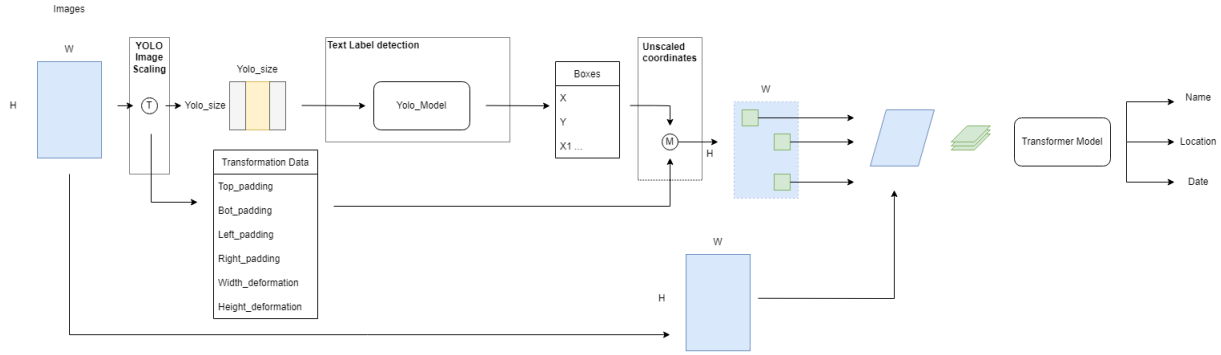


Figure 3.8: Model Inference Logic Simplified

extracting the text labels. our first step requires that the original image needs to transform into a a square image with the resolution series of (320 + 96 * N) x (320 + 96 * N) n ∈Z. We recommend setting the size the same as training and the crop parameter the same as the pre-processing YOLOv4 dataset creation parameters. By saving relevant data of each transformation this will allow map the inference bounding boxes of the smaller image inputed to YOLOv4 to the original full resolution image to obtain the maximum resolution on the crops. The last step is then we simply just compile all the images and input to our Transformer.

Chapter 4

Training and Results

As referred the YOLOv4 model was trained using a pretrained checkpoint on the ms-coco dataset, using the developed training module pre-configured from the package suite. This model was trained until three hundred epoch, and is still a bit far to be one hundred percent accurate but this module is pretty accurate over hall for this task. The task of this model is to identifying the text labels present in the sheets, this simple task can be proven to be arduous. Since these sheets came from multiple institution around the globe, with different standards, so each label is likely to have a lot of visual variation from one another proving this challenge can be a arduous for the YOLOv4 Model.

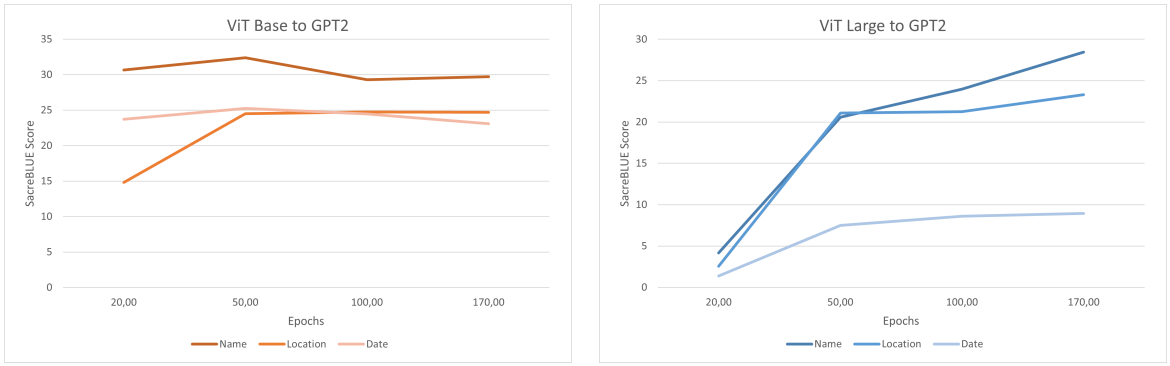
One of the obstacles that we faced when evaluating the performance of our Transformer model is that since this architecture is a non end to end model. The error could be generated by another "link the chain". This is normal and is derivative from the nature of the non end to end architectures, optimally we had two dataset ready for individual validation but since we didn't had enough time to annotate more 200 images to generate more examples we will be testing the full capability of both model working in sequence. Since the transformer model is implemented using hugging face we had to test using other encoder and decoder combinations like reference along this work. For that tested with multiple combinations. This test were done using the available checkpoint from the huggingface pretrained for each of the following models; As encoders we tested model like ViT, Deit, and Beit; The decoders tested were Roberta, and GPT2. All this model were trained until the ten thousand iterations, on a data set with a little over eight hundred entries this is around thirteen/fourteen epochs and checked the results. In multiple combinations the error tended to infinite or the text generated was close to gibberish leading us to quit from most combination. One of the tested combination stood out by giving semi-usable outputs since the fifth epoch of training making us choose clearly and test this combination of the model ViT with GPT2 model. With this decision made we performed tests using the multiple sizes available of these models. So we will start by presenting the early tests with a comparison between the ViT Large with GPT2 versus the ViT Base with GPT2 while sharing our opinion of the results. Lastly we will pick the best performer from the early test and try to obtain our final results of training with the concluding thoughts.

4.1 Training Module - Transformer model

Our Training module is fully implemented using PyTorch, this is a very well known library with many training options and optimization made available. We decided to train our transformer mostly using our fake labels since these are a well behave examples that correspond to a direct transcript from the prediction labels. Our training module allow to use up to two dataset for training one of these dataset is used frequently while another is only set to be used during some epochs, we will use this to induce noise while training for possible achieving better results. This list of epochs is defined by variable num_noise_epochs by default we use this periodically like every five epochs. For the optimizer we use AdamW that is a improvement on original Adam optimizer that allow the model to be good at generalizing tasks, there is a general perception by the community and has been recorded that SGD optimizer is slower but it's a good generalizing optimizer and Adam allows for faster training but is worst at generalization, hence the creation of AdamW as a mid term algorithm.

4.2 Tests - ViT large vs ViT base

For this model performance comparison we be analysing and sharing some thoughts by analysing the metric BLEU using sacreBLEU [31] implementation. The BLEU metric and its implementation is a model used to obtain a score based on perceptibly and similarity between the predicament string and a set of acceptable References Labels or Reference Label, optimally we want this score closest to one hundred.



(a) ViT Base to GPT2 SacreBLUE Score Results

(b) ViT Large to GPT2 SacreBLEU Score Results

Figure 4.1: Early Tests

Our model test achieved a maximum BLEU score around 30% to 40% depending on the decoder. We suspect that our model is not complex enough for this problem and or using wrong training parameters or methodology. Usually for this metric is only considered a good score around 70% score is enough for the human perception to understand what is written and to be considered similar to the original.So clear our early results were not good enough to solve this problem.

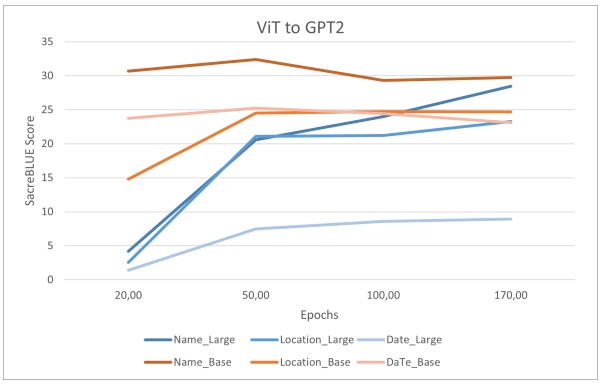


Figure 4.2: ViT to GPT2 SacreBLUE Score Results

When overlapping the results we obtain we can see a clear winner but the results were unexpected. Even though there is and advantage for the ViT large in the input resolution, from the original 224x224 of the base model to 384x384 on the large model. The ViT base manage to achieve better results overall. We suspect that this decrease of score came from the only difference between when modeling this models. When choosing ViT large versus the Base there is a increases the hidden_size tensors thus requiring a extra linear layer between the encoder and decoder to make them compatible. This layer is only initialized when there is a miss match on the sizes between the encoder and the decoder, leading us to believe this could be our culprit for the results seen. And since this extra layer is not require and or present when using the ViT Base since this hidden_sizes already is match-up with the GPT2 this could lead less errors. with this in mind we will realize our final tests only using the ViT-base with GPT2 since these are the better performer.

4.3 Results - ViT base to GPT2

For this analysis we will present and obtain results using the BLEU metric using sacreBLEU [31] with a metric that is absolute word error rate - WER [32] [33]. The WER metric is a hard metric this is calculated using a simple formula that can be expressed as:

$$WER = \frac{(Correct\_words + Inserted\_words + Deleted\_words)}{Lenght\_original\_phrase} \tag{4.1}$$

This metric correlates with the difference between two phrases and has in count: the extra word inserted, word deletion, and the correct words in relation to the original phrase size. This optimally should be closest to zero. We will present results for two distinct datasets: one that is similiar to our training dataset that is a standarized by out data process; Another obtained using a random data that is not part of our training dataset thus been not standarized in any way and may contain some errors.

4.3.1 Results - Similiar Dataset

The results presented are from another dataset generated based on the same as the pre-processing as the training dataset. We will present this result as semi-valid due to the randomness in generating false labels. These results present a very different reality than the previous tests. These results with BLEU similarity score above 80% means that this model is very good at extracting information. Which means one of two things, that this model may be overfitted to our traino data set and the layout of the fake labels or simply that the complexity of the real world data is too random for our model to be able to work, which we will present next.

Similar Data of the Training dataset:

BLEU:

Decoder Name: 91,71

Decoder Location: 81,10

Decoder Date: 100

WER:

Decoder Name: 0,074

Decoder Location: 0,18

Decoder Date:0,0

YOLO:

Failed to identify Labels on 2 images of 128 images.

4.3.2 Results - Real World Data

The data used in this metric was a set of images that were filtered only by our pre-processing module, but were not included in our training dataset because they were not manually annotated using VIA. Thus, this data was not fully standardized and may have some errors. The results are not satisfactory, proving that our model is not suitable for extracting information from all cases and only from semi-standardized data.

Real World Data:

BLEU:

Decoder Name: 3,86

Decoder Location: 2,43

Decoder Date:6,29

WER:

Decoder Name: 1,06

Decoder Location: 1,18

Decoder Date:0,99

YOLO:

Failed to identify Labels on 17 images of 1078 images.

This model results can be very good in case of standardized data but its no suitable for real world cases. While our results were not satisfactory in relation of usability on the field, we were time constrained for tweaking and fine tuning our parameters of training. Based on the result obtained by the unpublished paper for tOCR by Microsoft [2] proves that Deit and Beit paired with RoBERTa [8] can obtain better result than ViT paired with RoBERTa for the solely task of OCR, we also believe these models could also have great results for our task, given the right training.

Chapter 5

Conclusions

The results of this model are inconclusive but it can get good results when the data is standardized but it is not good enough to generalize to random real world data. Further testing would have to involve increasing the number of cases in the training dataset, by hand annotating more images along with adding more empty labels and fonts to increase the training cases found. Transformers are known in general not to be very good at generalization and very dependent on how they are trained and on the training data used. Although the results were subpar to other attempts this type of multiple encoder to multiple decoder architecture still has a lot of potential. We keep that the original thought that this problem is very complex in nature to any single model to handle, a separation by language or other is need. As a final thought is that multiple decoders can be exploited for the attention to extract data or achieve different goals. For our use case we exploited the attention mechanism to function as a NER with some grade of success, confirming our hypotheses.

5.1 Achievements

Despite our results we have implemented and tested a model design that is yet to be studied in depth (multiple encoder to multiple decoders). There are multiple fields where this model design could have great success : one in example is in the field of CAD when modeling 3d objects across multiple 2d views in order to generate a single 3d model and also the reverse could be possible with deconstructions of a 3d object into 2d views. Another example can be in the field of multimedia communication in case of video encoding or decoding that could use motion vector and image frames to generate more efficient encoders/decoders and also in the field of NLP with multi translations output with many possible use cases. Since our design has been available on GitHub and Documented hopefully sheds some light and creates curiosity on possible more implementation of this type of models. Especially implementing such a well known library with great support and continuous improvements provided by the huggingface team and a striving AI community will allow more people to be exposed to these types of models designs and testing on more use cases.

5.2 Future Work

As an improvement to our transformer model we could limit the data available for the Location and Date decoders to avoid errors since the information is really only available on the text labels from the label encoder generated words. Another possible improvement is to add the support to the complex module for multiple decoder attention mechanism to the hugging face inference function, generate() natively. While our solution to bypass the generate() limitation works, this still requires loading the model N times the decoders to run inference over the same inputs, this process could use parallelism computing to greatly improve its speed. Also the possibility of our region segmentation model to look for the bar code since its represents directly the specimen name encoded.

Bibliography

[1] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[2] M. Li, T. Lv, L. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li, and F. Wei. Trocr: Transformer-based optical character recognition with pre-trained models. September 2021. URL https://www.microsoft.com/en-us/research/publication/trocr-transformer-based-optical-character-recognition-with-pre-trained-models/.

[3] 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, 2020.

[4] H. Bao, L. Dong, and F. Wei. Beit: Bert pre-training of image transformers. arXiv preprint arXiv:2106.08254, 2021.

[5] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In International Conference on Machine Learning, pages 10347–10357. PMLR, 2021.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[8] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

[9] R. S. licensed under CC-BY-SA. paperswithcode state of the art, 2018. URL https://paperswithcode.com/sota.

[10] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2014. URL http://arxiv.org/abs/1405.0312. cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list.

[13] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[14] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

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

[16] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. Lecture Notes in Computer Science, page 346–361, 2014. ISSN 1611-3349. doi: 10.1007/978-3-319-10578-9_23. URL http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[17] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2017.

[18] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation, 2018.

[19] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon. Cbam: Convolutional block attention module, 2018.

[20] @Tianxiaomo. pytorch-yolov4. https://github.com/Tianxiaomo/pytorch-YOLOv4, 2020.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

[23] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

[24] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. OpenAI blog, 2017.

[25] A. M. Dai and Q. V. Le. Semi-supervised sequence learning, 2015. URL https://arxiv.org/abs/1511.01432.

[26] D. Owen, L. Livermore, Q. Groom, A. Hardisty, T. Leegwater, M. van Walsum, N. Wijkamp, and I. Spasić. Towards a scientific workflow featuring natural language processing for the digitisation of natural history collections. Research Ideas and Outcomes, 6:e55789, 2020.

[27] M. Dillen, Q. Groom, S. Chagnoux, A. Güntsch, A. Hardisty, E. Haston, L. Livermore, V. Runnel, L. Schulman, L. Willemse, Z. Wu, and S. Phillips. A benchmark dataset of herbarium specimen images with label data. Biodiversity Data Journal, 7:e31817, 2019. ISSN 1314-2836. doi: 10.3897/BDJ.7.e31817. URL https://doi.org/10.3897/BDJ.7.e31817.

[28] A. Dutta and A. Zisserman. The VIA annotation software for images, audio and video. In Proceedings of the 27th ACM International Conference on Multimedia, MM '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi: 10.1145/3343031.3350535. URL https://doi.org/10.1145/3343031.3350535.

[29] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016.

[30] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, and T. Wolf. Datasets: A community library for natural language processing. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 175–184, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.21.

[31] M. Post. A call for clarity in reporting BLEU scores. In Proceedings of the Third Conference on Machine Translation: Research Papers, pages 186–191, Belgium, Brussels, Oct. 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-6319.

[32] J. Woodard and y. . .-j. . W. t. . A. Nelson, J.T.

[33] A. Morris, V. Maier, and P. Green. From wer and ril to mer and wil: improved evaluation measures for connected speech recognition. 01 2004.

Appendix A

Example of collate function

```
def CollateCustom(batch):
    # its required to pad the batch check max size and pass this information for the model
    Sizes = []
    for item in batch:
        Sizes.append(len(item[4])+1)
    YName = []
    YLocation = []
    YDate = []
    XFullImage = []
    # alocate a NP array of empty objects
    XCrop = np.empty((len(batch), max(Sizes)), dtype=object)
    for i in range(len(batch)):
        YName.append(batch[i][0])
        YLocation.append(batch[i][1])
        YDate.append(batch[i][2])
        XFullImage.append(batch[i][3])
        XCrop[i][0:len(batch[i][4])] = batch[i][4]
    return YName, YLocation, YDate, XFullImage, XCrop, Sizes
```

Appendix B

Dataset Graphics

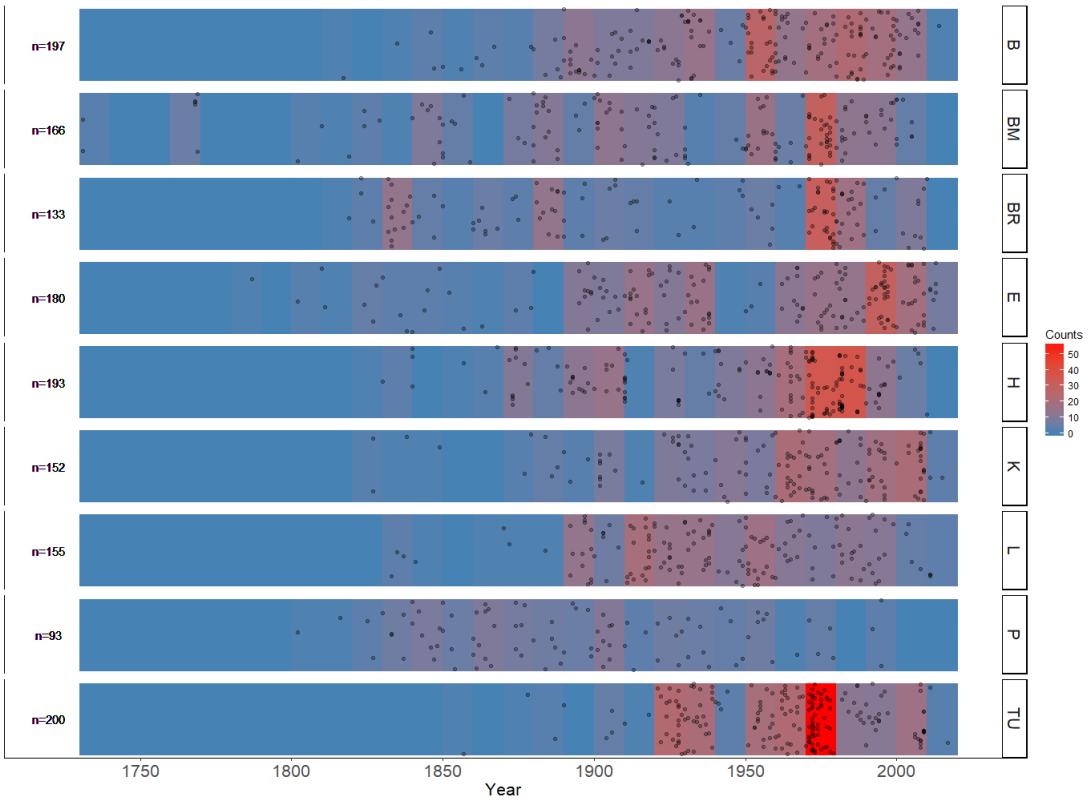


Figure B.1: Image of the distribution samples over the years with separation of language

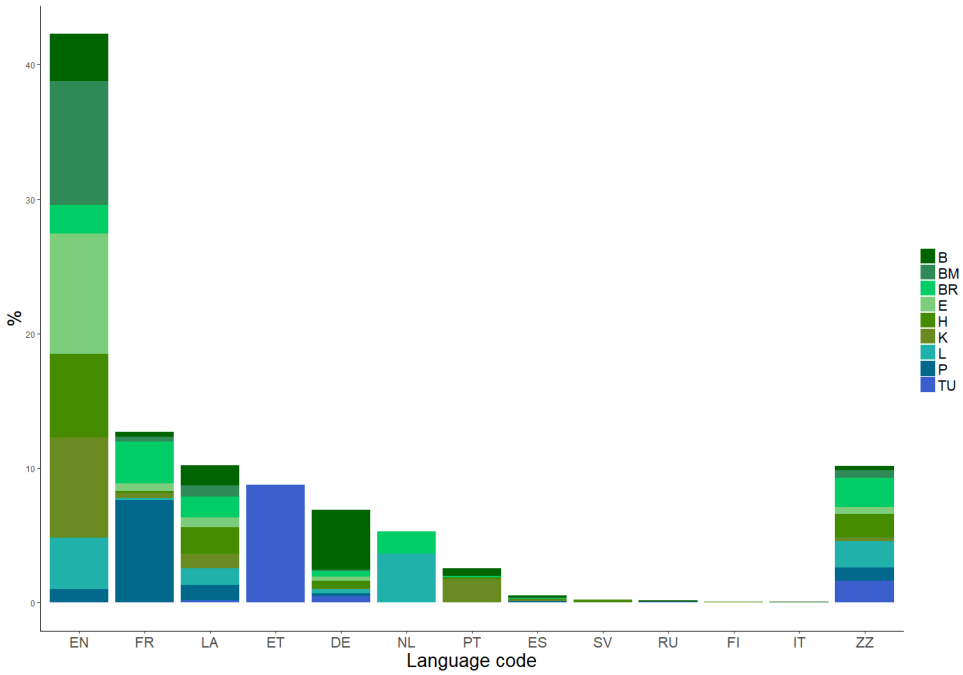


Figure B.2: Distribution of the datasets by language

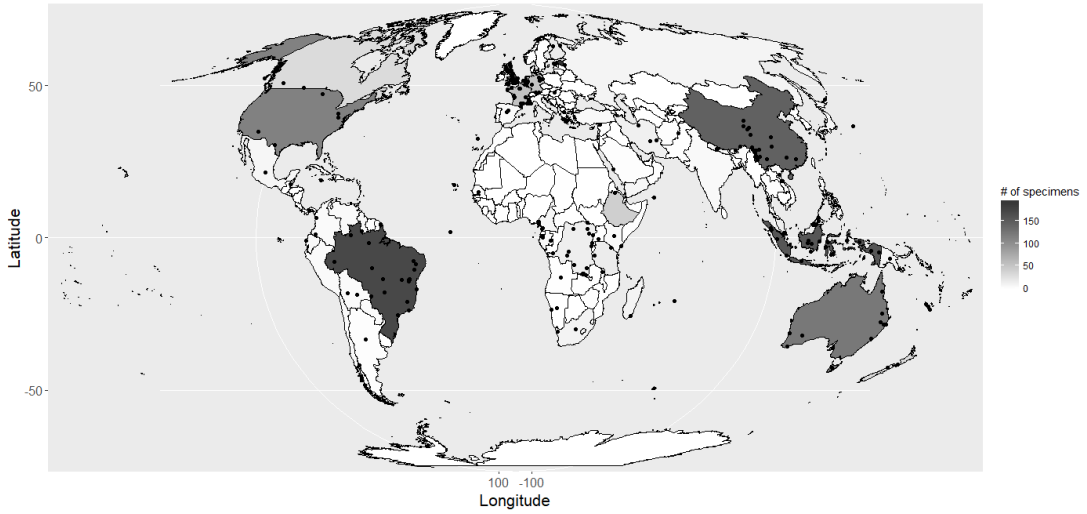


Figure B.3: Global Coverage of the specimens

This image set is self-plagiarized from A benchmark dataset of herbarium specimen images [27].

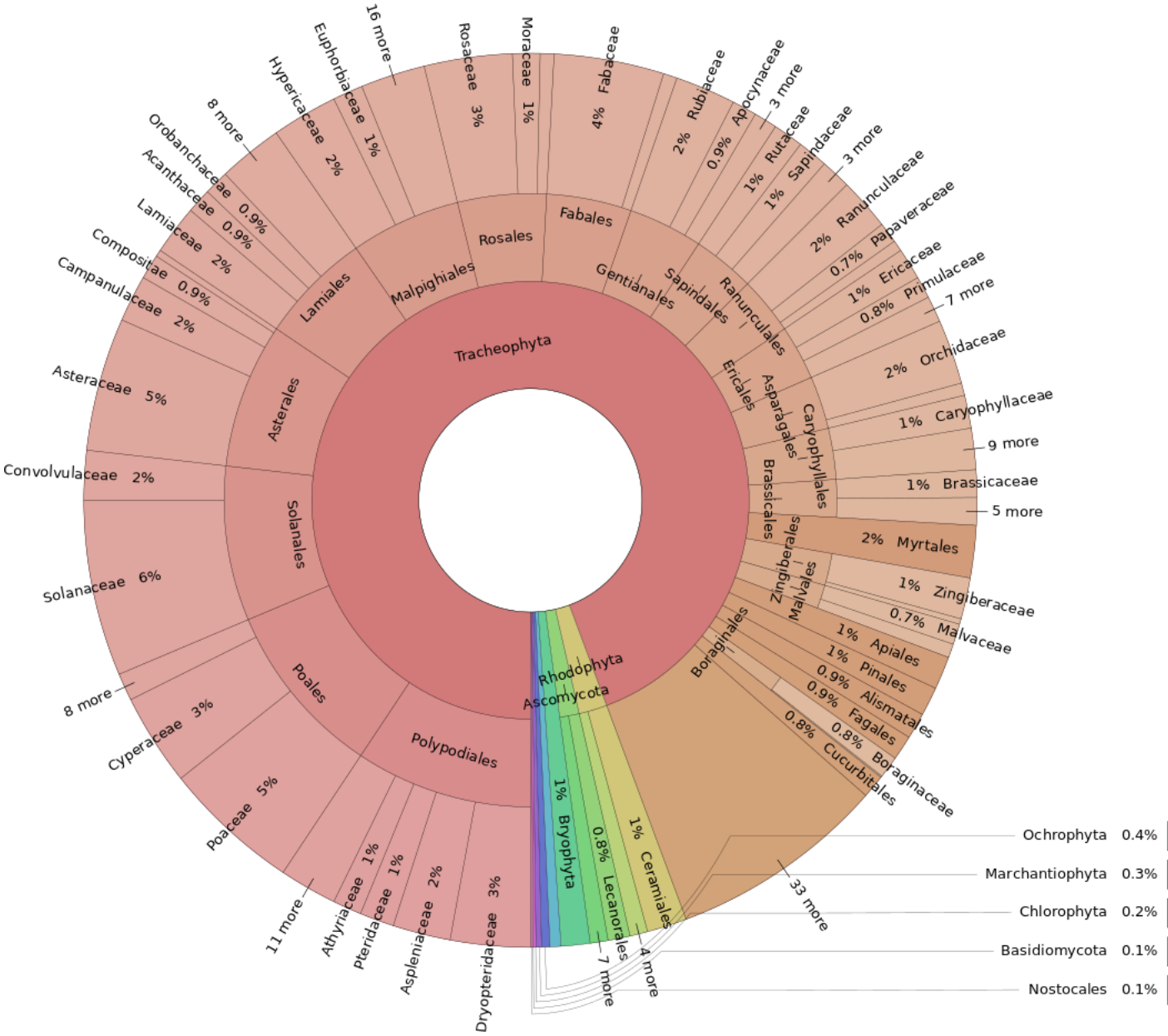


Figure B.4: Image of the distribution of herbaria specimen