Deep Learning Methods for Processing Digitized Herbarium Specimens

Filipe Lucas Borges Seleiro Martins
filipe.seleiro.martins@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

June 2022

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, Herbarium, Text generation based on Images, Computer Vision, Machine Learning

1. Introduction

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. This preservation task is mostly done by 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 SOTA Ai model, thus cutting in half the manual labour. 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 CNN and a top charting model called YOLOv4 for the task of Image Detection, Recognition, and Classification. This model will identify the text labels presented on the herbarium sheet. For later cropping text and to be used as a input for our huggingface [13] transformer model that will aim to extract relevant data for cataloging purposes. Despite the unsatisfactory results of our model to solve this problem, we were able to retrieve some conclusions about this architecture of multiple decoders and encoders that would allow other fields to benefit it.

2. Background

In this chapter, we will review the multiple models architectures that are used in our solution, by reviewing each implementation and the implementation of new techniques that made they achieve top performance.

2.1. Object Recognition and Region Segmentation - YOLOv4 The First Step For Our Solution

The YOLOv4 (You Only Look Once)v4 [1] a model that is designed for the task of Object Recognition and Region Segmentation. This CNN single stage detector based model will be utilized for identifying and extracting regions of written data where most of our information resides on herbarium sheets, we will explain the model architecture and techniques used to compose this top charting model.

This model employs multiple techniques that improve on the last iterations . Starting by the Backbone : The CSPdarknet53 as the name hints, this

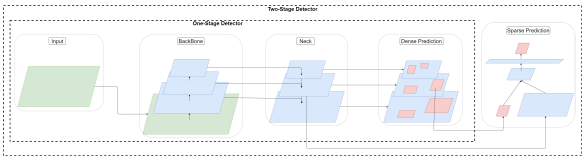

Figure 1: State of the Art typical CNN Architecture for object detection.

block is composed by 53 layers of convolutions also that now on YOLOv4 make use of the CSPnet technique in order to prevent a problem called "Vanishing Gradient". This technique splits the input in two parts , one is routed without any modification, and another is applied the convolution as part of feature extraction, in the end of the process it simply concatenates at the end the result of each path. The neck is composed by two layers the first is a SPP: this techniques if done to increase the receptive field this techniques applies multiple parallel convolution with different kernel sizes and strides to get multiple contexts for the same layer. Following by a PANet modified, this process of instance segmentation by preserving spatial information that might improve the detection head.This technique was a replacemente for the FPN implemented on the YOLOv3 achieving great results. The head remained the same as YOLOv3 and its main function to locate bounding boxes and performing classification.

	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 1: Structure of Darknet-53 from Yolov3

Since the original Yolov4 implementation was on Darknet, we used a repository made by Tianxiaomo 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 into PyTorch checkpoint. Since this model was not created by us, we will give full credits to the original creator Tianxiaomo on Github [10]. We fine-tuned the YOLOv4 model using the pretrained checkpoint in the MS COCO dataset leading us to a shorter training period and great results.

2.2. 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) [11]. With this new architecture came some important advancements on attention mechanisms and performance, these networks called transformers networks were crafted with a self-attention mechanism in mind and made huge advancements on the computational efficiency. These networks are design to be highly parallel computational by design. These networks have been developed by Google, and multiple models based on this architecture have been created with a huge success like Bert[2] achieving the best results for the NLP [12]. 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 [4] lead to some great results already paving the path to be promising model of Vision Transformers.

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 has been tensor to tensor or sequence to sequence.

There is some pre-processing required for our transformer to work as intended. This process begins with the *embeddings*, which consists of 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 a positional encoder. Taking the size of our token and modulating through a sinusoidal function with different frequencies according to the size.

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

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

Where pos is the current position and i is the total dimension. The first one is a multi-head atten-

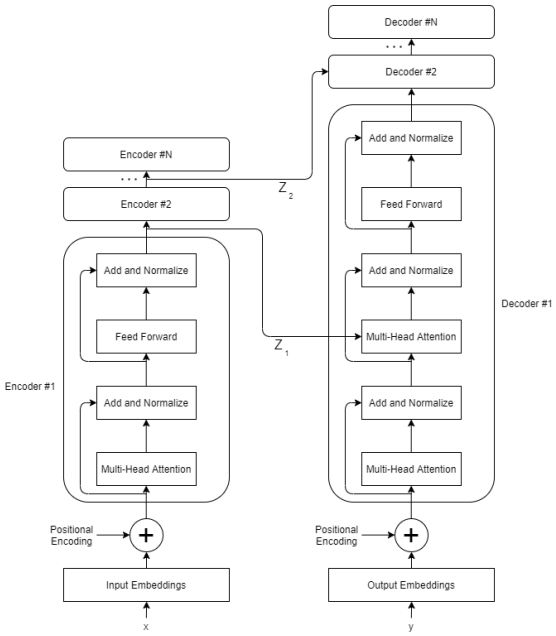

Figure 2: The Transformer neural network architecture.

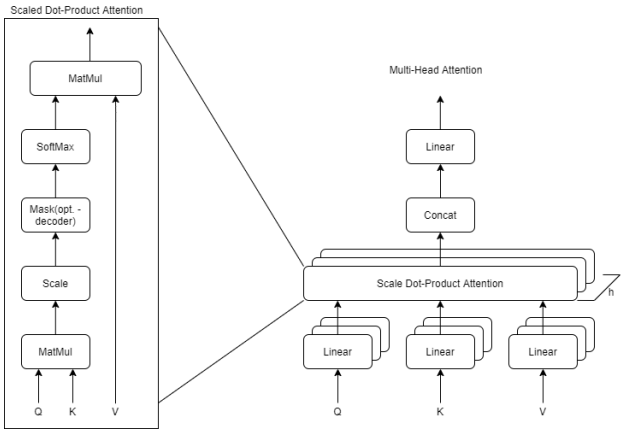

Figure 3: Scaled-Dot product and Multi head-Attention.

tion mechanism that focuses on how surrounding positions affect the current one, it achieves this by doing an all to all comparison and calculating a attention value. 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 predictions 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 achieve this is by calculating an attention value for each input that can be described as a 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.
The scaled dot-product attention can be described 3 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:

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

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 our model to 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:

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

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

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

2.3. ViT - The Image Captioning model
The ViT model is a particular transformer since it doesn't use a decoder stack. Introducing the Transformer networks for image recognition, starting with the first implementations that follow the

traditional Transformer, and the network the Vision Transformer (ViT) [4]. ViT doesn't use encoder-decoder methodologies, it just borrows the self-attention layers of the encoder but when paired with a NLP decoder this transformer can be used for OCR. 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 $X_p \in \mathbb{R}^{N\times\left(P^2\times C\right)\times D}$.

This model also borrows the concept introduced on BERT's transformer networks [2]. 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 instead of their absolute position. They achieve this by using $1-$dimensional for the Relative Attention.

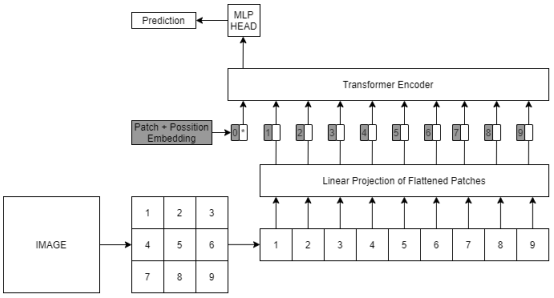

Figure 4: ViT model Architecture

3. Implementation

Before introducing our implementation of our model, we have to explain our pre-processing module used for data preparation and training data set creation. We will go over our data analysis from the acquired data following the process of filtering, transformation and augmentation of our data set with computer image generation.

3.1. Pre-Processing

The dataset used is an already established and compiled collection with over 1,800 herbarium specimens [3] for the purpose of cataloging. This dataset 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 dataset has present multiple languages that the text labels could be written. All these languages provide us with a good distribution around the globe and variety of the data presented on the labels. 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 standardization was a tool that serves to manually annotate our images from this dataset. The VGG [6] - Image Annotator (VIA) [5], 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 and to characterize the quality of the sheet and label with notes about missing or miss-matched information.

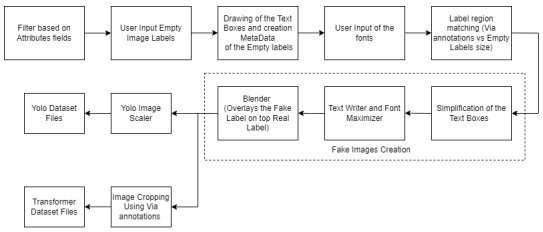

Figure 5: Pre-Processing Module Functions.

Our Pre-Processing module has multiple functions that work in conjunction to manipulate our data in order to reach data standardization. This module as multiple task: To Filter; Computer generate more examples from the manual annotated images; And create the dataset files used on the training of each model. One early 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 variances 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 middle steps begin the process of computer-generated images. These processes will ask the user to provide, images files of empty herbarium labels and Fonts to each be

placed on a specific path. Then the user will be prompt annotate each empty label image provided, using rectangles to indicate the useful space in the image of the multiple text fields are located. Theses rectangles will be simplified and eventually turned into line like textbox where our computer-generated text will be placed. Before starting to generate computer generated labels or "fake" labels, we decided to do a resolution matching between the Via annotation of the label's sizes with the empty label's 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 textbox like. For this to happen we will just explain briefly the code used. The code used is heavily recursive that test multiple relations boxes to box in order to combine and or associate them based on the type and relative position to one another. The output can be seen on the 6.

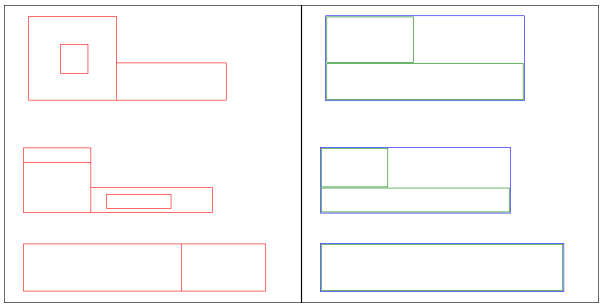

Figure 6: Boxes Simplification provided with our demo script

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 mean 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 7 applied to two sentences: "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".

As for methodologies for blending, we tested multi-

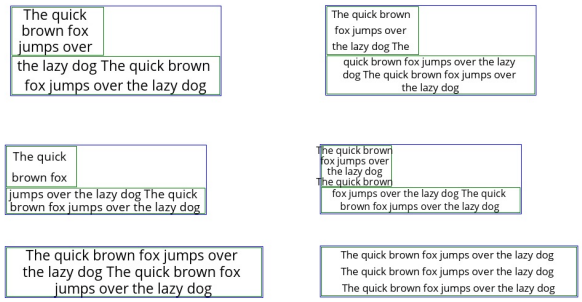

Figure 7: Text Algorithm applied to the Boxes Simplification example

ple 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 to 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.

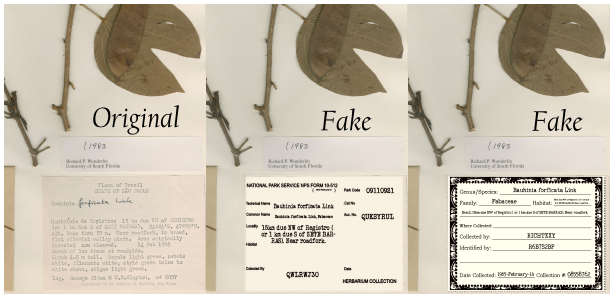

Figure 8: Results of generated fake labels blend onto a real image

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

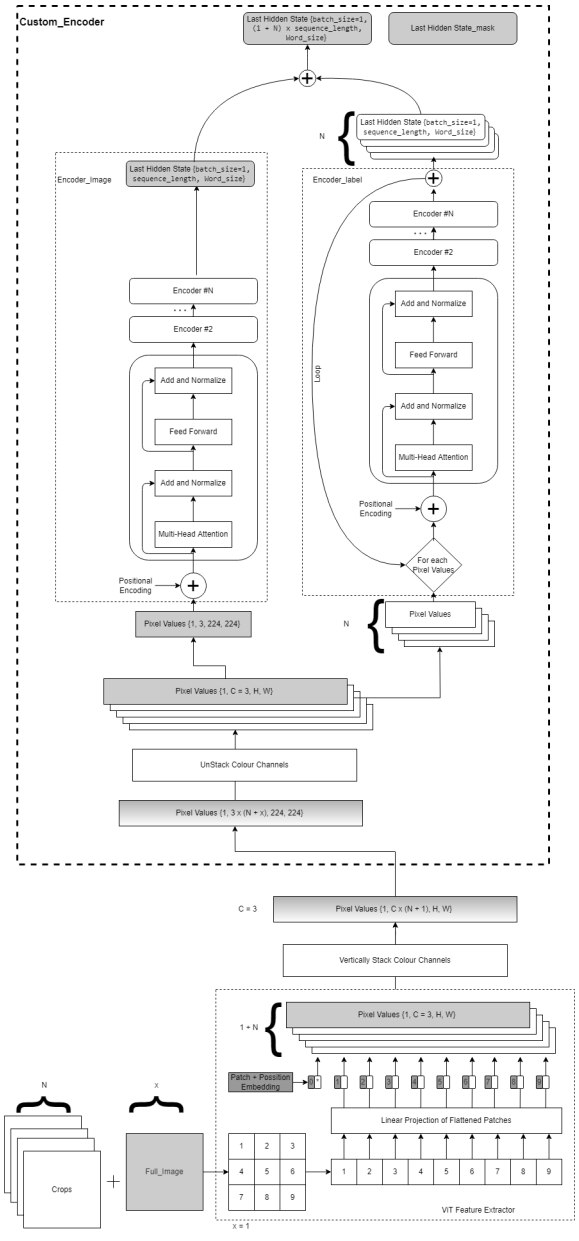

Figure 9: Model design encoder part using ViT encoder as example with Single Batch processing

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 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.

With the same present logic that the attentions mechanism can be exploited to do different tasks. Our implementation of the custom decoder module 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 a 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 datafrom 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 decoder 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 [13], 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 already 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 [13]. We found what each dimension from the input tensor has four dimensions comprised of:

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

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 a 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 implement a loop inside the forward function of our Custom_Encoder for each crop Pixel_Values to be encoded on the encoder_label. As seen 9. 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:

$$Pixel\_Values_{(1sheet\ \&\ 1crop)} = (Batch, 2 * 3, Width, Height)$$

$$Pixel\_Values_{(1sheet\ \&\ 2crop)} = (Batch, 3 * 3, Width, Height)$$

$$Pixel\_Values_{(1sheet\ \&\ Ncrop)} = (Batch, (1 + N) * 3, Width, Height)$$

After the image embedding have been processed by each encoder, 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 these models.

$$Encoder\_hidden\_States\_mask = (Batch, N\_Words, hidden\_size)$$

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 [7] library by huggingface for Dataloader that generates this value.

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

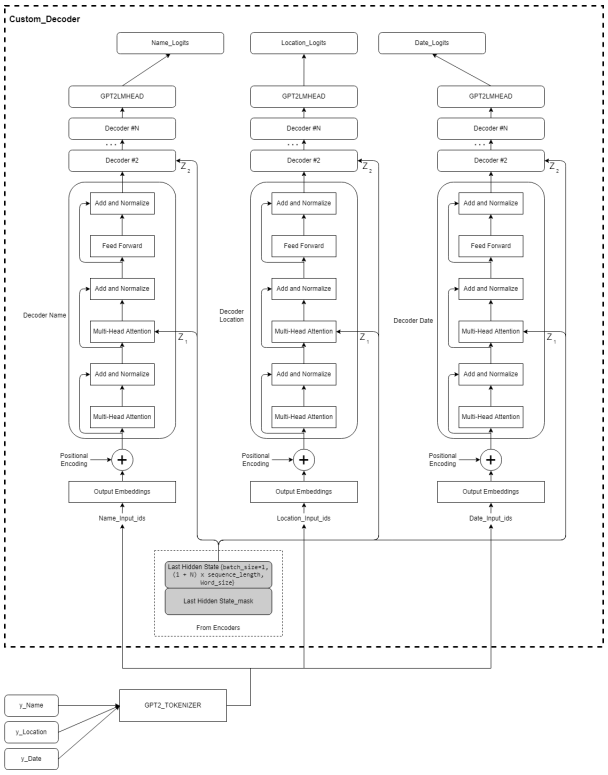

Figure 10: Model design decoder part using GPT2 with a LM head as example and Single Batch processing

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 run 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 it's 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 module simply outputs the result of the three inference runs from each the decoder weights.

4. Results

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 tendency to infinite or the text generated was close to gibberish leading us to quit from most combination. From this point we will reference the combined model as simple the model.The one that stood out was the combination using the ViT with GPT2. This combination gave a semi-usable output since the fifth epoch of training making us choose clearly this combination for the rest of the results presented.

4.1. Tests

For these early test we will compare and analyse only the ViT large and base models and try to get some conclusions Our full model as a total accuracy for predicting the labels is around thirty percent depending on the decoder. We suspect that our model is not complex enough for this type of analysis since it requires OCR and also attends to the herbarium specimen characteristics.

For this model performance comparison we be analysing and sharing some thoughts by analysing the metric BLEU using sacreBLEU 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.

When overlapping the tests, we can see a clear winner but the results were unexpected. Despite the advantage of the ViT large(384x384) input resolution in relation of ViT base (224x224), 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 initial-

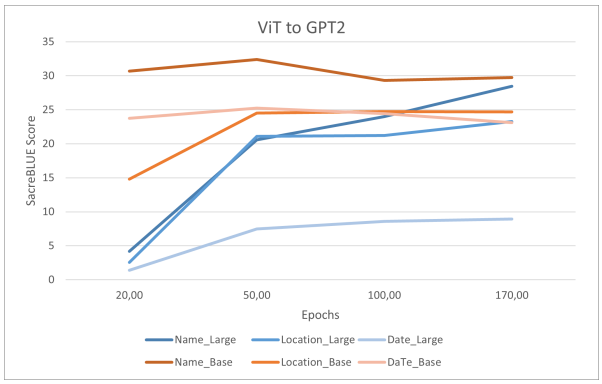

Figure 11: ViT to GPT2 SacreBLUE Score Results

ized 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. By keeping in mind this tests our results will be using the ViT-base with GPT2 since these are the better performer.

4.2. Results

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, we choose this due to be well representative of the real world data.

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.

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.

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.

The results are not satisfactory, proving that our model is not suitable for extracting information from all cases and only from semi-standardized data.

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 [8] proves that Deit and Beit paired with RoBERTa [9] 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.

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 examples on the training dataset. 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. 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.

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.

Acknowledgements
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.

References

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.