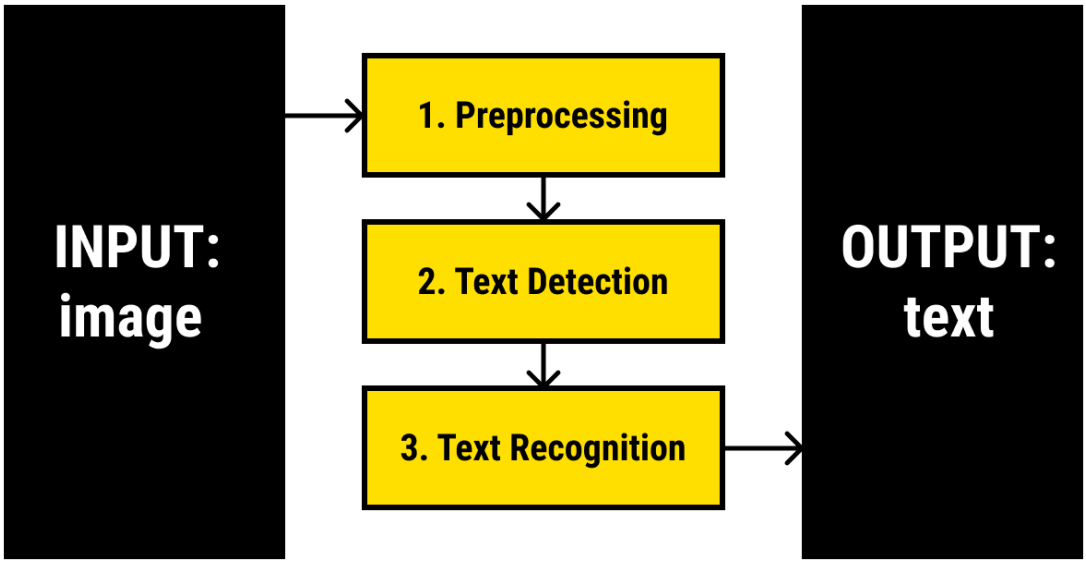
Document Image Analysis

- Data Collection
- Data Preprocessing
- Convert scanned image to text



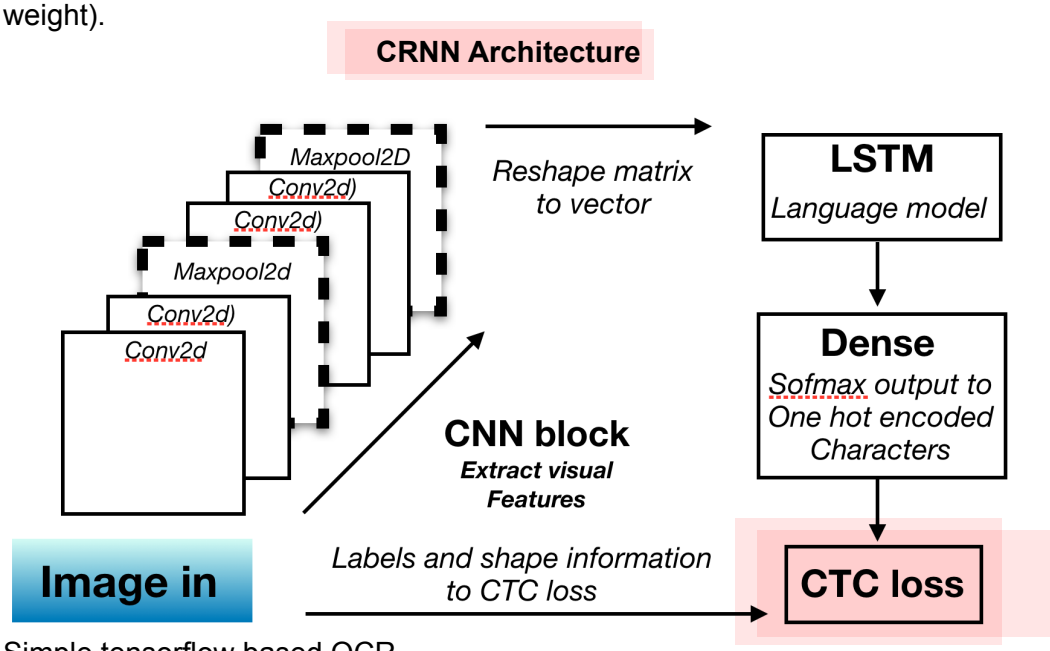
  - Preprocessing
    - This OCR step includes simplification, detection of meaningful edges, and defining the outline of the text characters. This is a common step for any task that has an image recognition component in it.
  - Text detection
    - This step of an OCR project requires drawing a bounding box around the pieces of text found on the image. A few of the legacy techniques used for this step include SSD, real-time (YOLO) and region-based detectors, sliding window technique, Mask R-CNN, EAST detector, etc.
  - Text recognition
    - The final OCR step is to recognize the text that was put in the bounding boxes. For this task, one or a combination of convolutional and recurrent neural networks and attention mechanisms is frequently used. Sometimes this step may also include the interpretation step, which is characteristic for more complex OCR tasks like handwriting recognition and IDC.

Deep Learning OCR with a Convolutional Recurrent Neural Network (CRNN)

- This method follows the two steps after the images were preprocessed for OCR:
  - **Convolutional neural network (CNN) to extract the features;**
  - **Recurrent neural network (RNN) to predict the location and value of the text characters.**
  - CNNs are one of the best techniques to use for deep learning OCR for the step of text detection. Convolution layers are commonly used for image classification tasks due to their efficiency in feature extraction. They allow detecting the meaningful edges in an image and (on a higher level) shapes

and complex objects. Compared to fully-connected layers, for example, convolutional layers decrease the complexity of a machine learning OCR algorithm by reusing the pattern-detection filters throughout an image.

- RNNs are used next to identify the relationship between the characters. Recurrent networks are great at processing the sequences of inputs that have variable lengths, such as speech recognition or unstructured text (e.g., handwriting recognition for OCR). Most commonly, long short-term memory (LSTMs) cells are used to avoid the vanishing gradient problem (when the weight prescribed to signify the value of each specific input, the gradient may be too small to have the necessary effect on updating the value of the weight).



- Simple tensorflow based OCR .
  - https://github.com/kutvonenaki/simple_ocr
  - https://towardsdatascience.com/get-started-with-deep-learning-ocr-136ac645db1d
  - https://github.com/kutvonenaki/simple_ocr/blob/master/OCR_simple.ipynb
- CTC Loss
  - https://distill.pub/2017/ctc/
- Text Recognition Data Generator
  - https://textrecognitiondatagenerator.readthedocs.io/en/latest/index.html

Attention Mechanisms and Transformers in OCR Algorithms: Why Add Them to Your Neural Nets?

- Additions to CRNN models can be used to improve the prediction of the text in the input images. One such popular addition is an attention mechanism that is commonly added to optical character recognition algorithms to create attention-OCR models.

- Attention was initially introduced for the neural machine translation approach. Attention is used to predict the target text units based on the context vectors, as well as previously-generated target data pieces. Attention vector allows evaluating the weight of a target data piece (such as a word for an OCR model) by its correlation

with other data pieces (words). To put it simply, attention mechanisms are used for long-range dependency prediction that CRNNs and LSTMs are not capable of on their own.
https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/

- The accuracy can be further improved with multi-head attention: this is when an attention mechanism is run in parallel several times. This allows us to separately evaluate different dependencies (e.g., long-term vs short-term). The resulting concatenated output then can be further used to make the predictions of the deep learning OCR algorithm more precise.

- What about transformers? They are another popular way of increasing the accuracy of OCR architectures. A transformer basically performs a similar function to LSTM with the difference that, unlike an RNN, it doesn't require the input data to process in order (that is, from beginning to end). This can significantly decrease the time necessary to train such an OCR algorithm. A few well-known and widely acknowledged NLP transformer models are BERT, as well as GPT-2 and GPT-3. You can read more about transformers following this link, with examples and a few visualization schemes.
https://arxiv.org/abs/1810.04805
https://www.linkedin.com/pulse/can-gpt-3-replace-humans-just-yet-karyna-naminas/?trackingId=X03ShN1cQV%2B4R%2B6fv4YRSg%3D%3D
https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

RAM and DRAM: Recurrent Attention Models in Deep Learning OCR

As neural networks are vaguely based on the functioning of the biologic brains, similarly recurrent attention models (RAMs) use the idea that a certain part of a new image attracts the attention of a human eye. During the visual attention OCR process, an image is divided into “glimpses” of data to be processed for information. This allows creating glimpse vectors that contain meaningful features from every piece of an image. An RNN then processes these glimpse vectors to predict other pieces of an image to process next. Backpropagation is used to ensure the accuracy of the output data.

DRAM (Deep Recurrent Attention Model) is similar to RAM but uses two RNNs instead of one, which makes the OCR processing of the image with the text more efficient. The first RNN is dedicated to analyzing the next glimpse location. The second RNN is used for the classification task as it assigns the data labels to the text characters.

- https://github.com/kevinzakka/recurrent-visual-attention#:~:text=The%20Recurrent%20Attention%20Model%20%28RAM%29%20is%20a%20neural,up%20a%20dynamic%20internal%20representation%20of%20the%20image.
- https://papers.nips.cc/paper/2014/file/09c6c3783b4a70054da74f2538ed47c6-Paper.pdf
- https://deepai.org/publication/recurrent-models-of-visual-attention
- https://medium.com/@sunnerli/visual-attention-in-deep-learning-77653f611855#:~:text=As%20the%20result%2C%20the%20deep%20recurrent%20attention%20model,is%20that%20it%20uses%20two%20stacked%20LSTM%20unit.

- https://arxiv.org/abs/1706.03581
- https://github.com/ablavatski/EDRAM

Spatial Transformer Networks

Spatial Transformer Networks, augment input images by applying affine transformations so that the trained model is robust to variations in data.

The network consists of a localisation net, a grid generator and a sampler. The localisation net takes an input image and gives us the parameters for the transformation we want to apply on it. The grid generator uses a desired output template, multiplies it with the parameters obtained from the localisation net and brings us the location of the point we want to apply the transformation at to get the desired result. A bilinear sampling kernel is finally used to generate our transformed feature maps.

- https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html#:~:text=Spatial%20transformer%20networks%20boils%20down%20to%20three%20main,transformation%20and%20applies%20it%20to%20the%20input%20image.
- https://arxiv.org/abs/1506.02025
- https://github.com/kevinzakka/spatial-transformer-network

Building your own Attention OCR model

We will use attention-ocr to train a model on a set of images of number plates along with their labels - the text present in the number plates and the bounding box coordinates of those number plates. The dataset was acquired from here.

The steps followed are summarized here:

1. Gather annotated training data
2. Get crops for each frame of each video where the number plates are.
3. Generate tfrecords for all the cropped files.
4. Place them in models/research/attention_ocr/python/datasets as required (in the FSNS dataset format). Follow this link or the following sections of this blog.
5. Train the model using Attention OCR.
6. Make predictions on your own cropped images.

https://nanonets.com/blog/attention-ocr-for-text-recogntion/

Datasets to Use in an OCR Deep Learning Project

In order for the OCR algorithm to function properly, it is necessary to train it. There are a lot of datasets that can be used for a deep learning OCR model to train, and their specificity depends on the tasks the model is to solve. Here are a few of the most popular datasets:

MNIST Dataset: it trains an OCR neural network by showing it one of the numerical digits at a time;

SVHN (Street View House Number) Dataset: training of an OCR model on house numbers, as the name suggests, with the challenge that each number is written in a digger size, font, shape, and writing style;

SVT (Street View Text) Dataset: an OCR algorithm trains on images captured outdoors, which means they are often noisy, of low quality, with additional artifacts and complex backgrounds;

Scene Text Dataset: a combination of text and digits allows to train an optical character recognition model in English and Korean languages;

Devanagari Character Dataset: this is an example of the dataset for OCR training in a different language from English.

Document Extraction

The problem is quite simple to define. You have documents say invoices, receipts, Purchase orders, etc that come into your company's numerous workflows. These documents are manually digitized by a human operator and fed into a software system which is time consuming and error-prone. You want to automate this digitization using Deep Learning.

Understanding The Data

Firstly we need to get the OCR results on the images. Since we'll be attempting to use visual as well as text features. OCR for the most part can be assumed to be a solved problem and there are many OCR APIs like Tesseract.

Different OCR engines provide us with a different kind of output. Take for example your OCR outputs consist of blocks of information. Each block is associated with a type. For us the relevant ones are the ones of block type Text. Extracting all blocks with block type text lets us see what each block of text looks like. Inside each block of text, there are lines and within each line, there are characters. The blocks, lines as well as characters are associated with bounding box information. For Tesseract, the format is different and the bounding box information can be acquired with line numbers, paragraph numbers. It is also possible to obtain character boxes as well as text boxes. To learn more, visit this blog post.

We can build a pipeline that utilizes said bounding box information or we can build an engine completely only based on line and block information. Depending on the use-case and the kind of data one might have to deal with, the approach must change.

Template Matching & Why It Does Not Work

To build a template based OCR solution, we will have to create a list of patterns that each field might represent and write regular expressions for each. Every line would then be iterated over to search which lines match which pattern. Depending on which pattern is matched, we will be able to infer which field an extracted line belongs to. Once we have all the fields extracted, we can finally put them in an organised format for future use. You can find a great resource to pattern matching here.

The above mentioned approach has many limitations like being vulnerable to changing layouts, getting confused with multiline items and the risk of ending up with a product built on lines and lines of unreadable code. Pattern matching is not advisable for data which is highly unstructured and when the structure itself varies often. Also creating the list of patterns that might work for a generalised invoice can be a daunting task in itself and often, this is only applied in situations where a tool is being utilized in-house for specific kinds of documents that are set in their layouts.

Solving the Problem with Machine Learning

Machine learning approaches provide us with an alternative solution that allows us to circumvent our dependence on knowing document templates to extract text information and automate processes. We will look at a few of these approaches, try to understand the concepts associated with them and try to chalk out an end to end solution for our problem.

Named Entity Recognition

Named Entity Recognition allows us to evaluate a chunk of text and find out different entities from it - entities that don't just correspond to a category of a token but applies to variable lengths of phrases. The models take into consideration the start and end of every relevant phrase according to the classification categories the model is trained for.

All the lines we extracted and put into a dataframe can instead be passed through a NER model that will classify different words and phrases in each line into, if it does find any, different invoice fields.

The SpaCy NER tags include DATE for invoice dates, ORG for vendor names, MONEY for prices and sum total, PRODUCT for items sold, CARDINAL for phone numbers, invoice numbers, etc. We can apply transfer learning and retrain NER models provided by popular NLP libraries like SpaCy, NLTK and Stanford CoreNLP by adding our new entities in their vocabulary and retraining the models. We can also search for patterns based on POS tagging using regular expressions or one of the above mentioned tools.

The key here is designing a strong NER tagging model. To do so, we need to be sure of two things -

How we represent word and character level features
What algorithms and architectures are used to make predictions.
Training Models for NER Tagging
We can utilize TF-IDF Vectorizer, n-grams or skip-grams to extract our feature representations, utilize GloVe Word2Vec for transfer word embeddings weights and re-train our embeddings using Keras, Tensorflow or PyTorch. Using embeddings will substantially increase our model strength since it will allow us to capture semantic information better.

We can build our model using traditional machine learning algorithms like SVMs and Naive Bayes as detailed here, deep learning architectures like RNNs, BiLSTMs, CRFs as is done here and here or using transformers like BERT/XLNet as done here. This repository provides all the transformer architectures and example training and inference scripts for several NLP

tasks, including NER tagging. The example script utilizes NER tagging for text summarization but can be repurposed to fit our task.

Once each block is classified into an invoice field, we can put together all the text present in different blocks to produce our results, organised in a JSON or XML format.

Understanding the Output

The way to create our final outputs will depend on how we want to look at the fields extracted. Considering we are going line by line in our pipeline, it would be easier and less prone to errors if we were to club together different extracted named entities into a sort of nested structure. How different lines and chunks of text are extracted and tagged into invoice fields from the predictions derived using NER tagging on phrases and words is explained in the inference evaluation section.

Take for example the items bought that are mentioned in the invoice. It is important that each item is mapped with its price and the quantity. To do this we can create an object called 'Inventory' which will include different named entities like the product, the price and the quantity. This is similar to what is described here as nested entities.

Building a Dataset
For all of these situations, besides the strength of our learning algorithms, a deciding factor on how our model performs will be the strength of the data we train our model on. We might need to create a custom dataset for different fields and the phrases that represent these fields. For example,

The field 'Balance' can be associated with

'Balance due',
'Payment due',
'Amount',
'Amount to be paid',
'To be paid',
'Amount due', etc.
This process can be done semi-automatically by extracting OCR outputs, finding chunks of text that contain some key words like 'Amount', 'Due', 'Balance', etc. and extract all the phrases found in all the invoices that match the patterns. But creating an exhaustive list is usually a difficult task and we would want to consider using word embeddings and document embeddings to extract features out of these phrases and map them to our fields. So if we encounter a new phrase or word, we can compare our word embeddings to find if the field is similar in its semantic information to do our classification. This is similar to what SpaCy documentation called entity linking using a knowledge base.

Evaluating the NER Tagging Model

When we are able to extract named entities, it is usually done by classifying words or phrases into different fields. Besides finding out the accuracy of our named entity recognition during training, we also need some way of knowing how our tagging model is performing on unseen data. We will, in the coming sections, look at how to evaluate our training process, how to evaluate a continuous training loop and how to measure our inference performance.

NER Training

We evaluate the training process by creating a testing dataset and finding some key metrics -

Accuracy: the ratio of correct predictions made against the size of the test data.
Precision: the ratio of true positives and total predicted positives.
Recall: the ratio of true positives and total actual positives.
F1-Score: harmonic mean of precision and recall.

Different metrics take precedence when considering different use-cases. In the case of invoice processing, we know that a goof-up in the numbers or missing an item can lead to losses for the company. This means that besides needing a good accuracy, we also need to make sure the false positives for money related fields are at a minimum - so aiming for a high precision value might be ideal. We also need to make sure that details like invoice number and dates are always extracted and done correctly since they are needed for legal and compliance purposes, so maintaining a high recall value for these fields might take precedence.

NER Re-training

Digitization of documents is a difficult task to completely automate and trying to aim for 100% automation is seldom practical. When models are encountered with unseen data, building humans in the loop solutions that allow humans to review the information extracted from text, correct predictions and allow for re-training can help constantly improve model accuracy while minimizing the costs algorithmic errors might incur.

Ideally, you'd want to set up a pipeline where you collect data for a few days/weeks/months (depending on the scale you are operating), retrain your model on this new data and evaluate the predictions to see if your model has improved or not.

Here's what the DevOps pipeline for Microsoft Azure looks like -

source
Versioning datasets, writing differential tests and including them in your continuous integration pipelines can help keeping the most stable algorithms deployed. The differential tests are meant to evaluate, in this case, the metrics we mentioned above, compare the models before and after re-training and decide if deploying the new models is a viable option. The differential tests will not allow models that are inferior post re-training to get deployed.

A note on human error and it's costs
Though automating the process of digitizing invoices reduces workload for humans and the chances of fatigue related errors, wrong annotations can cost the model, and not always equally. This paper talks about calculating annotation costs (they use a metric based on the length of the text re-annotated incorrectly) and trying to minimize the same in your retraining loop. The metric mentioned in the above paper will not be applicable to the use-case of

invoice processing but devising an algorithm that limits re-annotations with higher possible costs can help maintain your prediction quality for a long time.

NER Inference

Remember how we split our XML data into lines and chunks? OCR solutions, especially the commercial ones produce one of the most reliable OCR outputs for digital documents in the industry. Often, an entire chunk will belong to the same invoice field. For an unseen document, assuming our OCR predictions are satisfactorily accurate and that we have a different way of dealing with nested entities (more on this in the next paragraph), we can devise a way to measure confidence in our predictions for each chunk instead of just relying on the confidence provided by our NER tagging model.

We can extrapolate the named entity prediction information for words and phrases to assign invoice fields to each of the chunks and lines extracted from text. We can measure confidence for each chunk by aggregating the named entity prediction confidence for each field that occured in the chunk.

For chunks with single field predictions, the mechanism is pretty straight forward. An average value for all the named entity confidence scores is returned.

If multiple fields occur multiple times in the same chunk, we extract as many confidence scores as the number of fields present in the chunk. Here we need to define a threshold value.

Let's do this with a basic example.

Here, we pick a threshold confidence of 0.5.
Say our algorithm detected 2 fields.
We have two NEs for field 1 with confidence scores 0.6 and 0.7.
We have three NEs for field 2 with confidence scores 0.3, 0.5 and 0.4.

Clearly, the mean confidence for field 2 is below the threshold, so the block would be classified as field 1 with a confidence of 0.65.

What if for the second field, the confidence scores are 0.5, 0.5 and 0.6?

This would mean two fields cross the threshold. In this case we would check if the two fields together fall into a nested field template. For example, a line item would consist of a product as well as the price. If that is the case, we aggregate all the confidence values and return the line item as the predicted field for the block with a confidence of, in this case, 0.58.

If it doesn't match any such nested field template, we simply return the field with the higher confidence, like we did in the first thresholding example.

We can also devise more complex block inference values calculations and the corresponding metrics on a weighted average approach instead of a thresholding approach where relations between different entities are either defined via rules according to our use-case and the data we might encounter more frequently or can be learned using relation extraction methods as is done here.

Using Embeddings and CNNs

Tesseract OCR provides us with bounding box information of every chunk of text, every line and every character detected in invoices. This provides us with rich spatial information which unfortunately has been left unexploited in all the approaches mentioned above. This spatial information can be utilized by placing our OCR text in a grid form and using convolutional networks to learn how different named entities correlate with the spatial information as is done here.

The gridded texts are formed with the proposed grid positional mapping method, where the grid is generated with the principle that is preserving texts' relative spatial relationship in the original scanned document image. The rich semantic information is encoded from the gridded texts at the very beginning stage of the convolutional neural network with a word embedding layer. This allows it to access spatial as well as semantic information of the document and indeed, for receipts and invoices, the model has performed better than NER tagging based approaches. The authors use a spatial pyramid based CNN architecture for this purpose.

Using Graph Convolutional Networks

The idea of merging spatial and semantic information for information extraction has been applied in this paper as well. The graph embeddings produced by graph convolution summarize the context of a text segment in the document, which are further combined with text embeddings for entity extraction using a standard BiLSTM-CRF model.

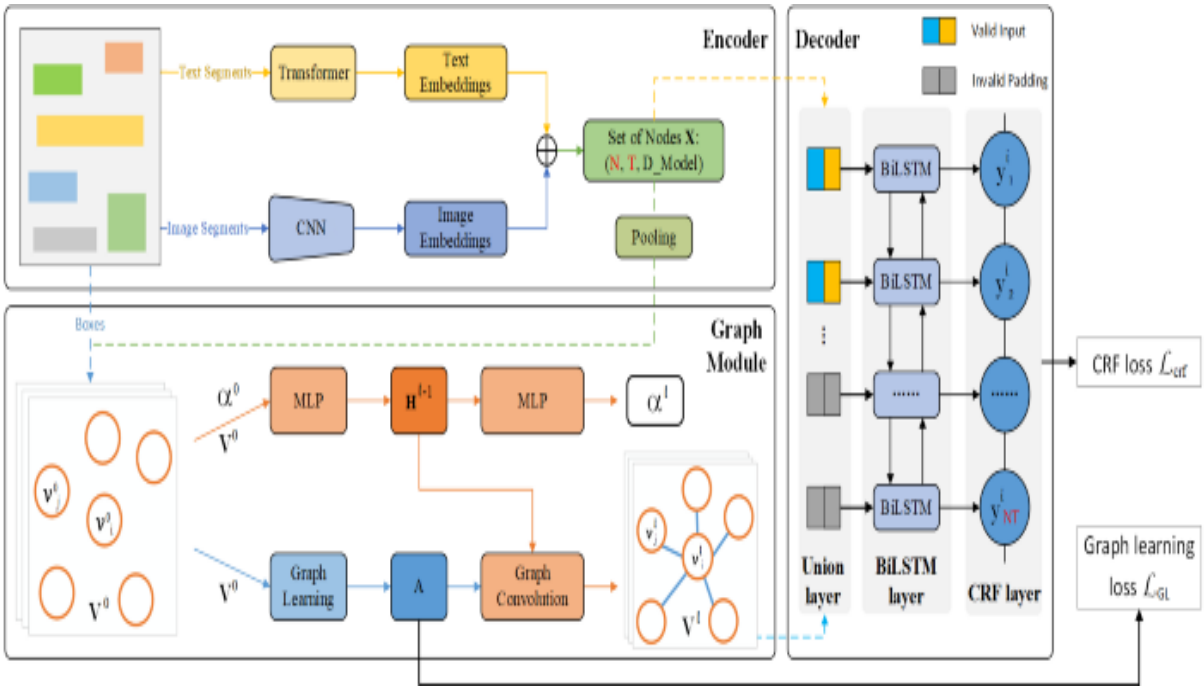
Here each word is modeled as a node and its spatial relationships with its neighboring words modeled as edges. Every node in the graph is associated with a label, which in our case would be our invoice fields, and we want to predict the label of the nodes without ground-truth. The ablation studies revealed that their field extraction did not get too affected if only graph embeddings were used but saw a substantial setback in performance when only word embeddings were utilized.

Summary

We looked at several different machine learning approaches to extract relevant fields from invoices that included out of the box NER tagging methods, training our own NER tagging models, using machine learning classifiers as well as deep learning architectures for the same. We also looked at a few methods for information extraction from text that made use of spatial information along with word embeddings. We discussed creating custom datasets and evaluating our training procedures as well as inference procedures. We looked at continuous learning loops and spoke of some best practices regarding the same.

PICK-PYTORCH

https://arxiv.org/pdf/2004.07464.pdf



PICK- Processing Key Information Extraction from Documents using improved Graph Learning Convolutional NetworKs, is a robust and effective method to improve extraction ability by automatically making full use of the textual and visual features within documents. PICK incorporates a graph learning module into existing graph architecture to learn a soft adjacent matrix to effectively and efficiently refine the graph context structure indicating the relationship between nodes for downstream tasks instead of predefining edge type of the graph artificially. Besides, PICK makes full use of features of the documents including text, image, and position features by using graph convolution to get richer representation for KIE. The graph convolution operation has the powerful capacity of exploiting the relationship generated by the graph learning module and propagates information between nodes within a document. The learned richer representations are finally used to a decoder to assist sequence tagging at the character level. PICK combines a graph module with the encoder-decoder framework for KIE tasks.

https://github.com/wenwenyu/PICK-pytorch

LayoutLM

LayoutLM is a simple but effective multimodal pre-training method of text, layout and image for visually-rich document understanding and information extraction tasks, such as form understanding and receipt understanding. LayoutLM archives the SOTA results on multiple datasets.
https://github.com/BordiaS/layoutlm
https://www.microsoft.com/en-us/research/uploads/prod/2020/02/layoutlm.pdf
https://huggingface.co/transformers/model_doc/layoutlm.html