Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
*** * ***
Université de Carthage
*** * ***
Institut National des Sciences
Appliquées et de Technologie

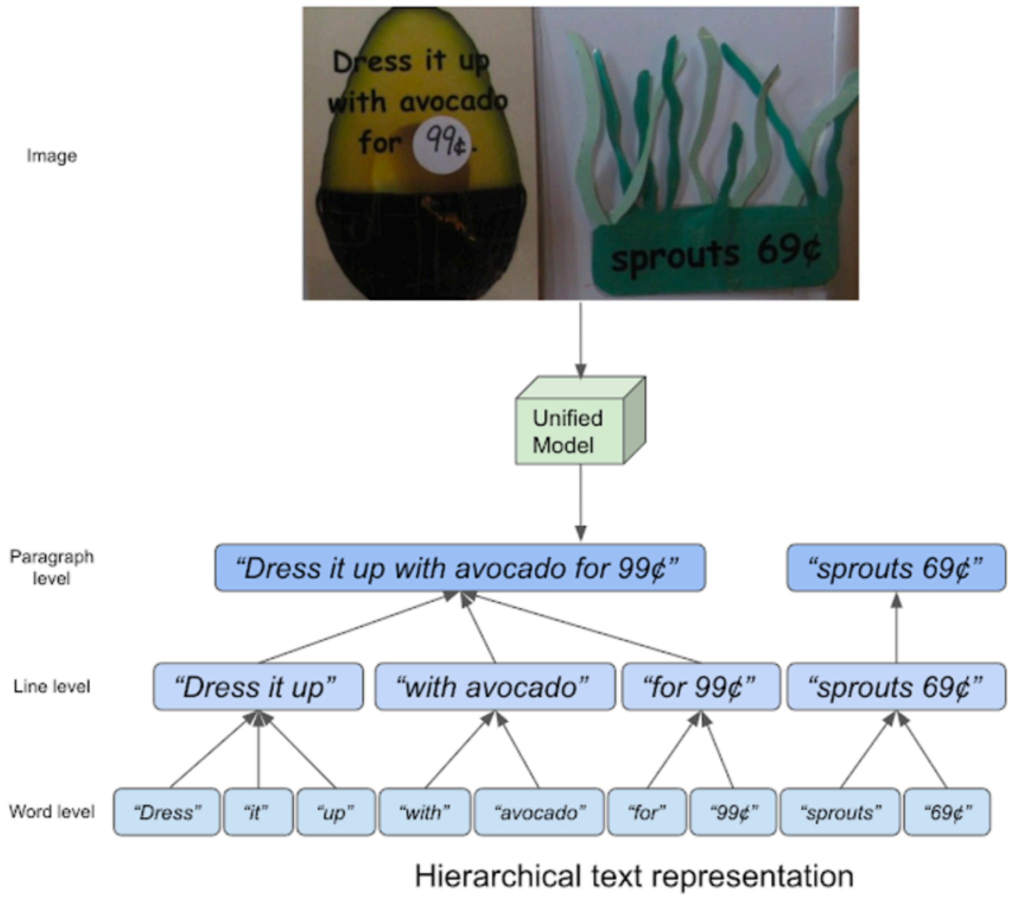
"GlyphNet: Enhancing OCR Accuracy with Post-OCR Glyph Embedding Correction"

- Presented by : Malek Zitouni

- Academic Year : 2024-2025

- The last few decades have witnessed the rapid development of Optical Character Recognition (OCR) technology, which has evolved from an academic benchmark task used in early breakthroughs of deep learning research to tangible products available in consumer devices and to third party developers for daily use. These OCR products digitize and democratize the valuable information that is stored in paper or image-based sources (e.g., books, magazines, newspapers, forms, street signs, restaurant menus) so that they can be indexed, searched, translated, and further processed by state-of-the-art natural language processing techniques.
- Research in scene text detection and recognition (or scene text spotting) has been the major driver of this rapid development through adapting OCR to natural images that have more complex backgrounds than document images. These research efforts, however, focus on the detection and recognition of each individual word in images, without understanding how these words compose sentences and articles.
- Layout analysis is another relevant line of research that takes a document image and extracts its structure, i.e., title, paragraphs, headings, figures, tables and captions. These layout analysis efforts are parallel to OCR and have been largely developed as independent techniques that are typically evaluated only on document images. As such, the synergy between OCR and layout analysis remains largely under-explored. We believe that OCR and layout analysis are mutually complementary tasks that enable machine learning to interpret text in images and, when combined, could improve the accuracy and efficiency of both tasks.

- Google announced the *Competition on Hierarchical Text Detection and Recognition* (the HierText Challenge), hosted as part of the 17th annual International Conference on Document Analysis and Recognition (ICDAR 2023). The competition is hosted on the Robust Reading Competition website, and represents the first major effort to unify OCR and layout analysis. In this competition, we invite researchers from around the world to build systems that can produce hierarchical annotations of text in images using words clustered into lines and paragraphs



The concept of hierarchical text representation.

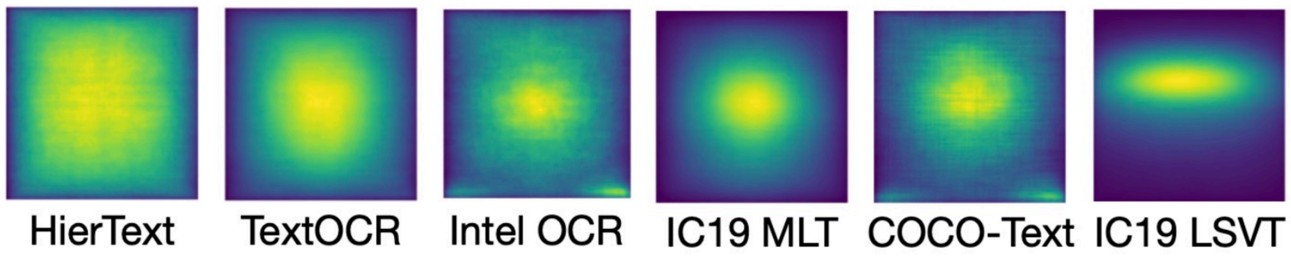
- In this competition, we use the HierText dataset that we published at CVPR 2022 with our paper "Towards End-to-End Unified Scene Text Detection and Layout Analysis". It's the first real-image dataset that provides hierarchical annotations of text, containing *word*, *line*, and *paragraph* level annotations. Here, "*words*" are defined as sequences of textual characters not interrupted by *spaces*. "*Lines*" are then interpreted as "*space*"-separated clusters of "*words*" that are logically connected in one direction, and aligned in spatial proximity. Finally, "*paragraphs*" are composed of "*lines*" that share the same semantic topic and are geometrically coherent.

- To build this dataset, we first annotated images from the Open Images dataset using the Google Cloud Platform (GCP) Text Detection API. We filtered through these annotated images, keeping only images rich in text content and layout structure. Then, we worked with our third-party partners to manually correct all transcriptions and to label words, lines and paragraph composition. As a result, we obtained 11,639 transcribed images, split into three subsets: (1) a train set with 8,281 images, (2) a validation set with 1,724 images, and (3) a test set with 1,634 images. As detailed in the paper, we also checked the overlap between our dataset, TextOCR, and Intel OCR (both of which also extracted annotated images from Open Images), making sure that the test images in the HierText dataset were not also included in the TextOCR or Intel OCR training and validation splits and vice versa. Below, we visualize examples using the HierText dataset and demonstrate the concept of hierarchical text by shading each text entity with different colors
- HierText is currently the most dense publicly available OCR dataset. Below we summarize the characteristics of HierText in comparison with other OCR datasets. HierText identifies 103.8 words per image on average, which is more than 3x the density of TextOCR and 25x more dense than ICDAR-2015. This high density poses unique challenges for detection and recognition, and as a consequence HierText is used as one of

Dataset	**Training split**	**Validation split**	**Testing split**	**Words per image**
ICDAR-2015	1,000	0	500	4.4
TextOCR	21,778	3,124	3,232	32.1
Intel OCR	19,1059	16,731	0	10.0
HierText	8,281	1,724	1,634	103.8

the primary datasets for OCR research at Google.

- We also find that text in the HierText dataset has a much more even spatial distribution than other OCR datasets, including TextOCR, Intel OCR, IC19 MLT, COCO-Text and IC19 LSVT. These previous datasets tend to have well-composed images, where text is placed in the middle of the images, and are thus easier to identify. On the contrary, text entities in HierText are broadly distributed across the images. It's proof that our images are from more diverse domains. This characteristic makes HierText uniquely challenging among public OCR datasets.



Spatial distribution of text instances in different datasets.

- **Glyph embedding :**

Glyph embedding is a concept in natural language processing (NLP) and computer vision that integrates the visual or graphical features of characters (glyphs) into machine learning models. This approach is especially relevant in languages where visual representation plays a crucial role, such as Chinese, Japanese, or Korean, where characters carry semantic or phonetic information visually.

What Are Glyphs?

Glyphs are the visual symbols or shapes representing a character in a specific typeface or script. In the context of text processing, glyph embeddings capture these visual structures, often incorporating them into models alongside traditional textual features.

Why Use Glyph Embeddings?

1. **Rich Semantic Information**: Languages like Chinese have ideographic characters where the structure of the glyph carries meaning.
2. **Handle Rare Characters**: Glyph embeddings can help deal with rare or unseen characters by using their visual similarity to known characters.
3. **Improved Multimodal Models**: They bridge the gap between vision and text models by incorporating visual information.

How Glyph Embeddings Are Created

1. **Feature Extraction from Glyphs**:
   - **Bitmap Representation**: Convert glyphs into pixelated bitmaps and use convolutional neural networks (CNNs) to extract visual features.

    - **Vector Representation**: Use scalable vector graphics (SVG) to represent glyphs, focusing on their geometric properties.
2. **Fusion with Textual Embeddings**:
    - Combine glyph features with textual embeddings (e.g., Word2Vec, GloVe, or Transformer embeddings).
    - Use concatenation, attention mechanisms, or specialized layers to integrate the information.

What is a Bitmap?

A bitmap is a type of digital image format that represents a picture as a grid of pixels. Each pixel in the grid corresponds to a specific color or intensity value, forming the complete image.

How Bitmaps Work for Glyphs

When used for glyphs:

1. **Glyph Conversion to Bitmap**:
    - Each glyph is rendered as an image, where the shape of the character is drawn onto a grid of pixels.
    - The glyph is typically black (foreground) on a white (background) canvas.
2. **Pixel Representation**:
    - Each pixel is assigned a value. For monochrome (black-and-white) bitmaps, this might be binary (e.g., 1 for black, 0 for white).
    - For grayscale, each pixel has an intensity value (e.g., 0 to 255).
    - In color bitmaps, each pixel may have multiple channels (e.g., RGB values).
3. **Resolution**:
    - The grid's resolution (e.g., 28x28 pixels for small glyphs or higher for detailed representations) determines the image's quality and the level of detail captured for the glyph.

Using Bitmaps for Feature Extraction

- **Input to Convolutional Neural Networks (CNNs)**:
    - CNNs take the bitmap as input, treating the pixel values as features.
    - The CNN applies filters (kernels) to the pixel grid, detecting edges, shapes, and other patterns in the glyph.
    - This process helps in understanding the visual structure of the glyph and creating meaningful embeddings.
- **Advantages of Bitmap Representation for Glyphs**:
    - Simple and direct way to encode visual information.
    - Effective for capturing the spatial structure and details of the glyph.
    - Works well with modern machine learning models, particularly CNNs.

Applications of Glyph Embedding

1. **Language Modeling**: Improved understanding of logographic languages in NLP tasks like machine translation and sentiment analysis.
2. **Handwriting Recognition**: Extracting features from glyphs enhances the performance of models trained to recognize handwriting or printed text.
3. **OCR (Optical Character Recognition)**: Glyph embeddings improve recognition accuracy for languages with complex scripts.
4. **Cross-Language NLP**: Helps bridge languages that share script features, such as Kanji in Japanese and Chinese characters.

Examples of Models Using Glyph Embedding

- **Glyph-aware Transformers**: Extensions of Transformer architectures that incorporate glyph embeddings into their layers for improved understanding of visually complex scripts.
- **Hybrid CNN-LSTM Models**: Convolutional layers extract glyph features, and LSTM layers process sequential text information.

-Converting images into digital formats has become increasingly practical for preserving ancient documents (Avadesh and Goyal, 2018; Narang et al., 2020), understanding real-time road signs (Wu et al., 2005; Kim, 2020), multimodal information extraction (Liu et al., 2019; Sun et al., 2021) and evaluating text-guided image generation (Petsiuk et al., 2022; Zhang et al., 2023; Rodríguez et al., 2023), just to name a few. These digital representations can be further processed using language models to extract underlying features. A high-quality OCR model can benefit various domains. However, the performance of optical character recognition (OCR) is often limited due to factors such as image quality or inherent model limitations.

- Glyph embedding captures the visual characteristics of the characters, rather than solely focusing on semantic aspects like most conventional embeddings

-In the initial phase of our study, we employ the extensively researched and widely employed ICDAR 2013 dataset as our primary data source for OCR tasks. The results obtained from this phase will subsequently serve as inputs for the next step, where we evaluate the performance of our postOCR correction model. Additionally, to facilitate training of the glyph embedding in our postOCR correction model, we rely on the Chars74K dataset .

- **Chars74k dataset :**

Dataset Structure

The Chars74K dataset is organized into three primary components:

1. English Characters

- Total Images: Approximately 64,000.
- Categories: 62 classes (26 uppercase letters, 26 lowercase letters, and 10 digits).
- Sources:
  - Handwritten: Characters written by multiple individuals.
  - Printed: Characters rendered in a variety of computer fonts.
  - Natural Scenes: Photographs of characters captured in real-world scenarios (e.g., signboards, license plates).

2. Kannada Characters

- Total Images: Around 7,000.
- Categories: 657 classes representing the Kannada script.
- Sources:
  - Focuses on handwritten characters written by multiple contributors.

3. Tamil Characters

- A smaller subset focusing on Tamil script characters, also handwritten.

Challenges in the Dataset

1. Variations in Appearance:
   - Different fonts, colors, and sizes of characters.
   - Handwritten characters vary significantly based on the writer's style.
2. Natural Scene Challenges:
   - Characters in natural scenes are often distorted, rotated, or affected by lighting conditions, occlusion, and noise.
3. Multilingual Complexity:
   - Recognizing characters in different scripts like Kannada and Tamil requires specialized approaches.

Dataset Organization

The dataset is organized into folders, typically grouped by class (e.g., folders for 'A', 'B', '1', '2', etc.), and each folder contains images corresponding to that character.

File Details:

- Format: Images are provided in various formats, including .jpg.

- Image Size: The images vary in size and resolution.
- Labels: Each image is labeled with its corresponding character for supervised learning tasks.

Applications

The Chars74K dataset is commonly used for:

1. Character Recognition:
   - Developing models for OCR, particularly for recognizing handwritten and scene-text characters.
2. Scene Text Recognition:
   - Training models to detect and classify text in natural images, such as street signs or advertisements.
3. Font and Style Analysis:
   - Analyzing variations in character shapes across different fonts and writing styles.
4. Multilingual OCR:
   - Recognizing characters from Kannada and Tamil scripts.

Machine Learning Approaches

To use Chars74K for character recognition tasks, various deep learning and traditional approaches can be applied:

1. Traditional Methods:
   - Feature extraction (e.g., SIFT, HOG) followed by classification with SVMs or Random Forests.
2. Deep Learning Models:
   - Convolutional Neural Networks (CNNs): Effective for image-based character classification.
   - Recurrent Neural Networks (RNNs): Useful for sequence modeling, especially for handwritten data.
   - End-to-End Scene Text Recognition Models: Combine object detection and classification for natural scene data.

Limitations

1. Imbalanced Classes:
   - Some characters or styles might have fewer samples than others, making the dataset slightly unbalanced.
2. Dataset Size:

        ○ While large, it is smaller compared to modern large-scale datasets used for deep learning tasks, such as ImageNet or COCO.
3. Scene Text Variety:
    ○ Scene text samples are limited and may not cover all real-world complexities.

- The ICDAR 2013 dataset and the recently introduced ICDAR 2023 dataset serve as prominent benchmarks for evaluating Optical Character Recognition (OCR) models. The ICDAR 2013 dataset encompasses a diverse collection of document images, encompassing both printed and handwritten text captured under a variety of conditions, including different languages, font styles, sizes, and distortions. It serves as a robust benchmark for assessing the accuracy and robustness of OCR systems in a wide array of real world scenarios. In contrast, the ICDAR 2023 dataset expands upon the achievements of its predecessor by introducing additional challenges that push the boundaries of OCR technology. It incorporates more intricate document layouts, degraded text, low resolution images, and challenging background clutter, with the goal of evaluating OCR model performance in increasingly demanding scenarios
- Both datasets will undergo evaluations using three models: EasyOCR2 , PaddleOCR3 , and TrOCR (Transformer-based OCR) .The EasyOCR model will be utilized specifically for single word detection boxes, while the other two models will provide single line detection box functionality. Consequently, the outputs from EasyOCR will consist solely of single words, whereas PaddleOCR will generate output in the form of sentences.
- The Chars74K dataset is employed for training the glyph embedding. This dataset encompasses scene images of English and Kannada characters, although for this particular experiment, only English alphabets are utilized. In addition to the Chars74K dataset, we capture screenshots of Korean and Hebrew characters to serve as samples for the garbage class in open-set classification. For training the post-OCR correction model, the data consists of the outputs obtained from EasyOCR, PaddleOCR, and TrOCR in the initial phase

- SOTA(STATE OF THE ART MODELS) OCR models :

1. EasyOCR

EasyOCR is an open-source OCR library that combines efficiency and accuracy, widely used for extracting text from images. It relies on two key components:

1. Text Detection: Uses the CRAFT algorithm (Character Region Awareness for Text detection) by Baek et al., 2019.

2. Text Recognition: Based on the CRNN architecture (Convolutional Recurrent Neural Network) proposed by Shi et al., 2016.

1.1 Detection: CRAFT Algorithm

- CRAFT is a character-level text detector.
- It focuses on detecting individual characters instead of entire words or lines, which allows it to better handle irregularly shaped text.
- Steps:
    1. Region Score Map: Predicts the likelihood of text regions in the image.
    2. Affinity Score Map: Links detected characters to form words.
    3. Text Region Grouping: Groups characters into coherent text regions or words.

1.2 Recognition: CRNN

The CRNN architecture combines convolutional neural networks (CNNs) for feature extraction and recurrent neural networks (RNNs) for sequence modeling. Here's how it works:

1. Feature Extraction:
    - A ResNet or VGG backbone extracts robust visual features from the image.
    - This reduces the input to a sequence of feature vectors.
2. Sequence Labeling:
    - An LSTM (Long Short-Term Memory) processes the sequential data and models relationships between characters.
3. Decoding:
    - Uses CTC (Connectionist Temporal Classification) loss for decoding the sequence into readable text without requiring explicit character segmentation.

---

2. PaddleOCR

PaddleOCR is an OCR framework built on the PaddlePaddle deep learning platform. It offers robust text detection and recognition, particularly for multilingual and scene text.

2.1 Detection: PP-OCRv3

- PP-OCRv3 is the latest iteration of PaddleOCR's text detection model.
- It combines improvements in:
    1. Backbone: Uses an efficient lightweight backbone for better performance and lower computational cost.

2. Detection Head: Implements a text detection algorithm similar to EAST (Efficient and Accurate Scene Text Detector) or DBNet, focusing on text regions.
3. Optimization: Includes post-processing optimizations to refine detected bounding boxes.

2.2 Recognition

PaddleOCR's text recognition pipeline includes:

1. Lightweight CRNN:
    - Extracts features using a convolutional backbone (e.g., MobileNet) and processes them using a recurrent network (GRU or LSTM).
    - Uses CTC loss for sequence-to-sequence prediction.
2. Transformer-Based Models (Advanced Option):
    - Offers transformer-based models for recognition tasks requiring high accuracy and multilingual capabilities.

2.3 Features

- Multilingual Support: Supports over 80 languages.
- Scene Text Recognition: Handles curved, rotated, or noisy text with ease.
- End-to-End OCR: Integrates detection and recognition in a streamlined workflow.

---

3. TrOCR

TrOCR (Transformer-based OCR) is a cutting-edge OCR model developed by Microsoft. It is entirely based on transformers, following the success of models like BERT and GPT.

3.1 Transformer Architecture

1. Encoder:
    - Extracts visual features from the input image using a Vision Transformer (ViT) or CNN-based backbone.
    - Encodes the image into a sequence of embeddings.
2. Decoder:
    - A transformer decoder predicts text in an auto-regressive fashion, similar to how language models like GPT generate text.
    - Each output token is conditioned on the previous ones, enabling TrOCR to capture contextual relationships between characters.

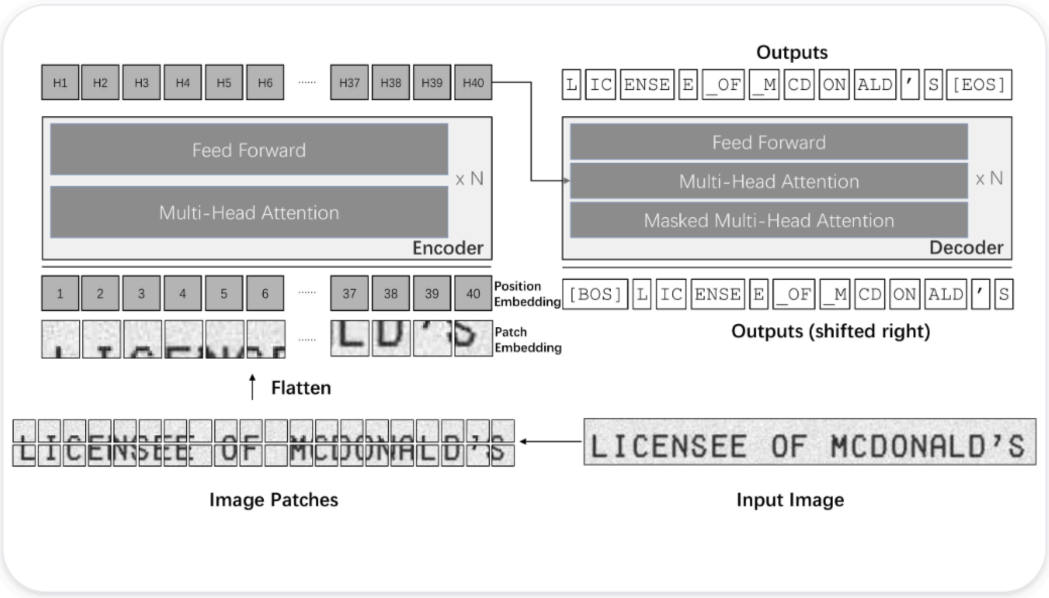
3.2 Key Advantages

- Contextual Understanding:

    - Unlike traditional CNN-based OCR models, transformers excel at modeling global context, which is useful for recognizing text in noisy or complex environments.
- Pretrained Language Models:
    - TrOCR can be fine-tuned using pre-trained language models, enhancing its ability to understand linguistic patterns and semantics.
- End-to-End Training:
    - Combines image-to-text prediction in a unified framework, simplifying the training process.

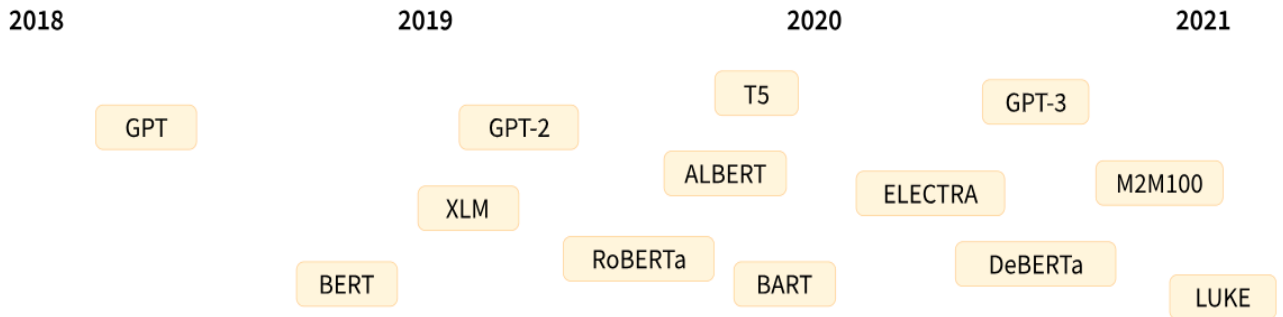
3.3 Applications

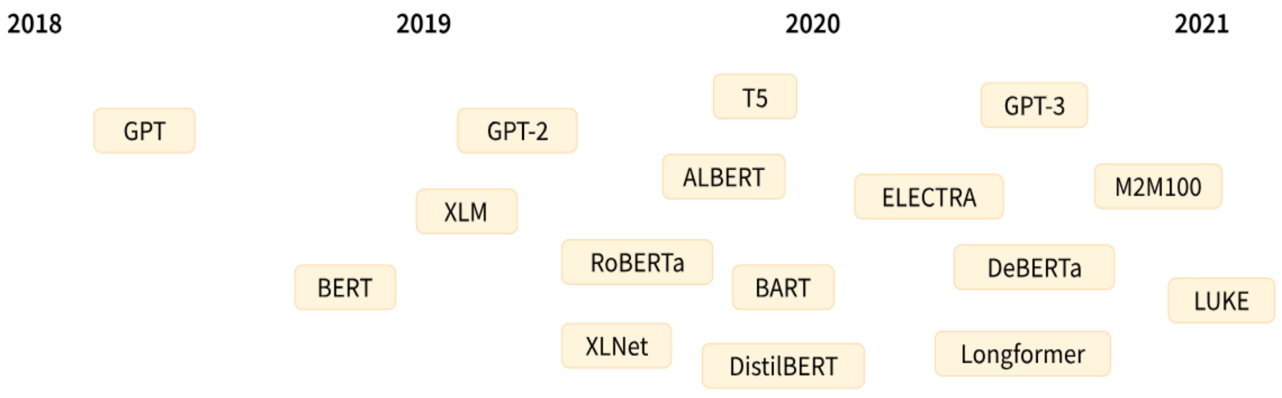
- Scene Text Recognition: Recognizes text in real-world images (e.g., street signs, billboards).
- Handwritten Text Recognition: Performs exceptionally well on handwritten datasets, such as I AM or CVL.



TrOCR architecture. Taken from the original paper.

- Transformers :





The Transformer architecture was introduced in June 2017. The focus of the original research was on translation tasks. This was followed by the introduction of several influential models, including:

- June 2018: GPT, the first pretrained Transformer model, used for fine-tuning on various NLP tasks and obtained state-of-the-art results
- October 2018: BERT, another large pretrained model, this one designed to produce better summaries of sentences
- February 2019: GPT-2, an improved (and bigger) version of GPT that was not immediately publicly released due to ethical concerns
- October 2019: DistilBERT, a distilled version of BERT that is 60% faster, 40% lighter in memory, and still retains 97% of BERT's performance
- October 2019: BART and T5, two large pretrained models using the same architecture as the original Transformer model (the first to do so)
- May 2020, GPT-3, an even bigger version of GPT-2 that is able to perform well on a variety of tasks without the need for fine-tuning (called *zero-shot learning*)

This list is far from comprehensive, and is just meant to highlight a few of the different kinds of Transformer models. Broadly, they can be grouped into three categories:

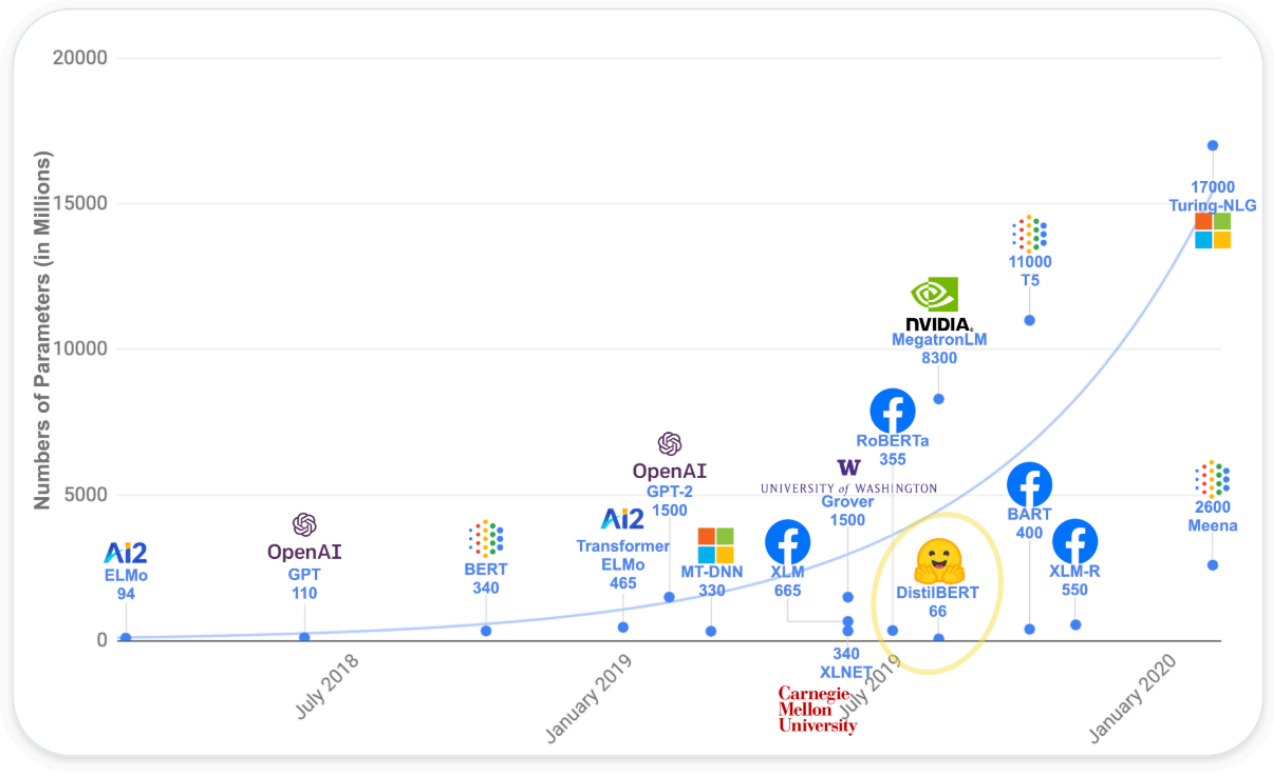
- GPT-like (also called *auto-regressive* Transformer models)
- BERT-like (also called *auto-encoding* Transformer models)
- BART/T5-like (also called *sequence-to-sequence* Transformer models)

-All the Transformer models mentioned above (GPT, BERT, BART, T5, etc.) have been trained as *language models*. This means they have been trained on large amounts of raw text in a self-supervised fashion. Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model. That means that humans are not needed to label the data!

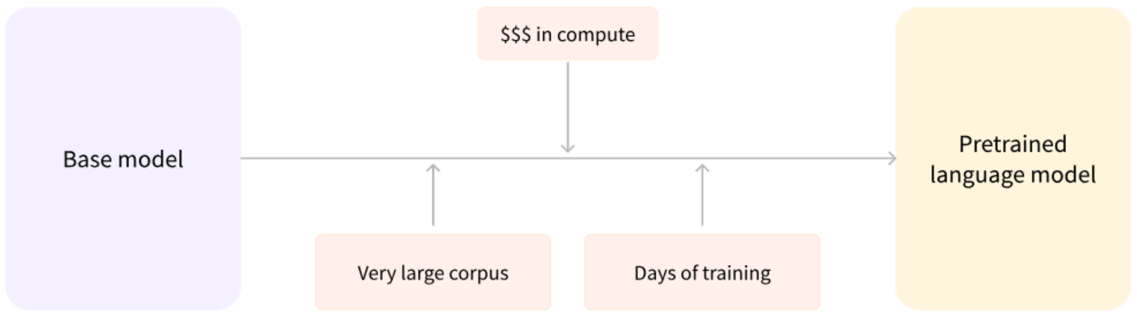
This type of model develops a statistical understanding of the language it has been trained on, but it's not very useful for specific practical tasks. Because of this, the general pretrained model then goes through a process called *transfer learning*. During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

An example of a task is predicting the next word in a sentence having read the *n* previous words. This is called *causal language modeling* because the output depends on the past and present inputs, but not the future ones.

--*Pretraining* is the act of training a model from scratch: the weights are



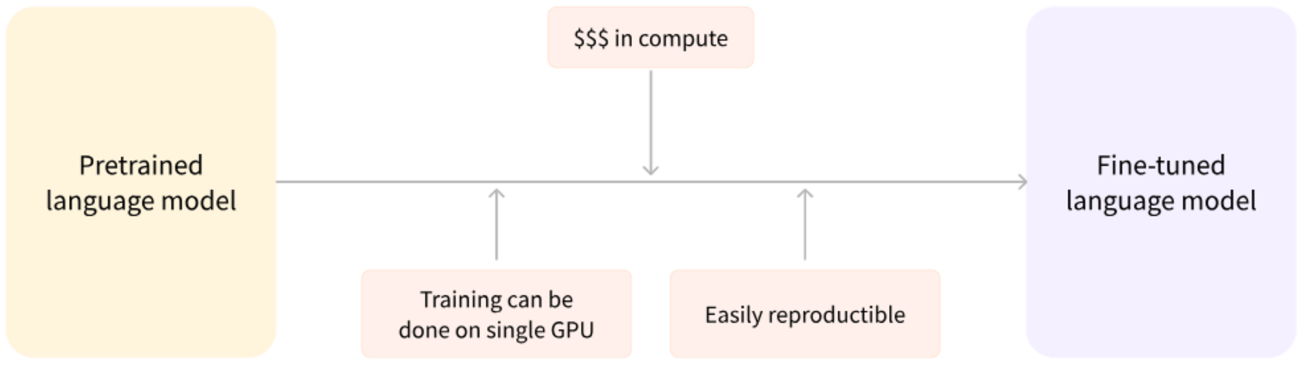
randomly initialized, and the training starts without any prior knowledge.



--*Fine-tuning,* on the other hand, is the training done after a model has been pretrained. To perform fine-tuning, you first acquire a pre-trained language model, then perform additional training with a dataset specific to your task. Wait — why not simply train the model for your final use case from the start (scratch)? There are a couple of reasons:

- The pretrained model was already trained on a dataset that has some similarities with the fine-tuning dataset. The fine-tuning process is thus able to take advantage of knowledge acquired by the initial model during pretraining (for instance, with NLP problems, the pretrained model will have some kind of statistical understanding of the language you are using for your task).
- Since the pretrained model was already trained on lots of data, the fine-tuning requires way less data to get decent results.
- For the same reason, the amount of time and resources needed to get good results are much lower.

For example, one could leverage a pretrained model trained on the English language and then fine-tune it on an arXiv corpus, resulting in a science/research-based model. The fine-tuning will only require a limited amount of data: the knowledge the pretrained model has acquired is “transferred,” hence the term *transfer learning*.


$$$ in compute
Pretrained language model
Fine-tuned language model
Training can be done on single GPU
Easily reproductible

-Each of these parts can be used independently, depending on the task:

- Encoder-only models: Good for tasks that require understanding of the input, such as sentence classification and named entity recognition.
- Decoder-only models: Good for generative tasks such as text generation.
- Encoder-decoder models or sequence-to-sequence models: Good for generative tasks that require an input, such as translation or summarization.

Attention layers

-This layer will tell the model to pay specific attention to certain words in the sentence you passed it (and more or less ignore the others) when dealing with the representation of each word.To put this into context, consider the task of translating text from English to French. Given the input "You like this course", a translation model will need to also attend to the adjacent word "You" to get the proper translation for the word "like", because in French the verb "like" is conjugated differently depending on the subject. The rest of the sentence, however, is not useful for the translation of that word. In the same vein, when translating "this" the model will also need to pay attention to the word "course", because "this" translates differently depending on whether the associated noun is masculine or feminine. Again, the other words in the sentence will not matter for the translation of "course". With more complex sentences (and more complex grammar rules), the model would need to pay special attention to words that might appear farther away in the sentence to properly translate each word.

The same concept applies to any task associated with natural language: a word by itself has a meaning, but that meaning is deeply affected by the context, which can be any other word (or words) before or after the word being studied.

-To speed things up during training (when the model has access to target sentences), the decoder is fed the whole target, but it is not allowed to use future words (if it had access to the word at position 2 when trying to predict the word at position 2, the problem would not be very hard!). For instance, when trying to predict the fourth word, the attention layer will only have access to the words in positions 1 to 3.

- As we dive into Transformer models in this course, you'll see mentions of *architectures* and *checkpoints* as well as *models*. These terms all have slightly different meanings:

- Architecture: This is the skeleton of the model — the definition of each layer and each operation that happens within the model.
- Checkpoints: These are the weights that will be loaded in a given architecture.

  For example, BERT is an architecture while bert-base-cased, a set of weights trained by the Google team for the first release of BERT, is a checkpoint.

  - Encoder Models :

Encoder models use only the encoder of a Transformer model. At each stage, the attention layers can access all the words in the initial sentence. These models are often characterized as having "bi-directional" attention, and are often called *auto-encoding models*.

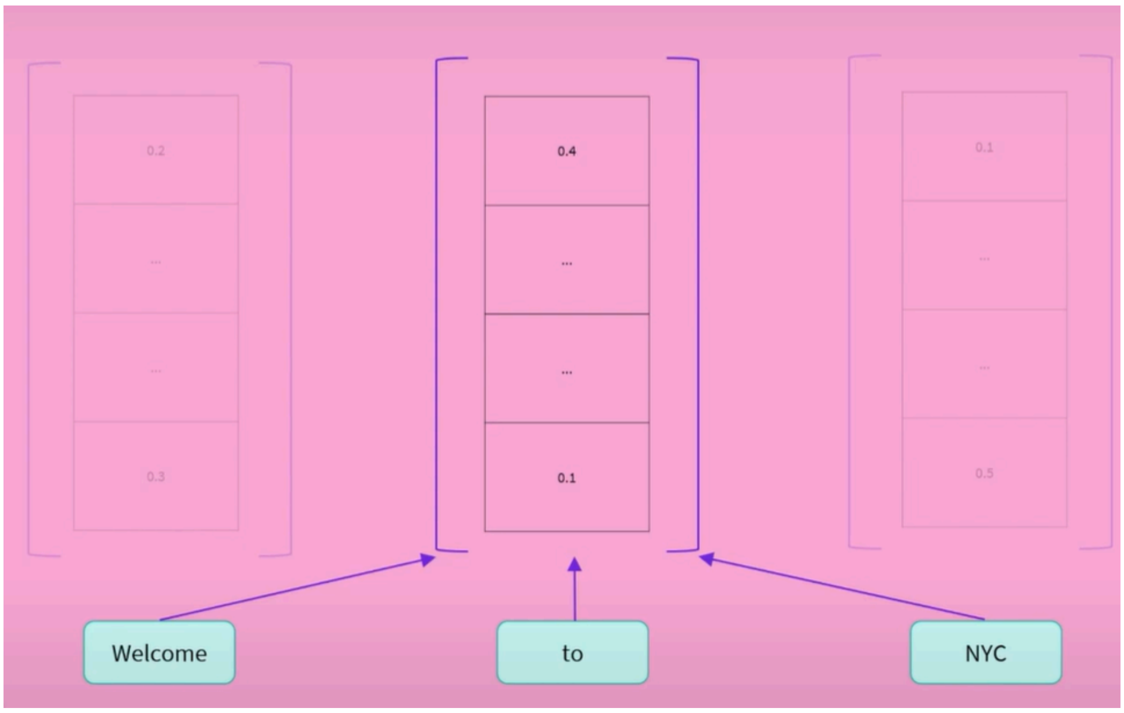
The pretraining of these models usually revolves around somehow corrupting a given sentence (for instance, by masking random words in it) and tasking the model with finding or reconstructing the initial sentence.

Encoder models are best suited for tasks requiring an understanding of the full sentence, such as sentence classification, named entity recognition (and more generally word classification), and extractive question answering.

Representatives of this family of models include:

- ALBERT
- BERT
- DistilBERT
- ELECTRA
- RoBERTa

- Decoder :

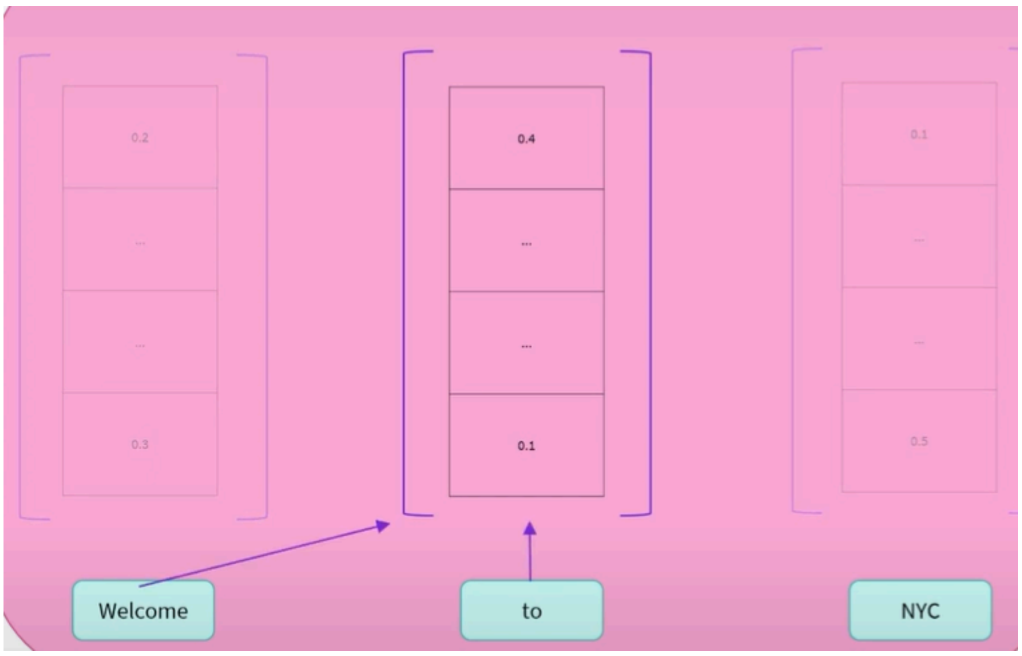


Decoder models use only the decoder of a Transformer model. At each stage, for a given word the attention layers can only access the words positioned before it in the sentence. These models are often called *auto-regressive models*.

The pretraining of decoder models usually revolves around predicting the next word in the sentence.

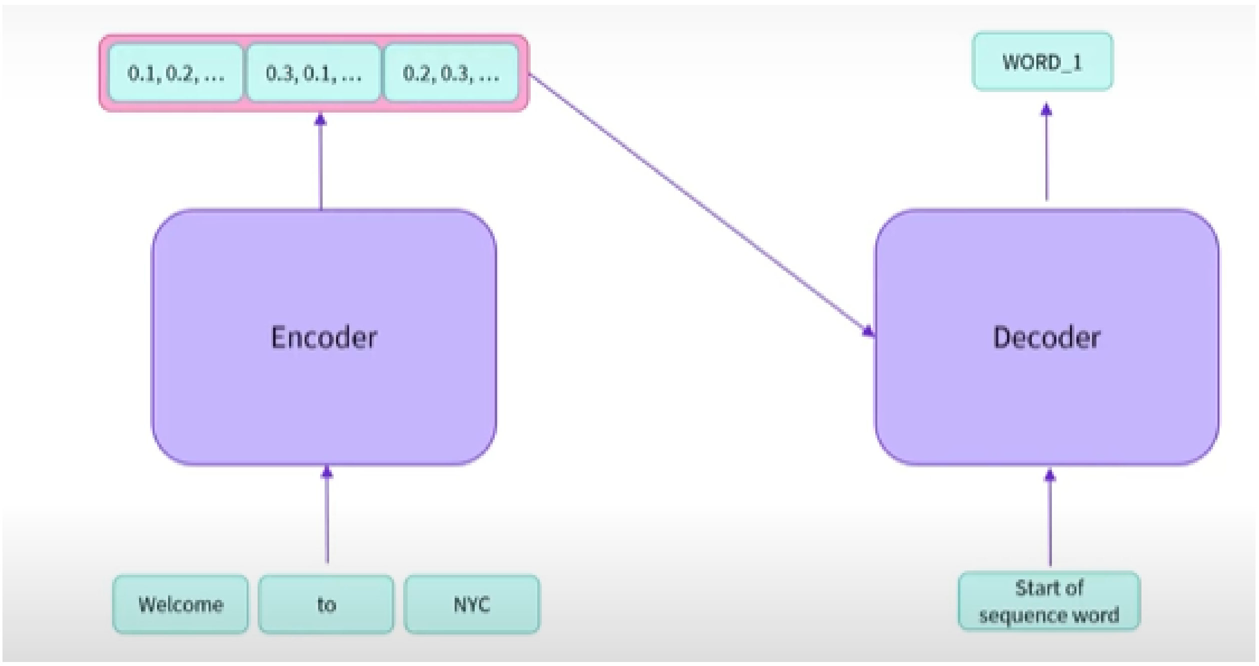
These models are best suited for tasks involving text generation.

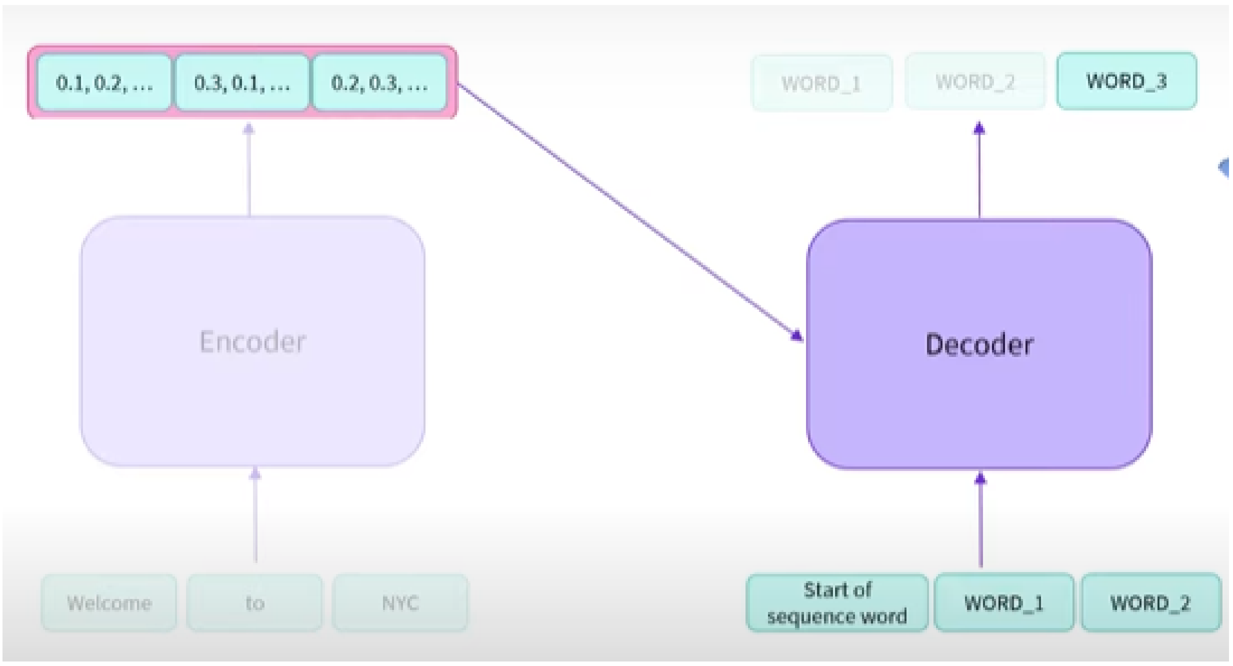
Representatives of this family of models include:

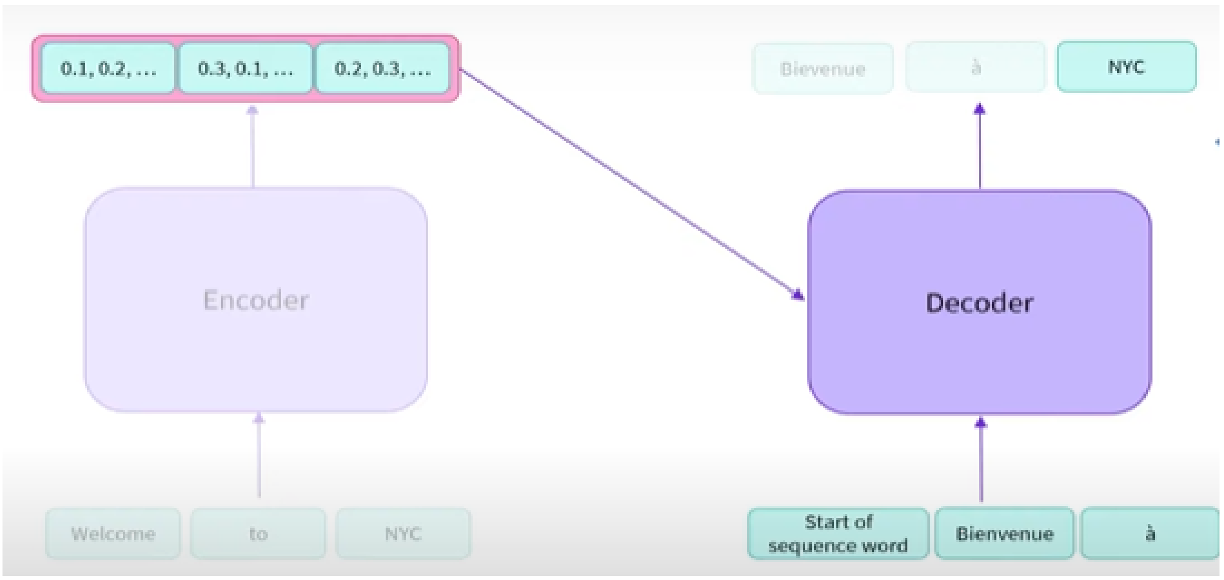


- CTRL
- GPT
- GPT-2
- Transformer XL

- Encoder Decoder :


0.1, 0.2, …
0.3, 0.1, …
0.2, 0.3, …
WORD_1
Encoder
Decoder
Welcome
to
NYC
Start of sequence word


0.1, 0.2, …
0.3, 0.1, …
0.2, 0.3, …
WORD_1
WORD_2
WORD_3
Encoder
Decoder
Welcome
to
NYC
Start of sequence word
WORD_1
WORD_2


0.1, 0.2, …
0.3, 0.1, …
0.2, 0.3, …
Bievenue
à
NYC
Encoder
Decoder
Welcome
to
NYC
Start of sequence word
Bienvenue
à

Encoder-decoder models (also called *sequence-to-sequence models*) use both parts of the Transformer architecture. At each stage, the attention layers of the encoder can access all the words in the initial sentence, whereas the attention layers of the decoder can only access the words positioned before a given word in the input.

The pretraining of these models can be done using the objectives of encoder or decoder models, but usually involves something a bit more complex. For instance, T5 is pre-trained by replacing random spans of text (that can contain several words) with a single mask special word, and the objective is then to predict the text that this mask word replaces.

Sequence-to-sequence models are best suited for tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.

Representatives of this family of models include:

- BART
- mBART
- Marian
- T5

-CharBERT: Character-aware Pre-trained Language Model

Most pre-trained language models (PLMs) construct word representations at subword level with Byte-Pair Encoding (BPE) or its variations, by which OOV (out-of-vocab) words are almost avoidable. However, those methods split a word into subword units and make the representation incomplete and fragile.

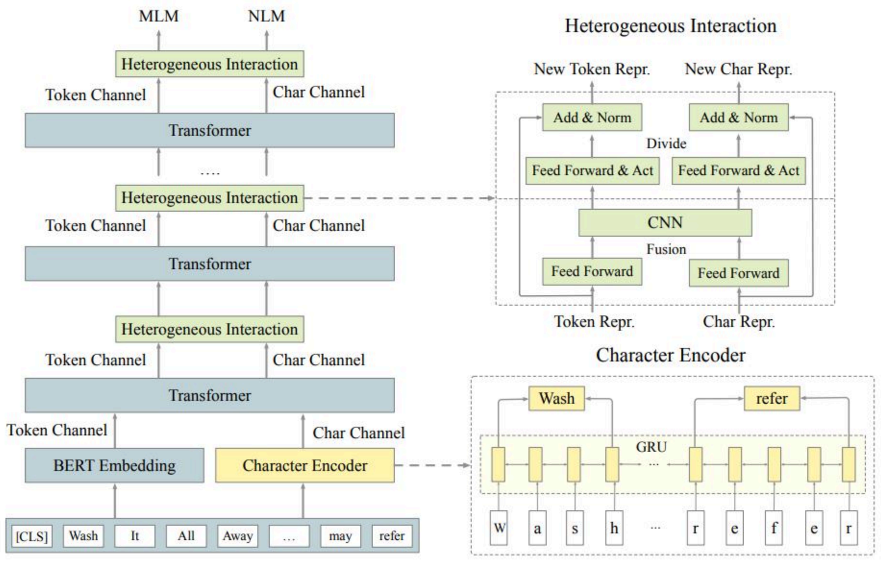


Figure 2: The neural architecture of CharBERT. The left part is the main structure of CharBERT based on the original pre-trained models like BERT. The modules in the right part are the heart of CharBERT: character encoder and heterogeneous interaction. *(best viewed in color)*

-CharBERT is a model architecture designed to enhance traditional text embeddings by incorporating character-level information into word-level embeddings, improving performance on various natural language processing (NLP) tasks. It extends the **BERT (Bidirectional Encoder Representations from Transformers)** model by integrating character-level features alongside token embeddings, making it particularly effective for tasks involving morphologically rich languages or noisy text (e.g., typos, social media text).

1. **Character-Level Embeddings**:
    - CharBERT adds a character-level encoder to capture fine-grained subword information, improving the handling of misspellings, rare words, and unseen vocabulary.
    - This character-level encoder often uses a convolutional neural network (CNN) or recurrent neural network (RNN) to process sequences of characters.
2. **Fusion of Word and Character Representations**:
    - CharBERT fuses character-level and word-level embeddings into a unified representation.
    - This fusion ensures the model retains the high-level semantic context from BERT while benefiting from the detailed morphological information of characters.
3. **Transformer Backbone**:
    - The core of CharBERT is a standard BERT-like Transformer architecture.

- ○ The model integrates character embeddings at the input or intermediate levels to enhance the word embeddings used by the Transformer layers.

Architecture of CharBERT

1. **Character Encoder**:
   - Converts characters within a word into embeddings.
   - Uses CNNs (for local feature extraction) or LSTMs (for sequential character information).
2. **Word Embedding Fusion**:
   - Combines character-level representations with pre-trained word embeddings (e.g., BERT embeddings).
   - Techniques include concatenation, gating mechanisms, or attention layers.
3. **Transformer Layers**:
   - Applies BERT's multi-head self-attention and feed-forward layers on the fused embeddings.
4. **Output Layers**:
   - Provides outputs for downstream tasks such as text classification, named entity recognition (NER), question answering, or text generation.

Benefits of CharBERT

1. **Robustness to Noisy Text**:
   - Handles typographical errors, abbreviations, and rare words better than standard BERT.
2. **Improved Generalization**:
   - Incorporates morphological details, making it effective for languages with complex word structures.
3. **Seamless Integration**:
   - Builds upon pre-trained BERT models, leveraging their semantic understanding while adding morphological insights.

Applications

Text Classification: Improved accuracy in classifying text with noisy or domain-specific language.

Named Entity Recognition (NER): Better recognition of rare or misspelled named entities.

Question Answering: Enhanced understanding of out-of-vocabulary terms in context.

Social Media Analysis: Effective in handling informal language and non-standard spelling.

Unsupervised pre-trained language models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have achieved surprising results on multiple NLP benchmarks. These models are pre-trained over large-scale open-domain corpora to obtain general language representations and then fine-tuned for specific downstream tasks. To deal with the large vocabulary, these models use Byte-Pair Encoding (BPE) (Sennrich et al., 2016) or its variations as the encoding method. Instead of whole words, BPE performs statistical analysis of the training corpus and split the words into subword units, a hybrid between character- and word-level representation. Even though BPE can encode almost all the words in the vocabulary into WordPiece tokens without OOV words, it has two problems: 1) incomplete modeling: the subword representations may not incorporate the fine-grained character information and the representation of the whole word; 2) fragile representation: minor typos can drastically change the BPE tokens, leading to inaccurate or incomplete representations. This lack of robustness severely hinders its applicability in real-world applications. We illustrate the two problems by the example in Figure 1. For a word like backhand, we can decompose its representation at different levels by a tree with a depth of 3: the complete word at the first layer, the subwords at the second layer, and the last characters. BPE only considers representations of subwords on the second layer and misses the potentially useful information at the first and last layer. Furthermore, if there is noise or typo in the characters (e.g., missing the letter 'k'), the subwords and its number at the second layer will be changed at the same time. Models relying purely on these subword representations thus suffer from this lack of robustness

We propose a new pre-training method CharBERT (BERT can also be replaced by other pre-trained models like RoBERTa) to solve these problems. Instead of the traditional CNN layer for modeling the character information, we use the context string embedding (Akbik et al., 2018) to model the word's fine-grained representation. We use a dual-channel architecture for characters and original subwords and fuse them after each transformer block. Furthermore, we propose an unsupervised character learning task, which injects noises into characters and trains the model to denoise and restores the original word. The main advantages of our methods are: 1) character-aware: we construct word representations from characters based on the original subwords, which greatly complements the subword-based modeling. 2) robustness: we improve not only the performance but also the robustness of the pre-trained model; 3) model-agnostic: our method is agnostic to the backbone PLM like BERT and RoBERTa, so that we can adapt it to any transformer-based PLM. In summary, our contributions in this paper are: • We propose a character-aware pre-training method CharBERT, which can enrich the word representation in PLMs by incorporating features at different levels of a word • We evaluate our method on 8 benchmarks, and the results show that our method can significantly improve the performance compared to the strong BERT and RoBERTa baselines; • We construct three character attack test sets on three types of tasks. The experimental results indicate that our method can improve the robustness by a large margin

-Character Encoder :

we need to form token-level embeddings with the input sentences as sequences of characters. We first convert the sequences of tokens into characters and embed them into fixed-size vectors then apply a bidirectional GRU layer (Cho et al., 2014) to construct the contextual character embeddings

- **GRU :** The GRU (Gated Recurrent Unit) is a type of recurrent neural network (RNN) architecture used in natural language processing (NLP) tasks. In the context of CharBERT, GRUs are particularly employed to process character-level embeddings and learn sequential dependencies within textual data at the character level.

Role of the GRU in CharBERT

- **Character-level Embeddings**:
    - For each word (or subword), CharBERT generates embeddings from its characters.
    - These character embeddings are passed through a GRU layer (or sometimes a BiGRU, the bidirectional version of GRU) to capture sequential dependencies among characters.
- **Why GRU?**:
    - GRUs are computationally efficient compared to other RNNs like LSTMs while still capturing long-term dependencies.
    - GRUs are well-suited for processing sequences where memory of previous steps is crucial, as in understanding how characters contribute to forming a word or subword.

Role of the GRU in CharBERT

- **Character-level Embeddings**:
    - For each word (or subword), CharBERT generates embeddings from its characters.

- These character embeddings are passed through a GRU layer (or sometimes a BiGRU, the bidirectional version of GRU) to capture sequential dependencies among characters.

- **Why GRU?**:
  - GRUs are computationally efficient compared to other RNNs like LSTMs while still capturing long-term dependencies.
  - GRUs are well-suited for processing sequences where memory of previous steps is crucial, as in understanding how characters contribute to forming a word or subword.

- ## Transformer

The core Transformer in CharBERT remains similar to BERT but operates on the enriched embeddings.

Key Components of Each Transformer Layer:

1. **Multi-Head Self-Attention**:
   - Enables tokens to attend to other tokens in the sequence, capturing contextual relationships bidirectionally.
   - In CharBERT, the character-level enhancements in embeddings provide additional granularity, allowing the self-attention mechanism to capture morphological and subword nuances.
2. **Feed-Forward Networks (FFN)**:
   - Processes the output of the self-attention mechanism.
   - The FFN helps refine token representations by applying two dense layers with a non-linear activation (e.g., ReLU).
3. **Layer Normalization and Residual Connections**:
   - Residual connections help preserve information across layers.
   - Layer normalization stabilizes the network, making training more robust.
4. **Heterogeneous Interaction**

The embeddings from characters and the original token-channel are fed into the same transformer layers in pre-trained models. The token and char representations are fused and split by the heterogeneous interaction module after each transformer layer. In the fusion step, the two representations are transformed by different fully-connected layers. Then they are concatenated and fused by a CNN layer, which can be formulated by

$$t_i'(x) = W_1 * t_i(x) + b_1 \;\; ; \;\; h_i'(x) = W_2 * h_i(x) + b_2$$
$$w_i(x) = [t_i'(x); h_i'(x)] \;\; ; \;\; m_{j,t} = tanh(W_3^j * w_{t:t+s_j-1} + b_3^j)$$

where ti(x) is the token representations, W, b are parameters, wt:t+sj−1 refers to the concatenation of the embedding of (wt ,...,wt+sj−1), sj is the window size of jth filter, and m is the fusion representation with the dimension same with the number of filters. In the divide step, we transform the fusion representations by another fully connected layer with the GELU activation layer (Hendrycks and Gimpel, 2016). We then use the residual connection to retain the respective information from the two channels.

The left part shows the overall structure of **CharBERT**, which is based on BERT and enhanced by incorporating a **character encoder** and **heterogeneous interaction layers**.

A. Token Channel

- This is the standard BERT processing pipeline.
- Tokens are embedded using BERT's **WordPiece embeddings**, along with segment embeddings and positional embeddings.
- The token embeddings are fed into a stack of **Transformer layers** for contextualization, just like in vanilla BERT.

B. Character Channel

- Each token (or subword) is further broken down into its individual **characters**.
- These characters are processed by the **Character Encoder** to generate a **character-level representation** for each token.
- The character representations are then fed into the **heterogeneous interaction** layers, where they interact with token-level features.

C. Heterogeneous Interaction

- After each Transformer layer, the token and character representations are updated through the **heterogeneous interaction mechanism**.
- This mechanism allows the model to fuse character-level and token-level information iteratively.
- The updated representations are passed back to the Transformer layers for further processing.

Final Outputs:

- The final token and character representations are used for downstream tasks like Masked Language Modeling (MLM) or Next Sentence Prediction (NSP), or fine-tuning for specific NLP tasks.

Final Outputs of the Token Channel

- The token channel goes through multiple layers of Transformers, where token representations are continuously refined by integrating context from

surrounding tokens (via self-attention) and from character-level representations (via heterogeneous interaction).
- After the last Transformer layer, the token channel produces **contextualized token embeddings**:
  - These embeddings capture the meaning of each token (subword) in the context of the entire sentence.
  - They are the core output used for downstream NLP tasks.

---

2. Final Outputs of the Character Channel

- The character channel processes the token's character-level representations through GRU layers and heterogeneous interaction mechanisms.
- The character embeddings are updated at every layer and fused with token embeddings in the heterogeneous interaction modules.
- By the end, the character channel outputs **contextualized character representations**:
  - These embeddings capture subword-level morphological features and are particularly helpful for handling typos, rare words, or morphologically complex languages.

---

3. Combined Outputs: Task-Specific Representations

The final outputs depend on the specific task for which CharBERT is being used. The token and character representations can either be used individually or fused further for the task. Below are common use cases:

A. Masked Language Modeling (MLM)

- MLM is a pre training task where random tokens in the input sequence are masked (e.g., replaced with [MASK]), and the model predicts the original tokens.
- Final **token embeddings** from the last Transformer layer are passed through a softmax classifier to predict the masked tokens.
- Character-level features help make better predictions for rare or misspelled tokens.

B. Next Sentence Prediction (NSP)

- NSP is another pretraining task where the model predicts whether one sentence follows another in the corpus.
- The final embedding of the [CLS] token (a special token added to the input) is used as a **sentence-level representation**.
- This [CLS] embedding is passed through a classifier to predict whether the second sentence is the next sentence.

C. Fine-Tuning for Downstream Tasks

When CharBERT is fine-tuned for specific NLP tasks, the final outputs are adapted as follows:

1. **Sequence Classification (e.g., Sentiment Analysis)**
   - The final embedding of the [CLS] token is passed to a classifier.
   - The [CLS] token summarizes the information from both token and character channels, making it suitable for classification tasks.
2. **Named Entity Recognition (NER)**
   - The final embeddings of each token (from the token channel) are used to classify each token as belonging to an entity type (e.g., person, organization, location) or not.
   - Character-level information is particularly useful for identifying named entities with uncommon spellings.
3. **Question Answering (QA)**
   - For tasks like SQuAD, the final embeddings of all tokens are used to predict:
     - The **start position** and **end position** of the answer span within the input text.
   - Character-level features help improve predictions for text with typos or uncommon words.
4. **Part-of-Speech Tagging (POS)**
   - The final embeddings of each token are classified into their respective part-of-speech categories.
   - Morphological patterns captured by the character channel are particularly beneficial.

---

4. Fusion of Token and Character Representations

The heterogeneous interaction ensures that the final outputs of the token and character channels are **highly synergistic**:

- **Token embeddings** carry semantic and contextual information.
- **Character embeddings** carry subword-level morphological and spelling-related information.
- Together, they provide a rich representation for handling a wide range of NLP tasks.

Summary of Outputs

- **Token Outputs**: Contextualized token representations for all tokens in the sequence, derived from BERT-style processing.
- **Character Outputs**: Contextualized character-level representations, enriched by GRU and CNN processing.

- **Task-Specific Outputs**:
    - [CLS] embedding: Used for classification tasks (e.g., sentiment analysis, NSP).
    - Token embeddings: Used for token-level tasks (e.g., NER, POS tagging, QA).

Role of the [CLS] Token in CharBERT

- The [CLS] token is inserted as the very first token in the input sequence during training or inference.
- Its primary purpose is to aggregate information about the entire input sequence, including all tokens and their relationships.
- As the sequence passes through the layers of the Transformer, the [CLS] token's embedding is continuously updated to encode contextualized information from the entire sequence.

[CLS] Sentence A [SEP] Sentence B [SEP]

[CLS]: Marks the start of the sequence.
[SEP]: Separates Sentence A from Sentence B.

-The paper presented at ICDAR 2023 is titled "TrOCR Meets Language Models: An End-to-End Post-correction Approach."

TrOCR, a Transformer-based Optical Character Recognition model, was a subject of interest at the 17th International Conference on Document Analysis and Recognition (ICDAR 2023), held from August 21-26, 2023, in San José, California. ICDAR 2023

One notable paper presented at the conference was "TrOCR Meets Language Models: An End-to-End Post-correction Approach."ACM Digital Library

This study aimed to enhance handwritten text recognition performance and domain adaptability by integrating an OCR model with a language model serving as a corrector. The integration addressed challenges in recognizing handwritten text by combining optical character recognition with language modeling techniques.

Additionally, the ICDAR 2023 Competition on Reading the Seal Title featured methods employing Transformer-based networks for seal text recognition. One approach utilized a Vision Transformer encoder and Transformer decoder, leveraging the Transformer's global self-attention mechanism to improve performance on curved seal text.RRC

These contributions highlight the ongoing research and application of Transformer-based models like TrOCR in advancing OCR technologies, as showcased at ICDAR 2023.

Link : https://icdar2023.org/program/accepted-papers/

- Reading seal title text is a challenging task due to the variable shapes of seals, curved text, background noise, and overlapped text. However, this important element is commonly found in official and financial scenarios, and has not received the attention it deserves in the field of OCR technology. To promote research in this area, we organized ICDAR 2023 competition on reading the seal title (ReST), which included two tasks: seal title text detection (Task 1) and end-to-end seal title recognition (Task 2). We constructed a dataset of 10,000 real seal data, covering the most common classes of seals, and labeled all seal title texts with text polygons and text contents. The competition opened on 30th December, 2022 and closed on 20th March, 2023. The competition attracted 53 participants and received 135 submissions from academia and industry, including 28 participants and 72 submissions for Task 1, and 25 participants and 63 submissions for Task 2, which demonstrated significant interest in this challenging task. In this report, we present an overview of the competition, including the organization, challenges, and results. We describe the dataset and tasks, and summarize the submissions and evaluation results. The results show that significant progress has been made in the field of seal title text reading, and we hope that this competition will inspire further research and development in this important area of OCR technology . Based on the flourishing of deep learning methods, we have witnessed the maturity of regular and general OCR technology, including scene text detection and recognition. However, as a common element which can be seen everywhere in official and financial scenarios, seal title text has not gained its attention. And the task of reading seal title text is also faced with many challenges, such as variable shapes of seal (for example, circle, ellipse, triangle and rectangle), curved text background noise and overlapped text, as shown in Figure 1 - 3. In order to promote the research of seal text, we propose a competition on reading the seal title. Considering there are no existing datasets for seal title text reading. We construct a dataset including 10,000 real seal data, which covers the most common classes of seal. In the dataset, all seal title texts are labeled with text polygons and text contents. Besides, two tasks are presented for this competition: (1) Seal title text detection; (2) End-to-end seal title recognition. We hope that the dataset and tasks could greatly promote the research in seal text reading

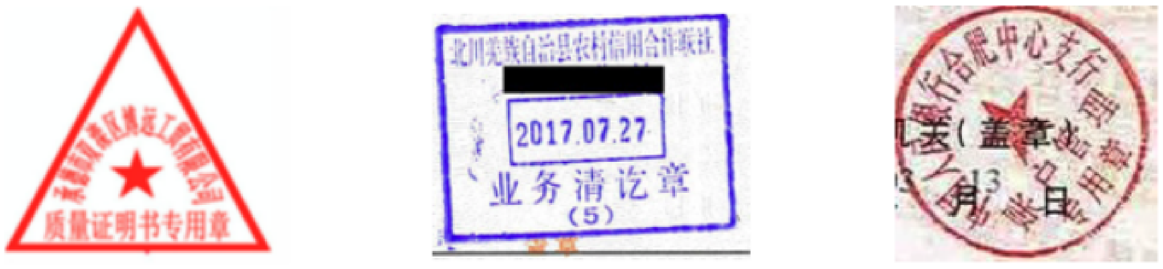


Fig. 1. Different shapes of seals samples in the ReST.

<u>Task 1: Seal Title Text detection</u>

The aim of this task is to localize the title text in seal image, The input examples are shown in Figure 4

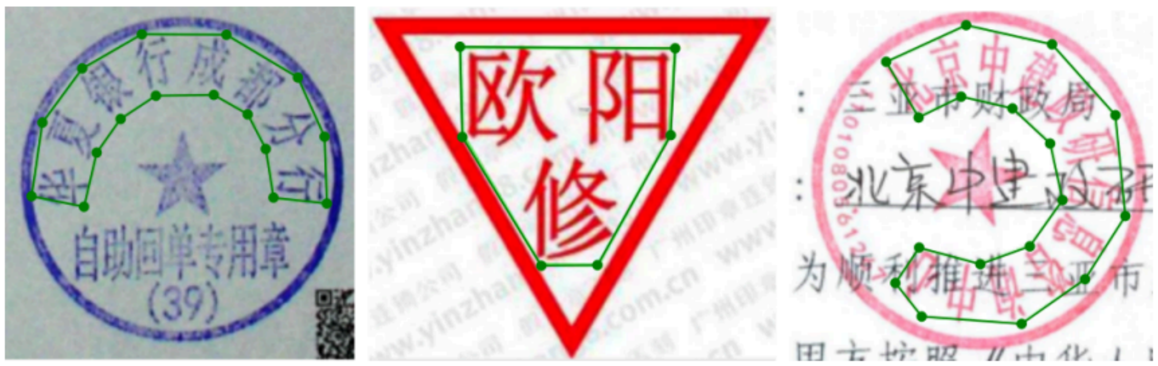


Fig. 4. Example images of the Seal Text dataset. Green color binding lines are formed with polygon ground truth format.

Submission Format. Participants will be asked to submit a JSON file containing results for all test images. The results format is:

```
{
"res_1": [
"points": [[x1, y1], [x2, y2], . . . , [xn, yn]], "confidence" : c],
"res_2": [
"points": [[x1, y1], [x2, y2], . . . , [xn, yn]] , "confidence" : c ],
. . . . . .
}
```

Evaluation Protocol. For Task 1, we adopt IoU-based evaluation protocol by following CTW1500 [10,3]. IoU is a threshold-based evaluation protocol, with 0.5 set as the default threshold. We will report results on 0.5 and 0.7 thresholds but only H-Mean under 0.7 will be treated as the final score for each submitted model, and to be used as submission ranking purpose. To ensure fairness, the competitors are required to submit confidence score for each detection, and thus we can iterate all confidence thresholds to find the best H-Mean score. Meanwhile, in the case of multiple matches, we only consider the detection region with the highest IOU, the rest of the matches will be counted as False Positive. The calculation of Precision, Recall, and F-score are as follows:

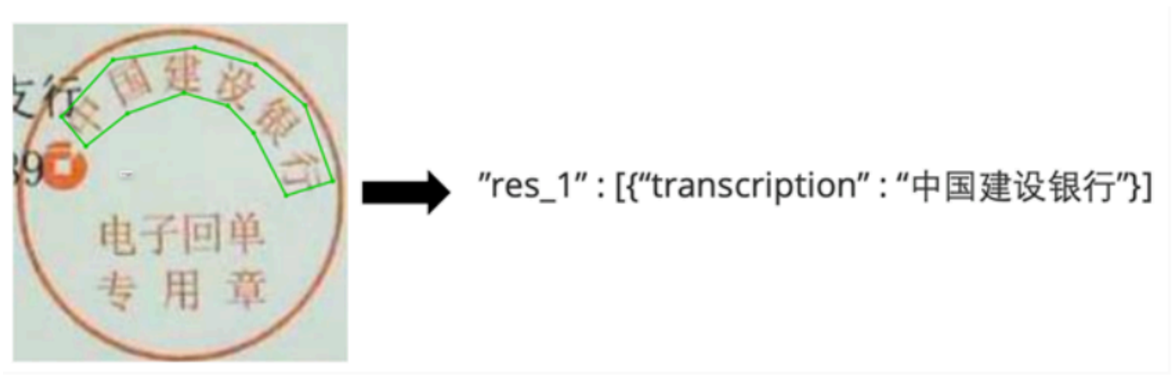
$$\begin{aligned} \text{Precision} &= \frac{TP}{TP+FP}, \\ \text{Recall} &= \frac{TP}{TP+FN}, \end{aligned}$$

$$F = \frac{2 * \text{ Precision } * \text{ Recall}}{\text{Precision } + \text{ Recall}}$$

where TP, FP, FN and F denote true positive, false positive, false negative and H-Mean, respectively

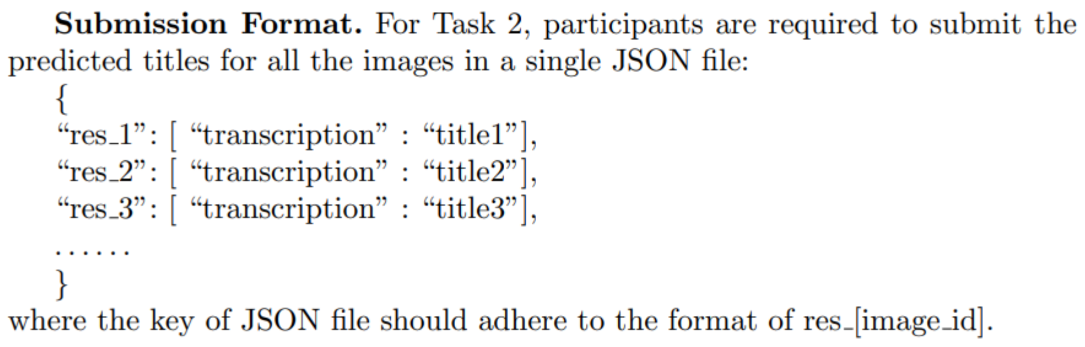
Task 2: End-to-end Seal Title Recognition.

The main objective of this task is to extract the title of a seal, as shown in Figure 5, the input is a whole seal image and the output is the seal's title.



"res_1" : [{"transcription" : "中国建设银行"}]

Fig. 5. Example of the task2 input-output.



Submission Format. For Task 2, participants are required to submit the predicted titles for all the images in a single JSON file:

```
{
“res_1”: [ “transcription” : “title1”],
“res_2”: [ “transcription” : “title2”],
“res_3”: [ “transcription” : “title3”],
......
}
```

where the key of JSON file should adhere to the format of res_[image_id].

* Evaluation Protocol:

-Metrics for this task is case-insensitive word accuracy. We will compute the ratio of correctly predicted titles and the total titles.

-Visualization of ICDAR 2013 :
https://datasets.activeloop.ai/docs/ml/datasets/icdar-2013-dataset/

Book : “Enhancing OCR Performance through Post-OCR Models: Adopting Glyph Embedding for Improved Correction”

The paper titled **"Enhancing OCR Performance through Post-OCR Models: Adopting Glyph Embedding for Improved Correction"** by Yung-Hsin Chen and Yuli Zhou explores the potential of post-OCR correction models to improve the accuracy of Optical Character Recognition (OCR) systems. The authors propose a novel approach that incorporates **glyph embeddings**—visual representations of characters—into the post-OCR correction process, alongside traditional semantic embeddings like CharBERT. The study demonstrates that this combination leads to significant improvements in OCR accuracy, especially for correcting individual words and sentences.

Key Points:

1. **Introduction**

- **OCR Limitations**: OCR systems often struggle with poor image quality, complex layouts, and degraded text, leading to errors in text recognition.

- **Post-OCR Correction**: The authors propose using post-OCR correction models to address these limitations. These models leverage language models (LMs) like GPT and BERT to correct errors in OCR outputs by analyzing context and semantics.

- **Glyph Embedding**: The novelty of their approach lies in incorporating **glyph embeddings**, which capture the visual characteristics of characters, rather than just their semantic meaning. This helps the model better understand and correct OCR errors, especially in cases where the visual form of characters is crucial.

2. **Data**

- **Datasets**: The study uses two main datasets:

  - **ICDAR 2013**: A widely-used benchmark for OCR tasks, containing diverse document images with printed and handwritten text in various languages, fonts, and conditions.

  - **Chars74K**: Used for training the glyph embedding, this dataset contains images of English and Kannada characters. The authors also capture screenshots of Korean and Hebrew characters to create a "garbage class" for open-set classification.

- **OCR Models**: The authors evaluate three OCR models: **EasyOCR**, **PaddleOCR**, and **TrOCR**. EasyOCR is used for single-word detection, while PaddleOCR and TrOCR are used for sentence-level detection.

3. **Method**

- **OCR Models**: The study evaluates state-of-the-art OCR models, including:

  - **EasyOCR**: Uses the CRAFT algorithm for text detection and CRNN for recognition.

  - **PaddleOCR**: Built on the PaddlePaddle platform, it supports multiple languages and scene text recognition.

  - **TrOCR**: A transformer-based OCR model that leverages contextual information for improved accuracy.

- **Post-OCR Correction**: The authors propose a post-OCR correction model that uses two parallel encoders (one for CharBERT embeddings and one for glyph embeddings) and a transformer decoder. The glyph embeddings are trained using **ResNet** models, with a focus on capturing the visual features of characters.

- **Glyph Embedding Training**: The glyph embedding is trained using open-set classification, where non-English characters are classified into a "garbage" class. This helps the model distinguish between relevant and irrelevant characters.

4. **Results**

- **OCR Model Evaluation**: The authors evaluate the OCR models using metrics like **Character Error Rate (CER)** and **Word Error Rate (WER)**. They find that post-OCR correction using GPT-3.5 significantly improves the accuracy of OCR outputs, especially for sentence-level text.

- **Glyph Embedding Performance**: The glyph embedding models show improved accuracy when more layers of the ResNet model are unfrozen during training. The "garbage class" approach outperforms the adapted softmax approach.

- **Post-OCR Correction Model**: The inclusion of glyph embeddings leads to significant improvements in WER and CER scores at the sentence level. However, the impact on single-word correction is minimal, as the model already performs well at this level.

5. **Discussion**

- **Post-OCR Correction Effectiveness**: The study finds that post-OCR correction can mitigate the limitations of OCR models, especially for sentence-level text. However, for single-word inputs, the improvement is less pronounced.

- **Glyph Embedding Impact**: The glyph embedding significantly enhances the performance of the post-OCR correction model, particularly for sentence-level correction. The authors suggest that including additional training data for special characters and punctuation could further improve the model's performance.

Conclusion:

The study demonstrates that **post-OCR correction models**, especially those incorporating **glyph embeddings**, can significantly improve the accuracy of OCR systems. By combining visual and semantic information, the proposed model achieves superior results, particularly for correcting sentence-level text.

The authors highlight the potential for further improvements by expanding the training data to include more diverse characters and punctuation.

Key Contributions:

1. **Glyph Embedding**: The introduction of glyph embeddings, which capture the visual characteristics of characters, is a novel contribution to the field of post-OCR correction.

2. **Lightweight Model**: The proposed post-OCR correction model achieves results comparable to GPT-3.5 but with a significantly lighter architecture, making it more efficient.

3. **Improved OCR Accuracy**: The study shows that post-OCR correction can effectively address the limitations of OCR models, especially in challenging scenarios like degraded text or complex layouts.

Future Work:

The authors suggest that future research could focus on expanding the glyph embedding framework to include more characters, such as punctuation and special symbols, to further enhance the model's performance. Additionally, exploring the application of this approach to other languages and scripts could be a valuable direction for future studies.