End-to-end Document Recognition and Understanding with Dessurt

Brian Davis[1], Bryan Morse[1], Bryan Price[2], Chris Tensmeyer[2], Curtis Wigington[2], and Vlad Morariu[2]

[1] Brigham Young University, Provo UT, USA {`briandavis,morse`}`@byu.edu`
[2] Adobe Research, USA {`bprice,tensmeye,wigingto,morariu`}`@adobe.com`

Abstract. We introduce Dessurt, a relatively simple document understanding transformer capable of being fine-tuned on a greater variety of document tasks than prior methods. It receives a document image and task string as input and generates arbitrary text autoregressively as output. Because Dessurt is an end-to-end architecture that performs text recognition in addition to the document understanding, it does not require an external recognition model as prior methods do. Dessurt is a more flexible model than prior methods and is able to handle a variety of document domains and tasks. We show that this model is effective at 9 different dataset-task combinations.

Keywords: Document understanding, end-to-end, handwriting recognition, form understanding, OCR

1 Introduction

Document understanding is an area of research attempting to automatically extract information from documents, whether that be specific key information, answers to natural language questions, or other similar elements. While there have been many approaches, the research community has begun to gravitate around pre-trained transformers as general purpose solutions. Beginning with LayoutLM [35], these models began as BERT-like transformers incorporating spatial/layout information and later visual features. In general, we refer to these as the LayoutLM family. The LayoutLM family of models are pre-trained on a large corpus of document images and then fine-tuned to their particular tasks.

The LayoutLM family consists of encoder-only transformers, meaning predictions are only made for the input tokens. These state-of-the-art models are two-stage models, where text recognition is first performed by an external OCR model to obtain the input text tokens for the transformer. We see two limitations coming from these architecture choices:

1. A limited output space, having predictions only for individual input tokens. While they can classify the input tokens, they cannot produce additional outputs, e.g., arbitrary text or token relationships, without additional submodules.

Fig. 1. The LayoutLM family of document transformers require OCR and output is tied to the tokens. Dessurt does not require any separate models and can generate arbitrary text to solve a variety of tasks.

Table 1. Model class capabilities

	Handwriting	Arbitrary output	Apply to different visual domain
LayoutLM family	OCR dependant	✗	Fine-tune two models
Dessurt	✓	✓	Fine-tune single model

2. Dependence on high quality external OCR text segmentation and recognition. Encoder-only transformers are incapable of inserting new tokens if the OCR missed or under-segmented text. A single incorrectly recognized character in an OCR'd word can cause a wrong word embedding to be used or cause the word to be out of vocabulary. Relatedly, discrete input tokens lack the uncertainty the text recognition model may have in its predictions. For clean, modern documents, this generally isn't an issue as the OCR models used are quite robust. However, for handwritten or degraded historical documents, OCR quality can be poor and lead to prediction errors.

To combat these flaws we introduce **Dessurt**: **D**ocument **e**nd-to-end **s**elf-supervised **u**nderstanding and **r**ecognition **t**ransformer. Dessurt is a novel, general document understanding architecture that can perform a great variety of document tasks. Dessurt operates in an end-to-end manner with a single pass: text segmentation and recognition are learned implicitly. Dessurt takes only the image and task text as input and can auto-regressively produce arbitrary text as output. Fig. 1 compares Dessurt to the LayoutLM family at a high level architecturally. The first limitation of the LayoutLM family is easily solved with Dessurt's auto-regressive output. Because text recognition is implicit, rather than provided as explicit OCR results, Dessurt is able to resolve text recognition uncertainty or ambiguity in a task-focused way. Additionally, the auto-regresssive output decouples Dessurt's output from the text recognition. These together address the second limitation. See Table 1 for a comparison of architecture features.

Because Dessurt takes both an input image and text and can output any arbitrary text, it can complete a greater variety of tasks compared to the LayoutLM family of transformers. Particularly we solve a form parsing task (form

image to JSON) that the LayoutLM family cannot handle without additional modules. Also, when retraining for a different visual domain, Dessurt's simple end-to-end design means only one model needs to be fine-tuned. For the LayoutLM family, both the recognition and transformer models would need to be fine-tuned.

Like prior methods, we pre-train on the IIT-CDIP dataset [17], a large collection of document images, with a masked language modeling task. We also introduce three synthetic document datasets to better capture natural language, structured documents, and handwriting recognition. Finally, we introduce new pre-training tasks to teach Dessurt to read and located text, and to parse structured documents.

We validate our claims of Dessurt's flexibility by applying it to six different document datasets across six different tasks: 1) Document question answering, with both DocVQA [23] and HW-SQuAD [22], 2) Form understanding and 3) Form parsing, with both the FUNSD [14] and NAF [5] datasets, 4) Full-page handwriting recognition and 5) Named entity recognition on the IAM handwriting database, and 6) Document classification with the RVL-CDIP dataset. Of particular interest, both NAF and IAM datasets require handwriting recognition, the NAF being comprised of difficult historical documents. These are domains in which the LayoutLM family would need to fine-tune its recognition model as well, but Dessurt can fine-tune on without adjustments. We note that Dessurt does not achieve state-of-the-art results on the most tasks evaluated, but it is capable of operating on a larger range of tasks than individual state-of-the-art models.

In summary, our primary contributions are

- Dessurt, a novel, general document understanding architecture capable of both performing text recognition and document understanding in an end-to-end manner and producing arbitrary text output,
- A collection of synthetic datasets and tasks for pre-training an end-to-end document understanding model for a variety of possible final tasks,
- An evaluation of Dessurt fine-tuned on 9 dataset-task combinations, and
- Our code, pre-trained model, and datasets which will be made available at `https://github.com/herobd/dessurt`

2 Related Work

2.1 LayoutLM Family

Document understanding has become largely dominated by transformer architectures. Beginning with LayoutLM(v1) [35] the goal was to bring the success of transformers like BERT [8] in the natural language space into the more visual domain of documents. LayoutLM pre-trained in a very similar manner to BERT, but included 2D spatial position information.

BROS [12], TILT [24], and LayoutLMv2 [34] improved the architecture by introducing spatially biased attention, making the spatial information even more

Fig. 2. Dessurt architecture

influencial. LayoutLMv2 also introduced visual tokens as many layout cues are captured more visually than spatially.

Visual tokens can be overshadowed by textual tokens. In an effort to make the visual processing more important, DocFormer [1] forced feature updates to be from both textual and visual features.

We note that TILT and DocFormer use only visual features extracted near the text tokens spatially, making them blind to areas of the form without text. LayoutLMv2 extracts visual tokens across the entire document.

2.2   End-to-end Models

Models in the LayoutLM family have been evaluated without taking text recognition into account. Many document understanding datasets come with pre-computed OCR results used by everyone. While this is useful in making comparisons, text recognition is an essential task and for visually difficult documents can become a challange in itself.

One aim of an end-to-end method can be to accomplish both recognition and understanding in one pass. Another aim might be to learn the output text in a manner that allows arbitrary output predictions. DocReader [16] is an end-to-end method for key information extraction. While it does rely on external OCR, it uses an RNN to predict arbitrary text.

We note that a concurrent pre-print work on end-to-end document understanding, Donut [15], has been introduced, and shares an architecture similar in design to Dessurt. It also utilizes a Swin [19] encoder but uses a BART-like [18] decoder. Donut differs from Dessurt primarily in how the cross attention occurs and in pre-training. Donut shares many of the same advantages of Dessurt.

3   Model

Dessurt is a novel end-to-end framework for handling general document understanding problems. It takes as input an image and query text, and outputs

arbitrary text appropriate for the given tasks. It handles character recognition, including handwriting recognition, implicitly as part of the network.

The architecture is shown in Fig 2. The model processes three streams of tokens: 1) Visual tokens that encode visual information about the document image, 2) Query tokens that encode the task the model is to perform, and 3) Autoregressive response tokens where the output response is formed. The model progresses through three main stages: Input encoding, cross-attention, and output decoding.

Input encoding: The input consists of an image and a query token string. Because the Swin [19] layers we use require a fixed size image input, we use an input image size of 1152×768. This large size is needed as we process an entire page at once and must ensure small text is legible. The input image is 2-channeled, one being the grayscale document, the other being a highlight mask used in some tasks. The query tokens begin with a special task token indicating the desired task and then potentially have some text providing context for the task (e.g., the question text). The response tokens are initialized with a task specific start token and during training contain the previous ground truth token for teacher-forcing.

The first step of the model is to encode the inputs into feature arrays to initialize the three streams. The input image is tokenized by passing it through a small downsampling CNN and adding learned 2D spatial embeddings. The input query text and response text are tokenized using the same process as BART [18] with standard sinusoidal position encoding. These feature arrays are then passed to their respective token streams.

Note that the model does not require as input any OCR tokens corresponding to the image. The network implicitly recognizes the text.

Cross-attention: The three streams then pass through a series of cross-attention layers to allow them to share information and transfer that information into the response. The visual array is processed by Swin [19] layers modified to not only attend to the other elements in the local window but also the query array. (We note that the biased attention remains for the visual elements.) The query array has standard Transformer [31] attention, but attends to the entire visual array in addition to the query array. The response array has standard autoregressive attention to previous response elements but also attends to the visual and query arrays. The arrays pass through series of eight of their respective cross-attention layers. The last two layers of the model update only the query and response arrays, with both layers attending to the final visual features.

Output decoding: The final response array is decoded into text using greedy search decoding (where the most likely token is selected at each step), allowing it to predict text not found in the document. Additionally, we also output a pixel mask for use in training. This is produced by a small upsampling network using six transpose convolutions that process the final visual features.

Specific implementation details for the model and its layers can be found in the accompanying Supplementary Materials.

Fig. 3. Examples of data used in pre-training. (a) IIT-CDIP dataset image with Text Infilling task highlighting channel: highlight is magenta (value of 1), removed text is turquoise (value of -1). (b) Synthetic Wikipedia text. (c) Synthetic handwriting. (d) Synthetic form and its parse JSON.

4   Pre-training Procedure

The goal of the pre-training is to teach Dessurt to perform text recognition and document understanding and to have general language model capabilities like BERT. We pretrain several datasets with each dataset having multiple tasks associated with it. An example from each dataset is in Fig. 3.

4.1   IIT-CDIP dataset

The IIT-CDIP dataset [17] is a pre-training dataset used by several other document understanding transformers [35,34,1]. The OCR method we applied to the IIT-CDIP dataset is in the Supplementary materials.

There are several tasks defined with this dataset all of which are described in the Supplementary Materials. For brevity, we only describe the most important ones here. The primary task (occurring 66% of the time) is a Text Infilling task. It is a masked language modeling task inspired by the text infilling used to train BART [18]; instead of replacing the removed text with a blank token, we delete them from the image. The entire block of text and the deleted areas are marked (with different values) in the input highlight channel, as seen in Fig. 3 (a). The model then must predict the text of the entire block, filling in the deleted regions. We also do a variant of this task where a single word is blanked from the image and the model must predict that single word. There are several reading based tasks as well, such as to read on from the text provided in the query.

4.2   Synthetic Wikipedia

We want our pre-training to help the model understand natural language; however, the IIT-CDIP dataset only represents a skewed slice of natural language. Additionally, it represents a limited range of font styles. We choose to create an

on-the-fly dataset by selecting random text from Wikipedia[3] [9] and rendering it as paragraphs in random locations with random fonts.

We pick a random article, random column width, random font, random text height, and random spacing (between word and new line). We render the words using the font and text height. We place the words in column/paragraph form, adjusting the column width to fit as much of the article as possible. We find blank space in the image where the paragraph can be added. If one is found the paragraph is added and we attempt to add another paragraph; otherwise, the image is complete. An example generated image is seen in Fig. 3 (b).

To obtain our font database, we scrape all the free-for-commercial-use fonts from 1001fonts.com, giving us a set of over 10,000 fonts. The script we used to scrape the fonts will be made available. More details on these fonts and our synthetic dataset creation are found in the Supplementary Materials

This dataset uses the same distribution of tasks as the IIT-CDIP dataset.

4.3   Synthetic Handwriting

Dessurt must be able handle handwriting as several document understanding tasks require this. The IIT-CDIP dataset contains little handwriting and while our font database has "handwritten" fonts, they do not capture the variation present in real handwriting. There is, unfortunately, not a publicly available dataset of handwriting comparable in size to the IIT-CDIP datset. The IAM handwriting database [21] is frequently used, but with fewer than 800 instances to train on, an autoregressive transformer could overfit during pre-training.

We choose instead to use synthetic handwriting. This allows us to generate a larger breadth of text, but at the cost of realism. We use the full line handwriting synthesis method of Davis et al. [7] to generate 800,000 lines of sequential text from Wikipedia articles, with a randomly sampled style for each line. We compose a document by sampling a random number of consecutive handwriting lines (to maintain language flow), selecting a random text height, random newline height, and random starting location on the page, and then placing the lines in the document in block/paragraph style. We additionally apply warp grid augmentation [32] to each line to further add to the visual variation. An example image can be seen in Fig. 3 (c).

For the learning task, the model must read the entire page of handwriting.

4.4   Synthetic Forms

We want Dessurt to be capable of extracting the information from highly structured documents, but given the lack of structured information present in our IIT-CDIP annotations, we decided to generate synthetic forms with known structure. The structure is based on the annotations of the FUNSD [14] dataset, which is primarily label-value pairs (or question-answer pairs) which are occasionally grouped under headers. We also include tables.

---

[3] https://huggingface.co/datasets/wikipedia

To come up with label-value pairs, we use GPT-2 [25] to generate synthetic "forms". We give GPT-2 a prompt text (e.g., "This form has been filled out.") followed by an example label-value pair, newline and a label with colon (e.g., "Date: 23 Mar 1999\nName:"). GPT-2 then usually generate a series of label-value pairs separated by colons and newlines, which is easily parsed. All the label-value pairs from one generation are a label-value set in our dataset. We sometimes use Wikipedia article titles as part of the prompt (e.g. "This form should be filled out regarding <u>Marvel Universe</u>") which then become the header for that label-value set. We reuse previously generated labels and label-value pairs as new form prompts. The quality of GPT-2 output is limited, but we hope it reflects at least some of the semantics of label-value relationships.

The data for tables is more random. The row and column headers are random 1 to 3 word snippets from Wikipedia. A cell value is either a random number (with various formatting) or a random word.

A document is composed by randomly placing label-value sets and tables until a placement fails due to there not being enough room. Some cells and values are blanked. More details on the form generation process can be found in the Supplementary Materials.

The primary task on the forms (occurring about half the time) is to parse it into a JSON capturing the text and structure of the form. An example synthetic form and its corresponding JSON are seen in Fig. 3 (d). We also have tasks where the query has an entity on the form and the model must predict the class of the entity and then read the entities it is linked to. To ensure an understanding of tables, there are also table-specific tasks such as retrieving a cell based on a query with the row and column header, or listing all the row/column headers for a table. All the tasks used are described in the Supplementary Materials.

4.5 Distillation

Because Dessurt has a unique architecture, we could not use pre-trained transformer weights to initialize our model (like Donut [15] or models in the LayoutLM family). This is clearly a disadvantage, so we attempt to infuse pre-trained language knowledge into Dessurt in a different way: cross-domain distillation. We feed text to a pre-trained transformer teacher, and then render that text in a document image to pass to the student, Dessurt. Then we apply the standard distillation loss introduced by Hinton et al. [11], which guides the logit predictions of the student to match the teachers logits (the "dark knowledge").

Distillation is generally applied with a student and teacher getting the exact same inputs. We are attempting something fairly unique which is to apply distillation across domains, textual to visual.

To ensure architectural similarity, we need the teacher to be an autoregressive model. For this we use BART, an encoder-decoder transformer where the decoder is an autoregressive model with cross attention to the encoder (a vanilla transformer encoder). Both BART and Dessurt will be given the Text Infilling task which BART was pre-trained with. BART gets the masked text as input

to its encoder, and Dessurt gets the rendered text with deleted regions as input (and the query token indicating the Test Infilling task) and then they both autoregressively output the input text with the blanks filled in.

The token probabilities Dessurt predicts for a blanked region reflect not only its language modeling, but also the uncertainty it has in reading the other words on the page. For BART the probabilities are only the language modeling; it has no uncertainty about reading. We minimize the reading uncertainty Dessurt has when performing distillation by selecting a subset of "easy" fonts and reducing the variability with which the documents are rendered. More details on this are in the Supplementary Materials.

4.6   Training

We employ a simple curriculum to to prioritize certain aspects during early training. This is due to the need for recognition to be learned (to a certain degree) before the understanding tasks can be solved and the difficulty of learning recognition on dense multi-line documents in a semi-supervised fashion.

We first train Dessurt on small images (96×384) of synthetic Wikipedia text with simple reading tasks for 150,000 iterations. Not only is the visual space small, but the output sequence length is short. We then use full-sized synthetic Wikipedia text documents for 200,000 iterations with primarily reading tasks. Finally, the model enters normal pre-training.

The iterations we outline here are what were used for the ablation models. For our primary evaluation we use a model that was pre-trained in total for over 10 million iterations (110k weight update steps) during development (meaning datasets and tasks were added throughout the training), but followed roughly the same curriculum. The ablation models were pre-trained for 1 million iterations with all datasets and tasks being introduced at once.

We used data parallelism over 6 Nvidia Tesla P100s, which can each only hold a batch size of 1. We use gradient accumulation of 128 iterations, leading to an effective batch size of 768, with approximately 7,800 weight update steps for the ablation models. The last 4 million iterations of the final modal used gradient accumulation of 64 iterations, meaning the effective batch size was 384. We use the AdamW optimizer [20] and a learning rate of 10^{-4} and weight decay of 0.01.

5   Experiments

To demonstrate the flexibility of Dessurt, we evaluate it on the six document datasets and six diverse taskslisted in Table 2. The RVL-CDIP dataset [10] is a page classification dataset, which requires understanding overall layout and text topics. The DocVQA dataset [23] requires both reading and layout comprehension. HW-SQuAD [22] is more focused on reading comprehension, but has difficult text (synthetic handwriting) to recognize. Both the FUNSD [14] and NAF [5] datasets require form understanding, with a focus on label-value pairs

Table 2. A summary of the end tasks we use to evaluate Dessurt and their attributes. The term "special output" refers to whether the tasks requires more than standard token prediction employed by most in the LayoutLM family

Dataset	Task	Domain	Requires handwriting recognition	Requires special output	Train set size
RVL-CDIP [10]	Classification	Modern printed	No	No	320K
DocVQA [23]	Question answering	Modern	Occasionally	No	39K
HW-SQuAD [22]	Question answering	Synthetic handwriting	Yes (easier)	No	68K
FUNSD [14]	Entity / Relationship detection	Modern printed forms	No	No/Yes	130
FUNSD	Form parsing	Modern printed forms	No	Yes	130
NAF [5]	Line / Relationship detection	Historic forms	Yes	No/Yes	921
NAF	Form parsing	Historic forms	Yes	Yes	921
IAM [21]	Full page recognition	Handwriting	Yes	Yes	747
IAM NER [30]	Named entity recognition	Handwriting	Yes	No	747

in forms. The FUNSD dataset includes modern business documents, but the NAF dataset is uniquely challenging because it contains historical records with a both printed and handwritten text. We take a task from Tüselmann et al. [30], specifically named entity recognition over the IAM handwriting database (IAM NER), requiring both handwriting recognition and NLP capabilities. We also evaluate full-page handwriting recognition on the IAM database [21]. Each of these, and our experimental protocol for them, are discussed in more detail in the Supplementary Materials. We also present an ablation study at the end of this section.

5.1   RVL-CDIP

We compare Dessurt to several other models in Table 3 on document classification with the RVL-CDIP dataset. Dessurt performs slightly below the state-of-the-art, but is comparable to the other models. We note that this problem requires a holistic view of the document and is likely benefiting from a strong vision model. We note that while Dessurt uses a Swin architecture, it is shallower and narrower than the one used by Donut.

5.2   DocVQA and HW-SQuAD

For DocVQA, the model must locate the text that answers a textual question. The results are presented in Table 3 with ANLS, a text edit-distance metric that accounts for multiple correct answers. Unlike RVL-CDIP, understanding

Table 3. Results on RVL-CDIP and DocVQA datasets

	use OCR	# params	RVL-CDIP accuracy	DocVQA ANLS
BERT$_{BASE}$ [8]	✓	110M +OCR	89.8	63.5
LayoutLM$_{BASE}$ (w/ img) [35]	✓	160M + OCR	94.4	-
LayoutLM$_{BASE}$ [35]	✓	113M + OCR	-	69.8
LayoutLMv2$_{BASE}$ [34]	✓	200M + OCR	95.3	78.1
LayoutLMv2$_{BASE}$ w/ Tesseract OCR	✓	200M + OCR	-	48.2
DocFormer$_{BASE}$ [1]	✓	183M + OCR	**96.2**	-
TILT$_{BASE}$ [24]	✓	230M + OCR	93.5	**83.9**
Donut [15]		156M	94.5	47.1
Donut +10k trainset images [15]		156M	-	53.1
Dessurt (ours)		127M	93.6	63.2

the text in DocVQA is critical, likely leading to both Dessurt's and Donut's comparatively limited performance. Other models rely on strong external recognition methods; LayoutLMv2's performance significantly drops when using a weaker OCR. Dessurt outperforms Donut, likely due to its language-focused tasks and real data in pre-training. Dessurt's weakest areas for DocVQA are Figures/Diagrams and Image/Photo. This makes sense because the pre-training datasets are almost exclusively textual.

The HW-SQuAD dataset [22] is the popular question answering benchmark SQuAD [26] rendered with handwritten fonts and noise. We evaluate on the task of machine comprehension, where the single document containing the answer is fed to the model. Unfortunately, the only prior method on this ([29]) was doing text snippet retrieval, not question answering, and so is incomparable. We use ANLS as our metric as it seems well suited to the task and achieve 55.5%.

5.3   FUNSD and NAF

Form parsing is the most difficult task we tackle, particularly with the NAF dataset, which is comprised of historical forms containing a good deal of handwriting. In our full form parsing task the model must reproduce the entire contents of the form in a structured manner, including recognition of text. We have the model predict JSON using the same format used in pre-training (Fig. 3 (d)). Normalized tree edit-distance (nTED) has been introduced by Hwang et al. [13] as a metric for comparing document parses. However, nTED is not permutation invariant, which is undesirable due to the lack of a canonical read order for forms. We introduce a modified metric, Greedily-Aligned nTED or GAnTED, which is more robust to permutation. GAnTED is discussed in detail in the Supplementary Materials. We compute GAnTED for FUDGE [6] by running Tesseract[4] on the bounding boxes it predicts and using the class and relationship predictions to build the JSON output.

---

[4] https://github.com/tesseract-ocr/tesseract

Table 4. Results on FUNSD dataset

	GT OCR used	# params	Entity Fm	Rel Fm	GAnTED
LayoutLM$_{BASE}$ [35]	boxes + text	-	78.7	42.8 [12]	-
BROS$_{BASE}$ [12]	boxes + text	138M + OCR	83.1	**71.5**	-
LayoutLMv2$_{BASE}$ [34]	boxes + text	200M + OCR	82.8	-	-
DocFormer$_{BASE}$ [1]	boxes + text	183M + OCR	**83.3**	-	-
Word-FUDGE [6]	boxes	17M + OCR	72.2	62.6	-
FUDGE [6] (+Tesseract)	none	17M (+OCR)	66.5	56.6	34.8
Dessurt (ours)	none	127M	65.0	42.3	**23.4**

Table 5. Results on NAF dataset

	# params	Line Fm	Rel Fm	GAnTED
Davis et al. [5]	1.8M	**73.8**	49.6	-
FUDGE [6]	17M	73.7	**57.3**	-
Dessurt (ours)	127M	49.3	29.4	42.5
Dessurt w/ census pretraining	127M	50.2	30.3	**38.8**

We also compare using standard F-measure for entity detection and relationship detection. We do this by aligning Dessurt's predicted strings to the GT strings. This means our results are dependant on the text recognition of Dessurt. This is in contrast to other models that use the GT word boxes for tokens and need only identify the correct box(es) rather than produce the correct text. Thus we end up below what prior methods achieve. Our results on both the FUNSD and NAF datasets are presented in Tables 4 and 5 respectively. On the NAF dataset, no models rely on external recognition models.

The visual domain of NAF is very different from modern documents, meaning two-stage methods require a specialized recognition model. We compare Dessurt's recognition ability to a CNN-LSTM [32] trained on the NAF dataset in the Supplementary Materials. We also report results pre-training Dessurt on images taken from the U.S.A. 1940 Census (visually similar to NAF data) in Table 5. Details for this pre-training are in the Supplementary Materials.

5.4   IAM Database

There have been several specialized approaches for doing full-page handwriting recognition, where line segmentation is done implicitly or explicitly. Dessurt is trained to do full-page recognition during its pre-training. We compare it to other full-page recognition models in Table 6. The metrics used are character error rate (CER) and word error rate (WER) across an entire page (or paragraph; the IAM dataset has one paragraph per page). Dessurt performs quite favorably compared to these specialized approaches and even achieves the lowest WER. We note that our pre-training includes synthetic handwriting derived from the IAM training set, so Dessurt is uniquely suited to solve this task on the IAM dataset. The fact that Dessurt's WER is relatively better than its CER is unusual and is likely a result of the word-part token prediction (other models use character prediction)

Table 6. Results on IAM page/paragraph recognition

	# params	CER	WER
Bluche [2]	-	7.9	24.6
Chung and Delteil [3]	-	8.5	-
Start, Follow, Read [33]	-	6.4	23.2
OrigamiNet [36]	16.4M	4.7	-
Vertical Attention Network [4]	2.7M	**4.5**	14.6
Dessurt (ours)	127M	4.8	**10.2**

Table 7. Results on IAM NER. Reported in macro F-measure

	Split RWTH Task 6 classes	Custom 6 classes	RWTH 18 classes	Custom 18 classes
Toledo et al. [28]	34.0	37.4	14.9	18.0
Rowtula et al. [27]	47.4	54.6	32.3	30.3
Tüselmann et al. [30]	**70.7**	**76.4**	**52.0**	**53.6**
Dessurt (ours)	62.0	71.5	40.4	48.5
Dessurt w/ IAM pretraining	59.5	71.1	39.5	45.3

and the language modeling capabilities learned in pre-training. We note that the number of parameters in Dessurt is one or two orders of magnitude higher than the other models.

We also evaluate using the IAM NER task introduced by Tüselmann et al. [30] as part of a set of named entity recognition problems for handwriting datasets. Tüselmann et al. use a two-stage approach constructed specifically for this problem. They use a word level handwriting recognition model, with its outputs fed to a RoBERTa-based NER model (which sees the whole document). We fine-tune Dessurt on both line level NER and document level NER. In both cases Dessurt sees the entire handwriting image but has the lines it is supposed process highlighted. It performs transcription along with the classification with two tasks: (1) first reading a word, and then predicting its class, and (2) the reverse with class predicted first. This ensures we know which word Dessurt is predicting a class for. We randomly replace words in the teacher-forcing with close edit-distance words to decrease reliance on the recognition output. Additionally, we apply warp grid augmentation [32] on the lines of the document. We also experimented with adding recognition on IAM words to the pre-training (more details in Supplementary Materials).

Our results for IAM NER are presented in Table 7. While Dessurt is moderately successful, it falls short of the customized two-stage approach presented by Tüselmann et al. They report that the CER of the HWR model they use is 6.8, which is the same CER as Dessurt. We assume this indicates that (unsurprisingly) RoBERTa is a stronger language model than Dessurt and is responsible for this superior performance.

Table 8. Ablation results. The top four rows show the pre-training ablation with I=IIT-CDIP dataset, W=synthetic Wikipedia dataset, H=synthetic handwriting dataset, F=synthetic form dataset, D=distillation from BART. The lower three rows show ablations to the model: removing supervision with output mask, removigin supervision with output mask and reducing Swin window size to 7, removing cross attention from image to question tokens. Results for DocVQA are evalutated using the validation set. "PT IAM" indicates IAM data added to last 200k iters of pre-training

	DocVQA (valid) ANLS	IAM NER	Macro Fm IAM PT	FUNSD Entity Fm	Rel Fm	NAF Entity Fm	Rel Fm	RVL CDIP acc.
Max iterations	500k	200k	200k	34k		300k		500k
I	44.0	42.3	43.4	19.7	10.2	28.7	12.6	89.0
W+I	43.2	45.2	49.0	29.5	16.0	31.0	13.7	89.1
H+W+I	44.4	50.1	49.7	29.3	16.5	31.6	14.9	88.9
F+H+W+I	**46.5**	47.6	50.0	44.8	28.2	**36.5**	**17.6**	**89.5**
D+F+H+W	43.1	**52.7**	**53.3**	39.4	22.0	31.5	14.3	88.5
All=D+F+H+W+I	45.5	50.4	52.5	**47.8**	**29.5**	34.6	15.3	89.0
All, no mask loss	44.9	45.7	49.7	47.3	26.2	33.2	15.1	88.3
All, no mask loss w=7	44.4	44.8	51.3	45.9	28.6	31.8	15.3	88.6
All, 1-way cross attn.	44.9	42.9	46.9	41.0	25.2	33.7	15.9	88.8

5.5 Ablation

We performed an ablation study of the different sources used in our model's pretraining as well as some of the architectural choices (Table 8). We begin the data ablation with only the IIT-CDIP [17] dataset (I). We then incrementally add the synthetic Wikipedia (W), synthetic handwriting (H), synthetic forms (F), and distillation from BART (D). We ablate out the the predicted spatial mask used in pretraining, and change the Swin window size from 12 to 7. We also ablate the 2-way cross attention by instead only having the query and response tokens attend to the visual tokens without the visual tokens attending to the query tokens. This is very similar to Donut, which lacks 2-way cross attention.

As can be seen each pre-training data source adds something to the model. The synthetic handwriting and synthetic forms are aimed at particular downstream tasks (IAM NER and form understanding respectively), but we note that their inclusion generally helps other tasks as well. Only the distillation appears selectively helpful and may not contribute significantly. In general, the ablated model components are helpful to the full model, but not necessary. The results with the RVL-CDIP dataset shows that the data a model is pre-trained with appears to be relatively irrelevant to its performance.

6   Conclusion

We have introduced Dessurt, an end-to-end architecture for solving a wide variety of document problems. Dessurt performs recognition within its single pass,

removing reliance on an external recognition model, which most document understanding approaches require, making it a much simpler method. Because Dessurt uses arbitrary text as its output, it is also more flexible in the range of problems it can solve. We evaluate Dessurt on a wider range of tasks than any previous single method has done and show results ranging from promising to state-of-the-art.

References

1. Appalaraju, S., Jasani, B., Kota, B.U., Xie, Y., Manmatha, R.: Docformer: End-to-end transformer for document understanding. In: International Conference on Computer Vision (ICCV) (2021)
2. Bluche, T.: Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. Advances in Neural Information Processing Systems (NIPS) (2016)
3. Chung, J., Delteil, T.: A computationally efficient pipeline approach to full page offline handwritten text recognition. In: International Conference on Document Analysis and Recognition Workshops (ICDARW). IEEE (2019)
4. Coquenet, D., Chatelain, C., Paquet, T.: End-to-end handwritten paragraph text recognition using a vertical attention network. IEEE Transactions on Pattern Analysis and Machine Intelligence (2022)
5. Davis, B., Morse, B., Cohen, S., Price, B., Tensmeyer, C.: Deep visual template-free form parsing. In: International Conference on Document Analysis and Recognition (ICDAR). IEEE (2019)
6. Davis, B., Morse, B., Price, B., Tensmeyer, C., Wiginton, C.: Visual fudge: Form understanding via dynamic graph editing. In: International Conference on Document Analysis and Recognition (ICDAR). Springer (2021)
7. Davis, B., Tensmeyer, C., Price, B., Wigington, C., Morse, B., Jain, R.: Text and style conditioned gan for generation of offline handwriting lines (2020)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) (2019)
9. Foundation, W.: Wikimedia downloads, `https://dumps.wikimedia.org`
10. Harley, A.W., Ufkes, A., Derpanis, K.G.: Evaluation of deep convolutional nets for document image classification and retrieval. In: International Conference on Document Analysis and Recognition (ICDAR)
11. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015). arXiv preprint arXiv:1503.02531 **2** (2015)
12. Hong, T., Kim, D., Ji, M., Hwang, W., Nam, D., Park, S.: Bros: A pre-trained language model focusing on text and layout for better key information extraction from documents. arXiv preprint arXiv:2108.04539 (2021)
13. Hwang, W., Lee, H., Yim, J., Kim, G., Seo, M.: Cost-effective end-to-end information extraction for semi-structured document images. In: Conference on Empirical Methods in Natural Language Processing (EMNLP) (2021)
14. Jaume, G., Ekenel, H.K., Thiran, J.P.: Funsd: A dataset for form understanding in noisy scanned documents. In: International Conference on Document Analysis and Recognition Workshops (ICDARW). IEEE (2019)
15. Kim, G., Hong, T., Yim, M., Park, J., Yim, J., Hwang, W., Yun, S., Han, D., Park, S.: Donut: Document understanding transformer without ocr. arXiv preprint arXiv:2111.15664 (2021)
16. Klaiman, S., Lehne, M.: Docreader: Bounding-box free training of a document information extraction model. In: International Conference on Document Analysis and Recognition (ICDAR) (2021)
17. Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., Heard, J.: Building a test collection for complex document information processing. In: ACM SIGIR Conference on Research and Development in Information Retrieval (2006)

18. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L.: Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: 58th Annual Meeting of the Association for Computational Linguistics (ACL) (2020)
19. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: International Conference on Computer Vision (ICCV) (2021)
20. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (ICLR) (2019)
21. Marti, U.V., Bunke, H.: The iam-database: an english sentence database for offline handwriting recognition. International Journal on Document Analysis and Recognition **5**(1) (2002)
22. Mathew, M., Gomez, L., Karatzas, D., Jawahar, C.: Asking questions on handwritten document collections. International Journal on Document Analysis and Recognition (IJDAR) **24**(3) (2021)
23. Mathew, M., Karatzas, D., Jawahar, C.: Docvqa: A dataset for vqa on document images. In: Winter Conference on Applications of Computer Vision (WACV) (2021)
24. Powalski, R., Borchmann, Ł., Jurkiewicz, D., Dwojak, T., Pietruszka, M., Pałka, G.: Going full-tilt boogie on document understanding with text-image-layout transformer. In: International Conference on Document Analysis and Recognition (ICDAR). pp. 732–747 (2021)
25. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog (2019)
26. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: Squad: 100,000+ questions for machine comprehension of text. In: Conference on Empirical Methods in Natural Language Processing (EMNLP) (2016)
27. Rowtula, V., Krishnan, P., Jawahar, C., CVIT, I.: Pos tagging and named entity recognition on handwritten documents. In: International Conference on Natural Language Processing (ICNLP) (2018)
28. Toledo, J.I., Carbonell, M., Fornés, A., Lladós, J.: Information extraction from historical handwritten document images with a context-aware neural model. Pattern Recognition **86** (2019)
29. Tüselmann, O., Müller, F., Wolf, F., Fink, G.A.: Recognition-free question answering on handwritten document collections. arXiv preprint arXiv:2202.06080 (2022)
30. Tüselmann, O., Wolf, F., Fink, G.A.: Are end-to-end systems really necessary for ner on handwritten document images? In: Lladós, J., Lopresti, D., Uchida, S. (eds.) International Conference on Document Analysis and Recognition (ICDAR) (2021)
31. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Neural Information Processing Systems (NIPS) (2017)
32. Wigington, C., Stewart, S., Davis, B., Barrett, B., Price, B., Cohen, S.: Data augmentation for recognition of handwritten words and lines using a cnn-lstm network. In: International Conference on Document Analysis and Recognition (ICDAR) (2017)
33. Wigington, C., Tensmeyer, C., Davis, B., Barrett, W., Price, B., Cohen, S.: Start, follow, read: End-to-end full-page handwriting recognition. In: European Conference on Computer Vision (ECCV) (2018)
34. Xu, Y., Xu, Y., Lv, T., Cui, L., Wei, F., Wang, G., Lu, Y., Florencio, D., Zhang, C., Che, W., Zhang, M., Zhou, L.: LayoutLMv2: Multi-modal pre-training for visually-rich document understanding. In: 59th Annual Meeting of the Association for Computational Linguistics (ACL) (2021)

35. Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., Zhou, M.: LayoutLM: Pre-training of text and layout for document image understanding. In: International Conference on Knowledge Discovery & Data Mining (KDD) (2020)
36. Yousef, M., Bishop, T.E.: Origaminet: weakly-supervised, segmentation-free, one-step, full page text recognition by learning to unfold. In: Computer Vision and Pattern Recognition (CVPR) (2020)

Supplementary Materials for End-to-end Document Recognition and Understanding with Dessurt

Brian Davis[1], Bryan Morse[1], Bryan Price[2], Chris Tensmeyer[2], Curtis Wigington[2], and Vlad Morariu[2]

[1] Brigham Young University, Provo UT, USA {briandavis,morse}@byu.edu
[2] Adobe Research, USA {bprice,tensmeye,wigingto,morariu}@adobe.com

In these Supplementary Materials we provide details on the model, training, and evaluation of Dessurt. There also additional examples of the synthetic data. The contents are as follows:

1 Model Details

The down-sampling/tokenization CNN used by Dessurt was inspired by the CNN component of a CNN-LSTM [?]. We originally pre-trained the CNN as part of a line recognition OCR model, but found that this did not improve training (it could learn just as well from scratch with small images). The CNN down-samples the input image by a factor of 8, meaning the input visual tokens of Dessurt are $144 \times 96 = 13,824$ total. This CNN has 7 convolution layers detailed in Figure 1. We found having an aggressive down-sampling on the computationally light CNN (as opposed to Swin layers) was vital for being able to fit our model in memory when running on large images.

Dessurt has 8 full cross attention layers layers and 2 more which only update the textual tokens (using the last visual tokens). We down-sample the visual tokens (as Swin [?] does) after the first 4 layers. The initial width of the visual tokens is 128 and the becomes 256 after the down-sampling. We note this is quite

Fig. 1. Architecture of the visual encoder. Each convolution layer (orange, yellow, red) except the last is followed by Group Norm [**?**], dropout, and ReLU. The last convolution is followed by Layer Norm. All yellow convolution layers have 3x3 kernels

small and was necessary due to the large image size we process. The width of the textual tokens is 768. The reverse-bottleneck of the fully connected layers for the textual tokens goes up to 3072. The model processes a maximum of 20 query tokens and 800 response tokens. The response must be long for form parsing, where the entire document is predicted.

Swin [**?**] doesn't use 2D position embedding, but instead relies on relative position attention bias. We include 2D spatial embedding because the textual tokens are attending to the visual tokens and may need location information.

2   Pre-training Procedure

2.1   IIT-CDIP Dataset

There isn't a standard OCR for the IIT-CDIP [**?**] used by researchers and it hasn't been investigated what impact this might make in pre-training.. We processed the dataset using Tesseract[3], an open source OCR engine. Tesseract makes many errors and doesn't capture the layout of the documents very well. We perform some post processing including rotating the image upright and attempting to extract the block or paragraph structure by doing layout analysis using Publaynet [**?**] and PrimaNet [**?**] models available on LayoutParser [**?**]. We will make try to make our OCR results available for other researcher as the dataset is quite large and this is a very long process.

To check rotation, we examine the average confidence score C_{mean} returned by Tesseract as well as the average width-to-height ratio for the returned word boxes $\frac{width_i}{height_i} = R_i$. If $C_{mean} > 80$ and $R_{mean} > 1$ we assume the rotation is correct. If not we then run Tesseract on $90°$, $270°$, and $180°$ rotations of the image. If any passes the before-mentioned threshold, we accept it as the correct rotation. If none do, we compare the product score $C_{mean}R_{mean}$ of the

---

[3] https://github.com/tesseract-ocr/tesseract

four rotations and accept the one with the largest product score above 55. If none have a product score above 55, the image is removed from the dataset. This removes images without words and images Tesseract particularly struggled with. When we accept a rotation, we not only use that OCR result, but also use that rotated image in the dataset.

We discard Tesseracts block and paragraph groupings. We then run both the Publaynet and PrimaNet models on the image and append the returned bounding boxes. We remove "super" layout boxes that are superfilous by removing any bounding box covering more than 90% of the text boxes when most text lines overlap with multiple layout bounding boxes. We then go though each text line and assign it to a layout bounding box based on how many lines also overlap with that bounding box and how full the layout bounding box is with text lines. We then collapse the layout bounding boxes to the text lines assigned to them. We then find highly overlapping layout bounding boxes and merge them together. These remaining layout bounding boxes, and any text lines which were not assigned a layout bounding box, are the final blocks used as our layout annotations for the IIT-CDIP data.

Because some tasks are reliant on the block or paragraph structure of a document, and sometimes the extracted block structure is poor, we look at the area of each block covered by its words to heuristically decide if the block structure was accurately extracted or not. For each block we compute height to width ratio (tall is good as it probably has multiple lines) and how much of the block is covered by its text lines (more is better as its dense text). These get averaged, with each block getting weight equal to the number of text lines in it. If this above a threshold, we accept the block structure as good for layout based tasks.

If not stated otherwise, the model is supervised to predict a pixel mask for whatever text it reads (outputs). Wherever text is removed from the document image, has a -1 on the input mask (also called highlight mask).

We now list all the tasks for pre-training with the IIT-CDIP dataset, and their frequency. One can note in the provided Figures many errors resulting from our OCR and layout analysis steps. Tasks with "*" require good block annotation.

- 66.1% Text Infilling (Fig. 2): This is a MLM task inspired by the text infilling used to train BART [?]; however instead of replacing the removed text with a blank token, we delete them from the image, replacing them with white. The area is marked so the model knows something was removed and the entire text block is highlighted. The model then must predict the text of the entire block, filling in the blanked regions. This is easier than the infilling task used by BART for two reasons: (1) the length of what should be filled in can be approximated by the physical blank-space, and (2) we do not allow a blank area of 0 tokens (inserted blank token).
- 16.5% Word Infilling (Fig. 3): This is a potentially more difficult MLM task. A single word in the document is removed and the model must predict that word. In the above task the model is forced to place the text in context by

generating the entire block. For this task it predicts the word in isolation, and thus must capture language context in its hidden states somewhere.

- 4.1% Place Word (Fig. 4): A different flavor of MLM. Several words, of roughly the same length, are removed from the document image. The query contains one of the removed words. The model must predict a pixel mask at the location(s) the given word occurs.
- 4.1% Highlight Block* (Figs. 5 and 6): The query contains a small snippet of text (and randomly the text is highlighted in the input) and the model must predict a pixel mask covering all the words in the block the text belongs to. This is intended to teach document layout.
- 3.7% Read On* (Figs. 7 and 8): The query contains a short text snippet (and randomly the text is highlighted in the input) and the model is to read starting after that text, following newlines, until the end of the block. This teaches text recognition from both a finding and reading standpoint.
- 2.1% Get Blanked (Fig. 9): The query has a snippet of text, but one word is replaced by a blank token. The model must read the word that fits in the blank token. It randomly has the text snippet highlighted or not.
- 2.1% Re-read Replaced (Fig. 10): The model is given a snippet of text, but one word is replaced by a random word of the same length. The model then must read the text using the correct word. It randomly has the text snippet highlighted or not.
- 0.4% Highlight Text (Fig. 11): The query has a snippet of text and the model predicts a mask for it.
- 0.4% Read Highlight (Fig. 12): A text line is highlighted for the model to read.
- 0.2% Read Line Above: The query has a snippet of text and the model must read the text line above it.
- 0.2% Read Line Below: The query has a snippet of text and the model must read the text line below it.

IN: N/A

OUT: American-blend market and 80 percent filter," says Jensen. "As | said, we are presently the leader of every important segment. We led the development of the low 'tar' category which will account for 20 percent in 1981."

Fig. 2. Example Text Infilling task: Magenta is highlight, turquoise is deleted text.

IN: N/A

OUT: interest

```
3.  Number of reprints (Papers in prepar
4.  Number of manuscripts 2
5.  Progress Report No. 2 covering perio
    (8 page)
6.  Progress Report Abstract
7.  For your ████    (a) Progress Repor
    period 1/1/91 - 12/31/91 (b) 2 artic

AVID STONE

S/mla
```

Fig. 3. Example Word Infilling task: Turquoise is deleted word.

IN: studies

OUT:

```
DATE:    August 29, 1985

SUBJECT: ACCEPTABILITY OF BARCLAY AND OTHER B&W PRODUCTS AMONG BARCLAY
         KS AND 100 SMOKERS AT DUPONT/292

In the past ████ several Dupont acceptability ████ evaluated how
BARCLAY KS and BARCLAY 100 smokers rated BARCLAY and other B&W lights
and full-flavor products.  These are ████ products as well as new
████

In the tests where BARCLAY ████ was evaluated, ████ was found to
be most acceptable.

Attached are descriptions and tar deliveries of the products.

The Dupont study I.D.s were 5DE1, 6DE1, 6DF1, 7DF1, 2DG1, and 7DH1.

A. LK.

A. L. K.

ALK/vr

1557k
```

Fig. 4. Example Place Word task: Turquoise is deleted words, green is GT output mask.

IN: of the 10mm gections was almost completely

OUT:

```
internal shape in the vacuum chamber were used,
and a vacuum was induced in the vacuum chamber
(Figure 5).  The plugs were carried by the belt
to the vacuum spacing wheel position satisfactorily
as long as they were removed from the plug drum
grooves.  However, the plugs were removed from the
grooves in a random order.  They tended to stick
in the grooves.  The plug support portion of the
section cutter was adjusted to decrease its
pressure on the plugs.  This onlypartially eliminated
the sticking problem.

Test #2 - Test #1 was repeated except that sectioned 60mm
           filter plugs were used.  The sectioned plug also
           tended to stick in the grooves of the drum.  In
           fact, they did not come out as readily as the
           unsectioned plug.  When the 10mm sections did clear
           the plug drum they tended to scatter randomly on
           the belt.

Test #3 - Test #1 and #2 were repeated except that no vacuum
           was induced in the vacuum chamber.  Fewer plugs
           were removed from the plug drum.  However, the
           scattering of the 10mm sections was almost completely
           eliminated.  The scatter that did occur appeared
           to be the result of the belt being not flat.
```

Fig. 5. Example Highlight Block task: Yellow is GT output mask.

IN: | understand that your Kool is made in Macan and, unlike our US DF offering, requires a Health

OUT:

Fig. 6. Example Highlight Block task (with input highlight): Red is highlight input, Yellow is GT output mask. Notice this has a block segmentation error.

IN: Tax and Budzet bills raised the tobacco excise tax by 10-cents per pack of cigareties\in the year 2000, and an additional S-cent per

OUT: pack in 2002. Wholesalers are opposed to any\additions] increase in the tobaceo excise tax.\This sxcise tax increase meags on the date that thess higher texes go into effect, wholeselers\will have o pay 2 floor stock tax on any whacca products held in inveatory. This is a fiscal\penalty that cannot be recouped by the whelesaler.‡

Fig. 7. Example Read On task: Yellow is GT output mask.

IN: will be located at the end of their pier. The Living Classroom Foundation has plans to\build a new Environmental Education Pavilion and Observation Tower, also at the end of their
RESPN: pier (See\article at the end of this section)‡

OUT: pier (See\article at the end of this section)‡

Fig. 8. Example Read On task (with input highlight): Magenta is highlight input, Yellow is GT output mask.

IN: B DORAL's .3 point Share of Segment loss accounted for ø of the decline

OUT:

Fig. 9. Example Get Blanked task: "∅" is blank character. Magenta and red are highlight input. Red is GT output mask.

IN: three months\each year (Grade II symptoms), was six times that in non-smoking men; the\prevalence of these symptoms in a lesser degree (Grade I symptoms) was only\twice as large, the Institute says. Among women, cigarette smokers had three\times as much Grade II cough and phlegm and twice as much Grade I as non-smokers.,\Neither ex-smokers nor pipe end cigar smokers differed ethologists from the\non-smokers except in the oldest age group vwhere the proportion with Grade II\, symptoms was less in non-smokers. The prevalence of symptoms increased with\ege in men vho smoked cigarettes; no trend of this type wes seen among non-smokers.

OUT: three months\each year (Grade II symptoms), was six times that in non-smoking men; the\prevalence of these symptoms in a lesser degree (Grade I symptoms) was only\twice as large, the Institute says. Among women, cigarette smokers had three\times as much Grade II cough and phlegm and twice as much Grade I as non-smokers.,\Neither ex-smokers nor pipe end cigar smokers differed appreciably from the\non-smokers except in the oldest age group vwhere the proportion with Grade II\, symptoms was less in non-smokers. The prevalence of symptoms increased with\ege in men vho smoked cigarettes; no trend of this type wes seen among non-smokers.

Fig. 10. Example Re-read Replaced task: Yellow is GT output mask.

IN: to regulation as a drug

OUT:

Fig. 11. Example Highlight task: Yellow is GT output mask.

IN: N/A

OUT: suitable management program for these insects.‡

Fig. 12. Example Read Highlight Text task: Magenta and red are highlight input. Red is GT output mask.

Conservative Party
Tom Ognibene - former Queens City Councilman (1992–2002) and Republican minority leader who supported tax cuts, education reform and opposed Mayor Bloomberg's smoking ban. Ognibene was endorsed by the leaders of the Queens County Republican Committee on February 10, 2005 and was expected to win the endorsement of the Conservative Party . He received 8,100 signatures, 600 more than the necessary 7,500 signatures to appear on the primary ballot. However, the Bloomberg campaign challenged many signatures, leaving Ognibene with 5,848 eligible signatures. Ognibene ran as the Conservative Party nominee. He challenged the Republican nomination in a hearing on Thursday, August 25. but lost.
Green Party
Anthony Gronowicz - Green Party candidate, and history professor at Borough of Manhattan College, who sought to strengthen affordable housing, supported renewable sources of energy and sought to provide free tuition to City University of New York. He was featured in articles on third party candidates in The Village Voice and in The Villager in 2005.
Libertarian Party
Audrey Silk - former police officer, community activist and founder of NYC Citizens Lobbying Against Smoker Harassment, nominated by the party on April 16, 2005. She supported lowering parking fines.

Oisseau-le-Petit is a commune in the Sarthe department in the region of Pays-de-la-Loire in north-western France.

See also

Communes of the Sarthe department

Snorre s platform is located in the northern part of the field and is a semi-submersible integrated drilling, processing and accommodation steel facility. Oil from Snorre s is piped 45 km t Statfjord s platform for storage and export.
The Snorre field is operated by Statoil. In 2004, Statoil started a project to upgrade the offshore production complex. The Norwegian Petroleum Directorate is requesting Statoil to build a new platform at the field.

Marriage and children

In 1815, she married Nathan Tyson the son of Elisha Tyson, a Quaker and abolitionist of Baltimore, was the Baltimore Chamber of Commerce's first president. He was also the first president of the Baltimore Corn and Flour Exchange. He had a "gracious love story" with his wife and they had a relaxed attitude about some Quaker conventions. Tyson was described as a "woman of much sweetness and dignity of bearing, possessed of an exceedingly cultivated mind and many accomplishments."

Valley View Airport , is a privately owned, public use airport located northeast of Estacada in Clackamas County, Oregon, United States.

ACIDIC C92+ STORE
THIS IS A BLANKET TERM THAT ENCOMPASSES A SPECTRUM OF ACIDIC VESICLES THAT INCLUDE ENDOSOMES, LYSOSOMES AND LYSOSOME-RELATED ORGANELLES AND SECRETORY VESICLES AND ACIDOCALCISOMES. THEY ARE A HIGHLY DYNAMIC CONTINUUM OF VESICLES WITH A RICH VARIETY OF ESTABLISHED BIOCHEMICAL ROLES IN CELLS TO WHICH C92+ STORAGE CAN NOW BE ADDED THEIR LUMINAL PH IS ONE CHARACTERISTIC THAT DISTINGUISHES A GIVEN VESICLE CLASS FROM ANOTHER WHERE ENDOSOMES ARE WEAKLY ACIDIC (PH 6-6.5). LYSOSOMES ARE TYPICALLY THE MOST ACIDIC (PH 4.5-5.0) AND SECRETORY VESICLES ARE TYPICALLY PH 5.5. C92+ IS SEEN TO BE INCREASINGLY IMPORTANT FOR ENDO-LYSOSOMAL FUNCTION E.G. TRAFFICKING AND AUTOPHAGY ABERRATIONS IN C92+ SIGNALS CAN HAVE PATHOPHYSIOLOGICAL CONSEQUENCES INCLUDING LYSOSOMAL STORAGE DISEASES SUCH AS NIEMANN PICK C AND MUCOLIPIDOSIS IV

WHEN NAADP MOBILIZES C92+ FROM THESE STORES THE PH OF THE STORES CONCOMITANTLY INCREASES (BECOMES MORE ALKALINE) AS TESTIFIED BY STUDIES IN SEA URCHIN EGG MAMMALIAN HEART AND PANCREAS WHETHER THIS HAS CONSEQUENCES FOR VESICLE (OR NAADP) FUNCTION REMAINS TO BE SEEN BUT LUMINAL PH IS USUALLY CRUCIAL FOR RESIDENT PROTEIN ACTIVITY

Beechcraft Super King Air
Bombardier Global Express
Dassault 328JET
Embraer ERJ-145
Fokker 70
Learjet 55

Fig. 13. Examples of synthetic Wikipedia documents

2.2   Synthetic Wikipedia

We first detail the paragraph generation method used in creating the synthetic documents. We then discuss the collected font database in more detail. We also include additional synthetic document examples in Fig. 13.

Details on paragraph generation The document begins as a blank image the same size as our model input.

The column width is sampled in the range of the whole image width to 1/5 of the image. The text height is from the range of 8 to 32 pixels. We note that at 8 pixels, many fonts are illegible. When rendering text, we estimate the maximum height of that font by generating a placeholder string with ascenders and descenders, and the scale to resize this placeholder string to the selected text height is the scale used to resize the actually rendered text. We predict spacing based on an approximated Em at 1.6 times the text height[4], and then the minimum and maximum (horizontal) space as 0.2 to 0.5 times the Em[5]. The newline spacing is sampled between 1 pixel and the text height.

Each word is generated individually and then they are arranged in paragraph form, placing words in a line until the column width is reached and then wrapping onto a new line. There three different paragraph formats selected with the following probabilities: indented 80%, no indent 18%, inverse indent 2%. On an intended paragraph format we select and indent length from 0.3 to 6.0 times the Em and each first line of a paragraph is indented accordingly. For no indent, extra space is added at a newline, randomly from 0 to the selected newline height, whenever a new paragraph is starting. For inverse indent, all lines except the first are indented. When starting a newline, we randomly add a perturbation indent, from 0 to the horizontal space width, to add noise to the process.

If the height of the rendered article exceeds the image height, we increase the column width (and resample the horizontal, newline, and indent spacing) and replace the words.

Articles are repeatedly added to the image until one cannot be placed.

Font database The 10,566 fonts we scrape from 1001fonts.com are not curated and so some fonts are not actual text fonts (Wingdings-like). Many don't include numbers and/or punctuation and some have only upper case letters (the BART tokenization [?] we use is case sensitive). We test fonts to automatically determine some these features and take them into account when rendering. When we randomly select a font, if the selected font does not have numbers another font with numbers is select and is used whenever a word has a number. If the selected font has only uppercase, all GT text is converted to be uppercase.

There are a wide variety of fonts, including handwritten and stylized fonts, but we did not track metadata when scraping, so we don't metrics on the

---

[4] `https://en.wikipedia.org/wiki/Em_(typography)`

[5] `https://docs.microsoft.com/en-us/typography/develop/`
`character-design-standards/whitespace`

database's distribution. However, we do render 949 fonts in Fig. 14 as a qualitative sample. Some of the fonts are variants of others (bold or italicized).

All the code for scraping and pre-processing the fonts will be included in our released code.

2.3   Synthetic Handwriting

The full line handwriting synthesis method we use [?] to generate handwriting does not use a random style vector as input, but rather a distribution from the ones extracted from the data. We interpolate styles extracted from the IAM training set, the "Random" option in the generation script provided by the authors of [?].

We note there are more realistic handwriting generation works more recently developed, but [?] is the only one to generate full lines and has a convenient script in its released code for generating a dataset like this.

For the full page recognition task, in training half of the instances have the handwriting lines highlighted.

Additional examples of documents with synthetic handwriting can be seen in Figure 15.

2.4   Synthetic Forms

Here we include the details of the generation of label-value sets with GPT-2, and how they are rendered into documents. We also include additional examples of generated images and their parse JSON in Figs. 16 and 17.

GPT-2 generation details The typical process for generating text with an autoregressive model is to intialize the text generation with a prompt. In our case we use both a text prompt and a "structure" prompt, which is an example of what format the label-value pairs should be in. The text prompts used are:
  – "This form is to be filled out."
  – "This form has been filled out."
  – "Form X" where X is replaced half of the time with a random letter and the other half with a letter followed by a random integer less than 10,000
  – "This form is to be filled out regarding Y." where Y is replaced with a random Wikipedia article title
  – "This form contains information about Y." where Y is replaced with a random Wikipedia article title

We draw from the pool of generated labels and label-value pairs for the structure part of the initialization for a generation run. These pools to not contain duplicate entries. Because labels with numbers can have several thousand "near duplicates" we limit the number of labels with numbers to be about 0.002 of the pools. We initialize the label-value pool with 60 instances of "Date: X", where X is a random date with one of 6 formats. We initialize the label pool with: "Name:", "Location:", and "Details:"

Fig. 14. Rendering of the word "Dessurt" in 949 fonts from our database at a text height of 16

Fig. 15. Four rendered pages with synthetic handwriting

Fig. 16. Examples of synthetic form documents and their JSON parse

Fig. 17. Examples of synthetic form documents and their JSON parse. On the lower image we see GPT-2's degenerate repeated text.

To generate a label-value set, we first sample a text prompt, a label-value pair, and a label and compose them as the input for GPT-2 using Huggingface's interface[6]. We use a temperature of 0.85 and generate three outputs. For each output, we parse it into label-value pairs until the parsing fails (it will frequently degenerate/stop generating a form). The parsing attempts to prevent repeated values from being added (a frequent degeneration of autoregressive models) and will also parse a comma separated list of values. List values are generated in vertical/newline separated format when creating a synthetic form.

We generate 813,793 label-value sets, with over 7 million total label-value pairs.

We note that we accidentally split URLs into label value pairs (with the label of "http"). We filter these out in the document creation process.

Form generation details A synthetic form is generated by repeatedly adding a label-value set or table in an empty region of the image. After each empty region has had a failed generation, the document is complete. One empty region is the area of the document right of the rightmost content. Whenever a table or label-value set is added, a new empty region is created underneath it spanning the same horizontal space. Each label-value set is generated to fit the region it is being generated in, so this process attempts to pack the form densely.

Label-value set: There are three possible fonts selections, for the header, labels, and values, however 30% of the time the label font will be forced to be the same as the header font, and 50% of the time the value font will be forced to be the same as the label. This is to make the parsing more difficult, and does reflect a frequent scenario in the FUNSD dataset [**?**]. All labels and all values in a set will be rendered with the same respective font.

In 0.5% of rendered label-value pairs, we replace all values with binary checkboxes. The are rendered with boxes, parentheses, or brackets (depending on what the font has) and an 'X' or blank value.

A block width is randomly selected, but will be increased if the generation fails to place any label-value pairs. If the placement fails at the maximum width for the empty region, the region has a failed generation.

A uniformly random selection is made between 9 different relationship indicators which determine how the label-value pairs will be rendered in relation to one another. These are the possible relationships:
- Colon: A colon is added to the end of the label. See Fig. 18 (a)
- Line: An underline is added beneath the value (or a blank area). Line thickness randomly selected per pair. See Fig. 18 (c)
- Colon+Line: Both of the above
- Dotted line: A dashed or dotted underline. Frequency of dotting randomly selected.
- Colon+Dotted line: See Fig. 18 (b)
- Box: The value is put in a box. Thickness of box lines is randomly selected per pair. See Fig. 18 (d)

---

[6] https://huggingface.co/gpt2

– Colon+Box
– To Right: The values will be to the right of the label with the values and labels aligned horizontally and no other cues. See Fig. 18 (e)
– To Left: The value will be to the left of the label (instead of right), there will be a line or box, and the values and labels will be aligned horizontally. See Fig. 18 (f)
– Below: The label will be below the value (instead of above), with an single line separating the value and label. See Fig. 18 (g)

The value will be randomly to the right of or below the label for a set (except for To Right, To Left, and Below). If the values are to the right, it is randomly choosen whether they will align horizontally (they start at the same x-position), or not (except in To Right and To Left when it is always aligned).

When placing label-value pairs, there is a probability (which increases with the number of pairs in the column) to start a new column (if there is room horizontally to due so). If the column reaches the bottom of the image a new column is started, unless there isn't horizontal room, in which case the generation of the label-value set ends.

If the label-value set has a header is is either placed at the top-left corner or the top-middle of the label-value set, having a 50%/50% chance. If the header is going to be placed at the top-left corner, it has a 50% of having the label-value pairs begin after it's horizontal position (instead of it being above them).

The placing of the text is done in largely the same manner as the synthetic Wikipedia text.

Table: There is a 33% chance a header is added for the table. This is 1 to 6 random, non-stop words[7] from Wikipedia. A random font and text height are selected for the header, the row and column headers, and the cell text. A random number of rows (range $[2, 15]$) and columns (range $[2, 10]$) are selected. For each header, a length is selected: 81.4% one word, 18.6% two words, 6.9% three words, 2% four words. That number of non-stop words are randomly sampled from Wikipedia and appended together to form the header. For each cell, 50% of the time it will be a single non-stop word sampled from Wikipedia, the other 50% will be a number with one of the following ten formats (uniformly sampled) displayed in Table 1.

The headers and cells are then arranged in a table with some random spacing. The row headers are always on the right of the cells and the column headers are always above the cells. We leave 15% of cells blank. If the table exceeds the space available, the table generation fails (this protects against a bias towards having tables with few rows/columns).

We then draw the lines of the table. All lines have random thickness. There is always a line separating the headers from the cells. It is randomly deterimined to draw lines between cells and on the outside of the table. Each line is randomly placed (not parallel) in the space availble between the table elements.

---

[7] We use the stop words listed at https://www.ranks.nl/stopwords

Fig. 18. Examples of label-value relationships. (a) Colon, (b) Colon+Dotted line, (c) Line, (d) Box, (e) To Right (with header), (f) To Left, (g) Below (with header)

Table 1. Number formats for table cells

Description	Example
Integer in range [0,100]	16
Integer in range [0,9999]	4567
Integer in range [-999,999]	-453
Percent	45%
Percent with decimal	23.45%
Decimal in range [0,100)	15.87%
Decimal in range [0,1)	0.834
Negative decimal in range (-1,0]	-0.452
Dollar amount in range [0,9999]	$2567
Dollar amount in range [0,999]	$754

Training tasks We define several tasks for these forms, however the Parse to JSON tasks is the most important, as this is also an end task we evaluate on the FUNSD [**?**] and NAF [**?**] datasets. We will first detail our JSON format and then list all the tasks.

The JSON format was specifically designed to be easy for an autogressive model to predict. The format must capture the FUNSD dataset labeling, including classes and relationships, in addition to tables which we predict differently.

In general, an instance is represented as a single JSON object:

```
{"entity text": "class"}
```

This allows the model to read the text before deciding the class, and during training ensures the model is predicting the class for the right entity. If a header has links to other entities, they are listed as contents, e.g.:

```
{"Title Text": "header", "contents":[{"Q1": "question"}, {"Q2":
"question"}]}
```

If an answer has links, these are handled as answers, e.g.:

```
{"Question text": "question", "answers":["A1", "A2"]}
```

We list the answers as strings instead of objects as they should have nothing linked below them in the hierarchy and this is a more compact representation. Tables are an object with row headers, column headers, and cells, where the cells are a nested list in row major order, e.g.:

```
{"row headers":["R1", "R2"], "column headers":["C1", "C2"],
"cells":[["r1 c1", "r1 c2"], ["r2 c1", "r2 c2"]]}
```

We write out the elements in read order, treating a table, or a header with all its sub-elements as a single element. The read order is determined by first ordering the elements by verticle position. We then take the top element and find all other elements which fall inside a horizontal range slightly above and below it. This is intended to be elements on roughly the same horizontal line, taking

large elements (like tables) into account (lots of things can be parallel to them). If the current element is the left most, it is the in order, otherwise, the elements to its left as place before it and they are evaluated with their own horizontally parallel elements. This process makes the read out be roughly natural for how a human might read around blocks like tables.

We note, it would be more efficient and probably more accurate to have defined special tokens for the control characters of the JSON, but we did not do this.

Here is the list of all tasks used in training on the form images:

- (48.2%) Parse to JSON: The document is reproduced in a special JSON format which captures structure as well as the class of thee entities. Examples of the JSON can be seen in Fig. 16 and 17. There are two possible queries, one to parse the document from the beginning, the other includes some portion of the JSON in the query and the model must parse starting from that point of the JSON (similar to the Read On task). This is neccesary as many forms have a JSON longer than the model's longest output (800 tokens)
- (4.02%) Link All: The query contains a form entity either by text, highlight, or both, and the model is to predict the class of the entity and read the text of all entities it is linked to.
- (4.02%) Link Down: Same as the above task, but only read text of linked entities down the hierarchy
- (4.02%) Link Above: Same as the above tasks, but only read text of linked entities up the hierarchy
- (4.42%) Cell: The query contains the texts of a row and a column header and the model must read the corresponding cell
- (4.42%) Row Header: The query contains a cell's text and the model must read the row header
- (4.42%) Column Header: Same as the above task by reading the column header
- (4.42%) All Row Cells: The query contains text for a row header and the model must read all the cells in the row.
- (4.42%) All Column Cells: Same as above for column
- (4.42%) All Row Headers: The query contains a number i and the model must read the row headers for the ith table in the document
- (4.42%) All Column Headers: Same as above for columns
- (3.61%) Count Tables: The model must return the number of tables and predict a mask covering them.
- (4.42%) Highlight Table: The query contains a number i and the model must predict a mask for the ith table
- (0.402%) Not Present: One of the above tasks with a specific query is given, but the entity in the query isn't on the document. The model must respond with a not-present token
- (0.402%) Read On: The query as some text and the model must read on from that text to the end of the entity it belongs to

2.5   Selection of "Easy" Fonts for Distillation

We score each font by rendering the following strings in the font: "abcdefg", "hijklmn", "opqrst", "uvwxyz", "12345", "67890", "ABCDEFG", "HIJKLMN", "OPQRST", "UVWXYZ" We then run Tesseract over on these images and compute the edit-distance between the Tesseract output and the image's source string. The sum of these edit-distances become the score for that font. All fonts with a score less than 21 are used as our "easy" fonts. This may seem like a high threshold, but the word images passed to Tesseract are not padded (text generally extends to the end of the image) which is a domain that Tesseract struggles with.

There are 586 fonts in our "easy" set, and they can be seen in Fig. 19.

2.6   Pre-training Curriculum Details

It has been noted by [?] that (billion parameter) autoregressive models have training stabilized by a sequence length based curriculum. This may be related to the success of our curriculum.

The small image pre-training uses the following tasks with uniform probability:
 − Get Blanked
 − Re-read Replaced
 − Highlight Text
 − Read Highlight
 − Read On

The reading pre-training on full sized images uses the same tasks as normal training, but with uniform probability. As there are more reading focused tasks, this step of the pre-training is focused on teaching reading.

During the main pre-training, the datasets are not sampled uniformly. We assume that some are more important than others. For the final model they are sampled with the following frequency:
 − IIT-CDIP: 45%
 − Synthetic Wikipedia: 29%
 − Synthetic Handwriting: 1%
 − Synthetic Forms: 5%
 − Distillation: 20%

For the ablation experiments, the models using all datasets has the given frequencies, with all others having the same ratio between the datasets they do have:
 − IIT-CDIP: 53%
 − Synthetic Wikipedia: 35%
 − Synthetic Handwriting: 2%
 − Synthetic Forms: 7%
 − Distillation: 2%

The changed frequencies used for the final model reflect the uncertainty the ablation showed regarding the importance of the distillation.

Fig. 19. All "easy" fonts rendering the word "Dessurt" at a text height of 16

3   Data Augmentation

Images of all datasets are scaled to match the size of Dessurt's input. In all our pre-training and fine-tuning, we apply some basic image augmentations. For most datasets, we randomly re-scale the image to 0.9-1.1 its original size, sampled uniformly. The exception is the census data which is scaled in the range [1,1.15], and the FUNSD dataset, which has the range [0.8,1.2]. If the image is larger than Dessurt's input size (due to a re-scale), we randomly crop a region form the image of Dessurt's input size. We apply a random rotation from the normal distribution with a standard deviation of $1°$.

We apply brightness augmentation which adjusts the brightness and contrast between background and foreground for the synthetic handwriting, synthetic forms, and IAM full-page recognition. The method is the same as used by Tensmeyer et al. [?], but we use $\sigma = 20$.

4   GAnTED

Here we describe the greedy-aligned normalized tree edit-distance (GAnTED), the metric we use to evaluate form parsing.

This is simply a greedy optimization of the nTED [?] metric done by permuting child lists of the predicted tree. This is neccsary as there is not a canonical ordering for forms. While we create the parse JSONs in a read order, it can often appear ambiguous which elements should be read first. Additionally, the order of the elements should be irrelevant to the information extracted.

The process we use is quite simple, if somewhat slow. We first convert the JSON into a tree. We discard class information in this process. A header will have it's content as children, and a question will have it's answer(s) as a child/children. For tables, things are handed a bit differently. We could have the list of cells in each row be children of its respective row header (the column headers have no children), or have the columns of cells be children of the column headers (the row headers having no children). While the model is trained to predict row-major tables, we note that often errors are made where a table is not recognized as such, and thus the header-cell relationships are predicted instead. Our table annotation of the FUNSD dataset is heuristic (see Section 5.4) and sometimes erroneous leading to such label-value relationships in the GT. Thus we compute the GAnTED for all combinations of table-to-tree conversions and take the minimum score.

We use the variant of TED where the relabel cost for the nodes is the normalized Levenshtein distance between the predicted string and the GT string. This means the recognition errors should be balanced in relation to structure errors.

The alignment is done in a breadth first traversal of the predicted tree. At each node, we compute the nTED for the entire tree when the node is moved up to 10 positions forward or backward in its list of children. We then place it in the position that gave the minimum score. Each node gets re-positioned once in this

Table 2. nTED, GAnTED, GAnTED with two aligment passes on the FUNSD and NAF datasets

	FUNSD			NAF		
	nTED	GAnTED	2-GAnTED	nTED	GAnTED	2-GAnTED
FUDGE [?] w/ Tesseract	59.1	34.8	34.5	-	-	-
Dessurt (scrambled)	81.4	35.8	32.0	-	-	-
Dessurt	44.1	23.4	23.2	80.4	42.5	42.1
Dessurt w/ census train	-	-	-	73.0	38.8	38.3

process. After each node is re-positioned, the final nTED score is the GAnTED score.

This is clearly not optimal, but given that the model attempts to predicted in read order, it is quite stable, only changing the GAnTED slightly if the alignment is done again.

In Table 2 we show the the nTED score, GAnTED score, and the GAnTED score when the alignment is done twice. As can be seen, the greedy alignment dramatically improves the nTED score, likely giving much accurate measures of a model's performance at form parsing, not just how well it matches the order of the GT. We also evaluate computing GAnTED on Dessurt's results when each set of children in it's tree are randomly permuted. This leads to decreased performance and less stability, indicating that an approximate read order should be established before computing GAnTED. We feel this should be reasonably easy to do under most situations.

5   Experiment Details

For each dataset we fine-tune the long-pre-trained model with a learning rate drop and early stopping based the validation set. We took the parameters with the best validation set performance as the final model.

5.1   RVL-CDIP

This dataset has significantly more data than the others we evaluate on. We drop the learning rate at 175K iterations, but are able to continue training to a total of 1.5 million iterations with continuous improvement on the validation set.

5.2   DocVQA

For DocVQA, we drop the learning rate at 200K iterations and evaluate the model at 380K iterations.

5.3   HW-SQuAD

For HW-SQuAD, we drop the learning rage at 200K iterations and evaluate the model at 970K iterations.

5.4   FUNSD and NAF

The model frequently falls into the common autoregressive degeneration of repeating the same output (generally a JSON object). We counter this by post-processing the output and removing any sequence of at least 8 characters that is repeated consecutively at least 5 times. If the model fails to produce the end token, we use the last predicted tokens to form a new query for the model to parse from. We note that this can allow the model to recover from a repeat degeneration, as often it will continue repeating till the maximum token length, these are removed, and then a new query is made from the end of the non-degenerate prediction. We re-query a maximum of 5 times.

Despite the highly regular structure of our JSON output, the model often fails to produce valid JSON output, especially on more difficult forms. We craft a series of rules to transform various JSON syntax errors into valid JSON, generally favoring a simple, less structured, output. We assume our correction rules don't effect performance significantly as these are generally occurring where the model is making other prediction errors.

During training on the FUNSD [**?**] and NAF [**?**] datasets, we use the same task distribution as the pre-training on synthetic forms. While it may not seem intuitive to training on tasks that are not part of the evaluation, the non-JSON tasks do improve performance, possibly providing a regularizing effect. For the FUNSD dataset, we drop the learning rate at 10K iterations and evaluate the model at 51K iterations. For the NAF dataset, we drop the learning rate at 65K iterations and evaluate the model at 320K and 400K iterations for the normal and census pre-trained model respectively.

Table annotations for FUNSD The FUNSD dataset doesn't contain annotations for tables. However, tables generally show up distinctively in the annotation with values having two labels linked to them. We use this along with various spatial heuristics to determine if a set of links actually comprise a table. It is generally successful, failing on tables where the label-value linking was left incomplete in the FUNSD annotations.

Table annotations for NAF The NAF dataset does contain table annotations, however, the transcriptions for the cells is not present in the dataset. We simply omit the `cells` of the JSON so only row and column headers are predicted. This follows in line with [**?**], which omits tables from it's predictions.

U.S.A. 1940 Census pre-training The census images we pre-train on are publicly available on the U.S.A. National Archive at `https://www.archives.gov`. The training set we use is 10,000 images. And example image is found in Fig. 20. The NAF dataset was also derived from the U.S.A. National Archive and thus the census images represents a very similar domain, although they lack any variation in layout. We ensure no overlap between these datasets.

Fig. 20. Example image from the U.S.A. 1940 census

The proprietary annotations we use contain human transcriptions of select columns of the main table in the document: line number, household ID, full name, sex, age, relationship to head of household, race, and birthplace. In our annotations, ditto marks have been filled in with the respective value, and the model is trained to do this as well. The there are three tasks we use in the pre-training:

– List the full contents of the table, being the above mentioned fields for each row
– List all names: List the names on the document. This is the column with the most variation and we assume most handwriting recognition is going to be learned from this column
– List all ages: List the ages on the document. Similar to the above task, but ensuring the model can read numbers

We crop the images to be only the left-side of the image as the only columns we use are on the left side. This allows the document to fit the aspect ratio of our model better and have higher resolution, which is needed given how dense the handwriting is.

5.5   IAM Database

When scoring our IAM NER [**?**] predictions, we do an alignment between the predicted word transcriptions and the GT words, minimizing the total edit-distance. This allows us to match the class prediction even on words the model did not transcribe correctly.

For the experiment pre-training Dessurt on the IAM dataset for IAM NER, for the last 200k iterations of the pre-training, 47% of the training instances are synthetic documents, each containing two columns of words sampled randomly from three IAM pages (both pages' words are jumbled together). The model must predict the contents of the two columns (full page recognition). By having Dessurt read the words in random order we hope to prevent overfitting on the dataset. Each word is randomly rescaled to a height in the range of 18 to 48

We note that the IAM splits used for IAM NER are not the same splits used to train the handwriting generation method we used in our data creation [**?**]. This means there is a potential information leak of test set data via what the generation model has learned and is using to generate our synthetic pre-trianing data. We feel this would be making a very minor impact on performance especially given the Dessurt's performance on IAM recognition [**?**], which does not have information leakage, is roughly the same as the recognition on the IAM NER splits.