Rosetta: Large Scale System for Text Detection and Recognition in Images*

Fedor Borisyuk
Facebook Inc.

Albert Gordo
Facebook Inc.

Viswanath Sivakumar
Facebook Inc.

ABSTRACT

In this paper we present a deployed, scalable optical character recognition (OCR) system, which we call *Rosetta*, designed to process images uploaded daily at Facebook scale. Sharing of image content has become one of the primary ways to communicate information among internet users within social networks such as Facebook, and the understanding of such media, including its textual information, is of paramount importance to facilitate search and recommendation applications. We present modeling techniques for efficient detection and recognition of text in images and describe *Rosetta*'s system architecture. We perform extensive evaluation of presented technologies, explain useful practical approaches to build an OCR system at scale, and provide insightful intuitions as to why and how certain components work based on the lessons learnt during the development and deployment of the system.

CCS CONCEPTS

• **Applied computing → Optical character recognition**;

KEYWORDS

Optical character recognition, text detection, text recognition

ACM Reference Format:
Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. 2018. Rosetta: Large Scale System for Text Detection and Recognition in Images. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3219861

1 INTRODUCTION

One of the primary ways through which people share and consume information in social networks such as Facebook is through visual media such as photos and videos. In the last several years, the volume of photos being uploaded to social media platforms has grown exponentially to the order of hundreds of millions everyday [27], presenting technological challenges for processing increasing volumes of visual information. One of the challenges in image understanding is related to the retrieval of textual information from images, also called Optical Character Recognition (OCR), which represents a process of conversion of electronic images containing typed, printed, painted or scene text into machine encoded text.

*Authors of the paper have equal contribution to this work.

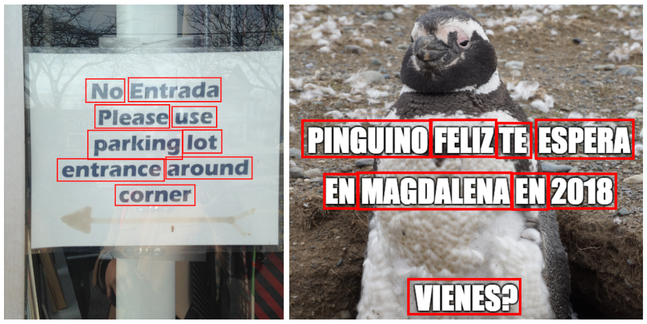


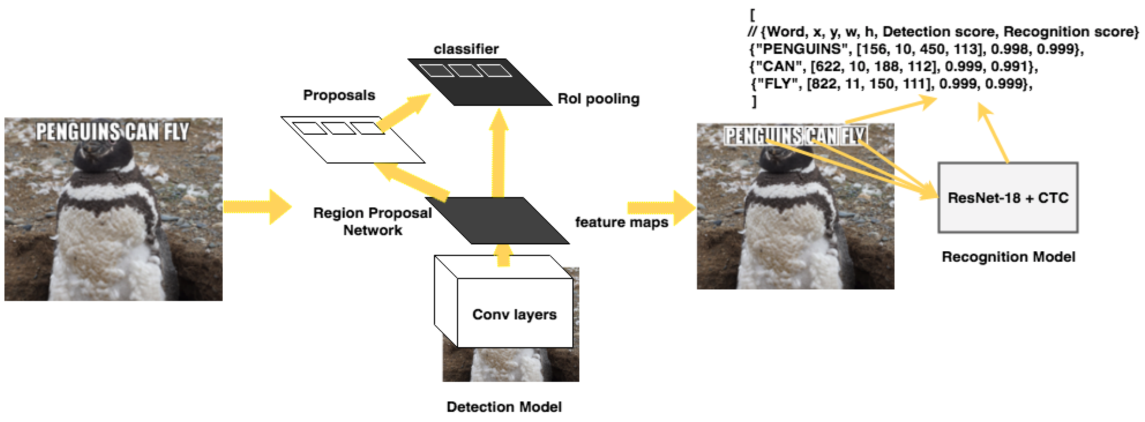
Figure 1: OCR text recognition using *Rosetta* system. An approach based on Faster-RCNN detects the individual words, and a fully-convolutional CNN produces the lexicon-free transcription of each word.

Obtaining such textual information from images is important as it facilitates many different applications, *e.g.* search and recommendation of images.

In the task of OCR we are given an image, and the OCR system should correctly extract the text overlaid or embedded in the image. Challenges to such a task compound as a number of potential fonts, languages, lexicons, and other language variations including special symbols, non-dictionary words, or specific information such as URLs and email ids appear in the image, and images tend to vary in quality with text in the wild appearing on different backgrounds. Another aspect of the problem arises from the huge volume of images in social networks uploaded daily that need to be processed. Due to the nature of downstream applications, we targeted to perform OCR in realtime once the image is uploaded, and that required us to spend significant time optimizing the components of the system to perform within reasonable latency constrains. Therefore, our problem can be stated as follows: *to build a robust and accurate system for optical character recognition capable of processing hundreds of millions of images per day in realtime.*

In this paper we present *Rosetta*, Facebook's scalable OCR system, which we have implemented and deployed in production and that powers downstream applications within Facebook. Following state-of-the-art systems, our OCR is divided into a text detection stage and a text recognition stage. Our text detection approach, based on Faster-RCNN model [24], is responsible of detecting regions of the image that contain text. After that, our text recognition approach, that uses a fully-convolutional character-based recognition model, processes the detected locations and recognizes the text they contain. Figure 1 shows some results produced by *Rosetta.*

The rest of the paper is organized as follows: §2 provides an overview of the related work. §3 describes our detection and recognition models, and §4 describes the design and architecture for *Rosetta*, §5 describes the experimental evaluation and the lessons learnt during



```
[
// {Word, x, y, w, h, Detection score, Recognition score}
{"PENGUINS", [156, 10, 450, 113], 0.998, 0.999},
{"CAN", [622, 10, 188, 112], 0.999, 0.991},
{"FLY", [822, 11, 150, 111], 0.999, 0.999},
]
```

Figure 2: Two-step model architecture. The first step performs word detection based on Faster-RCNN. The second step performs word recognition using a fully convolutional model with CTC loss. The two models are trained independently.

the system development. Finally, §6 describes the deployment and §7 concludes the paper.

2 RELATED WORK

Although the OCR problem has received a lot of attention during decades, most of this attention has been led by the document analysis community and it has been limited to document images (see *e.g.* [23] for some examples). It's only been during the last decade that the computer vision community has paid more attention to the problem of text detection and recognition in natural images, mostly due to the seminal work of Wang and Belongie [29]. Since then the topic has received a significant amount of attention, and large advances in text detection and text recognition in natural images have been produced, mostly driven by the success of convolutional neural networks (CNN).

The works of Jaderberg *et al.* [19, 20] are among the first to propose a CNN based approach for text recognition to classify words into a pre-defined set of dictionary words. Before the recognition stage, a combination of Edge Boxes [33] and an aggregate channel features (ACF) detector framework [11] is used to detect candidate locations for the words. Following Jaderberg's work, many new methods have recently appeared that use CNN architectures to detect words in images, including [16, 21, 25, 32]. CNN based detection methods have the benefit of providing an end-to-end learnable framework for text detection. We follow a convolutional approach as well to detect text in images by adapting state of the art object detection framework based on Faster-RCNN [24].

Similarly, many recent works on text recognition have also followed Jaderberg's work [20] and addressed its shortcomings. In particularly, and contrary to the original work, most recent character-based works do not require a dictionary (also known as lexicon) and can recognize words of arbitrary length that were not seen during training (*e.g.* [12, 22, 26, 30]). In general, this is achieved by using a fully-convolutional model that given an image produces a sequence of features. In some cases, attention mechanisms (unsupervised or supervised) or recurrent models can be leveraged to improve the quality of the features. At training time, a sequence-to-sequence loss such as CTC (Connectionist Temporal Classification) [14] is used to compute the error between the predicted features and the real transcription and to improve the model through backpropagation. At testing time, the sequence of features can be converted into a sequence of characters from where the transcription is obtained, either in a greedy manner, or using language model or a dictionary. Our fully-convolutional recognition model shares the key ideas of these works, *i.e.*, producing a sequence of features that can be trained with CTC and that can recognize words of arbitrary length that do not appear in a dictionary.

To train such models, obtaining high quality training data is of paramount importance. The works of [15, 19, 22] provide elegant solutions for preparation of artificial generated training dataset with hundreds to thousands of images being annotated with printed text combining different fonts, styles and font sizes to provide a variety of data for training robust text recognition classifiers. We have used similar approaches in this work to generate artificial training data for both detection and text recognition tasks.

3 TEXT EXTRACTION MODELS

We perform OCR in two independent steps: detection and recognition. In the first step we detect rectangular regions in the image potentially containing text. In the second step we perform text recognition, where, for each of the detected regions, a CNN is used to recognize and transcribe the word in the region. This two-step process has several benefits, including the ability to decouple training process and deployment updates to detection and recognition models, run recognition of words in parallel, and independently support text recognition for different languages. Figure 2 details this process.

In the following sections we describe our approach in detail.

3.1 Text Detection Model

For text detection we adopted an approach based on Faster-RCNN [24], a state-of-the-art object detection network. In a nutshell, Faster-RCNN simultaneously performs detection and recognition by i) learning a fully-convolutional CNN that can represent an image as a convolutional feature map (typically based on ResNet architectures [17]), ii) learning a region proposal network (RPN) that takes that feature map as input and produces a set of k proposal bounding boxes that contain text with high likelihood, together

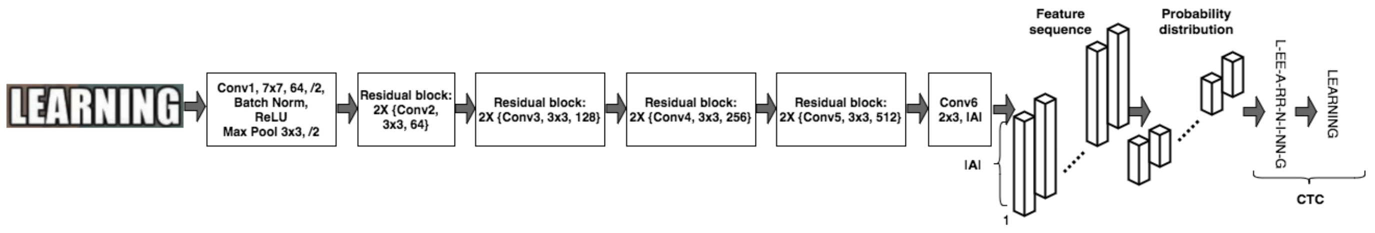

LEARNING
Conv1, 7x7, 64, /2, Batch Norm, ReLU Max Pool 3x3, /2
Residual block: 2X {Conv2, 3x3, 64}
Residual block: 2X {Conv3, 3x3, 128}
Residual block: 2X {Conv4, 3x3, 256}
Residual block: 2X {Conv5, 3x3, 512}
Conv6 2x3, IAI
IAI
1
Feature sequence
Probability distribution
L-EE-A-RR-N-I-NN-G
LEARNING
CTC

Figure 3: Architecture of Text Recognition model.

with their confidence score, and iii) extracting the features from the feature map associated with the spatial extent of each candidate box, and learning a classifier to recognize them (in our case, our categories are text and no text). To select the k most promising candidates, the proposals are sorted by their confidence score, and non-maximum suppression (NMS) is used to choose the most promising proposals. Additionally, bounding box regression is typically used to improve the accuracy of the produced bounding boxes. The whole detection system (feature encoding, RPN, and classifiers) are trained jointly in a supervised, end-to-end manner.

Our text detection model uses Faster-RCNN, but replaces the ResNet convolutional body with a ShuffleNet-based [31] architecture for efficiency reasons. As we show empirically in Section 5, ShuffleNet is significantly faster than ResNet and did not lead to inferior accuracy. The ShuffleNet convolutional trunk is pre-trained using ImageNet dataset [10]. To train the end-to-end detection system, the model was bootstrapped with an in-house synthetic dataset and then fine-tuned with COCO-Text and human annotated datasets to learn real-world characteristics, *cf*. §5.2.

3.2 Text Recognition Model

We experimented with different architectures and losses for text recognition. The first one is based on the character sequence encoding model (CHAR) of Jaderberg *et al.* [19]. The model assumes that all images have the same size (resized to 32x100 in [19] without preserving the aspect ratio) and that there is a maximum number of characters per word k that can be recognized ($k = 23$ in [19]). For longer words, only k of its characters will be recognized. The body of the CHAR model consists of a series of convolutions followed by k independent multiclass classification heads, each of which predicts the character of the alphabet (including the NULL character) at each position. During training, one jointly learns the convolutional body and the k different classifiers.

The CHAR model is easy to train using k parallel losses (softmax + negative cross-entropy) and provides a reasonable baseline, but has two important drawbacks: it can't recognize correctly words that are too long (for example URLs), and the number of parameters in the classifiers is very large, leading to big models that tend to overfit.

The second model architecture is a fully-convolutional model that outputs a sequence of characters. We refer to this model as CTC because it uses a sequence-to-sequence CTC loss [14] during training. The general architecture of the CTC model is illustrated in Figure 3. After the convolutional body, which is based on a ResNet-18 [17] architecture, it has a last convolutional layer that predicts the most likely character at every image position of the input word. Different from other works, we do not use an explicit recurrent network (such as an LSTM or GRU) stage or any attention mechanism, and produce directly the probability of each character. As noted by recent works (*e.g.* [12]), convolutions can still model the interaction between characters while being computationally more efficient at test time.

Algorithm 1 Curriculum Learning for Text Recognition Model

Input: Training dataset D; Number of warm-up epochs W; Number of epochs N; Initial maximum length of words l_0; Warm up word width w_w; Initial word width w_0; Learning rates α and β; Learning rate decay period t
Output: Trained CTC model.

```
1:  Note: The values of learning rates $\alpha$ and $\beta$ are determined
    empirically by observing training convergence. $\alpha$ is the highest
    learning rate that allows one to train the model after random
    initialization without diverging, while $\beta$ is the highest learning
    rate that allows one to train the model after initializing with
    pretrained weights.
2:  Set initial learning rate $lr = \alpha$.
3:  Set log-increment $\Delta = \frac{\log\beta - \log\alpha}{N}$.
4:  Set image width preprocessing to $w_w$
5:  Warmup training:
6:  Set $l = l_0$.
7:  for ($i = 1$; $i \leq W$; $i++$, $l++$) do
8:      Generate filtered training dataset $T$ by keeping only training
        examples from $D$ with labeled word length equal or smaller
        than $l$ characters.
9:      Train CTC model for one epoch using filtered training dataset
        $T$ and learning rate $lr$.
10:     Increase learning rate $lr = \alpha + 10^{i\Delta}$
11: end for
12: Post-warmup training:
13: Set $w = w_0$
14: for ($i = 1$; $i \leq N$; $i++$, $w += 8$) do
15:     Set image width preprocessing to $w$. Generate filtered training
        dataset $T$ by keeping only training examples from $D$ with
        labeled word length equal or smaller than the size of the feature
        map when using width $w$.
16:     Train CTC model for one epoch using filtered training dataset
        $T$ and learning rate $lr$.
17:     Set $lr = lr * 10^{-floor(i/t)}$
18: end for
19: Return model.
```

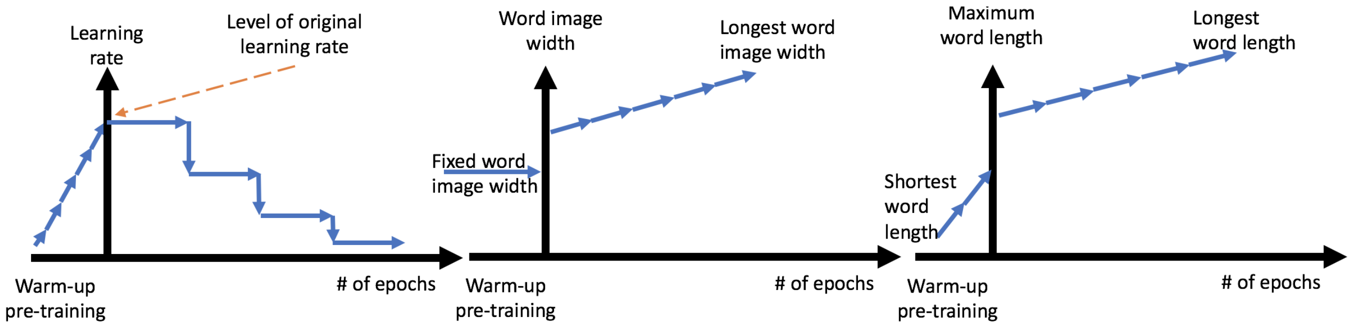


Figure 4: Schematic visualization for the behavior of learning rate, image width, and maximum word length under curriculum learning for CTC text recognition model.

This model is trained using the CTC loss, which computes the conditional probability of a label given the prediction by marginalizing over the set of all possible alignments paths, and that can be efficiently computed using dynamic programming. As shown in Figure 3, every column of the feature map corresponds to the probability distribution of all characters of the alphabet at that position of the image, and CTC finds the alignments between those predictions, which may contain duplicate characters or a blank character (−), and the ground truth label. For example, in Figure 3 we show that for the input training word *LEARNING*, the model might produce the sequence of characters "L-EE-A-RR-N-I-NN-G", which includes blanks and duplicates.

At inference time, computing the optimal labeling is typically intractable. Instead, one typically relaxes the problem: based on the assumption that the most probable path will correspond to the most probable labelling, one can find the best path decoding in linear time by greedily taking the most likely character at every position of the sequence. As a postprocessing, one then removes contiguous duplicate characters not delimited by the blank character.

This CTC model addresses the two problems present in the CHAR model, *i.e.*, it has significantly less parameters (because it does not require k independent fully-connected layers) and can predict words of arbitrary length (because the model is fully-convolutional). We also compared inference time of CHAR model versus CTC model and observed that the CTC model is 26% faster than the CHAR model, mostly due to the cost of the k fully connected layers that perform character classification in the CHAR model. Orthogonal to this, one can use different body architectures. We experimented with ResNet-18 and SqueezeNet [18], and obtained higher accuracy with ResNet-18 while experiencing only a very small overhead in computation.

Image Pre-processing: In most approaches to text recognition (starting with Jaderberg *et al.* [19]), the word image crops are pre-processed by resizing them to 32x100 pixels without preserving the aspect ratio. This leads to same characters looking very different depending on the length of the word, which in turn requires more model capacity to accurately represent. Instead, at training time, we resize the word images to 32x128 pixels distorting the aspect ratio only if they are wider that 128 pixels, and using right-zero-padding otherwise. This ensures that, for most words, no distortion is produced. Setting a fixed width is necessary from a practical point of view to be able to efficiently train using batches of images.

At testing time we resize the images to a height of 32 pixels preserving their aspect ratio (and so they can have an arbitrarily long width). This ensures that test and train images don't have, in general, a large domain shift. In preliminary experiments we noticed this approach to yield superior accuracy than to resize the images without preserving the aspect ratio.

Additionally, it is interesting to note that given a body of the neural network (in our case ResNet-18) and a word image, the number of character probabilities emitted depends on the width of the word image, not on the number of characters of the word (although obviously both are correlated). By stretching all training and testing images by the same, constant factor, we can control the number of probabilities emitted and approximately align it with the length of the transcription, which can affect training convergence and testing accuracy. We empirically found that a stretching factor of 1.2 leads to superior results than using the original aspect ratio.

Training of Text Recognition Model: We firstly train our text recognition models using synthetic data and then fine-tune on the application specific human rated data (described in §5.2) to achieve better results.

Interestingly, we found the CTC model much harder to train than the CHAR model: while the CHAR model converged quite rapidly and was not very sensitive to initial parameters such as the learning rate, the CTC model consistently either diverged after just a few iterations or trained too slow to be of practical use. We speculate that aligning sequences is a much harder problem than learning independent classifiers, and that that's the reason for the model not to train satisfactorily.

To effectively train the CTC model we considered two workarounds. The first one was to initialize the weights of the model body with the trained weights of the CHAR model, and then fine-tune those weights while simultaneously learning the last convolutional layer from scratch. The second approach was based on curriculum learning [6], *i.e.*, starting with a simpler problem and increasing the difficulty as the model improves. Instead of training with all training words directly, we warmed up the model for 10 tiny epochs that considered only very short words. We started training with words of length ≤ 3, for which the alignment would be easy and where the variations in length would be tiny, and increased the maximum length of the word at every epoch. We also reduced the width of the images to simplify the initial problem. Simultaneously, we started with a tiny learning rate and kept increasing it at every epoch until it reached the same initial learning rate of the CHAR model. This idea of increasing the learning rate during warm up has been used in other cases, see *e.g.* [13]. After this warmup standard training would follow, decreasing the learning rate after n epochs. However, even at this stage, we still apply the principles of curriculum learning: at the end of every epoch, besides the reduction of learning rate, we also increase the width of the words [1]. A representation of the changes in learning rate and word length can be seen in Figure 4, while Algorithm 1 describes the algorithm in detail.

Both the pretraining and the curriculum learning approaches worked well and led to very similar quantitative results. In the end, we adopted curriculum learning as our standard approach because of two reasons: first, it was slightly faster to perform the warm up than to train the CHAR model from start to end. Second, and most important, curriculum learning helped us to eliminate the need of training a separate CHAR model and copying the weights to the CTC model, which created an undesirable dependency between the models.

4 SYSTEM ARCHITECTURE

We now describe the system architecture of *Rosetta*, Facebook's realtime large-scale OCR system. *Rosetta* is deployed in production and is designed to operate on images uploaded daily at Facebook scale in a realtime fashion. Figure 5 outlines the high-level architecture of *Rosetta. Rosetta* utilizes a pull-based model where an image uploaded by a client application (step 1 in Figure 5) is added to a distributed processing queue. The inference machines in *Rosetta* pull the enqueued jobs when resources are available and process them asynchronously. Consumers can register callbacks when enqueuing jobs, which *Rosetta* invokes right after each job is processed for immediate usage of results by downstream applications. The processing queue is optimized for high throughput and availability, and utilizes RocksDB [8] underneath for persistence. This pull-based asynchronous architecture provides various benefits including better load-balancing, rate-limiting in scenarios of unexpected spikes in requests (for example, surge in photo uploads on Facebook), and the ability to optimize the system for throughput.

The online image processing within *Rosetta* consists of the following steps:

1. The image is downloaded to a local machine within *Rosetta* cluster and pre-processing steps such as resizing and normalization are performed.
2. The text detection model is executed (step 4 in Figure 5) to obtain location information (bounding box coordinates and scores) for all the words in the image.
3. The word location information is then passed to the text recognition model (step 5 in Figure 5) that extracts characters given each cropped word region from the image.
4. The extracted textual information along with the location of the text in the image is stored in TAO, Facebook's distributed graph database [9] (step 6 in Figure 5).
5. Downstream applications such as Search can access the extracted textual information corresponding to the image directly from TAO (step 7 in Figure 5).

[1]And, as a byproduct, the maximum length of the words that can be used for training, as the maximum accepted length is given by the size of the feature map, which in turn depends on the width of the image words.

In section §5, we show various experiments performed that helped us make the right trade-offs between accuracy of the system and inference speed during the development of *Rosetta*.

5 EXPERIMENTS

We present an extensive evaluation of *Rosetta* OCR system. Firstly we define the metrics used to judge the accuracy and processing times of the system, and describe the datasets used for training and evaluation. We follow the standard practice of training and evaluating our models on separate holdout datasets. We describe the evaluation of the proposed text detection and text recognition models, and explain the design decisions made for accuracy vs inference speed trade-offs. Finally we share various lessons learnt during the development and deployment of *Rosetta* that helped us launch at Facebook scale.

5.1 Metrics

We used a combination of the following accuracy and inference time metrics to evaluate *Rosetta*. We report all metrics relative to the baseline and some of the measurements are reported normalized to the interval $[0, 1]$ by min-max normalization. Most of the reported experiments are presented in the form of ablation study, where we list improvements in metrics over the baseline model configuration or baseline variant of the training data.

Performance of detection model: For detection tasks, the standard metric to measure performance is Mean Average Precision (mAP) at certain IoU (Intersection-over-Union) thresholds. The IoU between a predicted bounding box region and a ground-truth region is defined as the ratio of the area of the intersection of the boxes to the area of the union of the boxes. While precision and recall are single-valued metrics that measure performance at a given threshold, Average Precision (AP) takes into account the order of the predictions as well. Average Precision (AP) measures the area under the precision-recall curve for a given IoU threshold interval. Formulaically, $AP = \sum_n (R_n - R_{n-1}) P_n$, where P_n and R_n are the precision and recall at the n^{th} score threshold. Mean Average Precision (mAP) is calculated as average of AP's across the whole test set. As is standard for detection tasks, we report mAP with a 50% IoU threshold (mAP@0.5) and mAP@0.5:0.95 as the evaluation metric for quality of text detection model. mAP@0.5:0.95 is calculated as average of set of mAP's over different IoU thresholds, from 0.5 to 0.95 with step 0.05 (0.5, 0.55, 0.6 ..., 0.95). Higher mAP value for the detection model is desirable.

Performance of text recognition model: For the recognition model, we use accuracy and Levenshtein's edit distance as metrics to measure the performance. The accuracy is calculated based on statistics of prediction across test set by $Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$, where TP, TN, FP, and FN are the true positives, true negatives, false positives, and false negatives. For accuracy calculation, a prediction is counted as positive only if the predicted word matches exactly the ground-truth word. The accuracy metric gives information about the percentage of words that are correctly recognized, but does not give any information about how wrong the incorrect predictions are: incorrectly recognizing one character or incorrectly recognizing ten characters yields exactly the same accuracy, zero. To obtain more information about the incorrect transcriptions we use Levenshtein's edit distance, which counts the number of single

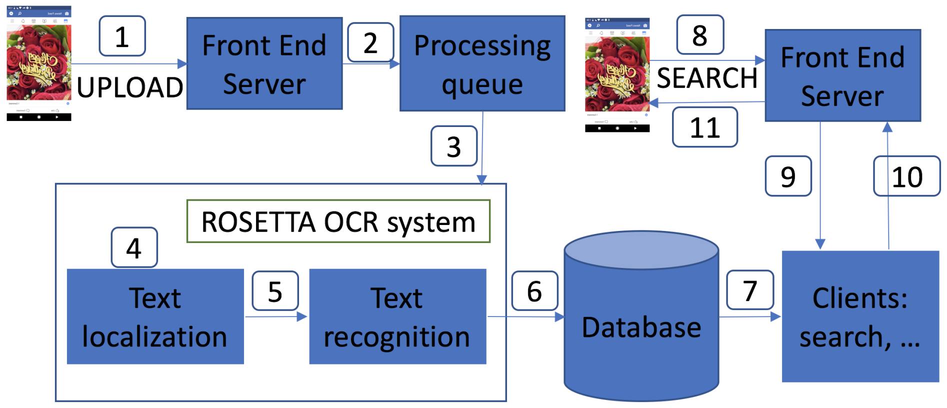


Figure 5: Architecture of *Rosetta*, Facebook's scalable text recognition system.

character edits (insertions, deletions and substitutions) between the predicted word and the ground-truth word, and can be seen as a surrogate of how much work would be needed to manually correct the transcription. We report the total edit distance for all items in the test set. Higher accuracy and lower edit distance are desirable.

Inference time: We used internal Facebook infrastructure to measure the inference runtimes of detection and recognition models. Inference time for detection model represents the time spent detecting and calculating the bounding boxes for all textual information observed in the image. Inference time for recognition model represents the time spent running the recognition model on every detected word in an image. The experiments were performed on hardware settings closely matching the production setup and using only one cpu and one single core. Lower inference times are desirable.

5.2 Training and Test Datasets

Having high quality training and test data is important for building robust supervised machine learning models. We used the COCO-Text [28] public dataset, which contains extensive annotations of text in the wild, to bootstrap training. COCO-Text contains more than 63,000 images and 145,000 text instances. However COCO-Text doesn't match the data-distribution of images uploaded to Facebook. For instance, a non-trivial portion of images on Facebook have text overlaid on them which COCO-Text fails to adequately capture. To address this, we generated a large synthetic dataset taking into account different use-cases. This was done by overlaying words on randomly selected public Facebook images. The images where filtered to select only those that do not already have text on them by using a separate image classification CNN that predicts the probability of text being present in an image. Image generation is part of the training process, where we automatically prepare the set of images at training initialization step, and remove the images after completion of the model training. The words to generate this synthetic dataset were picked at random from a dictionary, and were augmented to include special characters and to look like email addresses, URLs and phone numbers, and then overlaid on images with various fonts, sizes and distortions applied. We used around 400k synthetic images for training and 50k images for testing. We also manually annotated a dataset with thousands of images gathered using help of human raters and used it to fine-tune the models, which greatly improved the results. In the following sections of experimental evaluation we report most of the metrics on the human rated dataset mentioned here.

Table 1: mAP of Faster-RCNN detection model on the test set with different training datasets. Accuracies reported as relative improvements of mAP over training only on synthetic dataset. The → denotes finetuning, *i.e.*, $A \rightarrow B$ means train on A and then finetune on B.

	mAP (relative improvement)	
Training dataset	@IOU=0.5	@IOU=0.5:0.95
Synthetic	+0.0%	+0.0%
COCO-Text	+39.9%	+15.2%
Synthetic → COCO-Text	+41.2%	+16.6%
Synthetic → COCO-Text → Human rated	+57.1%	+35.2%

5.3 Detection Model Experiments

Text detection part of the implemented system is the most compute and latency intensive component during inference. Given our scale and throughput requirements, we spent significant amount of time improving the execution speed of text detection model while keeping the detection accuracy high. As we evaluated various approaches for the detection model, Faster-RCNN was a natural choice owing to its state-of-the-art results and readily available implementation within Facebook through Detectron [5]. Detectron is Facebook's state-of-the-art platform for object detection research.

Table 2: Inference runtimes of Faster-RCNN with various convolutional bodies. Numbers in the table reported as relative improvements to ResNet-50.

Convolutional body	Ratio of CPU Inference time
ResNet-50	1x
ResNet-18	1.9x faster
ShuffleNet	4.57x faster

Table 3: mAP of Faster-RCNN with ResNet-18 and ShuffleNet bodies evaluated on COCO-Text dataset. mAP numbers in the table reported as relative improvements to ResNet-18 with 3 RPN aspect ratios.

		mAP (relative improvement)	
Convolutional body	RPN Aspect Ratios	@IOU=0.5	@IOU=0.5:0.95
ResNet-18	3	+0.0%	+0.0%
ResNet-18	5	-3.4%	-1.1%
ResNet-18	7	+2.4%	+1.4%
ShuffleNet	3	+0.7%	+0.6%
ShuffleNet	5	+0.3%	+0.4%
ShuffleNet	7	+3.1%	+1.8%

Table 4: mAP of Faster-RCNN detection model on test set varying RPN_POST_NMS_TOP_N (number of top RPN proposal boxes to retain). Inference is 2x faster with 100 proposals compared to 1000. Numbers in the table reported as relative improvements to configuration of Faster-RCNN with RPN_POST_NMS_TOP_N=50.

	mAP (relative improvement)	
RPN Post NMS Top N	@IOU=0.5	@IOU=0.5:0.95
50	+0.0%	+0.0%
100	+5.9%	+2.3%
1000	+8.2%	+2.7%

Table 5: mAP of Faster-RCNN detection model on test set depending on different NMS methods for suppressing bounding boxes produced by the final regression head. An Intersection-over-Union (IoU) threshold of 0.7 is used across all settings. For Gaussian SoftNMS, we set σ=0.5 as in [7]. Numbers in the table reported as relative improvements to Standard NMS configuration.

	mAP (relative improvement)	
NMS Method	@IOU=0.5	@IOU=0.5:0.95
Standard NMS	+0.0%	+0.0%
Gaussian SoftNMS	+1.3%	+0.9%
Linear SoftNMS	+1.5%	+1.0%

Detectron is open-source [5] and we refer to its settings in this section while describing our experiments.

We evaluated various architectures for the convolutional body of Faster-RCNN including ResNet-50, ResNet-18 and ShuffleNet. Text detection model based on Faster-RCNN with ResNet-50 body incurs significant and impractical runtime on CPU per image whereas ShuffleNet is 4.5x faster. We show comparison of inference times for different convolutional bodies for Text Detection in Table 2. We further evaluated mAP of ResNet-18 and ShuffleNet on COCO-Text dataset as listed in Table 3, where we report mAP metrics as relative improvements to ResNet-18 with 3 RPN aspect ratios. ShuffleNet achieves competitive results in mAP in comparison to ResNet-18. The region proposal network (RPN) of Faster-RCNN was tweaked to generate wider proposals to handle text boundaries by modifying RPN.ASPECT_RATIOS setting in Detectron (with 7 aspect ratios and 5 different sizes, the RPN generates 35 anchor boxes per region), which showed consistently better results than the standard aspect ratios of 0.5, 1 and 2. All further experiments were performed on ShuffleNet since optimal inference latency was one of our primary goals.

We started development of the text detection model using COCO-Text dataset. However, we quickly observed that many applications of *Rosetta* have different production data distributions, which posed challenges during model development. For example, some applications have many images with printed text overlaid on top of original photos, where models trained just on COCO-Text showed suboptimal performance. At the same time, there were other applications where images with text occurring in natural circumstances is the norm. This was an **important lesson** we learnt in the beginning of our development, so we decided to introduce several datasets to solve the problem. Our solution to the problem was that we introduced artificially generated dataset with text overlaid images, and pre-trained models with it initially. Both detection and text recognition models were pre-trained on artificial dataset and then fine-tuned using COCO-text and human rated datasets specifically collected for client applications. In Table 1, we present an evaluation of text detection model improvements with fine-tuning. We initially trained using synthetic dataset, then fine-tuned using COCO-Text, and finally with human rated dataset achieving 57% improvement in mAP over the model trained on the synthetic dataset alone.

During inference, RPN_POST_NMS_TOP_N setting of Detectron, which controls the number of generated proposals to be fed through RoI Pooling and the final stage of detection network, was reduced to 100 from the default 1000 as shown in Table 4. This made inference 2x faster with acceptable drop in mAP, while reducing it below 100 gave diminishing returns in terms of inference time.

Another trade-off between performance and inference time of the detection model is related to the resolution of the input image. We experimented with a range of image resolutions by resizing the image to a particular size in the maximum dimension while maintaining its aspect ratio. We found that having images of side more than 800px increases the mAP@0.5 slightly, but significantly increases inference time.

We also experimented with a modification of NMS (Non Maximal Suppression) by replacing the final NMS in Faster-RCNN with SoftNMS following [7]. Result of the experiments are shown in Table 5, where we report relative gains in mAP@0.5 by using different strategies of SoftNMS. Overall we improved mAP@0.5 by absolute 1.5 points using SoftNMS.

One lesson that we learnt during the development of text detection model is related to the choice of metric for model's performance. Our initial choice of metric for Faster-RCNN model training was F1-score (a harmonic mean of precision and recall), which was at the initial moment of development a default evaluation metric in COCO-Text [28]. However, we observed that our reported validation set F1-score would fluctuate with a standard deviation of 5.87 points among training runs with the same configuration. We found the reason for the observed variation in performance to be because F1-score is measured at a single point in the precision-recall curve, which means we need to pick a threshold on the curve to compute

Table 6: Recognition model performance on different datasets. Higher accuracy and lower edit distance are better. Values in edit distance column are normalized by maximum value of occurred error for CHAR+Synthetic variant. Numbers in the table reported as relative improvements to configuration of CHAR model trained on Synthetic dataset.

Model	Training	Relative Accuracy	Normalized Relative Reduction of Edit Distance
CHAR	Synthetic	+0.0%	-0.0%
CTC	Synthetic	+6.76%	-11.23%
CHAR	Synthetic $\rightarrow$ Human rated	+42.19%	-67.01%
CTC	Synthetic $\rightarrow$ Human rated	+48.06%	-78.17%

Table 7: Normalized magnitude of drop of recall of detected words for combination of detection and recognition systems. Normalization is performed by taking a relative drop in recall and normalizing it by the maximum value of drop for "Case sensitive". The * symbol denotes that recognitions with one character error are still considered correct.

	Normalized magnitude of recall drop in word recognition
Case sensitive	-1X
Case insensitive	-0.94X
Case insensitive*	-0.63X

it. With minor variations in training runs such as random initialization of weights, the threshold fluctuates as well and is not a reliable way to measure the model's performance, while Average Precision on the other hand computes the area under the precision-recall curve and hence is robust to such variations. At the moment of writing this paper, newer tasks on COCO-Text such as ICDAR [3] use mAP for evaluation. Therefore, we have replaced F1-score with mAP and that helped to solve the problem.

5.4 Recognition Model Experiments

During the development of *Rosetta* we considered different architectural designs for the text recognition model, ranging from dictionary-based approaches where word recognition is modeled as a classification task amongst a predefined set of words [20], to character based approaches where all the characters are jointly recognized and the combination of those characters comprises the whole word. Dictionary based approaches have been shown to yield superior accuracy than character based word recognition when a dictionary or lexicon is known in advance, mainly due to the capacity of models to memorize a large yet limited set of dictionary words during training.

However, a predefined dictionary would be too limiting for many real-world applications, which require recognizing more than just simple words as in the case of URLs, emails, special symbols and different languages. Therefore, **an important architectural decision** and a natural choice was to use a character-based recognition model.

In the initial stages of development, we tested various efficient backbone architectures, including ResNet-18 [17] and SqueezeNet [18]. We evaluated these choices on our synthetic data and ended up selecting ResNet-18 as our main backbone architecture for the recognition body: ResNet-18 showed significantly better accuracy compared to SqueezeNet, while incurring only a negligible overhead in computation. The discrepancy between detection and recognition models with respect to the inference performance of ResNet-18 is because, compared to the detection model, the recognition model operates on much smaller input images: while the detection model operates on resolutions close to 600×800 pixels, the recognition one operates on 32×128 pixels. While the larger images allow architectures like SqueezeNet or ShuffleNet to show how efficient they are, the overheads in recognizing smaller images significantly reduce this efficiency gap. Therefore all the experimental numbers for text recognition model provided in this section are based on ResNet-18 backbone. Additionally, and contrary to many recent approaches, we do not use a recurrent model such as an LSTM or a GRU to model the interaction between characters, and instead use a purely convolutional model. The main argument is that recurrent models are slower at inference time, and a small loss in accuracy is affordable if the inference time is reduced. Additionally, a few works have started showing that recurrent models are not strictly necessary to accurately model the interaction between characters [12].

We also evaluated how case sensitive labelling would influence the accuracy of our model. We found that either ignoring and not ignoring the character case during training led to equivalent results when testing in a case-insensitive manner. However, training with case-sensitive annotations allowed us to perform case-sensitive inference, which can be useful in some downstream applications of *Rosetta*.

The CTC model achieves high accuracy on real-world validation set when trained with synthetic data and fine-tuned with manually annotated word crops. We also evaluated the total edit distance, *i.e.*, the total number of single character edits in the test set needed to correct the predicted transcriptions, as shown in Table 6. The fully convolutional CTC model (referred to as "CTC, Synthetic $\rightarrow$Human rated" in Table 6) improved the accuracy by 48.06% over the CHAR baseline.

We performed evaluation of the system to measure the gap of combining errors in text detection and text recognition, where in the case of perfect text detection we would get pure text recognition accuracy. We observed that in around 37% of text detection error cases our text recognition model could still correctly recover words making just one character error. In Table 7, we show relative drop in recall for overall number of detected words. For many downstream applications, single-character errors in recognized words are still acceptable and useful.

To improve the overall accuracy of the system, we augmented the training dataset for recognition model by introducing random jittering. The bounding box coordinates of ground-truth might be randomly shifted to model the behavior of noise from detection model. This resulted in 1.54% relative improvement in end-to-end performance. We observed that jittering becomes especially useful when there is less amount of training data available for certain application use-cases.

6 DEPLOYMENT

Rosetta service is deployed within Facebook at scale, offers a cloud API for text extraction from images, and processes a large volume of images uploaded to Facebook everyday. In *Rosetta*, the image is resized to 800px in the larger dimension and fed through the detection model which outputs bounding box coordinates around each word. The word patches are then cropped out, resized to a height of 32px while maintaining the aspect ratio, and processed by the recognition model. The inference runtime for the recognition model depends on the number of words detected in the image.

Rosetta service was deployed incrementally to client applications to anticipate any issues, with such a deployment plan consisting of weekly increase of traffic served initially to a predefined set of internal users, then to public traffic of 1%, 5%, 10%, 25%, 40%, 80% and finally 100%. We continued to evaluate resource utilization and incrementally add more machines to the processing fleet as deployment of the service continued until 100%.

The Faster-RCNN detection model was trained using the recently open-sourced Detectron framework [5] which is built on top of Caffe2 [2]. The text recognition model was trained using PyTorch [1] owing to its flexibility for quick prototyping and sequence modeling scenarios. Both models were deployed to production using Caffe2, with the text recognition model converted from PyTorch to Caffe2 using the intermediate ONNX format [4].

7 CONCLUSION

In this paper we presented approaches for building robust and efficient models for text detection and recognition, and discussed architectural approaches for building a scalable OCR system *Rosetta*. With thorough evaluation, we demonstrated trade-offs between achieving high efficiency in terms of scale and processing time and the accuracy of models. Our system is deployed to production and processes images uploaded to Facebook everyday. We have provided practical experience, trade-offs and shared lessons learnt building OCR system at scale.

8 ACKNOWLEDGMENTS

The authors would like to thank Anmol Kalia, Manohar Paluri, Peizhao Zhang, Sam Tsai, Fei Yang, Vajda Peter, Ross Girshick, Cristian Canton Ferrer, Simon Elmir, Alex Kapranoff, Denis Sheahan, Isaac Nwokocha, Tilak Sharma, Kevin Chen, Nikhil Johri, Shomir Dutt, Cristina Scheau, Yu Cao, Daniel Olmedilla, Brandon Chen, Ainulindale Yeqi Lu, and others who contributed, supported and collaborated with us during the development and deployment of our system.

REFERENCES

[1] 2016. PyTorch. (2016). http://pytorch.org/
[2] 2017. Caffe2. (2017). https://caffe2.ai/
[3] 2017. ICDAR2017 Robust Reading Challenge on COCO-Text. (2017). http://rrc.cvc.uab.es/?ch=5/
[4] 2017. Open Neural Network Exchange (ONNX). (2017). https://onnx.ai/
[5] 2018. Detectron: FAIR's research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet. (2018). https://github.com/facebookresearch/Detectron/
[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML*.
[7] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. 2017. Improving Object Detection With One Line of Code. *CoRR* abs/1704.04503 (2017).
[8] Dhruba Borthakur. 2013. Under the Hood: Building and open-sourcing RocksDB. (2013). https://code.facebook.com/posts/666746063357648/under-the-hood-building-and-open-sourcing-rocksdb/.
[9] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *USENIX Conference on Annual Technical Conference*.
[10] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.
[11] Piotr Dollar, Ron Appel, Serge Belongie, and Pietro Perona. 2014. Fast Feature Pyramids for Object Detection. *TPAMI* (2014).
[12] Yunze Gao, Yingying Chen, Jinqiao Wang, and Hanqing Lu. 2017. Reading Scene Text with Attention Convolutional Sequence Modeling. *CoRR* abs/1709.04303 (2017).
[13] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017).
[14] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*.
[15] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. 2016. Synthetic Data for Text Localisation in Natural Images. In *CVPR*.
[16] Dafang He, Xiao Yang, Chen Liang, Zihan Zhou, Alexander G. Ororbia II, Daniel Kifer, and C. Lee Giles. 2017. Multi-scale FCN with Cascaded Instance Aware Segmentation for Arbitrary Oriented Word Spotting in the Wild. In *CVPR*.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.
[18] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016).
[19] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. In *NIPS Deep Learning Workshop*.
[20] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2016. Reading Text in the Wild with Convolutional Neural Networks. *IJCV* (2016).
[21] Xuebo Liu, Ding Liang, Shi Yan, Dagui Chen, Yu Qiao, and Junjie Yan. 2018. FOTS: Fast Oriented Text Spotting with a Unified Network. *CoRR* abs/1801.01671 (2018).
[22] Zichuan Liu, YIxing Li, Fengbo Ren, Hao Yu, and Wangling Goh. 2018. Squeezed-Text: A Real-time Scene Text Recognition by Binary Convolutional Encoder-decoder Network. *AAAI*.
[23] George Nagy. 2000. Twenty years of document image analysis in PAMI. (2000).
[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *TPAMI* (2017).
[25] Baoguang Shi, Xiang Bai, and Serge J. Belongie. 2017. Detecting Oriented Text in Natural Images by Linking Segments. In *CVPR*.
[26] Baoguang Shi, Xiang Bai, and Cong Yao. 2016. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *TPAMI* (2016).
[27] Cooper Smith. 2013. Facebook Users Are Uploading 350 Million New Photos Each Day. (2013). http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9.
[28] Andreas Veit, Tomas Matera, Lukas Neumann, Jiri Matas, and Serge Belongie. 2016. Coco-text: Dataset and benchmark for text detection and recognition in natural images. *CoRR* abs/1601.07140 (2016). https://vision.cornell.edu/se3/coco-text-2/
[29] Kai Wang and Serge Belongie. 2010. Word Spotting in the Wild. In *ECCV*.
[30] Fei Yin, Yi-Chao Wu, Xu-Yao Zhang, and Cheng-Lin Liu. 2017. Scene Text Recognition with Sliding Convolutional Character Models. *CoRR* abs/1709.01727 (2017).
[31] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *CoRR* abs/1707.01083 (2017).
[32] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. 2017. EAST: An Efficient and Accurate Scene Text Detector. In *CVPR*.
[33] C. Lawrence Zitnick and Piotr Dollár. 2014. Edge Boxes: Locating Object Proposals from Edges. In *ECCV*.