Reconstruction of Occluded Text in Images for Optical Character Recognition

MASTERARBEIT

im Studiengang Internet Technologies and Information Systems (ITIS)

von

Diaa Eddin Esmail

Prüfer: Prof. Ralph Ewerth

Zweitprüfer: Prof. Sören Auer

Betreuer: M.Sc. Matthias Springstein

in Leibniz Universität Hannover

im November 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Leibniz Universität Hannover, November 20, 2019

Diaa Eddin Esmail

Contents

Abstract

Text occlusion is among the most intractable obstacles for the task of Optical Character Recognition (OCR) in Computer Vision. A typical example is when some characters of a text in an image are invisible due to occlusion them by arbitrary geometrical shapes. This causes distorting the text and missing some characters in the word because the holes arise in the image. It requires finding a method to reconstruct the image by filling in holes with plausible imagery before recognizing the whole text in an image. We introduce an approach to reconstruct an image by using Partial Convolution, which comprises a mask and renormalized convolution operation followed by a mask-update step to automatically generate an updated mask for the next layer as part of the forward pass. For the recognition task, we propose a no-recurrence sequence-to-sequence model that depends only on attention mechanism and dispenses completely with recurrences and convolutions. The model consists of an encoder and decoder, and uses stacked self-attention modules. The Experiments have shown that both proposed models can produce reconstructed images and improve the effectiveness of text recognition in these images.

Chapter 1

Introduction

This thesis aims to address the problem of scene text recognition and image reconstruction, and therefore to understand the dimensions of the problem that we cope with, let's give an example of use case that illustrates in which scenarios we could confront this problem. We assume that we have a lecture video recorded using a single still camera in a classroom, and suppose that the whiteboard in the image is covered by a moving object like the instructor, that results in like a mask (or holes) in some places of the content on the whiteboard, and makes the content invisible as the Figure 1.1 shows.

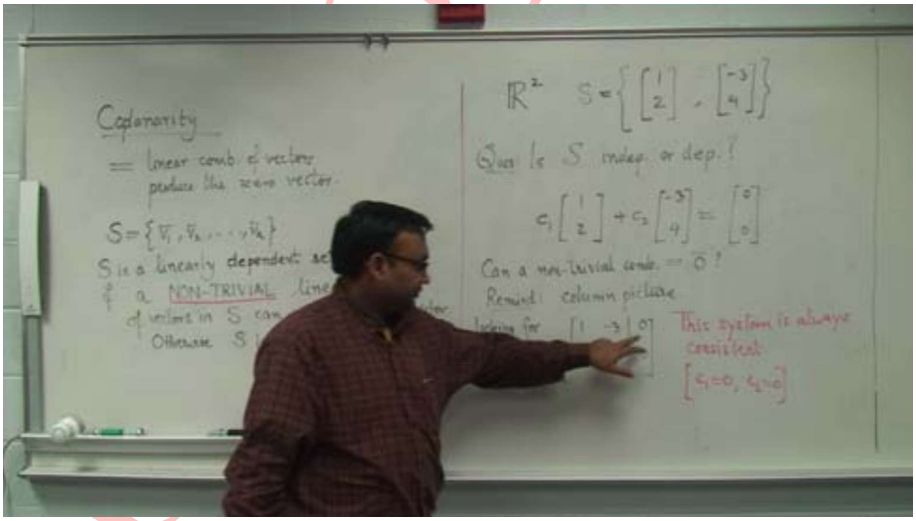


Figure 1.1: Raw Image captured by still camera in classroom [1].

What the image reconstruction model should do is to remove this mask that covers the content on the whiteboard, which it is in this case "the instructor", to get at the end a reconstructed image that only contains the content without the instructor, as Figure 1.2 shows, then we feed the recon-

structed image 1.2 into an optical character recognition engine to predict the text content.

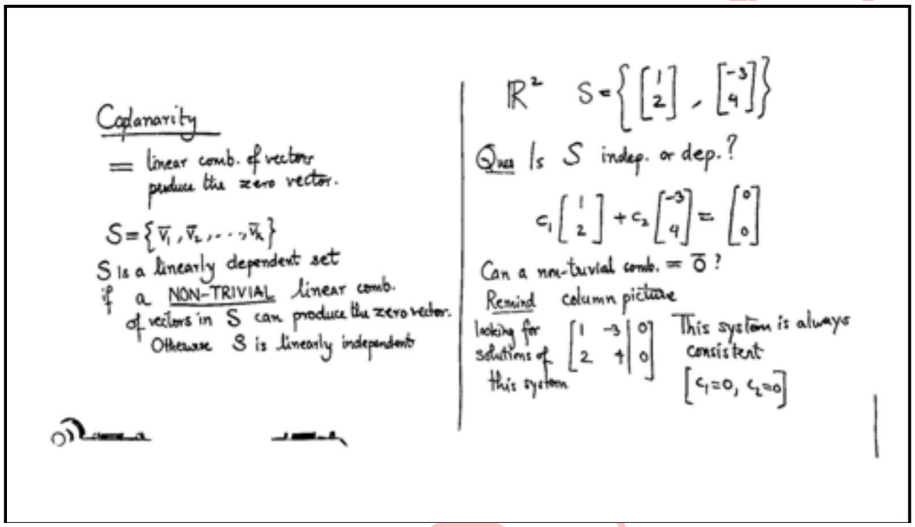


Figure 1.2: Reconstructed Image [1]

Based on the above example we can conclude that the task of reconstruction of occluded text in an image for optical character recognition comprises two major tasks: firstly, reconstruct the distorted part of the image that contains text by removing the shapes/objects that cover the characters, and secondly recognize these characters by using OCR engine. Both problems can be solved sequentially.

Image reconstruction also known as image inpainting is the process of reconstructing or filling in lost or damaged parts of images and considering the surroundings of the holes in an image to incorporate them smoothly with the rest of the image. Recent image inpainting approaches that do not use deep learning but do use patch sampling methods that are essentially based on statistics of the remaining image to fill in missing parts such as PatchMatch [2] is one of the state-of-the-art methods that is using randomized algorithm for finding approximate nearest-neighbor matches between image patches. The key idea of this algorithm is searching iteratively for the best suitable patches by finding random sampling, and then trying to produce smooth matches to be integrated in the surrounding areas. Although this approach generally is able to smoothly fill in the missing components using image patches from the surroundings, but it is limited by the available image statistics and does not allow to preserve semantically important structure in filled parts of an image.

There exist many traditional methods to do image inpainting like diffusion or patch-based methods such as [3],[4],[5] that typically use variational

algorithms or patch similarity to propagate information from the background regions to the holes. The Traditional methods usually work well for specific tasks. For example background inpainting when it is just simply required to repeat some patches from neighborhood to reconstruct the missing parts. Exemplar-based image inpainting [6]is one of those traditional methods that deals with the problem of removing large objects from image and replacing them with visually reasonable backgrounds. The algorithm is a combination of region-filling and texture synthesis approaches. However, non-Deep Learning methods still have problems with the missing content, they are not semantically-aware approaches. In contrast, deep neural networks do quite well because they can capture some semantic priors and learn meaningful hidden representations of an input image in an end-to-end fashion, which have been used for recent image inpainting efforts.

Previous proposed deep learning approaches are depending on initial hole values, by employing convolutional filters on images, and replacing the occluded content with a fixed value. However, these approaches have focused on rectangular shaped holes located at and around the center of an image. Pathak et al. [7] and Yang et al.[8] assume 64 x 64 square holes at the center of a 128 x 128 image. As a consequence, these approaches have some drawbacks, which often appear as lack of integration of the filled parts with the surrounding areas, texture in the hole regions, obvious color contrasts, or producing blurring in the in-painted parts, and may lead to overfitting to the rectangular holes, and eventually limit the utility of these models in application. These inpainting methods often rely on expensive post-processing to condition the generated output on the missing regions. For example, Iizuka et al. [9] present an approach for image completion with a fully-convolutional neural network and use global and local context discriminators to distinguish real images from completed ones. However, they perform post-processing using fast marching [10] and Poisson image editing [11] by blending the completed region with color of the surrounding pixels because of color inconsistencies in the filled regions. While Yu et al. [12] propose a deep generative model based on both sequential coarse and refinement networks that utilize surrounding image features as references during network training to make better predictions. However, the model still creates distorted structures or blurry textures inconsistent with surrounding areas.

In this thesis we investigate a model for image inpainting that operates well on irregular hole shapes (mask), and independent of the hole initialization values. Our work aims to achieve well-integrated hole predictions, and generate inpainted images without any additional post-processing, and ultimately feed these inpainted images into an OCR engine to recognize the text in image. The proposed image reconstruction model is based on U-Net architecture [13]. The U-Net network consists of an encoder and decoder, used for image segmentation with many successes. However, we will simulate the

"instructor" (see Figure 1.1) in our experiments by generating some irregular masks on scene text images, and thus to properly handle irregular masks we use so called *Partial Convolution Layer* that takes into account also binary masks, so that before doing the convolution we are multiplying an input patch of the image with a mask. Furthermore, we include a mechanism to automatically update the mask after each layer, and therefore the mask is shrinking from layer to layer and usually it disappears in the encoder phase. In the decoder phase, we do upsampling based on nearst-neighbor approach, and again we do partial convolution. We enable the use of skip connection as well, which can be quite important in this case to produce detailed output for inpainting areas in image.

For the text recognition task we adopt a no-recurrence seq2seq model, named (NRTR)[14], which depends only on attention-mechanism that connects the encoder and decoder together, and dispenses with recurrences and convolutions completely. The NRTR model is inspired by recent sophisticated transformer proposed in the paper "Attention Is All You Need" [15]. The transformer architecture is aimed at the problem of sequence transduction [16], which means any task where input sequences are transformed into output sequences, this includes speech recognition, text-to-speech transformation, machine translation etc. Basically, the goal is to design a single framework to handle as many sequences as possible. However, the transformer has made many outstanding successes in Natural Language Processing (NLP) such as machine translation tasks, where the experiments on two machine translation tasks (English-to-French , English-to-German) [15] showed that these models can achieve better quality while being more parallelizable and require significantly less time to train than the existing models including RNNs and CNNs based on encoder-decoder scheme. Another notable application of transformer [15] is GPT-2 [17] which stands for Generative Pretrained Transformer. GPT-2 is a model was trained to generate (predict) next token in a sequence of tokens in an unsupervised way [18]. It uses fine-tuned pre-trained models created by OpenAI [19], where this retraining approach became quite popular in 2018 and continues throughout 2019, and uses the transformer architecture as opposed to an RNN, LSTM, GRU. One more application of the transformer is Bidirectional Encoder Representations from Transformers (BERT) that is a recent paper [20] published by researchers at Google AI Language. BERT is presented as a model for NLP tasks, including Questing Answering (SQuAD V1.1), Natural Language Inference (MNLI), Next Sentence Predictions (NSP), and others. The conducted experiments on BERT have shown that it can outperform the state-of-the-art models in eleven NLP tasks. All these great successes of the transformer make us more excited about using it in our text recognition task. NRTR [14] takes advantage of the transformer architecture in addition to some improvements such as the stacked self-attention module, which is

used in the whole structure of NRTR (in encoder , decoder and modality-transform block) that allows to draw global dependencies between different input and output positions at once, rather than one by one in RNNs, and reduce the entire operation to a constant number unlike that in CNNs, which would allow for significantly more computation parallelization what it is exactly needed by the seq2seq model in scene text recognition. The main contributions of this thesis are:

- We introduce an inpainting model that is able to reconstruct the missing occluded parts of text in an image by using partial convolutional layers and automatic mask updating mechanism with skip links in U-Net network [13] to achieve good inpainting results without any post-processing or blending operations.
- For the text recognition task, we study a novel no-recurrence sequence-to-sequence model (NRTR) [14], inspired by the recent released transformer architecture [15] that connects the encoder and decoder through an attention mechanism, and show how the transformer architecture could be used for performing scene text recognition.

This thesis is organized into six chapters. Following this introduction, chapter 2 highlights the most recent related works to understand the state of the problem. Chapter 3 explains the deep leanring foundations. In chapter 4 we provide a detailed description of the proposed approaches and methods. Chapter 5 focuses on evaluating the effectiveness of the proposed models and conducting experiments. Finally, the last chapter 6 concludes the thesis and provides an outlook.

Chapter 2

Related Work

Before diving deeply into explaining our models and the underlying technologies behind them let's first have a look at the most recent methods and models, which have been proposed to deal with image reconstruction and scene text recognition problems. In this chapter we will give an overview for the most recent popular ideas and approaches that have been utilized to cope with these complicated standing problems. We will show where these methods have made great success and where have failed.

2.1 Image Inpainting

Dealing with image inpainting problem was done with two basic approaches: Non-learning approaches and Deep Learning based methods, each of them has its own fashions and algorithms to overcome this dilemma. Some of them have achieved a remarkable success, whereas others still have shortcomings and are applicable in a very narrow range and on specific cases.

2.1.1 Non-learning approaches

Most of non-learning approaches to image inpainting use information from the neighboring pixels to fill in the occluded part (the hole) using some mechanisms like distance field. For example, Ballester et. al.[3] present a variational approach for filling-in regions of missing data in digital images. The approach is based on joint interpolation of vector fields and gray levels. The key concept of this method is to smoothly propagate inside the hole both the vector field obtained from the image gradient and the corresponding gray values. Bartalmio et. al. [4] introduce an algorithm that attempts to smoothly propagate information from the surrounding areas in the isophotes direction. The approach uses Partial Differential Equations (PDEs) that first reconstruct the isophote (line of equal gray values) directions, then the gray levels by finding an image consistent with these gradi-

ent directions, which can be done using one dimensional transport equation. Telea et. al. [10] propose a similar image inpainting technique to [4] that is based on propagating an image smoothness estimator along the image gradient. The algorithm starts working firstly on inpainting an initial point (P) located on the boundary($\partial\Omega$) of the region to inpaint(Ω). By taking a small neighborhood B_ε(p) of size ε of the known image around the point (p). The inpainting of p should be determined by the values of the known image points close to p. Next, the point p is inpainted as a function of all points q in B_ε(p) by summing the estimates of all points q, weighted by a normalized weighting function. To inpaint the whole region (Ω) should apply the normalized weighting function iteratively to all the discrete pixels of the boundary ($\partial\Omega$). Afterwards, a fast-marching algorithm is applied as a post-processing to propagate the image information into the target region [10]. However, these methods still have some limitations such that they can only handle small, narrow holes where the color and texture variance is pretty low. If inpainting regions are big, then these methods can't reproduce plausible, meaningful textured regions, but rather produce blurring that is especially visible when sharp isophotes intersect the region's boundary almost tangentially. Moreover, big holes may cause over-smoothing or artifacts resembling Voronoi regions such as in [10].

On the other hand, other non-learning approaches try to address texture synthesis problems by using patch-based methods such as [5][21][22]. The main idea is to synthesize new texture by looking for relevant patches from existing textures of other source images or image's non-hole regions in an iterative fashion, then stitching them together in a consistent way. For improving the quality of synthesized texture is a Markov Random Field (MRF) used that allows to formulate the synthesis problem as a minimization of an energy function, which is optimized using an Expectation Maximization algorithm and can quantitatively measure quality of the synthesized texture. However, these methods often come at a large computation cost, furthermore due to the texture optimization technique, it can get stuck in local minima because it tries to decrease the energy function at each iteration. This seems evident as blurring or misalignment of texture elements in the output synthesized image. Other methods try to speed it up by proposing a faster similar patch searching algorithm, such as randomized correspondence algorithm (PatchMatch) [2] for quickly finding approximate nearest-neighbor matches between image patches. The main ideas driving PatchMatch algorithm are that some good patch matches can be found via random sampling, and that natural coherence in the imagery allows to propagate such matches quickly to surrounding areas. However, these approaches are still not fast enough for real-time applications because to obtain meaningful suitable patches the algorithm has to rerun many times with a new random seed, such repeated trials slow the algorithm down and burden it. Furthermore, when do extreme edits to an image can sometimes produce some artifacts such as ghosting or

feathering where the algorithm simply cannot escape a large local minimum basin, and in the end cannot select patch in a plausible manner.

2.1.2 Deep learning approaches

Deep learning methods have recently emerged as promising approaches for image inpainting. Initial efforts [23] [24] train convolutional neural networks for denoising and inpainting of small regions. These methods typically initialize the holes with some constant placeholder values e.g. the mean pixel value of ImageNet [25], which is then passed through a convolutional neural network. However, due to the resulting artifacts these methods typically require post-processing steps such as image blending operation to enforce color coherency near the hole boundaries, and to improve the effects of conditioning on the placeholder values. Context Encoders [7] learn both the presentation of appearance and semantics of visual structures, the algorithm driven by context-based pixel prediction, first embed the 128×128 image with 64×64 center hole into low dimensional feature space and then decode the feature to a 64×64 image. It is trained to reconstruct the content of 64×64 center hole in a 128×128 image, conditioned on its surroundings. When training context encoders, both pixel-wise reconstruction loss and generative adversarial loss have been used as the objective function. It has been observed that the generative adversarial loss function can produce much better sharper results than just only using the pixel-wise loss function, because it can handle multiple modes in the output. Although, the adversarial loss significantly improves the inpainting quality, the results are still quite blurry and contain artifacts, and it fails to produce reasonable results for larger inputs like 512×512 images, therefore it can not be generalized to high-resolution inpainting task. More recently, Iizuka et al. [9] and Li et al. [26] improved the result of Context Encoders [7] by introducing both global and local context discriminators as adversarial losses, then Iizuka et al. [9] apply Poisson blending as a post-processing. The global discriminator assesses if completed image is coherent as a whole, while the local discriminator focuses on a small area centered at the generated region to enforce the local consistency. In addition, Iizuka et al. [9] use dilated convolutions in inpainting network to replace channel-wise fully connected layer adopted in Context Encoders [7], both techniques are proposed for increasing receptive fields of output neurons. However, this approach is still limited to relatively small images and holes due to the spatial support of the model. Moreover, the experiments have shown some faults due to large inpainting regions, this is especially when the inpainting mask(hole) is at the border of the images, the model fails to realistically inpaint the mask.

Yu et al. [12] propose a coarse-to-fine generative neural network for image inpainting, where the network architecture follows the same input and output configurations as in [9] for training and inference. The model archi-

tecture consists of a coarse network trained with reconstruction loss, while the refinement network is trained with reconstruction loss, global and local GAN loss. Moreover, Yu et al. [12] add a contextual attention layer in the refinement network, which learns where to copy feature information from known background patches to generate missing patches in the image. Yang et al. [8] propose a multi-scale neural patch synthesis approach based on joint optimization of image content and texture constraint, which initializes the hole with the resulting output of Context Encoders [7], and then propagates the texture information from non-hole regions to fill the hole region as post-processing. It preserves not only contextual structures but also produces high-frequency details by matching and adapting patches with the most similar mid-layer feature correlations of a deep classification network. However, this approach still introduces discontinuity and artifacts when the scene is complicated, and is very slow due to the optimization process when it inpaints a large image. Song et al. [27] divide the inpainting task into two stages, firstly they train an Image2Feature network that initializes the hole with coarse prediction and extract its features. However, the output of this stage is blurry but contains high-level structure information in the hole. Secondly, the translation stage, they train a Feature2Image network that transforms the feature back into a complete image. Feature2Image network smooths the contents in the hole and outputs a complete image with sharp and plausible texture. The approach is very good at restoring a partially missing object. However, it fails if the image has complicated structures and patterns, or a major part of an object is missing such that Image2Feature network is not able to provide a good inference.

Some other deep learning approaches ignore the mask placeholder values. In Yeh et al. [28] search for the closest encoding in latent space of the corrupted image using context and prior losses, and then pass the encoding through the generative model to infer the missing content. However, this method still has limitations, such that the proposed GAN model works well for relatively simple structures, but fails to infer missing contents in complex scenes. Ulayanov et al.[29] demonstrated that the structure of a generator network is sufficient to re-construct the corrupted parts of an image prior to any learning. However, this approach requires a random initialization of a different set of hyper-parameters for every image, and applies several iterations to achieve good results. Moreover, the structure of the generator network is not able to use skip links that is very useful to produce detailed output. Harley et al. [30] recently introduced a segmentation-aware convolutional network with a soft attention mask. They made use of a masked and reweighted convolution operation, which allows to condition output only on valid inputs. In PixelCNN [31] it also has been used the same approach in [30] for full-image generation to condition the next pixel only on previously synthesized pixels. The main idea of PixelCNN is to use auto-regressive connections to model images pixel by pixel, decomposing the joint image

distribution as product of conditionals. Uhrig et al.[32] proposed sparsity invariant CNNs, which consider the location of missing data in an image during the convolution operation. The sparsity invariant network comprises reweighted convolution and max pooling based on mask updating mechanism for depth completion. The input of the network is a sparse depth map and a binary mask. Ren et al. [33] proposed a convolutional neural network with shepard interpolation layer, named ShCNN. The inputs of the shepard interpolation layer are images/feature maps and a mask indicating where interpolation should occur. The mask is a binary map of values, one for the known area and zero for the missing area, where the same kernel is applied for both image and mask convolutions. The mask can be automatically updated by the result of previous convolved mask. However, this model cannot handle big holes properly and introduces blurry components and artifacts when the dimensionality becomes high.

2.2 Scene Text Recognition

In recent years scene text recognition has become an active, interesting topic in the communities of computer vision. Researchers have proposed a large amount of novel methods and approaches. Generally, it has been observed that these methods in many studies are divided into two categories: in the most traditional works the text images has been considered as general objects, or as sequence objects in most recent works.

2.2.1 Text images as general objects

Since it is very difficult to directly segment characters in natural scene images, many traditional methods adopt bottom-up scheme by first detecting possible individual characters using sliding window [34] [35] or connected components [36] or Hough voting [37], then integrating these characters in the output text aided by a large dictionary or lexicon [35]. Wang et al. [34] proposed a method for locating specific words from natural scenes. Firstly, single characters are detected by sliding window. Then, possible combinations are scored according to the structural relationships between characters. Finally, the most similar combinations are selected from a given list as the output results. Unlike traditional text detection methods, this algorithm can only detect words in the given list and thus it is incapable of handling words out of the given list. However, a word list that contains all possible words is not always available for each image, and this makes the applicability range of this methods very limited and narrow. In a similar way to [34], Wang et al. [35] proposed an end-to-end text recognition engine. It is based on object recognition techniques by extending a previous work [34]. The first step is to detect potential location of characters in an image via sliding window classification. For the purpose of character detection is Random Ferns [38]

used, where the algorithm extracts some feature vector for each location in an image and computes a score that tells the likelihood of any character being in this location. The method considers words as a special kind of object, and characters as parts of the object. It searches the most possible detection and recognition results by modelling each single character and the spatial relationship between characters. Experiments show that this method obtains excellent performance on multiple standard dataset such that Chars74K dataset [39], ICDAR Robust Reading Competition dataset [40], and Street View Text dataset(SVT) [41]. However, this algorithm can only handle words that are within the given word list, thus it is not applicable to images without a word list (lexicon). Neumann et al. [36] proposed an end-to-end real-time text localization and recognition method. In the first stage, the character detection problem is achieved by an efficient sequential selection from the set of Extremal Regions (ERs). The ER detector extracts from original images the ER regions as candidates, and removes invalid candidates using a trained classifier. At a later stage for character classification and recognition, the remained candidates are grouped into text lines through a series of connection rules and select the most probable character segmentation. However, such connection rules can only adapt to horizontal or nearly horizontal text, therefore this algorithm is unable to handle texts with larger inclination angle.

Others adopt the top-down scheme, where text is directly recognized from the original image, then the text would be split into a sequence of single characters and detected one by one. The top-down information comes usually from the statistics of a large dictionary or lexicon. Almazan et al. [42] propose to embed both images and text strings into a common subspace, then finding the nearest neighbor of the input image in a dataset containing the lexicon embedded into the common subspace as well. Jaderberg et al. [43] propose an end-to-end system accomplished both text spotting and recognizing in natural scene images. The system is based on a region proposal mechanism for detection and deep convolutional neural networks for recognition. They address text recognition problem as an image classification problem. The deep convolutional neural network takes the whole word image as input to perform classification of the word and assign a class label to each English word across a huge dictionary such as the 90k-word dictionary.

2.2.2 Text images as sequence objects

Most recent models assume scene text recognition task as a sequence recognition problem, where image and text are separately represented as patch sequences and character sequences. Su et al. [44]. propose a scene text recognition based on Recurrent Neural Network (RNN). A word image is first represented as a sequential column vector based on Histogram of Oriented Gradient (HOG), then the RNN is adapted to classify the sequential feature

vectors into the corresponding word. First, a word image is converted into a sequence of column feature, where each column feature is generated by concatenating the HOG features extracted from the corresponding image patches in the same column of the input image. Then, a multi-layer of RNN with bidirectional Long-Term Memory (LSTM) [45] is trained to label the sequential data. Finally, the Connectionist Temporal Classification (CTC) [46] technique is utilized to find out the best match of a list of lexicon words based on the RNN output of the sequential feature. Shi et al. [47] propose an end-to-end trainable sequence recognition network. The network architecture is a combination of Convolutional Neural Network and Recurrent Neural Network (CRNN). The network architecture consists of three components from bottom to top. At the bottom, the convolutional layers are used as feature extractor to automatically extract a feature sequence from each input image. The output of convolutional layers would be fed into a recurrent network, which built for making prediction for each frame of the feature sequence. Finally, the transcription layer at the top to translate the per-frame predictions by the recurrent layers into a label sequence. Although CRNN is a combination of different component, but it is still possible to train the whole network with one loss function.

Another contribution of Shi et al. [48] they propose an attention-based end-to-end neural network model that comprises a rectification network to rectify text distortion and a recognition network. The rectification network transforms an input image into a new one, after rectifying the text in it by handling a variety of text irregularities. The recognition network is an attentional sequence-to-sequence model that predicts a character sequence directly from the rectified image. They adopted Thin-Plate-Spline (TPS) [49] as a transformation to achieve the objective of the rectification network, which is to crop along the word bounding box within the original image. TPS performs non-rigid deformation on images, and can rectify both perspective and curved text. The recognition network is an encoder-decoder-based architecture. The encoder consists of a combination of convolution neural network and a recurrent neural network. The convolutional layers in the encoder extract a feature map to be represented by a feature sequence that describes local image regions, whereas a LSTM network is employed over the convolutional layers to analyze the feature sequence bidirectionally, and capture long-range dependencies in both direction, then it outputs a new feature sequence of the same length. In the decoder, the feature sequence comes from the encoder is translated into a character sequence, and it can handle input and output sequences of arbitrary lengths. Furthermore, it is able to capture output dependencies, as it has access to the encoder output at every decoding step due to the attention module.

Lee et al. [50] propose recursive recurrent neural networks with attention modeling. The model first passes input cropped image of single word through recursive convolutional layers to extract encoded image fea-

tures, and then decodes them to output characters by a RNN with implicitly learned character-level language statistics. Moreover, an attention-based mechanism performs soft feature selection for better image feature usage. Cheng et al. [51] construct a focusing attention method that consists of two main components, an Attention Network (AN) that is responsible for recognizing character, and a Focusing Network (FN) that is responsible for adjusting attention on the target areas in the images. The model is designed as encoder-decoder framework. In the encoding stage, an image is transformed into a sequence of feature vectors by CNN-LSTM, and each feature vector corresponds to a region in the input image. In the decoding stage, AN can directly generate the target characters based on the encoded feature vectors. Gao et al. [52] propose an end-to-end attention convolutional network for scene text recognition. Instead of using RNNs, they apply stacked convolutional layers to effectively capture the contextual dependencies of the input sequence and in order to achieve a greater computational parallelism. The network architecture combines CNN with CTC module to generate label sequence without any recurrent connections. Firstly, a sequence-to-map operation transforms the feature sequence into a 2D map as the input of CNN to process the sequence simultaneously. Then, stacked convolutional layers can extract hierarchical contextual representations of the input sequence to model the long-term dependencies with a shorter path. Similar to [52] Yin et al. [53] avoid RNN-LSTM completely, and propose a scene text recognition method with character models on end-to-end convolutional feature maps. The framework of the proposed method consists of three parts: a sliding window layer extracts features from the window by CNN, on the top of sliding window there is a classification layer to predict a label distribution from the input features, and finally a transcription layer to translate the per-window predictions into the result sequence.

2.3 Approaches used in the thesis

In this thesis we combine two novel proposed models to achieve our goal that consists of two tasks. Firstly, for reconstruction of occluded text in image we adopt Image Inpainting for Irregular Holes Using Partial Convolutions [54]. The authors proposed the use of partial convolutions with an automatic mask update step, and demonstrate that substituting convolutional layers with partial convolutions and mask updates can achieve state-of-the-art inpainting results. Moreover, it is the first work that demonstrates the efficacy of training image inpainting models on irregularly shaped holes. The experiments on ImageNet dataset [55], Plcaces2 dataset [56], and CelebA-HQ [57] [58] have shown that the proposed model can achieve state-of-the-art image inpainting results and robustly handle holes of any shape, size location, or distance from the image borders.

Secondly, for recognition the sequence of characters in the reconstructed image produced by Image inpainting model [54], we adopt a no-recurrence seq2seq recognizer [14] dispensing with RNNs and CNNs, which is quite different from the above existing approaches explained in 2.2. NRTR is inspired by the latest framework, named Transformer [15] that uses only self-attention mechanism as its fundamental module. The transformer has shown great superiority of parallelism and efficiency on natural language processing tasks, where the properties of the transformer are also exactly desired by the seq2seq modeling in scene text recognition. Furthermore, the authors added necessary modifications to the transformer architecture such as modality-transform block as a pre-processing step to overcome text recognition task. Moreover, what makes NRTR attractive to use in this thesis that it is the first work of scene text recognition, which relies entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or CNNs. According to the conducted experiments of the authors [14] on four benchmarks "IIIT5K" [59], "IC03" [60], "IC13" [61], and "SVT" [41] NRTR could train faster and achieve state-of-the-art best results and sometimes outperform some of them.

Chapter 3

Deep Learning Foundations

To deeply understand the architecture of our both models we should first give a gentle overview about the deep neural networks, from which the core modules of our proposed models are derived.

3.1 Convolution Neural Network

Convolutional Neural Network (CNN) is a deep learning algorithm that takes images as inputs, which allows to encode certain properties into the architecture. The main role of CNN is to reduce the image's dimensions into a form which is easier to process without losing features, which are important for making a good prediction. A simple convolutional network is a sequence of layers. Each layer transforms an image matrix (volume) of dimension (Height x Width x Channel) to another dimension through a differentiable function. We use three main types of layers to build convolutional neural network architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

3.1.1 Convolution Operation

Visual data has really very rich spatial structure. Consequently, we can leverage the spatial structure in the input image to extract features and detect their presence in images automatically without any manual definition to construct a representation of the images. The input image is represented as an array of pixel values, and one way to immediately use the spatial structure that is inherent to this input is to connect patches of the input layer to a single neuron in subsequent layer. Another way to think about this is each neuron in a hidden layer only sees a particular region of what the input to that layer is. This not only reduces the number of weights in our model but also allows us to take advantage of the fact that the pixels that are spatially close to each other are probably somehow related. We can

define connections across the whole input by applying the same principle of connecting patches in the input layer to neurons in subsequent layer. We do this by actually sliding the patch window across the input image as Figure 3.1 shows.

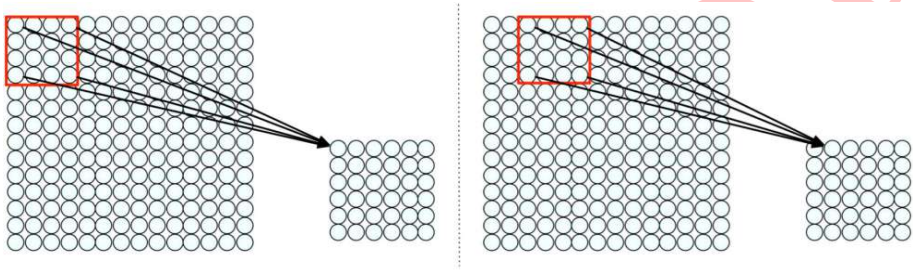


Figure 3.1: Spatial Structure, sliding the window two units [62]

In doing this we take into account the spatial structure that is inherent to the input, where the ultimate task is to learn visual features. However, the way to achieve this is by weighting these connections between the patch and the neuron in the next layer to detect particular features. Basically, we are applying a filter that is a set of weights to extract some sort of local features that are present in the input image. We can apply multiple different filters to extract different types of features. Furthermore, we can spatially share the parameters of each of these filters across the input, and thus the features that matter in one part of the image will still matter elsewhere in the image. In practice this amounts to this patchy operation that is called *Convolution Operation*. Suppose we need to compute the convolution of 5x5 representation of an image and 3x3 filter, to do this we need to cover the input image entirely by sliding this filter over the image and perform element-wise multiplication (Dot-Product) at each step and then add the outputs that result after each element-wise multiplication . The output matrix that we get after the convolution operation called Feature Map (or Activation Map). The convolution operation is expressed as [62]:

$$x' = \sum_{i=1}^{3}\sum_{j=1}^{3} w_{ij} X_{i+p,j+q} + b \tag{3.1}$$

Where 3×3 filter is matrix of weights w_{ij}, X_{ij} is the image input values, (p, q) for neuron in hidden layer, and b is bias. The following example illustrates how to compute the convolution of a 5×5 image and a 3×3 filter (ignoring the bias):

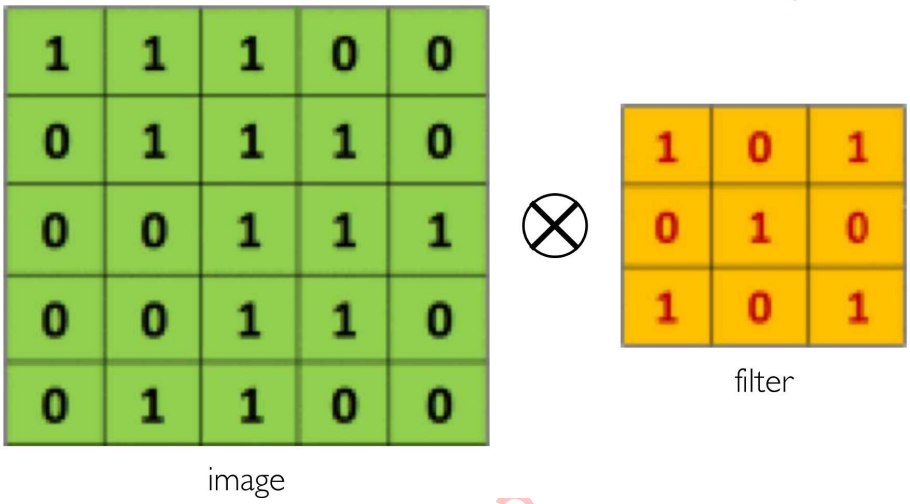


Figure 3.2: Compute the convolution of a 5×5 image and a 3×3 filter[62]

First, we start off in the upper left corner, we multiply this filter by the values of the patch of the input image and add the result, so we end up with the value 4.

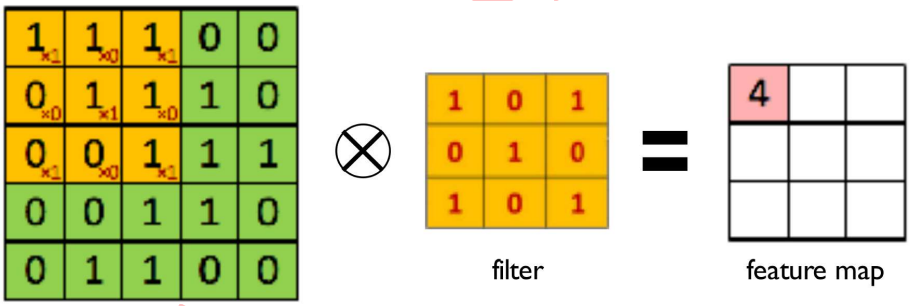


Figure 3.3: slide the 3x3 filter over the input image, element-wise multiply, and add the outputs[62]

We next slide 3×3 filter over by stride 1 to cover the next patch and repeat this element-wise multiplication in addition, this give us our second entry 3.

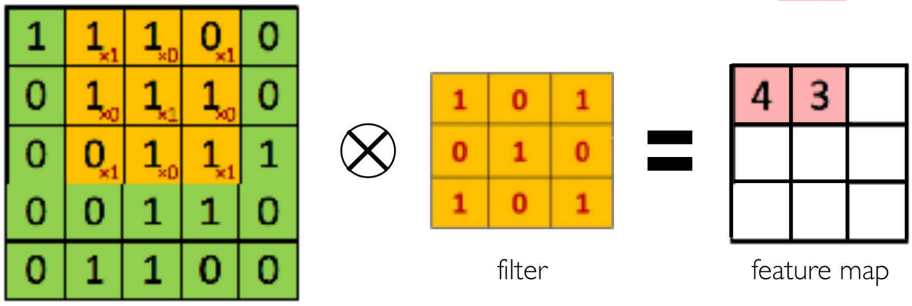


Figure 3.4: slide the 3x3 filter over the input image, element-wise multiply, and add the outputs[62]

We continue this process until we cover the input image entirely. Progressively sliding our filter to cover it, and doing this element-wise multiplication patch by patch and then adding the result and filling out the feature map.

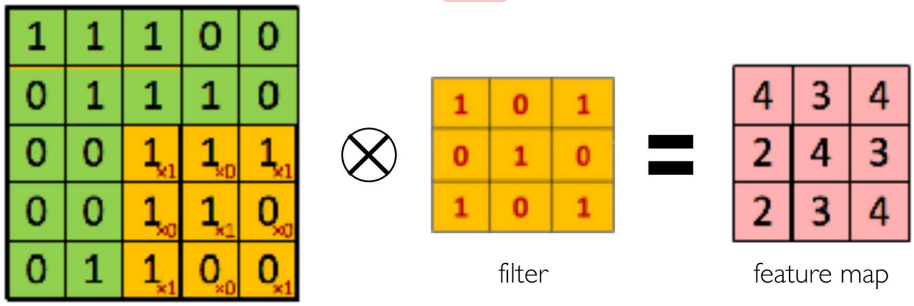


Figure 3.5: slide the 3x3 filter over the input image, element-wise multiply, and add the outputs[62]

This feature map reflects where in the input image was activated by this filter. To consider how powerful of the convolution operation, different weight filters can be used to produce distinct feature maps. As we see in these three examples [62], where three different filters are applied to the same input image to generate three very different outputs. By simply changing the weights of the filters we can detect and extract different features like edges and sharpen that are present in the input image. In the next section we explain partial convolution that is originally proposed to handle incomplete input data, such as images with holes.

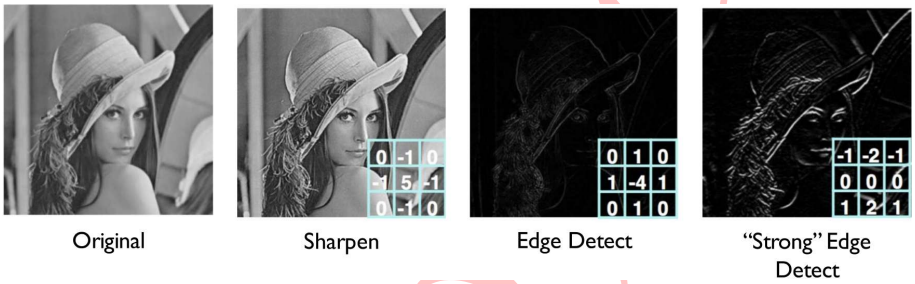


Figure 3.6: Producing Feature Maps[62]

3.1.2 Partial Convolutional Layer

Partial convolutional layer comprises a masked and renormalized convolution operation 3.1.1 followed by a mask-update step. The partial convolutional layer is basically a simple convolution but each input patch of image is multiplied with a binary mask. So, everywhere where the mask is, we set the pixels to zeros as Figure 3.7 shows and then we do the convolution operation.

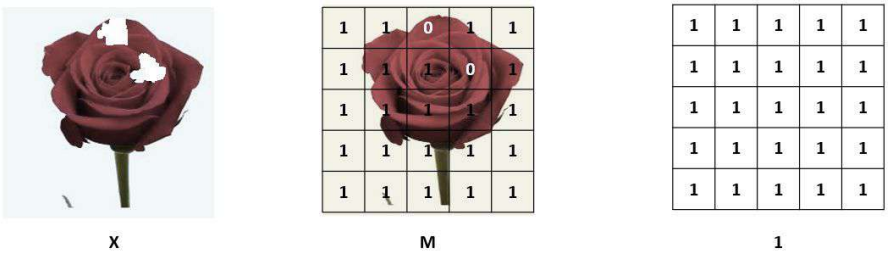


Figure 3.7: (left)Input image with holes. (middle) Multiplied mask with input image (holes are set to zeros). (right) Mask.

We refer to our partial convolution operation and mask update function jointly as the Partial Convolutional Layer. Let X be the feature values of

input image (pixel values) for the current convolution (sliding) window at the position (i,j) and M(i,j) be the corresponding binary mask, with the hole region being 0 and non-hole region being 1. The partial convolution (ignoring bias) at every location is similarly defined in [63] as:

$$x'_{(i,j)} = \begin{cases} W^T(X_{(i,j)} \odot M_{(i,j)})r_{(i,j)}, & \|M_{(i,j)}\|_1 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where

$$r_{(i,j)} = \frac{\|1_{(i,j)}\|_1}{\|M_{(i,j)}\|_1} \tag{3.3}$$

Where $\odot$ denotes element-wise multiplication, $1_{(i,j)}$ is the all-one vector with the same shape as $M_{(i,j)}$ and W is the filter weight matrix. We compute $x'_{(i,j)} = x'_{(i,j)} + b$ to account for an additional bias term (when $||M_{(i,j)}||_1 > 0$). As it can be seen, output values depend only on the unmasked inputs. The scaling factor $||1_{(i,j)}||_1/||M_{(i,j)}||_1$ applies appropriate scaling to adjust for the varying amount of valid inputs (unmasked inputs). We are only considering the pixels outside the masks and then we are doing the normalization, because convolution is based on sums, so if we are removing some elements then we need some normalization component, which just puts our activations back to the same level in irrespective of the mask size. So that is the difference with a normal convolution. One important thing is that the mask is also updated after each partial convolution operation. If the convolution was able to condition its output on at least one valid input value, then we mark that location to be valid. In other words, if we in one layer could reconstruct some pixels, when receptive field was covering some real pixels in the place of previous layer not the mask, then we are able to calculate some activations, and then we are updating this removing mask from this pixel. The mask updating function is expressed as:

$$m'_{(i,j)} = \begin{cases} 1, & \text{if } \|M_{(i,j)}\|_1 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

We are considering the reconstructed information in the subsequent layers. And thus, with sufficient successive applications of the partial convolution layer any mask will be shrinking from layer to layer and eventually be all ones if the input contained any valid pixels, then it usually disappears in the encoder part.

3.1.3 Introducing non-linearity

The next step is applying a non-linearity to the output of a convolutional layer. The most common activation function is Rectified Linear Unit (ReLU)

function, which is essentially a pixel-by-pixel operation that replaces all negative values that follow from a convolution with zero, and we can think of this as sort of a thresholding, where negative values after convolution indicate sort of negative detection of that associated feature (Figure 3.8).

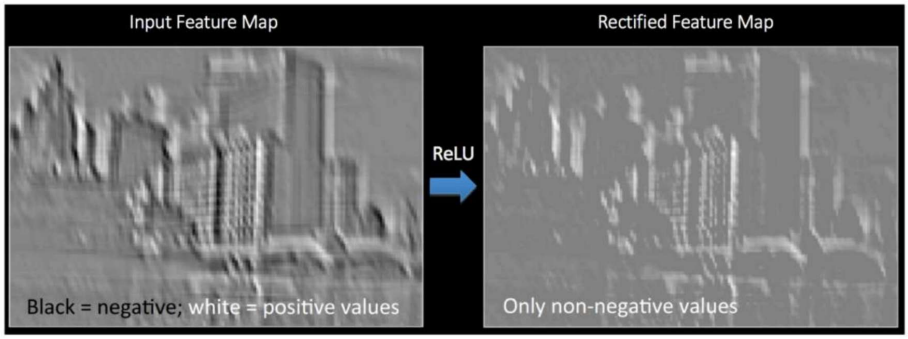


Figure 3.8: Applying ReLU after Conv Operation [62]

3.1.4 Pooling Layer

The final key operation in CNNs is pooling. Pooling is an operation that is used to reduce dimensionality from current layer to the subsequent one and to preserve spatial invariants. A common technique is Max Pooling as Figure 3.9 shows, we take the maximum value in a patch, where it is in this case a 2×2 patch that is being applied with a stride of 2 over this array.

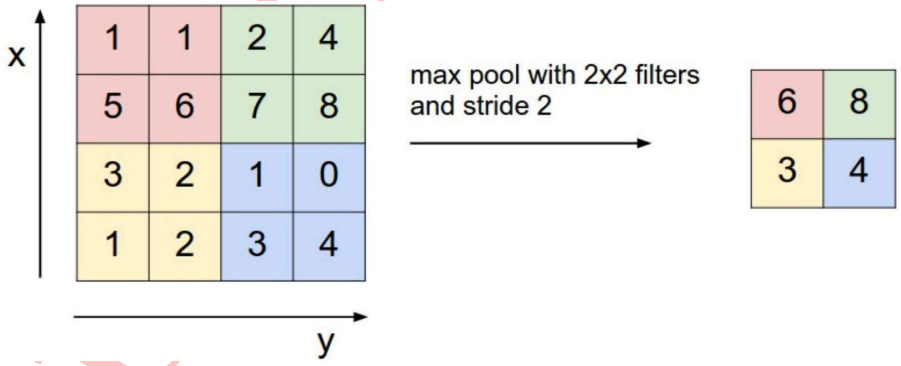


Figure 3.9: Max Pooling [62]

3.1.5 Fully Connected Layer

After we have flattened the high-level features' matrixes extracted by convolutional and pooling layers into a single vector, we can pass this vector

to the fully connected layer (Figure 3.10) to do the classification task. This fully connected layer can effectively output a probability distribution for the image's membership over a set of possible classes, and a common way to achieve that is by using softmax function whose output represents a categorical probability distribution over the set of classes that we are interested in.

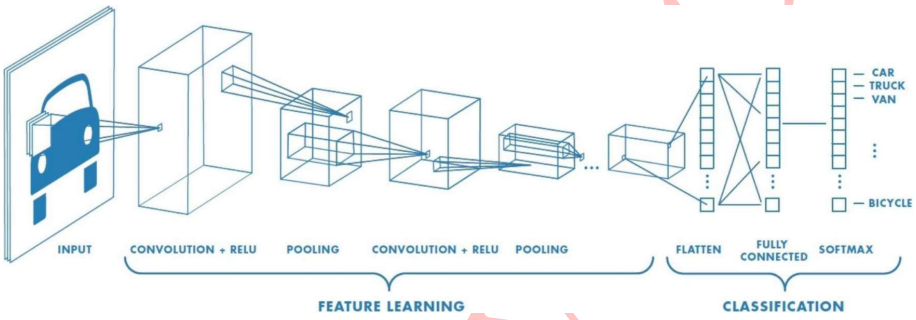


Figure 3.10: Complete CNN architecture [62]

3.2 Attention Mechanism in Deep Learning

Attention is one of the most influential ideas in the deep learning community. This mechanism has been used to solve various problem of sequence transduction [16] like image captioning, speech recognition, text-to-speech transformation and others. However, attention mechanism has shown the ability to produce state-of-the-art results in machine translation and other natural language processing tasks using sequence-to-sequence models that are composed of an encoder-decoder architectures such as the transformer [15]. The superiority of the transformer appeared due to ability to address the shortcomings of other approaches like recurrent and convolutional models to language understanding tasks such as language modeling, machine translation and question answering.

Recurrent Neural Networks (RNNs) with LSTM have in recent years become the typical network architecture for translation and processing language sequentially. However, reading one word at a time forces RNNs to perform multiple steps to make decisions that depend on words far away from each other. Prior research [64] has shown that the more such steps decisions are required, the harder it is for a recurrent network to learn how to make those decisions. The sequential nature of RNNs also makes it more difficult to fully take advantage of modern fast computing devices such as TPUs [65] and GPUs, which excel at parallel and not sequential processing. CNNs are much less sequential than RNNs, but in CNN architectures like ByteNet [66] or ConvS2S [67] the number of steps required to combine

information from distant parts of the input still grows with increasing the distance. In contrast, the transformer allows to draw global dependencies between different input and output positions at once, rather than one by one in RNNs, and reduce the entire operation to a constant number unlike that in CNNs, which would allow for significantly more computation parallelization, what it is exactly needed by the seq2seq model in scene text recognition.

In following sections we would like to present the transformer architecture, and what it consists of, and how its components work together to compute representations of its input and output without using sequence aligned RNNs or CNNs.

3.3 Transformer Architecture

The transformer follows the encoder-decoder framework. Both of the encoder and decoder have almost the same modules as follows: stacked self-attention, position-wise fully connected feed-forward network, and positional encoding as shown in the left and right halves of Figure 3.11, respectively.

3.3.1 Encoder

The encoder is composed of stack of N identical layers. Each layer is broken down into two sub-layers. The first one is a multi-head self-attention mechanism, and the second one is a position-wise fully connected feed-forward network. The encoder's inputs first flow through a self-attention layer that helps the encoder to look at other input sequences. The outputs of self-attention layer are then fed into a feed-forward neural network. Each sub-layer in each layer of the encoder has a residual connection [68] around it, and is followed by a layer normalization [69] to ease the training of the network as these residual connections are easier to optimize, whereas the layer normalization reduces significantly the training time. The output of each sub-layer is as follows:

$$LayerNorm(x + Sublayer(x)) \tag{3.5}$$

Where $Sublayer(x)$ is the function implemented by the sub-layer itself. In order to facilitate the residual connections all sub-layers in the model as well as the embedding layers produce the same output dimension.

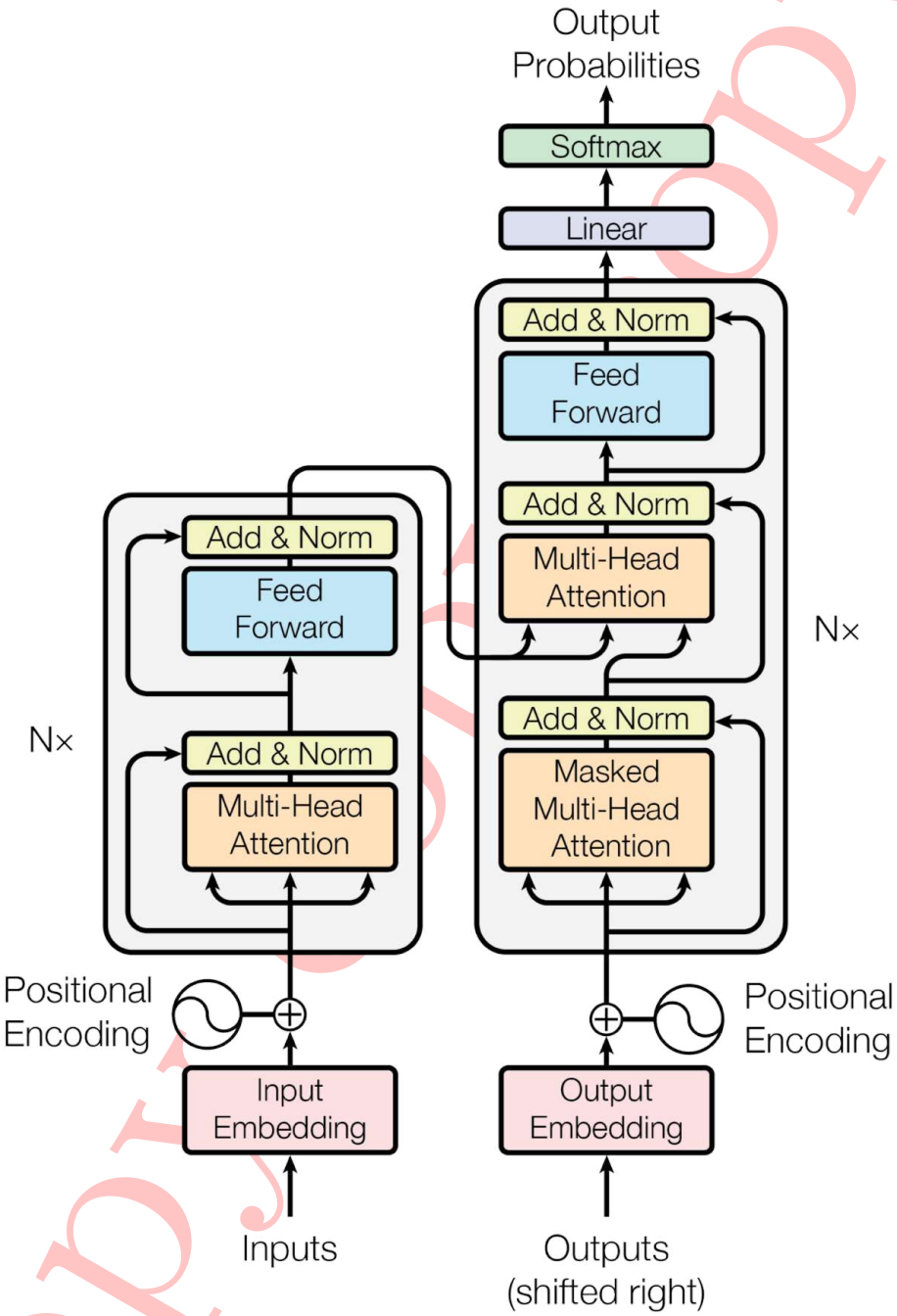


Figure 3.11: The Transformer Architecture.(left) The Encoder.(right) The Decoder [15]

Input Embedding

Like in most of sequence-to-sequence models, the learned embeddings are used to convert the input character targets to list of vectors, these lists of vectors called Input Embeddings (left half of Figure 3.11).

Positional Encoding

As the transformer now seems (Figure 3.11), it doesn't contain recurrent or convolution network which in turn leads to missing a way to account for the order of tokens the input sequence. To address this problem, the transformer adds some information or rather a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps to determine the relative or absolute position of the tokens in the input sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors when they are projected in Query/Key/Value vectors during Dot-Product attention, as we will see later.

The positional encodings will be added to the input embeddings at the bottom of the encoder and decoder stacks. The positional encodings have the same dimension as the embeddings, so that the two can be summed. In the transformer are sine and cosine functions of different frequencies used for positional encoding:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right) \tag{3.6}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right) \tag{3.7}$$

It is actually not the only possible method for positional encoding. There are many other choices such that learned and fixed positional encoding [70]. However, the used method in this transformer gives the advantage of being able to scale to unseen lengths of sequences.

3.3.2 Self-Attention Mechanism

As we've mentioned already, the encoder receives a list of vectors as input. It processes this list by passing these vectors into a multi-head self-attention, then into a feed-forward neural network as shown in the left half of Figure 3.11, afterwards it sends out the output upwards to the next encoder, since the encoder consists of stack of N identical layers (encoders). As a matter of fact, we have a stack of N encoders in the encoder block. However, the unit that is responsible for the self-attention computation is the multi-head attention layer (the right half of Figure 3.12). To understand how it works we have to dive inside this block and explain the layers inside it, and that what we are going to do in the following next sections. However, According

to conducted experiments explained in "Attention Is All You Need" [15], using self-attention would be summarized in the following set of benefits:

- The total computational complexity per layer would be reduced, where the sequential operations can be parallelized.
- The ability to learn the long-distance dependencies in the network despite the path length.

Scaled Dot-Product Attention

Self-attention is a mechanism to extract useful information from different positions of an input sequence for each position of the outputs. An attention function can be imagined as a mapping function that maps a query and a set of key-value pairs to an output, where the queries, keys, values, and the output are all vectors. In the transformer the self-attention function is called Scaled Dot-Product Attention (the left half of Figure 3.12). Specifically, it has three inputs: Queries (Q), Keys (K), and Values (V). The first step in calculating the self-attention is to create three vectors from each of the encoder's input vectors, in this case the input embeddings. For each input sequence, we create three vectors that are a query vector (Q) and a key vector (K) of the same dimension of key vector, whereas a value vector (V) has not necessarily the same dimension. These vectors are created by multiplying the embeddings by three matrices that they were trained during the training process. The second step is to compute the dot product (or what is known as score) of the query with all keys to determine how much focus to be placed on other parts of the input sequence at a certain position. The score is calculated by taking the dot product of the query vector with the key vector of the respective input we are scoring, then the second key of the same query and so on. The third and fourth steps are to scale the dot product by dividing the scores by $\sqrt{(d_k)}$ (the square root of the dimension of the key vectors). This leads to having more stable gradients, then pass the result through a softmax operation to obtain the weights on the values, where the softmax function normalizes the scores so they are all positive values and sum to 1. The goal of scaling the dot products by a scalar $(1/(\sqrt{(d_k)}))$ is to prevent softmax function from going into regions where it has extremely small gradients. The fifth step is to multiply each value vector by the softmax score. The objective of this step is to keep the input sequence that we would like to focus on and drop out the irrelevant ones. The last step is to sum up the weighted value vectors, which produces the output of the self-attention layer at a certain position as shown in the left half of Figure 3.12, queries, keys and values are represented as $K \in t_k \times d_k$, $Q \in t_q \times d_q$, $V \in t_v \times d_v$ respectively, where t means element numbers of corresponding inputs and d is the corresponding element dimensions. In general, we set $t_k = t_v$ and

$d_q = d_k$. We compute the matrix of outputs as:

$$Attention(Q, K, V) = softmax((QK^T)/\sqrt{d_k})V \tag{3.8}$$

This concludes the self-attention computation. The resulting vector is one, which is then sent along to the feed-forward neural network. However, this computation is done in matrix form for faster processing.

Multi-Head Attention

By adding a mechanism called Multi-Head Attention (the right half of Figure 3.12) instead of applying just one single attention function "Scaled Dot-Product Attention" (the left half of Figure 3.12) allows to improve the performance of the attention layer, where it expands the model's ability to jointly focus on different positions of the input sequence. Moreover, it gives the attention layer multiple representation subspaces and the model can attend to information from different representation subspaces at different positions. With representation subspaces we mean that the multi-head attention has not only one, but multiple sets of Query/Key/Value weight matrices which are called heads (h). Each of these sets is randomly initialized. Then after training, it is beneficial and effective to linearly project the input embeddings represented as (queries, keys, and values) h times with different and learned linear projections to d_k, d_k and d_v dimensions , respectively. On each of these projected versions of queries, keys and values is then scaled dot-product attention performed in parallel, resulting in d_v dimensional output values. However, that causes a bit of a challenge because the feed-forward layer is expecting to receive from multi-head attention layer a single vector and not h matrices. Thus, we need to compress these h matrices into one single vector. For this purpose, the outputs from multi-head attention are concatenated and once again projected to obtain the final d_{model} dimensional outputs:

$$MultiHead(Q, K, V) = Concat(head_1, ...head_h)W^o \tag{3.9}$$

Where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.10}$$

Q, K, V have the dimension of d_{model}, the predictions are parameter matrices $W_i^Q \in \mathcal{R}^{d_{model} \times d_k}$, $W_i^K \in \mathcal{R}^{d_{model} \times d_k}$, $W_i^V \in \mathcal{R}^{d_{model} \times d_v}$ and $W^O \in \mathcal{R}^{hd_v \times d_{model}}$. W^O is a weight matrix that was trained jointly with the model.

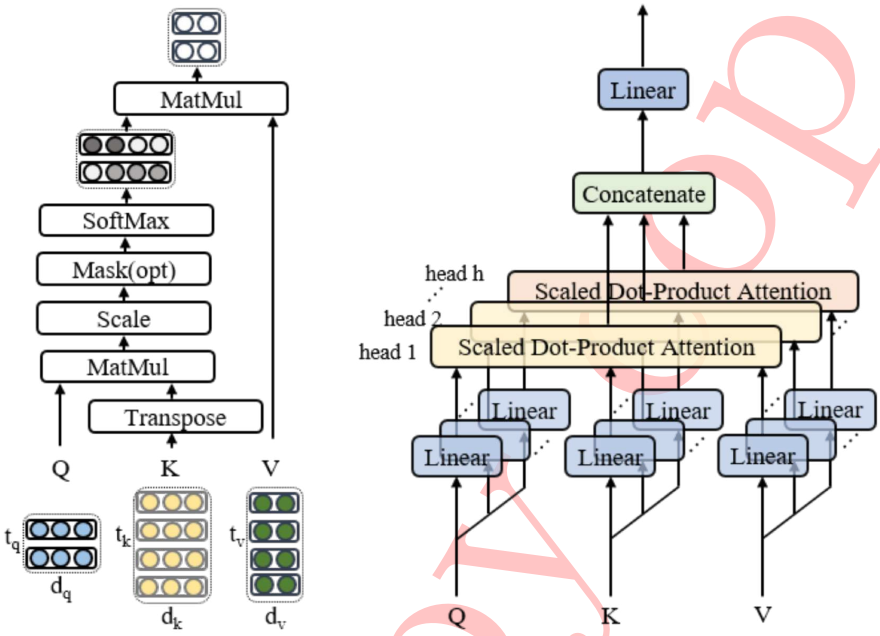


Figure 3.12: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several Scaled Dot-Product Attention running in parallel [14]

3.3.3 Position-wise Feed-Forward Network

Each of the layers in the encoder and decoder contains a fully connected feed-forward network as shown in Figure 3.11, which is applied to each position separately and identically. The output of the multi-head attention is a matrix that captures information from all the attention heads, which is then sent to the Feed-Forward Network (FFN). It consists of two linear transformations with a RELU activation 3.1.3 in between.

$$FNN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.11}$$

Where the weights are $W_1 \in \mathcal{R}^d_{model} \times {}^d_{ff}$ and $W_2 \in \mathcal{R}^d_{ff} \times {}^d_{model}$ and bias are $b_1 \in \mathcal{R}^d_{ff}$ and $b_2 \in \mathcal{R}^d_{model}$. The linear transformations are the same across different positions.

3.3.4 Decoder

As shown in the right half of Figure 3.11, the decoder is also composed of a stack of N identical layers. The decoder has almost the same components like those in the encoder 3.3.1. In addition to the two sub-layers in the encoder

that are a multi-head self-attention 3.3.2, and a position-wise fully connected feed-forward network 3.3.3, the decoder employs also a third sub-layer, which performs multi-head attention as well, but this time over the output of the encoder stack. Similar to the encoder, there are residual connections [68] around each of the sub-layers, followed by layer normalization [69] as well. We now take a look at how both of the encoder and decoder communicate and work together.

The output of the top encoder is then transformed into a set of attention vectors as a form of Key(K) and Value(V). These vectors are used by each multi-head attention layer 3.3.2 in the decoder layer which help the decoder to focus on appropriate places in the input sequence. The following steps still repeat the process until a special symbol is reached (For Example. <end of the sequence>) that is indicating that the decoder has completed its output. The output of each step is then fed to the bottom of the decoder in the next time step and the decoders keep their results to maintain the dependencies of the successive decoded sequence. Just like the encoder, the decoder embeds and adds positional encoding 3.3.1 to those decoder inputs to indicate the position of each input sequence. The multi-head attention layers in the decoder operate in a slightly different way than the ones in the encoder. In the decoder the multi-head attention layer receives its keys and values from the encoder outputs, and queries from the previous decoder block outputs. Moreover, at the bottom of the decoder there is a *Masked Multi-Head Attention Layer* that makes sure to only attend to earlier positions and prevent positions from attending to subsequent positions in the output sequence. This is done by masking future positions before the softmax step 3.8 in the self-attention computation, where the masking ensures that the predictions for position j can only depend on the previous known outputs at position less than j.

Linear transformation and softmax function

Finally, the decoder stack outputs a vector of floats, where a learned linear transformation and softmax function are used to covert these decoder output vectors into predicted next-token probabilities. The linear layer is a simple fully connected neural network 3.1.5 that transforms the output vectors produced by the stack of the decoders into a logits vector. Then, the softmax layer turns those logits into probabilities which all positive and add up to 1.

Chapter 4

Proposed Models

In this chapter we are going to introduce the models and the approaches that we use. We explain the theoretical and the technical issues of two models and provide more details about the techniques and methods that are applied to end up with a comprehensive composite model Figure 4.1, which is able to reconstruct occluded text in images by an inpainting model then recognize the text using an OCR engine.

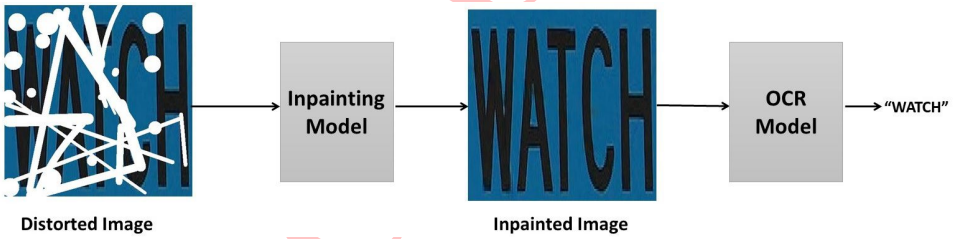


Figure 4.1: The Pipeline of both Reconstruction and OCR Model

4.1 Image Reconstruction Model

The architecture of the proposed model is based on U-Net network [13], which is based on encoder-decoder framework. However, instead of using normal convolutional operation 3.1.1, the network uses stacked partial convolution operation and mask updating steps, which together form a *Partial Convolutional Layer* 3.1.2 to perform image reconstruction. In this section we discuss the network architecture of the model and explain the loss function in detail.

4.1.1 Network Architecture

The network architecture is illustrated in Figure 4.2. The network is similar to U-Net architecture [13] but with some modifications to fit our task. It consists of an encoder and decoder with replacing all convolutional layers with partial convolutional layers that take into account also masks and using nearest neighbor up-sampling in the decoding stage. The straightforward implementation is to define binary masks of size C × H × W, and the same size with their associated images. Mask updating is implemented using a fixed convolution layer with the same kernel size (filter size) as the partial convolution operation, but with weights identically set to 1 and bias to 0.

The encoding stage consists of repeated application of partial convolutions with stride 2 for down-sampling. Each partial convolution is followed by a nonlinearity layer, for which is a rectified linear unit (ReLU) 3.1.3 is used and a batch normalization layer [69] except the first partial convolution layer. At each down-sampling step we double the number of feature channels (number of the filters) to be at the end up to 512 filters. The encoder contains 8 partial convolutional layers with specified filter size, where the first partial convolution has a filter size of 7×7 which results in 256×256 feature map, the second and third partial convolution have a filter size of 5×5, which produces a feature map of 128×128 and 64×64 respectively. And then, a repeated filter size of 3×3 for the remaining five partial convolutions, where the last partial convolutional layer (the eighth) at the end of the encoding stage sends out a 2×2 feature map forward to the decoder.

Every step of the decoding stage consists of an up-sampling layer of the feature map followed by a 3×3 partial convolution ("Up-partial convolution") that halves the number of feature channels (number of filters), and a concatenation (skip link) with the correspondingly cropped feature map from the encoding stage. Skip links are mentioned in Figure 4.2 as "concatenate", which concatenates the previous nearest neighbor up-sampled results with the corresponding partial convolution results from the encoder stage. For example, the number of filters at the fourth concatenation can be computed as following: $Concate4 = PConv4_{(encoder)} + NearestUpsample4_{(decoder)}$ which yields $512+512 = 1024$ filters, and for the fifth concatenation is $Concate5 = PConv3_{(encoder)} + NearestUpsample5 = 256 + 512 = 768$ filters, and so on. Each partial convolution of size 3×3 is followed by a LeakyReLU(0.2) as nonlinearity layer, and a batch normalization layer as well. In total the network including the encoder and the decoder has 16 partial convolutional layers.

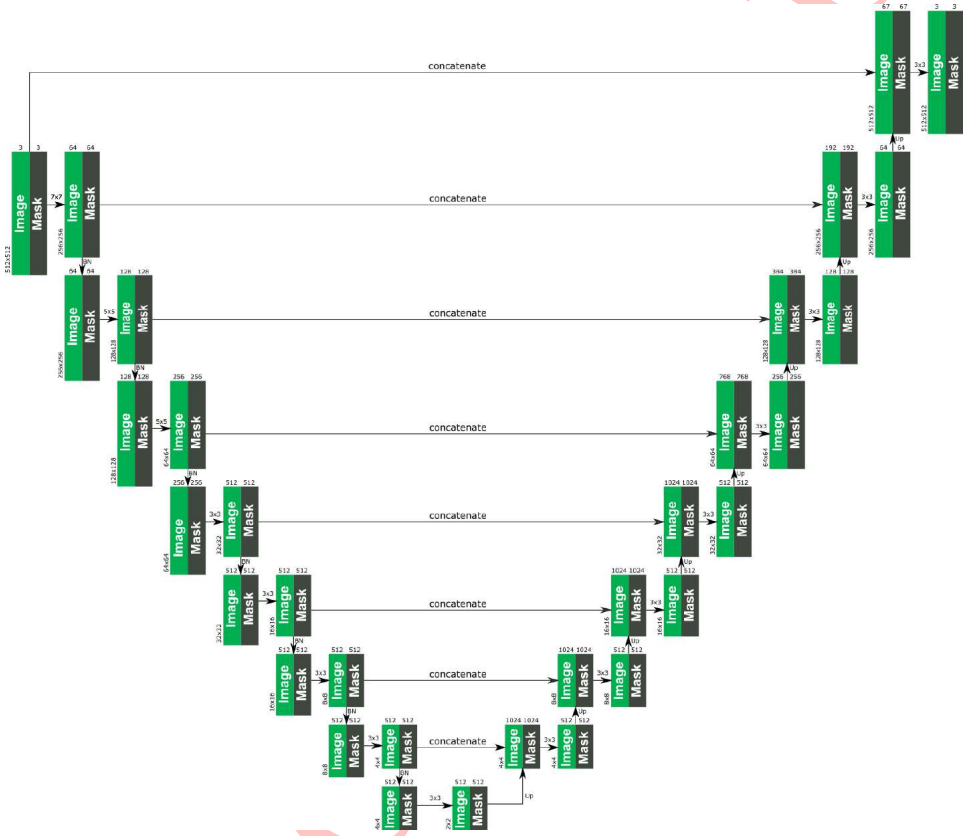


Figure 4.2: The U-Net Architecture. The Encoder (left), the Decoder (right) [71] [72].

4.1.2 Loss Function

To optimize the model parameters we need a loss function. Our loss function is composed of many components, which target both per-pixel reconstruction accuracy as well as how smoothly the predicted hole values integrate into their surrounding context.

Hole and Non-hole per Pixel Losses

The first loss component is based on per-pixel reconstruction error. Here we are considering two elements: one is calculated inside the mask and the other one is outside of the mask. We have two per-pixel losses both for masks and unmasked regions. Given input image with hole I_{in}, initial binary mask M (0 for holes), the network prediction I_{out}, and the ground truth image I_{gt}. The per-pixel losses are defined as follows:

$$\mathcal{L}_{hole} = \frac{1}{N_{I_{gt}}} \left\| (1 - M) \odot (I_{out} - I_{gt}) \right\|_1 \tag{4.1}$$

$$\mathcal{L}_{valid} = \frac{1}{N_{I_{gt}}} \left\| M \odot (I_{out} - I_{gt}) \right\|_1 \tag{4.2}$$

Where $N_{I_{gt}}$ denotes the number of elements in I_{gt}($N_{I_{gt}} = C * H * W$) C, H ,and W are the channel size, height and width of image I_{gt}. The both L_{hole} and L_{valid} are the L^1 losses on the network output for the hole and the non-hole pixels respectively, and they are all size-averaged.

Perceptual Loss

It is the second loss component of the loss function introduced by Gatys et al. [73]. It looks at two images (the ground truth image and the output image), but not in a pixel space rather in a higher-level feature space. We are extracting some features from a pre-trained VGG-16 model on ImageNet [25]. Here we are calculating this part by comparing these features for two outputs taking L^1 losses and summing over these free layers, and also we do this not only for our output image but also for so called "composite image" which is composed of the reconstructed mask and original pixels that are put around, so here in the whole formulation we put more attention to the inside of mask reconstruction error. We define the perceptual loss as:

$$\mathcal{L}_{perceptual} = \sum_{p=0}^{P-1} \frac{\left\| \Psi_p^{I_{out}} - \Psi_p^{I_{gt}} \right\|_1}{N_{\Psi_p^{I_{gt}}}} + \sum_{p=0}^{P-1} \frac{\left\| \Psi_p^{I_{comp}} - \Psi_p^{I_{gt}} \right\|_1}{N_{\Psi_p^{I_{gt}}}} \tag{4.3}$$

Here, I_{comp} is the raw output image but with the non-hole pixels directly set to ground truth. $N_{\Psi_p^{I_{gt}}}$ is the number of elements in $\Psi_p^{I_{gt}}$. $\Psi_p^{I_*}$ is the

activation map of the p^{th} selected layer given original input I_*. We use layers pool1, pool2 and pool3 of ImageNet-pretrained VGG-16 for our loss.

Style Loss

It is similar to Perceptual Loss but before taking L^1 we are performing an autocorrelation using some gram matrix on each feature map, and then after the autocorrelation we do the same more or less with some normalization factor depending on the size of our feature map taken from VGG-16, which is number of channels (C_n), height (H) and width (W) of our feature map.

$$\mathcal{L}_{style_{out}} = \sum_{p=0}^{P-1} \frac{1}{C_p C_p} \left\| K_p((\Psi_p^{I_{out}})^T(\Psi_p^{I_{out}}) - (\Psi_p^{I_{gt}})^T(\Psi_p^{I_{gt}})) \right\|_1 \tag{4.4}$$

$$\mathcal{L}_{style_{comp}} = \sum_{p=0}^{P-1} \frac{1}{C_p C_p} \left\| K_p((\Psi_p^{I_{comp}})^T(\Psi_p^{I_{comp}}) - (\Psi_p^{I_{gt}})^T(\Psi_p^{I_{gt}})) \right\|_1 \tag{4.5}$$

Here we note that the matrix operations assume that the high level features $\Psi(x)_p$ is of shape $(H_p W_p) \times C_p$, resulting in $C_p \times C_p$ Gram Matrix, and K_p is the normalization factor $1/(C_p H_P W_p)$ for the p^{th} selected layer. We include loss terms for both raw output and composited output. It should be mentioned that these two components Perceptual Loss and Style Loss are used also in other problems like style transfer, and they are more in line with human perception than the simple reconstruction error from the previous component.

Total Variation (TV) Loss

It is the last component in the loss functions. It is a kind of a penalty for non-smooth output on R [74]. We are calculating it in the area R which is our mask slightly enlarged by a deletion operation. Here we want the output to be smooth inside the mask and on the boundary between the mask and the original image. We define the total variation loss as:

$$\mathcal{L}_{tv} = \sum_{(i,j)\in R,(i,j+1)\in R} \frac{\left\| I_{comp}^{i,j+1} - I_{comp}^{i,j} \right\|_1}{N_{I_{comp}}} + \sum_{(i,j)\in R,(i+1,j)\in R} \frac{\left\| I_{comp}^{i+1,j} - I_{comp}^{i,j} \right\|_1}{N_{I_{comp}}} \tag{4.6}$$

Where $N_{I_{comp}}$ is the number of elements in I_{comp}.

Total Loss (L_{total})

The Total Loss is a combination of all above loss functions. It is a weighted sum of the whole of all these previous loss components.

$$\mathcal{L}_{total} = \mathcal{L}_{valid} + 6\mathcal{L}_{hole} + 0.05\mathcal{L}_{perceptual} \tag{4.7}$$
$$+120(\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + 0.1\mathcal{L}_{tv} \tag{4.8}$$

These weights were determined by performing a hyperparameter search on 100 validation images [54]. However, these weights depend on many factors. For example, on on-hand data and the pre-trained model that is used to compute the Perceptual and Style Loss. Consequently, one has actually to tune the weights to each particular problem and just monitor the contribution of each loss component during the training.

4.2 Scene Text Recognition Model

As we saw in related works chapter 2 the most previous sequence-to-sequence models are based on complex recurrent or convolutional neural networks or both together that is an encoder-decoder architecture. In this thesis we present a quite different sequence-to-sequence scene text recognition model that the mentioned existing approaches. Our model follows the encoder-decoder structure dispensing totally with RNNs and CNNs through an attention mechanism. Both of encoder and decoder are based on self-attention module to learn positional dependencies, which could be trained with more parallelization and less complexity. The network architecture recycles the transformer architecture proposed in "Attention Is All You Need" [15], besides to the core modules in the transformer [15] is a modality-transform block added in the encoder to transform an input image to the corresponding sequence, which could be considered as a pre-processing step. After explaining the transformer architecture (see 3.3) we would like now to discuss the overall architecture of NRTR [14], and how the modality-transform block is integrated in the transformer that forms at the end a consolidated model to recognize scene text.

4.2.1 Overall Architecture of NRTR

The overall architecture of NRTR [14] follows the encoder-decoder framework and contains the same core modules of the transformer (see 3.3), which perform the same functions that we explained previously but with some slight changes in the structure of the encoder and decoder.

Encoder

As shown in the left half of Figure 4.3 the encoder employs three normalization layers, whereas the encoder in the transformer 3.3.1 employs just two. In addition to the common previous modules in the transformer such that

scaled dot-product attention 3.3.2, multi-head attention 3.3.2 and position-wise feed-forward network 3.3.3, the encoder in NRTR inserts an essential module called *"Modality-Transform Block"* at the bottom before feeding the input image into the encoder as shown in left half of Figure 4.3.

Modality-Transform Block

It's considered as a preprocessing step for conversion of the input image. The modality-transform block consists of consecutive convolutional layers and in between batch normalization and ReLU layers. The intuition here is to convert each input image to the corresponding sequence that is compatible with the encoder length vector. This process is also known as "input encoding" because only after that the resulted image sequence can be fed into the encoder directly. Each input image goes through the layers of the modality-transform block to convert its dimension and then a concatenate operation is applied to reshape the previous image result into an encoder-length vector, and thereby it could flow into the encoder for the following modeling. However, we get the final input encoding after adding the positional encoding to the output of modality-transform block. Afterwards, it goes through a self-attention layer, then it is fed into a feed-forward neural network. Finally, the last encoder layer sends out the encoder's output upwards to the decoder.

Decoder

As shown in the right half of Figure 4.3 at the bottom of the decoder there is a character embedding to convert the input character targets to vectors that are compatible with the dimension of the decoder, and again the resulted sequences are then added with positional encoding and fed into a stack of N identical decoder layers. The *Masked Multi-Head Attention* receives the output encoding and sends it forward to the multi-head attention 3.3.2 that has keys and values come from the encoder outputs, and queries come from the previous decoder outputs as shown in the right half of Figure 4.3, then the resulted output is fed into a position-wise feed-forward network 3.3.3. At last, the decoder outputs are transformed to the probabilities of output classes by a linear projection and a softmax function 3.3.4.

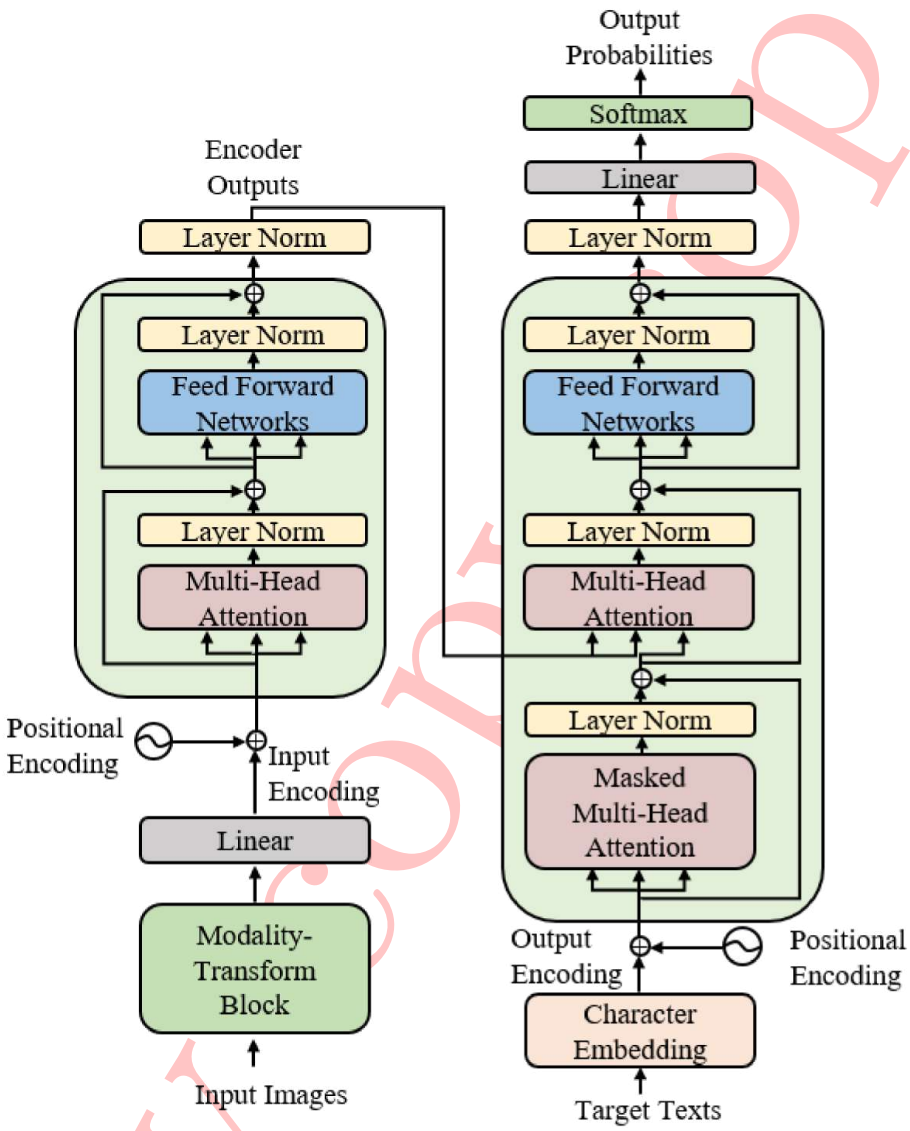


Figure 4.3: The overall architecture of NRTR [14]. (left) the Encoder, (right) the Decoder

4.2.2 Output of NRTR

We need a loss function that measures the cost incurred from incorrect predictions. This means, we want the output to be a probability distribution, in our toy example 4.4 we consider right now just a single character "h" of a sequence of characters "hello". However, since the model is not yet trained it is unlikely to happen just yet.

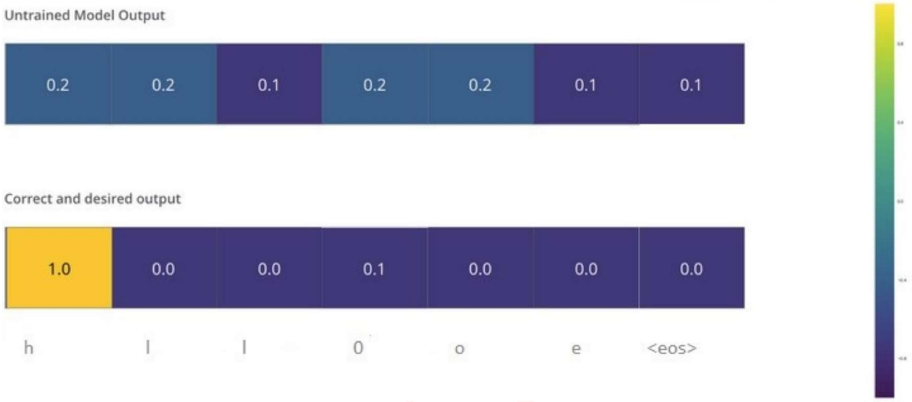


Figure 4.4: The probability distribution for untrained model [75].

Since the model's parameters (weights) are all initialized randomly, the untrained model produces a probability distribution with arbitrary values for each cell (character). We can compare it with the actual output, then improve the model's weights using adam optimizer [76] to make the output closer to the desired output. To compare two probability distributions we simply subtract one from the other such as cross-entropy [77] and Kullback-Leibler divergence [78]. However, the example 4.4 is very oversimplified just for one character. More realistically, we will use now a word longer than one character to be predicted as shown in Figure 4.5. We want our model to successively output probability distributions of all characters in the word where:

- Each probability distribution is represented by a vector of width vocabulary size (in our toy example is 7).
- The first probability distribution has the highest probability at the cell associated with the character "h". The second probability distribution has the highest probability at the cell associated with the character "e" and so on until the seventh output distribution has been reached that indicates '<end of the sentence>' symbol, which also has a cell associated with it.

Hopefully upon training, the model would output the right characters as we expect as shown in Figure 4.6. As long as the model produces the outputs

one at a time, we can assume that the model is selecting the character with the highest probability from the probability distribution and throwing away the rest.

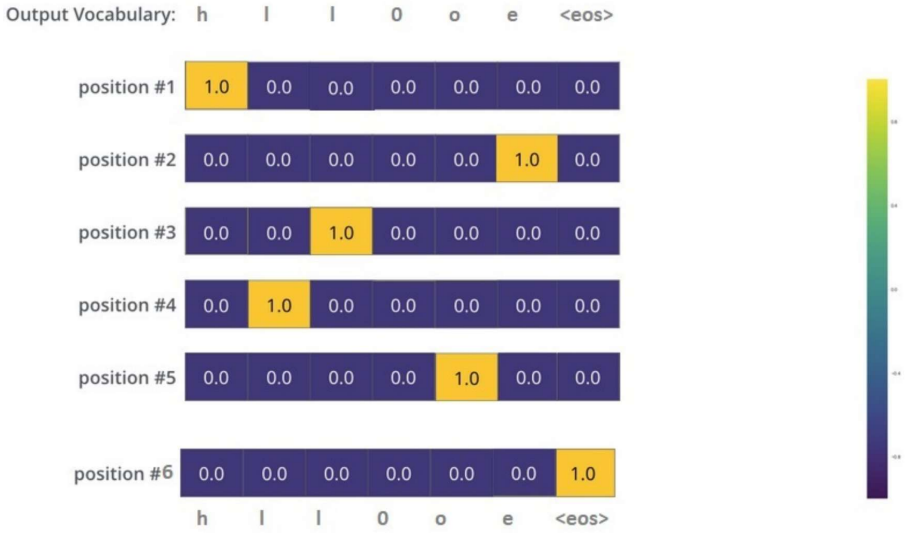


Figure 4.5: Target Model Output [75].

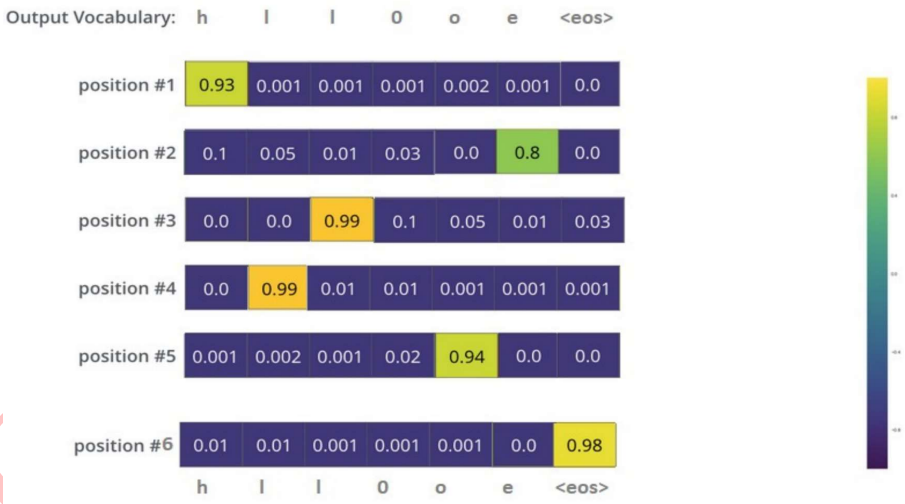


Figure 4.6: Trained Model Output [75].

Chapter 5

Experiment

To evaluate the effectiveness of the proposed image reconstruction and scene text recognition model, we conduct experiments on natural text image dataset. Further, in this section we explain the implementation details for both models. We show the architectures of networks, the stages of the training process, the hyperparameters that set the models and control the performance, the optimization functions which used to learn and optimize the models, and the accuracy functions as a quantitative indicator that determines how good the models are. In addition, we explore some parameters of the core modules in both models and investigate the structure of essential blocks.

5.1 Datasets

5.1.1 Training Dataset

For all experiments on both proposed models we use a new large-scale COCO-Text dataset [79] [80], which is based on MSCOCO dataset [81] that contains images of complex everyday scenes and based on real scene imagery (as opposed to synthetic images). The images are not collected with text in mind and thus they contain a broad variety of text instances. The text instances are categorized into a set of diverse categories such that machine printed and handwritten text, legible and illegible text, English script and non-English script. However, for our experiment we consider only COCO-Text words labeled as English and legible (machine printed 5.1, handwritten 5.2), and words longer than 3 characters. The ground-truth format (text labels) is one text file per split with one line per word with the format of (filename, transcription) as the Figure 5.3 shows, the transcription is utf-8 encoding. The COCO-Text dataset contains in total 63,686 cropped text images, which is split into 43,686 for train set, 10,000 validation set, and 10,000 test set with no public annotations.

Figure 5.1: Legible – Machine Printed Images

Figure 5.2: Legible – Handwritten Images

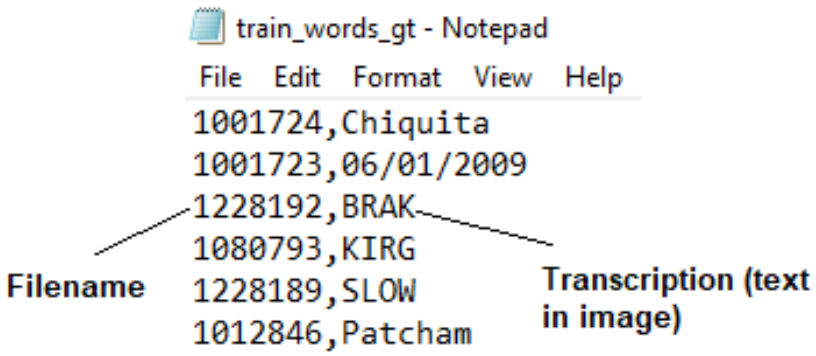


Figure 5.3: Groundtruth format for text images

5.1.2 Irregular Mask Dataset

Previous works 2.1 generate holes in their datasets by randomly creating rectangular regions within image datasets. We consider that not sufficient, because our work aims to handle diverse irregular hole shapes and sizes at different positions in the image, and not only rectangular regions that are located in certain position in the image. We create masks of random streaks and holes of arbitrary shapes as Figure 5.4 shows, where during the training we augment the masks and later perform random dilation, rotation and cropping. All the masks and images for training and testing are with the size of 512×512. However, the authors of Image Inpainting using PConv

[54] create irregular masks by using occlusion/dis-occlusion mask estimation method between two consecutive frames of videos [82], they generate 55,116 masks for the training and 24,866 masks for testing [83] Figure 5.5 shows some example of these masks.

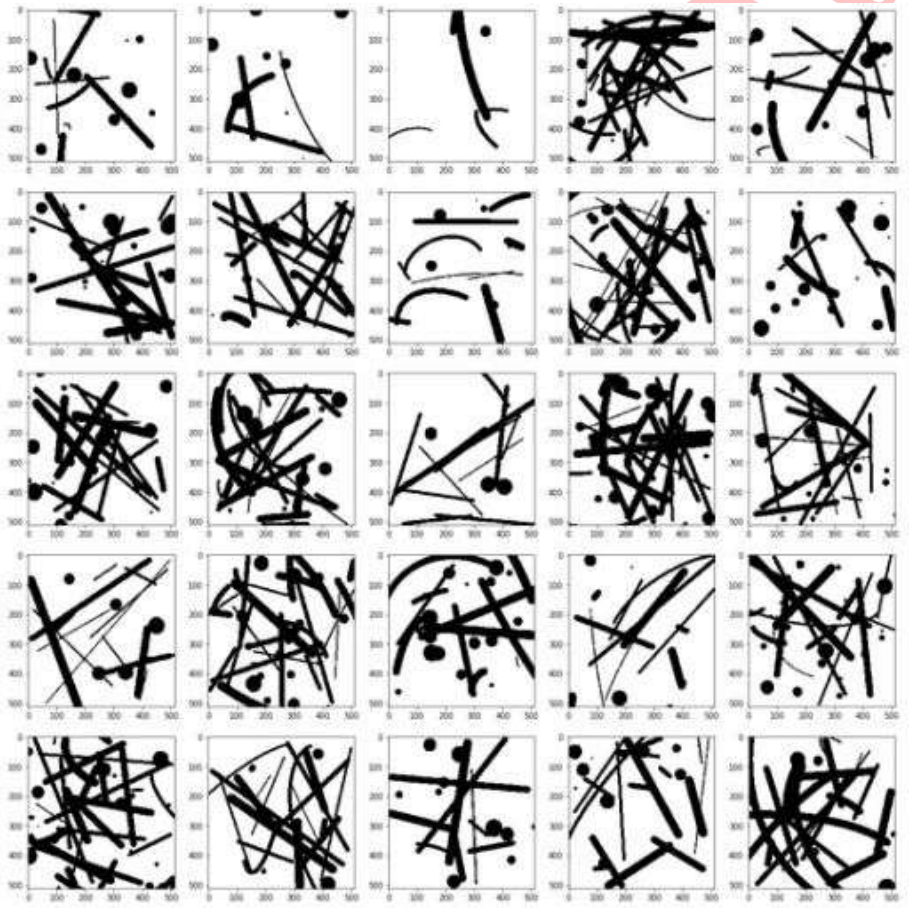


Figure 5.4: Some Examples of our irregular masks.

5.2 Image Reconstruction Model

5.2.1 Training Procedure

We train on a single NVIDIA Geforce GTX 1080 Ti (11 GB) with a batch size of 6, and each epoch is specified to be 10,000 batches long. Furthermore, for optimization we use Adam Optimizer [76], and initialize the weights using a robust initialization method described in [84] that particularly considers the rectifier nonlinearities. This method addresses the issues of training rectified models that are very deep (e.g., $>$ 8 convolutional layers) such that

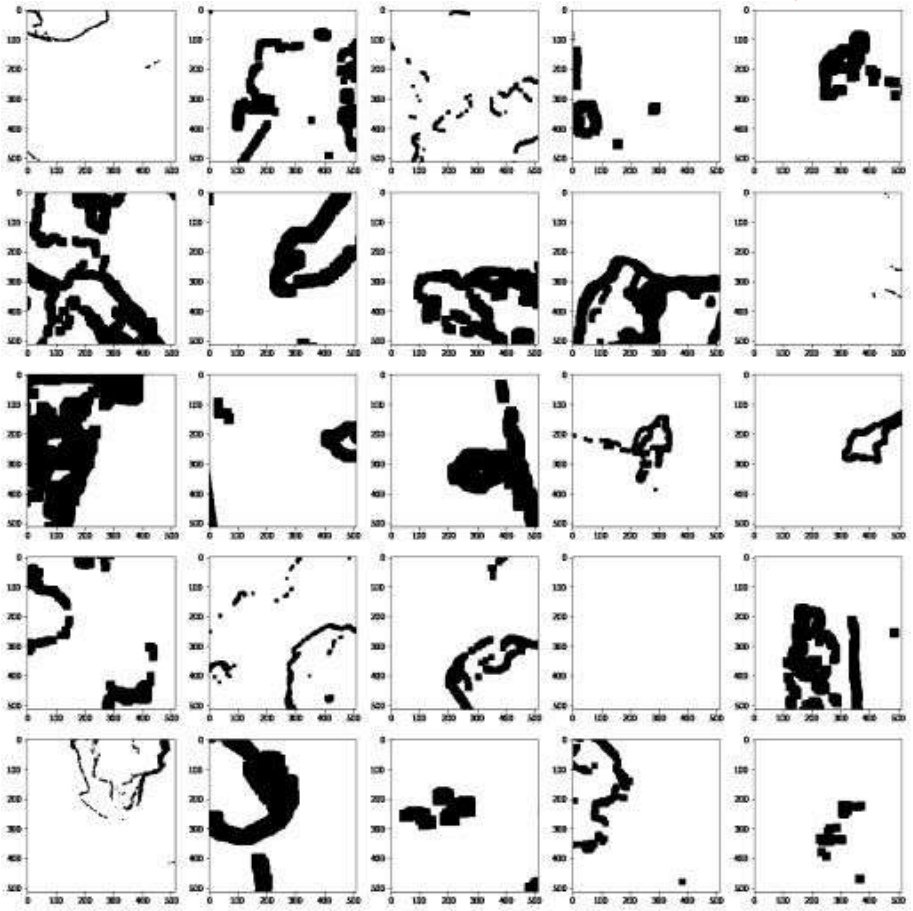


Figure 5.5: Some Examples of masks dataset [83] used in the paper [54]

the difficulties to converge as reported by VGG team [85]. The proposed initialization method enables us to train extremely deep rectified models like in our case (16 partial convolutional layers) directly from scratch. Moreover, this method avoids leading to a poorer local optimum.

Batch Normalization and Fine-Tuning

In general holes (mask) generate some problems with batch normalization, because the mean and variance will be computed for hole pixels, and thus various mask sizes affect the activation distributions, which could be seen as a kind of artifacts in the place of masks such that the non-smoothness at the masked locations in the image. The problem with it is that our model actually treats the boundaries of the image also as masks, and so when the model processes the middle part of the input image it gets some different activations because it is used to see a boundary around but now there is no

boundary and still image, so the activations slightly differ and in this extreme case when we do the reconstruction with no mask (the model just supposed that there is a mask) we get some problems with batch normalization that are also visible as artifacts. However, in order to use batch normalization in the presence of holes we can apply two-phase training technique:

- **Phase 1:** We first enable batch normalization in all layers for the initial training (in our experiment is 50 epochs) using a learning rate of 0.0002
- **Phase 2:** Then, we freeze the batch normalization (the trainable parameters of batch normalization layer) in all layers of the encoder part of the network, and fine-tune using a learning rate of 0.00005 (we have 50 epochs in this phase as well). However, we keep batch normalization parameters enabled in the decoder part of the network.

The two-phase technique does not only avoid the incorrect mean and variance issues, but also helps us to achieve faster convergence. It takes about 10 days for training the model on COCO-Text dataset for both phases. Table 5.2 and diagram 5.7 summarize the training process including the numeric evaluation.

5.2.2 Details of Network Architecture

In this section we explore the internal layers of the U-Net netwrok architecture 4.2. As Table 5.1 shows, PConv is defined as a partial convolutional layer 3.1.2 with the specified filter size (FS) and stride, and number of filters(#Filters). PConv1-8 are in the encoder stage, whereas PConv9-16 are in the decoder stage. The batch normalization column (BN) indicates whether PConv is followed by a batch normalization layer (Y) or not (-). The nonlinearity column (NL) shows what kind of nonlinearity function is used whether ReLU or LeakyReLU (LReLU). Skip links are shown as Concat# in the module name column, where the skip links concatenate the previous nearest neighbor upsampled results with the corresponding mentioned PConv# results from the encoder part.

5.2.3 Quantitative Evaluation

It has been observed in the recent related works of image inpainting 2.1 that there is no perfect numerical metric to evaluate image reconstruction results due to the existence of many possible solutions. Nevertheless, we follow the previous image inpainting works [8][12] by reporting our evaluation metric in terms of Peak Signal to Noise Ratio (PSNR) [86]. However, there is another evaluation metric, which has been used in some recent works called Inception Score (IS) [87] introduced for evaluating Generative Adversarial Networks (GANs), but it is not a suitable metric for evaluating image inpainting methods because inpainting task mostly focuses on background

Module Name	FS	#Filters	Stride	BN	NL
PConv1	7×7	64	2	-	ReLU
PConv2	5×5	128	2	Y	ReLU
PConv3	5×5	256	2	Y	ReLU
PConv4	3×3	512	2	Y	ReLU
PConv5	3×3	512	2	Y	ReLU
PConv6	3×3	512	2	Y	ReLU
PConv7	3×3	512	2	Y	ReLU
PConv8	3×3	512	2	Y	ReLU
NearestUpSample1		512	2	-	
Concat1(w/PConv7)		$512 + 512$		-	
PConv9	3×3	512	1	Y	LReLU(0.2)
NearestUpSample2		512	2	-	
Concat2(w/PConv6)		$512 + 512$		-	
PConv10	3×3	512	1	Y	LReLU(0.2)
NearestUpSample3		512	2	-	
Concat3(w/PConv5)		$512 + 512$		-	
PConv11	3×3	512	1	Y	LReLU(0.2)
NearestUpSample4		512	2	-	
Concat4(w/PConv4)		$512 + 512$		-	
PConv12	3×3	512	1	Y	LReLU(0.2)
NearestUpSample5		512	2	-	
Concat5(w/PConv3)		512+256		-	
PConv13	3×3	512	1	Y	LReLU(0.2)
NearestUpSample6		256	2	-	
Concat6(w/PConv2)		$512 + 128$		-	
PConv14	3×3	128	1	Y	LReLU(0.2)
NearestUpSample7		128	2	-	
Concat7(w/PConv1)		$512 + 64$		-	
PConv15	3×3	64	1	Y	LReLU(0.2)
NearestUpSample8		64	2	-	-
Concat8(w/Input)		$64 + 3$		-	-
PConv16	3×3	3	1	-	-

Table 5.1: Network configuration summary

filling (e.g. object removal case), and not on its ability to generate a variety classes of objects as in GANs.

Peak Signal to Noise Ratio (PSNR)

It is the simplest and most widely used full-reference quality metric for evaluating inpainted images defined by mean squared error (MSE) [88], and

Model	Training Phase	PSNR Train	PSNR Val	Loss Train	Loss Val	t/GPU
Image Inpainting	1	29.46	31.08	0.41	0.64	2h 20m
	2	29.96	29.75	0.35	0.61	

Table 5.2: Training procedure of image inpainting model. PSNR accuracies and losses for training set (Train) and validation set (Val) on distorted (masked) COCO-Text images 5.2.4 using irregular mask dataset 5.1.2. 't/GPU' indicates the time cost per epoch at training stage on GPU.

computed by averaging the squared intensity differences of distorted and original image pixels, it is usually expressed in terms of the logarithmic decibel scale. MSE and PSNR are defined as [89]:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{5.1}$$

$$PSNR = 10.\log_{10}(\frac{MAX_I^2}{MSE}) \tag{5.2}$$

$$= 20.\log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{5.3}$$

$$= 20.\log_{10}(MAX_I) - 10.\log_{10}(MSE) \tag{5.4}$$

The MAX_I is the maximum possible pixel value of the image, when the pixels are represented using 8 bits per sample, then MAX_I is 255, since is calculated as $2^B - 1$, B is bits per sample. However, in our experiment the input image is scaled within the range -2.11 to 2.64 because we use VGG16 model [90][91] with weights pre-trained on ImageNet [55], and thus we use the difference between these two values (that is 4.75) as MAX_I, the higher PSNR value is the better the performance of the model.

5.2.4 Results of Image Reconstruction Model

Some representative results of image reconstruction model are presented in Figure 5.6, including ground-truth images (GT), masked input images (Input) with various dilation of holes, and inpainted (reconstructed) images (PConv).

5.3 Scene Text Recognition Model

5.3.1 Training Procedure

We train and evaluate our Scene Text Recognition Model on original COCO-Text dataset 5.1.1, reconstructed COCO-Text images 5.2.4 by the Image Re-

Figure 5.6: Test result on COCO-Text dataset

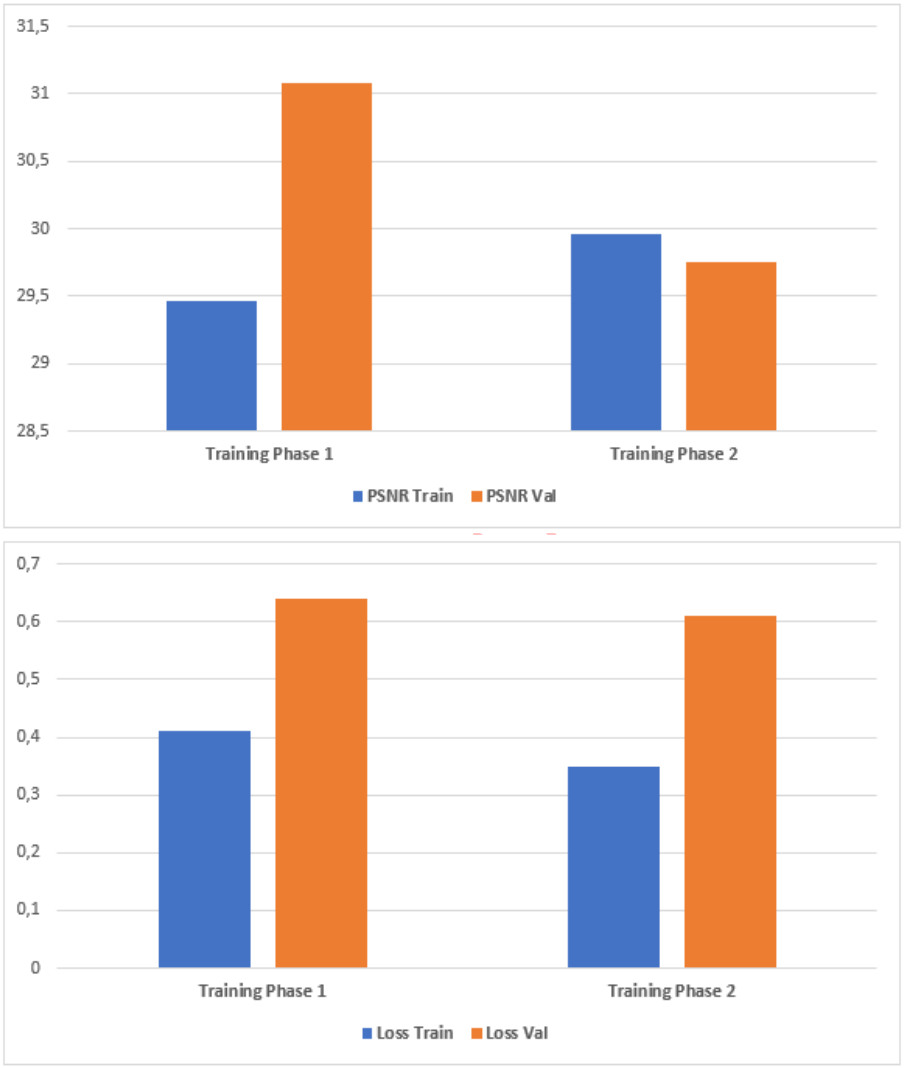


Figure 5.7: Plot of Image Inapinting loss and accuracy (PSNR) on Train and Validation set grouped by training phases.

construction Model, and distorted (masked) COCO-Text images 5.11 without any fine-tuning. In both training and inference stages, we set heights of the word images to 32, while widths are proportionally scaled with heights. We perform recognition on word images that contain alphanumeric characters, punctuation special symbols, and at least three characters. The output alphabet of target text consists of 98 classes, including 26 lowercase letters, 26 uppercase letters, 10 digital numbers, 32 ASCII punctuation symbols, space, furthermore end-of-sequence/padding/unknown tokens. In the training stage, samples are batched by 32 length of input encoding sequences.

Each training batch contains 2048 maximum number of image tokens and target tokens. We use Adam Optimizer [76] with following values of hyper-parameters $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-9}$, and vary the learning rate over the course of training according to the formula [14]:

$$lrate = d_{model}^{-0.5}.min(n^{-0.5}, n.warmup\_n^{-1.5}) \tag{5.5}$$

Where n represents the current training step number, and $warmup\_n = 16000$ controls over the learning rate, firstly increase then decrease. In order to prevent over-fitting, we set residual dropout to 0.1, where it is applied to the output of each sub-layer in the encoder 3.3.1 and decoder 3.3.4 before adding the residual information. We train our model on one machine with an NVIDIA Geforce GTX 1080 Ti (11 GB) for a total of 175000 steps which is about 128 epochs. Each epoch takes about 19 minutes, and in total the whole training phase takes about one day and 12 hours. The checkpoints are written at 1000 steps intervals in Tensorflow framework [92]. Table 5.3 and diagram 5.10 summarize the training process including the numeric evaluation.

5.3.2 Exploration of model architectures

In this section we investigate the contributions made by the key components of the proposed method, namely: the architectures of the encoder 3.3.1 and decoder 3.3.4 the position-wise feed-forward network 3.3.3 and modality-transform block 4.2.1.

Core module architectures

We explore the super parameters of the core modules in our model, including the encoder block number N_e, the decoder block number N_d and feed-forward inner dimension d_{ff}. The model dimension $d_{model} = 512$ and head number 3.3.2 h = 8 are all kept unchanged, and the modality-transform block only contains three convolutional layers all the time. We regard 6encoder6decoder model as our **base model**, we keep the total block number of the encoder and the decoder identical where $N_e = 6$, $N_d = 6$ and $d_{ff} = 2048$.

Modality-transform block architecture

Because the encoder itself also has strong feature extraction ability, we prefer to apply in the modality-transform block of our base model three convolutional layers with stride 2, number of filters 64, and kernel size 7 as Figure 5.8 shows. Each convolutional layer is followed by a batch normalization layer with following values of parameters (momentum= 0.997 and epsilon= 1e-5), and a non-linearity function (Relu) 3.1.3, where only the first convolutional layer is followed by a 2×2 max-pooling layer 3.1.4. We find that

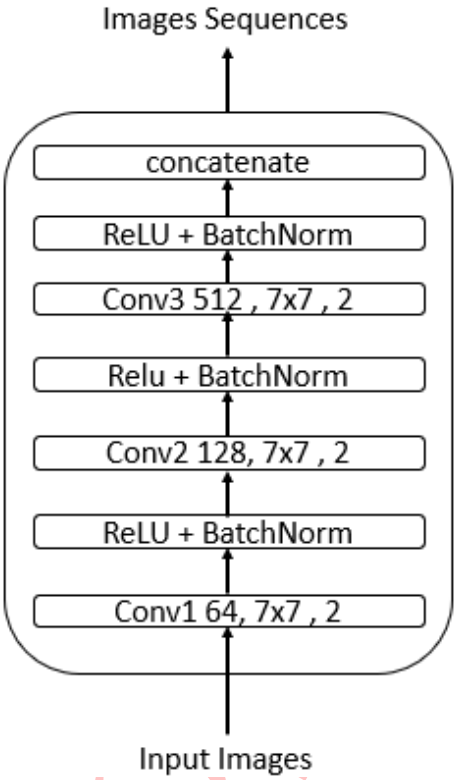


Figure 5.8: The proposed modality-transform block.

the batch normalization [93] technique is extremely useful for training network of such depth, where with the batch normalization layers the training process is greatly accelerated.

5.3.3 Results of Scene Text Recognition Model

Our final scene text recognizer is constructed by setting the encoder number $N_e = 6$, the decoder number $N_d = 6$, the feed-forward dimension $d_{ff} = 2048$ and three convolutional layers in the modality-transform block 5.8. Some representative results are presented in Figure 5.9. As can be seen, our model demonstrates excellent capability on recognizing extremely challenging text images, including but not limited to low resolution, low visual quality, complex geometric deformations and cluttered background, various font size, style and directions, moreover punctuations and special symbols, some of which are even hard to human to recognize.

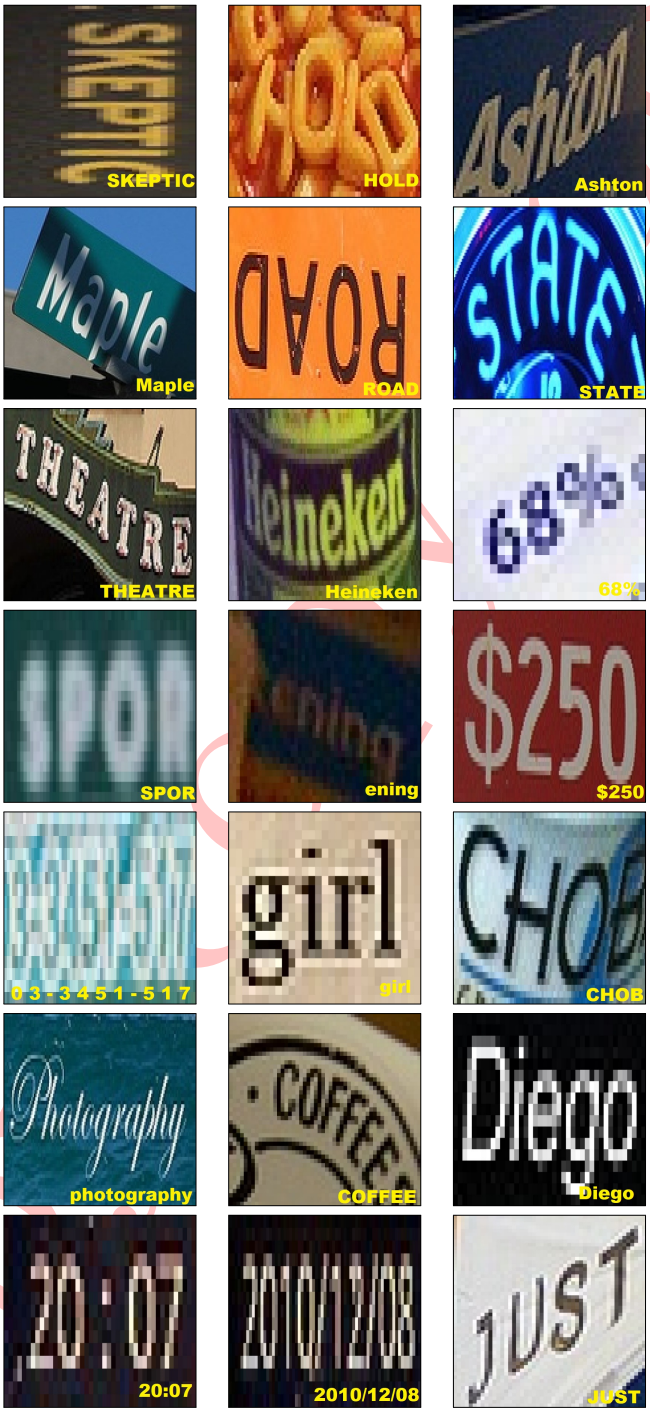


Figure 5.9: Test result on COCO-Text dataset. Correct recognitions with their text labels in yellow.

Model	Dataset	Accuracy Train	Accuracy Val	Loss Train	Loss Val	t/GPU
NRTR	Original	0.99	0.74	0.02	0.79	19m
	Inpainted	0.99	0.72	0.01	0.84	19m
	Distorted	0.99	0.37	0.04	1.84	19m

Table 5.3: Training procedure of NRTR. Recognition accuracies and losses(%) for training set (Train) and validation set (Val) on three versions of COCO-Text dataset. "Original" means original COCO-Text dataset 5.1.1, "Inpainted" reconstructed of masked COCO-Text images 5.2.4 by the Image Reconstruction Model, and "Distorted" masked COCO-Text images 5.4 5.11 that will be fed into image reconstruction model.'t/GPU' (minute) indicates the time cost per epoch at training stage on GPU.

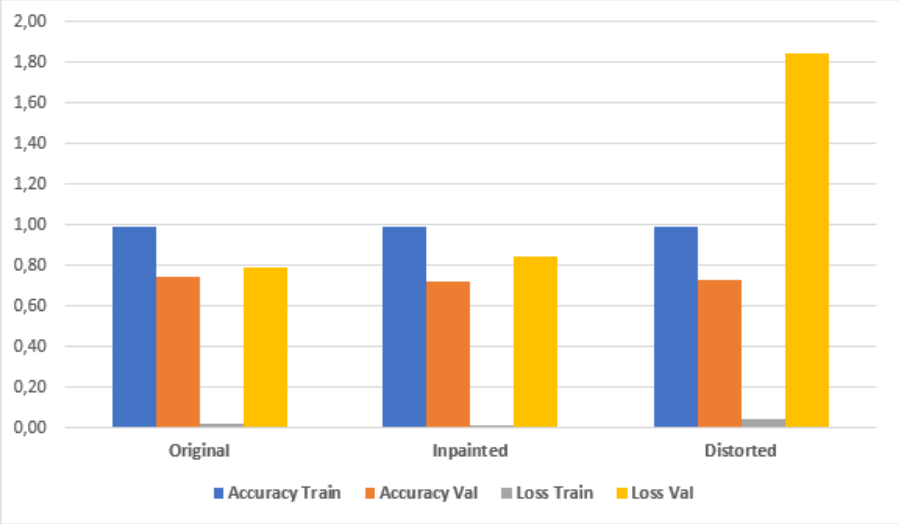
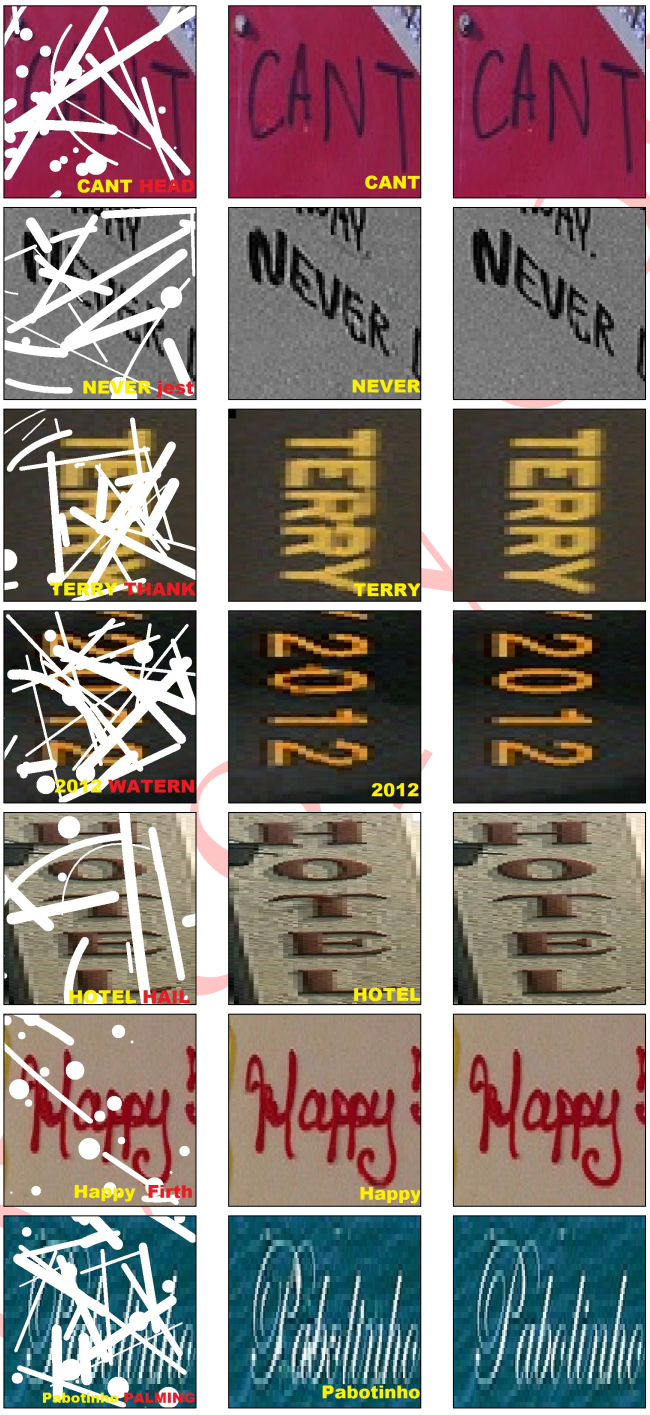


Figure 5.10: Plot of NRTR loss and accuracy on Train and Validation set grouped by Original, Inpainted, and Distorted Datasets.

5.4 Results of both models

Some representative results of both image reconstruction and text recognition model together are represented in Figure 5.11, including ground-truth images (GT) of COCO-Text dataset in the right column, in the left column there are the input masked images (Input) that will be fed into image reconstruction model, and in the middle column there are the predicted reconstructed images (PConv) by image reconstruction model. The experiment on this bunch of images is carried out in two phases: firstly, we feed the masked scene text images (the left column of Figure 5.11) into the our text recognition model to check up if the model is even able to recognize some

characters, as we can see the model is not able to recognize the text (text labels in yellow and incorrect outputs in red). However, we move on to the second phase of experiment and feed the input masked images into image inpainting model to reconstruct these images, and as the middle column of Figure 5.11 shows, the model can robustly handle the holes in these images and reconstruct them effectively. Afterwards, we feed the reconstructed images of middle column again into our scene text recognizer to see if it now can recognize the word images, as we can see in the middle column of Figure 5.11 ,the correct outputs in yellow are perfectly compatible with their text labels (word images).



(Input) (PConv) (GT)

Figure 5.11: Examples of results for both proposed models.
(left) Distorted (masked) input COCO-Text images with their text labels in yellow and incorrect outputs of NRTR model in red. (middle) Reconstructed images with their correct recognized text outputs in yellow. (right) Original images of COCO-Text dataset.

Chapter 6

Conclusion and Outlook

In this thesis we pointed out two problems laying in the field of computer vision, and explored a way to solve these challenging problems of image reconstruction and scene text recognition. For the task of image reconstruction, we proposed the use of a partial convolution layer with an automatic mask updating mechanism which it could achieve fantastic image inpainting results and robustly handle holes of any shape, size or dilation as seen in Figure 5.6. Further, the performance does not deteriorate catastrophically as holes increase in size as seen in the left column of Figure 5.11 for both word images "2012" and "TERRY".

For scene text recognition task, we present a novel no-recurrence seq2seq model (NRTR) that addresses the shortcomings happend in most current RNNs-based and CNNs-based scene text recognition methods by mitigating computation parallelization and complexity. NRTR follows the encoder-decoder framework but leverages self-attention inspired by the Transformer [15] as its fundamental component, combined with the proposed modality-transform block 5.3.2. The proposed architecture of NRTR could efficiently capture both contextual information and long-term dependencies. Extensive experiments over COCO-Text dataset show that the proposed NRTR can achieve both high accuracy and training speed. However, one limitation of NRTR model is that it fails to recognize texts that are occluded by other objects, e.g., mask (holes) in examples of Figure 5.11, which it was our motivation to propose image reconstruction model as a pre-step to reconstruct word images before being flowed into NRTR model.

For future research, we think that it might be helpful to test the effectiveness of the proposed models on more synthetic datasets and more public benchmarks such as IIT5K [59], Street View Text (SVT) [41], ICDAR (IC03) [60] and (IC13) [61]. Furthermore, the used methods in reconstruction of occluded text for OCR might be a starting point for extending the proposed ideas to method for automatic extraction of handwritten whiteboard content from lecture videos recorded with still cameras and generate

spatio-temporal index for the handwritten content in the video to increase the ability to find such videos online based on their content which it would also reduce the amount of manual effort required to make indexing and retrieval of such videos. For this propose, we need firstly to sample frames from the lecture video at sample rate FPS (frames per second) and each selected frame is then preprocessed for background estimation and removal, and then binarized using some methods such as a high-recall, low-precision machine learning binarizer. The final output of this step is a binary image where most handwritten components are kept and most noisy components will be removed. Then, we segment the whiteboard to estimate the whiteboard region and remove from binary frames any components that are not fully contained in the region, where in this step the location of handwritten pixels, and general background and foreground pixels like the instructor will be estimated. Afterward, we reconstruct the binary segmented frames by filling in the gaps where the handwritten were not visible due to occlusion by foreground elements like the instructor. Finally, we extract the handwritten content by dividing the reconstructed frame images using a grid and then classify each cell as handwritten content or noise, then group together handwritten content cells into blocks or text lines that can be used for generating spatio-temporal index using OCR engine.

Acknowledgements

I would like to thank Professor Ralph Ewerth for giving me the chance to complete my master's thesis in Visual Analytics Institute, my supervisor Mr. Matthias Springstein for the numerous fruitfull discussions we have had, for constructive comments, quick feedback, and technical support. Finally, I wish to thank my parents for their support and encouragement throughout my studies.

References

Literature

[1] Kenny Davila, Richard Zanibbi. *: Whiteboard Video Summarization via Spatio-Temporal Conflict Minimization.* 29 January 2018 (cit. on pp. 1, 2).

[2] Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B. *Patchmatch: A randomized correspondence algorithm for structural image editing, ACM Transactions on Graphics-TOG 28(3), 24.* 2009 (cit. on pp. 2, 7).

[3] Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G., Verdera, J. : *Filling-in by joint interpolation of vector fields and gray levels.* IEEE transactions on image processing 10(8), 1200-1211, 2001 (cit. on pp. 2, 6).

[4] Bertalmio, M., Sapiro, G., Caselles, V., Ballester, C.: *Image inpainting.* In: Pro-ceedings of the 27th annual conference on Computer graphics and interactive techniques. pp. 417-424. ACM Press/Addison-Wesley Publishing Co., 2000 (cit. on pp. 2, 6, 7).

[5] Efros, A.A., Freeman, W.T.: *Image quilting for texture synthesis and transfer.* In: In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 341-346., 2001 (cit. on pp. 2, 7).

[6] A. Criminisi, P. Perez and K. Toyama. *: Region filling and object removal by exemplar-based image inpainting.* Microsoft Research, Cambridge (UK) and Redmond (US), 2004 (cit. on p. 3).

[7] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A. *Context en-coders: Feature learning by inpainting.* In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2536-2544, 2016 (cit. on pp. 3, 8, 9).

[8] Yang, C., Lu, X., Lin, Z., Shechtman, E., Wang, O., Li, H. *High-resolution image inpainting using multi-scale neural patch synthesis.* In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)., 2017 (cit. on pp. 3, 9, 45).

[9] Iizuka, S., Simo-Serra, E., Ishikawa, H. *:Globally and locally consistent image completion.* In: ACM Transactions on Graphics (TOG) 36(4), 107, 2017 (cit. on pp. 3, 8).

[10] Telea, A. : *An image inpainting technique based on the fast-marching method. Journal of graphics tools 9(1), 23-34.* In: ACM Transactions on Graphics (TOG) 36(4), 107, 2004 (cit. on pp. 3, 7).

[11] $P_erez, P., Gangnet, M., Blake, A.$:. *Poisson image editing.* In: ACM Transactions on graphics (TOG) 22(3), 313-318, 2003 (cit. on p. 3).

[12] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S.: *Generative image inpainting with contextual attention.* arXiv preprint arXiv:1801.07892, 2018 (cit. on pp. 3, 8, 9, 45).

[13] Ronneberger, O., Fischer, P., Brox, T.: *U-net: Convolutional networks for biomedical image segmentation.* In: International Conference on Medical image computing and computer-assisted intervention. pp. 234-241. Springer, 2015 (cit. on pp. 3, 5, 30, 31).

[14] Fenfen Sheng, Zhineng Chen Bo Xu. *NRTR: A No-Recurrence Sequence-to-Sequence Model For Scene Text Recognition.* arXiv:1806.00926, 2018 (cit. on pp. 4, 5, 14, 28, 35, 37, 50).

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need.* arXiv:1706.03762., 2017 (cit. on pp. 4, 5, 14, 22, 24, 26, 35, 56).

[16] Alex Graves. *Sequence Transduction with Recurrent Neural Networks.* International Conference of Machine Learning (ICML) 2012 Workshop on Representation Learning., 14 Nov 2012 (cit. on pp. 4, 22).

[17] *How to Build OpenAI's GPT-2: "The AI That's Too Dangerous to Release".* https://blog.floydhub.com/gpt2/. Accessed: 2019-10-25 (cit. on p. 4).

[18] *Unsupervised Machine Learning.* https://towardsdatascience.com/. Accessed: 2019-10-25 (cit. on p. 4).

[19] *OpenAI.* https://openai.com/. Accessed: 2019-10-25 (cit. on p. 4).

[20] Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers forLanguage Understanding.* Google AI Language, arXiv:1810.04805v2, 24 May 2019 (cit. on p. 4).

[21] Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: *Texture optimization for example-based synthesis.* In: ACM Transactions on Graphics (ToG). vol. 24, pp. 795-802., 2005 (cit. on p. 7).

[22] Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: *Summarizing visual data using bidirectional similarity.* In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. pp. 1-8., 2008 (cit. on p. 7).

[23] R. Köhler, C. Schuler, B. Sch olkopf, and S. Harmeling.: *Mask-specific inpainting with deep neural networks.* In: German Conference on Pattern Recognition, pages 523–534.Springer, 2014 (cit. on p. 8).

[24] L. Xu, J. S. Ren, C. Liu, and J. Jia.: *Deep convolutional neural network for image deconvolution.* In: Advances in Neural Information Processing Systems, pages 1790–1798, 2014 (cit. on p. 8).

[25] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: *ImageNet Large Scale Visual Recognition Challenge.* In: International Journal of Computer Vision (IJCV) 115(3), 211-252, 2015 (cit. on pp. 8, 33).

[26] Li, Y., Liu, S., Yang, J., Yang, M.H.: *Generative face completion.* In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). vol. 1, p. 3, 2017 (cit. on p. 8).

[27] Song, Y., Yang, C., Lin, Z., Li, H., Huang, Q., Kuo, C.C.J.: *Image inpainting using multi-scale feature image translation.* .arXiv preprint arXiv:1711.08590, 2017 (cit. on p. 9).

[28] R. Yeh, C. Chen, T. Y. Lim, M. Hasegawa-Johnson, and M. N. Do.: *Semantic Image Inpainting with Deep Generative Models.* .arXiv preprint arXiv:1607.07539, 2016 (cit. on p. 9).

[29] Ulyanov, D., Vedaldi, A., Lempitsky, V.: *Deep image prior.* .arXiv preprint arXiv:1711.1092, 2017 (cit. on p. 9).

[30] Harley, A.W., Derpanis, K.G., Kokkinos, I.: *Segmentation-aware convolutional net- works using local attention masks.* In: IEEE International Conference on Computer Vision (ICCV)., 2017 (cit. on p. 9).

[31] van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al.: *Conditional image generation with pixelcnn decoders.* In: Advances in Neural Information Processing Systems. pp. 4790-4798, 2016 (cit. on p. 9).

[32] Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., Geiger, A.: *Sparsity invariant cnns.* arXiv preprint arXiv:1708.06500, 2017 (cit. on p. 10).

[33] Ren, J.S., Xu, L., Yan, Q., Sun, W.: *Shepard convolutional neural networks.* In: Advances in Neural Information Processing Systems. pp. 901-909., 2015 (cit. on p. 10).

[34] Kai Wang and Serge Belongie.: *Word spotting in the wild.* In European Conference on Computer Vision, pages 591–604.Springer, 2010 (cit. on p. 10).

[35] Kai Wang, Boris Babenko, and Serge Belongie.: *End-to-end scene text recognition.* In Computer Vision (ICCV), 2011 IEEE International Conference on, pages 1457–1464., 2011 (cit. on p. 10).

[36] Lukas Neumann and Ji Matas. *Real-time scene text localization and recognition.* In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pages 3538–3545., 2012 (cit. on pp. 10, 11).

[37] Cong Yao, Xiang Bai, Baoguang Shi, and Wenyu Liu. Strokelets: *A learned multi-scale representation for scene text recognition.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4042–4049, 2014 (cit. on p. 10).

[38] A. Bosch, A. Zisserman, and X. Munoz.: *Image classification using random forests and ferns.* In Computer Vision (ICCV), 2007 (cit. on p. 10).

[39] T. de Campos, B. Babu, and M. Varma.: *Character recognition in natural images.* In VISAPP, 2009. URL: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ (cit. on p. 11).

[40] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, andR. Young. *ICDAR 2003 robust reading competitions.IC-DAR.* 2003 (cit. on p. 11).

[41] K. Wang and S. Belongie. *Word spotting in the wild.* In ECCV, 2010 (cit. on pp. 11, 14, 56).

[42] Jon Almazan, Albert Gordo, Alicia Forn´es, and Ernest Valveny.: *Word spotting and recognition with embedded attributes.* IEEE transactions on pattern analysis and machine intelligence, 36(12):2552–2566, 2014 (cit. on p. 11).

[43] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Reading text in the wild with convolutional neural networks.* International Journal of Computer Vision, 116(1):1–20., 2016 (cit. on p. 11).

[44] Bolan Su and Shijian Lu. *Accurate scene text recognition based on recurrent neural network.* In Asian Conference on Computer Vision, pages 35–48. Springer, 2014 (cit. on p. 11).

[45] Graves, A., Schmidhuber., J.: *Framewise phoneme classification with bidirectional lstm and other neural network architectures.* In: Neural Networks. Volume 18. (2005) 602610, 2005 (cit. on p. 12).

[46] Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: *A novel connectionist system for unconstrained handwriting recognition.* In: Pattern Analysis and Machine Intelligence, IEEE Transactions on31(2009) 855–868, 2009 (cit. on p. 12).

[47] Baoguang Shi, Xiang Bai, and Cong Yao.: *An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition.* In: IEEE transactions on pattern analysis and machine intelligence, 39(11):2298–2304, 2017., 2017 (cit. on p. 12).

[48] Baoguang Shi, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. *Robust scene text recognition with automatic rectification.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4168–4176, 2016 (cit. on p. 12).

[49] F. L. Bookstein.Principal warps: *Thin-plate splines and the decomposition of deformations.* IEEE Trans. Pattern Anal. Mach.Intell., 11(6):567–585, 1989 (cit. on p. 12).

[50] Chen-Yu Lee and Simon Osindero. *Recursive recurrent nets with attention modeling for ocr in the wild.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2231–2239, 2016 (cit. on p. 12).

[51] Zhanzhan Cheng, Fan Bai, Yunlu Xu, Gang Zheng, Shiliang Pu, and Shuigeng Zhou. *Focusing attention: Towards accurate text recognition in natural images.* In 2017 IEEE International Conference on Computer Vision (ICCV), pages 5086–5094., 2017 (cit. on p. 13).

[52] Yunze Gao, Yingying Chen, Jinqiao Wang, and Hanqing Lu. *Reading scene text with attention convolutional sequence modeling.* arXiv preprint arXiv:1709.04303, 2017 (cit. on p. 13).

[53] Fei Yin, Yi-Chao Wu, Xu-Yao Zhang, and Cheng-Lin Liu. *Scene text recognition with sliding convolutional character models.* arXiv preprint arXiv:1709.01727, 2017 (cit. on p. 13).

[54] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. *Image inpainting for irregular holes using partial convolutions.* arXiv preprint arXiv:1804.07723, 2018 (cit. on pp. 13, 14, 35, 43, 44).

[55] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: *ImageNet Large Scale Visual Recognition Challenge.* International Journal of Computer Vision (IJCV) 115(3), 211-252, 2015. URL: https://doi.org/10.1007/s11263-015-0816-y (cit. on pp. 13, 47).

[56] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A.: *Places: A 10 million image database for scene recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017 (cit. on p. 13).

[57] Karras, T., Aila, T., Laine, S., Lehtinen, J.: *Progressive growing of gans for improved quality, stability, and variation.* arXiv preprint arXiv:1710.10196, 2017 (cit. on p. 13).

[58] Liu, Z., Luo, P., Wang, X., Tang, X.: *Deep learning face attributes in the wild. In:* Proceedings of International Conference on Computer Vision (ICCV), December-2015 (cit. on p. 13).

[59] Anand Mishra, Karteek Alahari, and CV Jawahar. *Scene text recognition using higher order language priors.* In BMVC 2012-23rd British Machine Vision Conference. BMVA, 2012 (cit. on pp. 14, 56).

[60] Simon M Lucas, Alex Panaretos, Luis Sosa, Anthony Tang, Shirley-Wong, Robert Young, Kazuki Ashida, Hiroki Nagai, Masayuki Okamoto, Hiroaki Yamamoto, et al. *Icdar 2003 robust reading competitions: entries, results, and future directions.* International Journal of Document Analysis and Recognition (IJDAR), 2003 (cit. on pp. 14, 56).

[61] Dimosthenis Karatzas, Faisal Shafait, Seiichi Uchida, Masakazu Iwamura, Lluis Gomez i Bigorda, Sergi Robles Mestre, Joan Mas, David Fernandez Mota, Jon Almazan Almazan, and Lluis Pere De Las Heras. *Icdar 2013 robust reading competition. In Document Analysis and Recognition (ICDAR).* 12th International Conference on, pages 1484–1493. IEEE., 2013 (cit. on pp. 14, 56).

[62] MIT 6S191. *Deep Learning for Computer Vision.* arXiv preprint arXiv:1709.01727, 2017. URL: http://introtodeeplearning.com/materials/2019_6S191_L3.pdf (cit. on pp. 16–19, 21, 22).

[63] Harley, A.W., Derpanis, K.G., Kokkinos, I.: *Segmentation-aware convolutional networks using local attention masks.* In: IEEE International Conference on Computer Vision (ICCV). vol. 2, p. 7, 2017 (cit. on p. 20).

[64] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, J urgen Schmidhube. *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies.* 2001 (cit. on p. 22).

[65] N. P. JouppiC. YoungN Patil. *In-Datacenter Performance Analysis of a Tensor Processing Unit.* In Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17., 2017 (cit. on p. 22).

[66] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, Koray Kavukcuoglu. *Neural Machine Translation in Linear Time.* 31 Oct 2016,last revised 15 Mar 2017 (this version, v2)) (cit. on p. 22).

[67] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, Yann N. Dauphin. *Convolutional Sequence to Sequence Learning.* (Submitted on 8 May 2017 (v1), last revised 25 Jul 2017 (this version, v3)) (cit. on p. 22).

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.: *Deep residual learning for image recognition.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016 (cit. on pp. 23, 29).

[69] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. *Layer normalization.* arXiv preprint arXiv:1607.06450, 2016 (cit. on pp. 23, 29, 31).

[70] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. *Convolutional sequence to sequence learning.* arXiv preprint arXiv:1705.03122v2, 2017 (cit. on p. 25).

[71] *U-Net Architecture using Partial Convolution.* https://github.com/zhixuhao/unet/blob/master/img/u-net-architecture.png. Accessed: 2019-06-25 (cit. on p. 32).

[72] *U-Net Architecture using Partial Convolution using Keras.* https://github.com/MathiasGruber/PConv-Keras/blob/master/data/images/architecture.png. Accessed: 2019-06-25 (cit. on p. 32).

[73] Gatys, L.A., Ecker, A.S., Bethge, M.: *A neural algorithm of artistic style.* arXiv preprint arXiv:1508.06576, 2015 (cit. on p. 33).

[74] Johnson, J., Alahi, A., Fei-Fei, L.: *Perceptual losses for real-time style transfer and super-resolution.* In: European Conference on Computer Vision. pp. 694-711. Springer, 2016 (cit. on p. 34).

[75] Jay Alammar. *The Illustrated Transformer.* http://jalammar.github.io/illustrated-transformer/. Accessed: 2019-10-14 (cit. on pp. 38, 39).

[76] Diederik Kingma and Jimmy Ba. Adam: *A method for stochastic optimization.* In ICLR, 2015 (cit. on pp. 38, 43, 50).

[77] *Cross-Entropy.* URL: https://colah.github.io/posts/2015-09-Visual-Information/ (cit. on p. 38).

[78] *kullback-leibler-divergence.* URL: https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained (cit. on p. 38).

[79] A. Veit, T. Matera, L. Neumann, J. Matas, S. Belongie. *COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images.* arXiv preprint arXiv:1601.07140, 2016 (cit. on p. 40).

[80] *COCO-Text: Dataset for Text Detection and Recognition.* https://vision.cornell.edu/se3/coco-text-2/. Accessed: 2019-04-14 (cit. on p. 40).

[81] *MS-COCO Dataset.* http://cocodataset.org/. Accessed: 2019-04-14 (cit. on p. 40).

[82] Sundaram, N., Brox, T., Keutzer, K.: *Dense point trajectories by gpu-accelerated large displacement optical ow.* 2010 (cit. on p. 43).

[83] *Image Inpainting for Irregular Holes Using Partial Convolutions.* http://masc.cs.gmu.edu/wiki/partialconv. Accessed: 2019-07-25 (cit. on pp. 43, 44).

[84] He, K., Zhang, X., Ren, S., Sun, J.: *Delving deep into rectiers: Surpassing human-level performance on imagenet classication.* In: Proceedings of the IEEE international conference on computer vision. pp. 1026-1034., 2015 (cit. on p. 43).

[85] K. Simonyan and A. Zisserman. *Very deep con-volutional networks for large-scale image recognition.* arXiv:1409.1556., 2014 (cit. on p. 44).

[86] Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P. *Image quality assessment: from error visibility to structural similarity.* IEEE transactions on image processing 13(4),600-612, 2004 (cit. on p. 45).

[87] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. *Improved techniques for training gans.* In: Advances in Neural Information Processing Systems. pp. 2234-2242., 2006 (cit. on p. 45).

[88] *Mean Squared Error.* https://en.wikipedia.org/wiki/Mean_squared _error. Accessed: 2019-07-25 (cit. on p. 46).

[89] *Peak signal-to-noise ratio.* https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. Accessed: 2019-07-25 (cit. on p. 47).

[90] *Models for image classification with weights trained on ImageNet.* https://keras.io/applications/#vgg16. Accessed: 2019-06-25 (cit. on p. 47).

[91] *VGG16 model ported from PyTorch to Keras.* https://github.com/ezavarygin/vgg16_pytorch2keras. Accessed: 2019-06-25 (cit. on p. 47).

[92] Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. *Tensorflow: Large-scale machine learning on heterogeneous distributed systems.* 2016 (cit. on p. 50).

[93] S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* 2015 (cit. on p. 51).