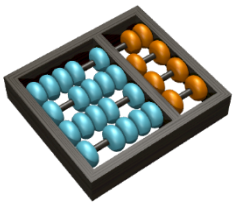
Universidade Estadual de Campinas
Instituto de Computação



Disciplina do 1º Semestre de 2024

IC - UNICAMP

Curso: Bacharelado em Ciência da Computação

MC920 - Introdução ao processamento de imagem digital

Trabalho 3

Alunos: Pedro Barros Bastos **RA**: 204481

Professor: Hélio Pedrini

Campinas – SP
2024

Introdução

A digitalização de documentos é amplamente utilizada para preservar e disseminar informações em formato digital. Um problema comum durante a digitalização é o desalinhamento dos documentos, resultando em um texto desalinhado horizontalmente que dificulta seu reconhecimento. Corrigir essa inclinação é crucial para o funcionamento adequado dos sistemas de reconhecimento óptico de caracteres.

Este relatório tem como objetivo explorar duas diferentes formas de se realizar o alinhamento de textos em imagens: a partir da projeção horizontal dos pixels pretos e a partir da transformada de Hough.

Será descrito como foi realizada a implementação do alinhamento pela projeção horizontal e seus resultados, bem como a implementação do alinhamento pelas funções prontas que permitem realizar a transformada de Hough. Depois será mostrada uma comparação dos resultados utilizando o Tesseract OCR.

Arquivos executáveis

- **alinhar.py** - script python que irá efetuar o alinhamento na imagem fornecida.
    - Parâmetros de execução:
        - ***--inputImg***: Path da imagem a ser alinhada
        - ***--outputImg***: Nome da imagem de saída alinhada
        - ***--mode***: Método a ser escolhido para realizar o alinhamento: PROJECTION ou HOUGH
        - ***--houghThreshold***: Se o modo de execução escolhido for HOUGH, o valor inteiro passado neste parâmetro será usado para extração das linhas de Hough. Valor default 400

Obs: Foi deixado um arquivo texto chamado **EXEMPLOS-EXECUCAO.txt** com exemplos de possíveis execuções dos scripts python.

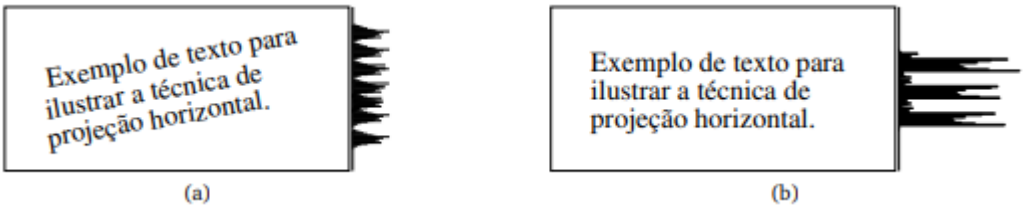
Pacotes utilizados

- **OpenCV** - v4.6.0
- **Numpy** - v1.26.4
- **argparse** - v1.4.0
- **Pytesseract** - v0.3.10

Obs: Foi utilizada a ferramenta **conda** para construção do ambiente de desenvolvimento para este trabalho, bem como utilização de SO **Ubuntu 22.04.4 LTS** inicializado em uma virtualbox.

Projeção Horizontal

A técnica da projeção horizontal consiste em variar o ângulo de rotação da imagem e projetar a quantidade de pixels pretos em cada linha da imagem. Com isso, o melhor alinhamento (melhor ângulo de rotação) é aquele que otimiza uma função objetivo calculada a partir de cada linha de pixels projetada. Neste desenvolvimento optou-se por utilizar como função objetivo a soma dos quadrados das diferenças dos valores em células adjacentes da projeção horizontal realizada.



Para efetuar o alinhamento por esta técnica, teve-se que levar em conta que o sentido da rotação para o alinhamento correto da imagem pode ser tanto horário quanto anti-horário (alinhamento com ângulo positivo VS alinhamento com ângulo negativo). Assim, foi decidido que o melhor alinhamento seria o maior valor da função objetivo obtido quando se rotaciona a imagem tanto no sentido horário quanto no sentido anti-horário. Para obter esse valor, os ranges de ângulos adotados para os testes foram de 0º a 90º (rotação em sentido anti-horário, seguindo convenção OpenCV) e de 0º a -90º (rotação em sentido horário, seguindo convenção OpenCV).

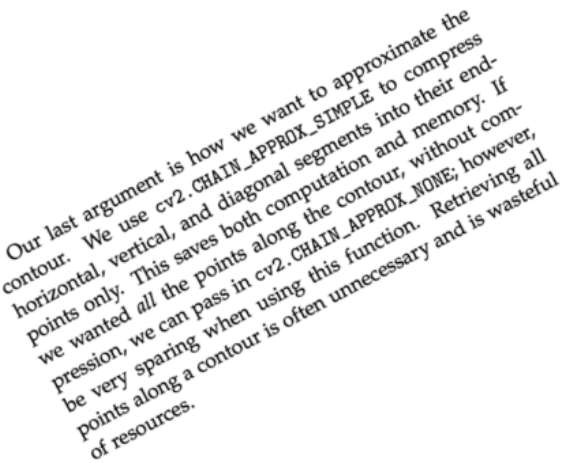


*Imagem **neg_28.png** original desalinhada*

Imagem rotacionada em -90º (totalmente rotacionada em sentido horário, segundo abordagem adotada)

Imagem rotacionada em 90º (totalmente rotacionada em sentido anti-horário, segundo abordagem adotada)

** Fazendo o plot das imagens rotacionadas, vê-se que neste caso partes do texto ficaram cortadas da imagem resultado. Não foi realizado nenhum tratamento deste caso durante os alinhamentos.*

*Imagem **pos_41.png** original desalinhada*

Imagem rotacionada em -90º (totalmente rotacionada em sentido horário, segundo abordagem adotada)

Imagem rotacionada em 90º (totalmente rotacionada em sentido anti-horário, segundo abordagem adotada)

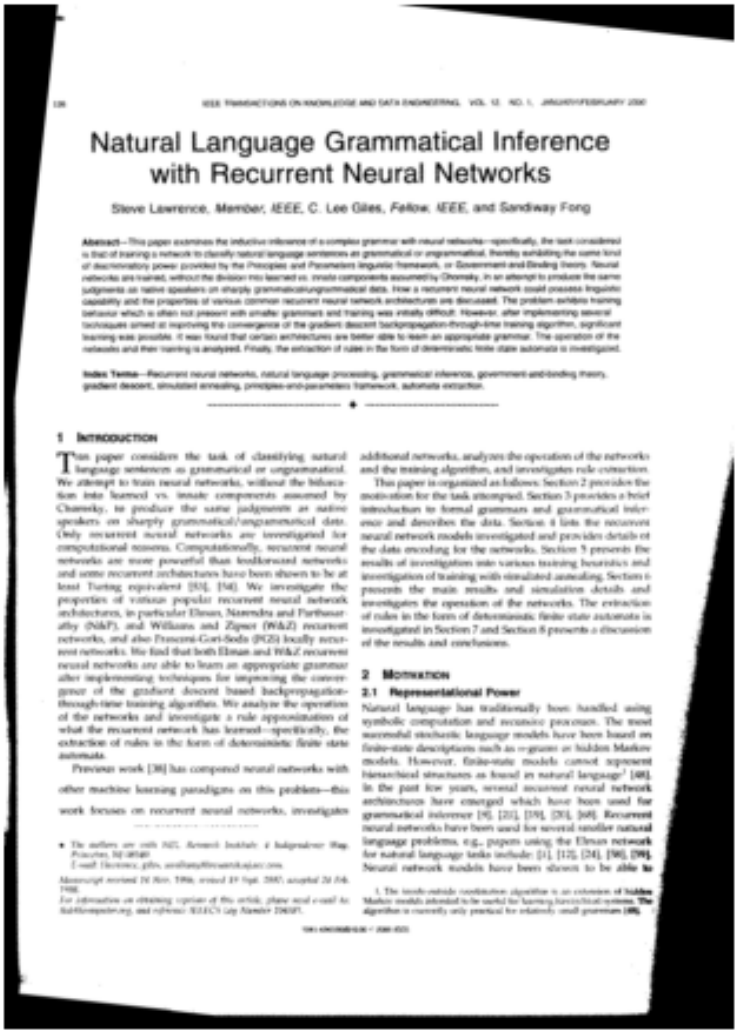
Nos exemplos acima, tanto para a **neg_28.png** quanto para a **pos_41.png**, temos que a imagem alinhada no eixo horizontal que se deseja está dentro dos limites de rotação adotados (-90º a 90º).

Para a implementação do alinhamento pelo projeção horizontal adotou-se uma variação do ângulo de 0.2º a cada iteração dentro do range de 0 a 90º e de 0 a -90º. Além disso, antes de se começar as iterações, binarizou-se a imagem de entrada para melhor identificação dos pixels pretos. Tal binarização se mostrou essencial para o alinhamento da imagem **partitura.png**, o que será melhor explicado logo adiante.

Como resultado das iterações de 0 a 90º e de 0 a -90º, obtém-se os dois maiores valores de função objetivo (respectivos aos dois ranges mencionados) e os ângulos de rotação associados a esses valores. O ângulo final é decidido a partir da comparação destes melhores valores, onde o maior valor corresponde à imagem que contém o alinhamento horizontal que é desejado. Como já mencionado, foi utilizado como função objetivo o cálculo da soma dos quadrados das diferenças dos valores em células adjacentes da projeção horizontal realizada. Tal cálculo foi realizado usando funções vetorizadas providas pelo numpy para conferir velocidade durante o processamento.

```
def funcao_objetivo(binaryImage):
    histograma = np.sum(binaryImage, axis=1)
    diff_squared = np.diff(histograma) ** 2
    sum_of_squares = np.sum(diff_squared)
    return sum_of_squares
```

Desta forma foi possível verificar com sucesso o alinhamento das imagens passadas como entrada. Seguem abaixo alguns resultados:



Natural Language Grammatical Inference with Recurrent Neural Networks

Steve Lawrence, *Member, IEEE*, C. Lee Giles, *Fellow, IEEE*, and Sandiway Fong

Abstract—This paper examines the inductive inference of a complex grammar with neural networks—specifically, the task considered is that of training a network to classify natural language sentences as grammatical or ungrammatical, thereby exhibiting the same kind of discriminatory power provided by the Principles and Parameters linguistic framework, or Government-and-Binding theory. Neural networks are trained, without the division into learned vs. innate components assumed by Chomsky, in an attempt to produce the same judgments as native speakers on sharply grammatical/ungrammatical data. How a recurrent neural network could possess linguistic capability and the properties of various common recurrent neural network architectures are discussed. The problem exhibits training behavior which is often not present with smaller grammars and training was initially difficult. However, after implementing several techniques aimed at improving the convergence of the gradient descent backpropagation-through-time training algorithm, significant learning was possible. It was found that certain architectures are better able to learn an appropriate grammar. The operation of the networks and their training is analyzed. Finally, the extraction of rules in the form of deterministic finite state automata is investigated.

Index Terms—Recurrent neural networks, natural language processing, grammatical inference, government-and-binding theory, gradient descent, simulated annealing, principles-and-parameters framework, automata extraction.

1 Introduction

This paper considers the task of classifying natural language sentences as grammatical or ungrammatical. We attempt to train neural networks, without the bifurcation into learned vs. innate components assumed by Chomsky, to produce the same judgments as native speakers on sharply grammatical/ungrammatical data. Only recurrent neural networks are investigated for computational reasons. Computationally, recurrent neural networks are more powerful than feedforward networks and some recurrent architectures have been shown to be at least Turing equivalent [51], [34]. We investigate the properties of various popular recurrent neural network architectures, in particular Elman, Narendra and Parthasarathy (N&P), and Williams and Zipser (W&Z) recurrent networks, and also Frasconi-Gori-Soda (FGS) locally recurrent networks. We find that both Elman and W&Z recurrent neural networks are able to learn an appropriate grammar after implementing techniques for improving the convergence of the gradient descent based backpropagation-through-time training algorithm. We analyze the operation of the networks and investigate a rule approximation of what the recurrent network has learned—specifically, the extraction of rules in the form of deterministic finite state automata.

Previous work [38] has compared neural networks with other machine learning paradigms on this problem—this work focuses on recurrent neural networks, investigates additional networks, analyzes the operation of the networks and the training algorithm, and investigates rule extraction.

This paper is organized as follows. Section 2 provides the motivation for the task attempted. Section 3 provides a brief introduction to formal grammars and grammatical inference and describes the data. Section 4 lists the recurrent neural network models investigated and provides details of the data encoding for the networks. Section 5 presents the results of investigations into various training heuristics and investigation of training with simulated annealing. Section 6 presents the main results and simulation details and investigates the operation of the networks. The extraction of rules in the form of deterministic finite state automata is investigated in Section 7 and Section 8 presents a discussion of the results and conclusions.

2 Motivation

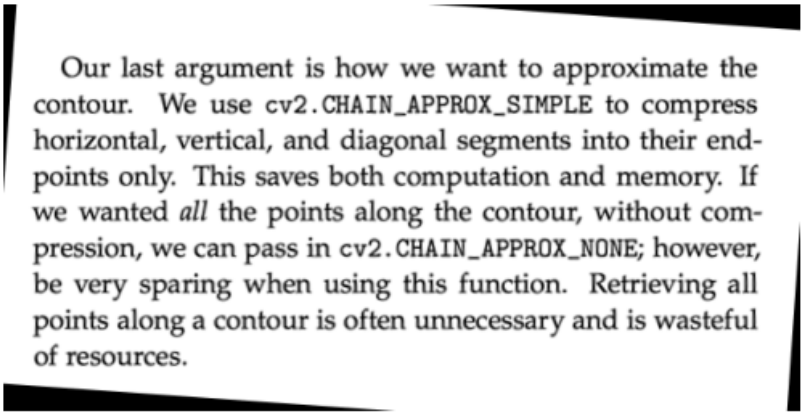
2.1 Representational Power

Natural language has traditionally been handled using symbolic computation and recursive processes. The most successful stochastic language models have been based on finite-state descriptions such as n-grams or hidden Markov models. However, finite-state models cannot represent hierarchical structures as found in natural language[1] [48]. In the past few years, several recurrent neural network architectures have emerged which have been used for grammatical inference [9], [21], [19], [20], [60]. Recurrent neural networks have been used for several notable natural language problems, e.g., papers using the Elman network for natural language tasks include: [11], [12], [24], [58], [59]. Neural network models have been shown to be able to

• The authors are with NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.
E-mail: {lawrence, giles, sandiway}@research.nj.nec.com.

Manuscript received 16 Nov. 1996; revised 19 Sept. 1997; accepted 24 Feb. 1998.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104207.

1. The inside-outside reestimation algorithm is an extension of hidden Markov models intended to be useful for learning hierarchical systems. The algorithm is currently only practical for relatively small grammars [48].



Mesmo sendo mencionado em sala de aula, rotacionar a imagem dentro do range de -90º a 90º foi suficiente para alinhar as imagens em qualquer rotação dada em qualquer um dos quatro quadrantes possíveis. Abaixo, a imagem *sample1.png* foi rotacionada dentro dos quatro quadrantes possíveis e depois cada rotação foi passada como entrada para o script de alinhamento, sendo as imagens da coluna à direita o resultado dos alinhamentos. A única diferença é que dependendo da rotação da imagem de entrada, o resultado do alinhamento pode acabar fornecendo uma imagem com o texto de cabeça para baixo, mas ainda assim alinhado horizontalmente.

Imagem sample1.png com rotação de 47º

Imagem sample1.png alinhada

Imagem sample1.png com rotação de 144º

Imagem sample1.png alinhada

Imagem sample1.png com rotação de 210º

Imagem sample1.png alinhada

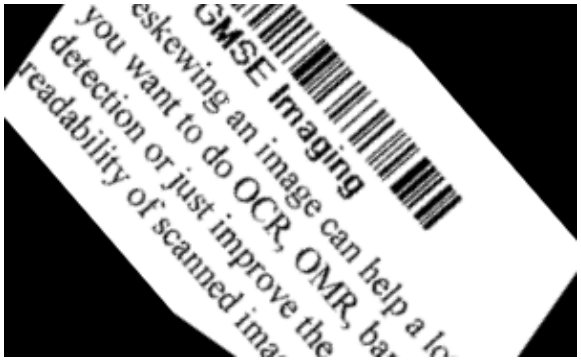


Imagem sample1.png com rotação de 311º

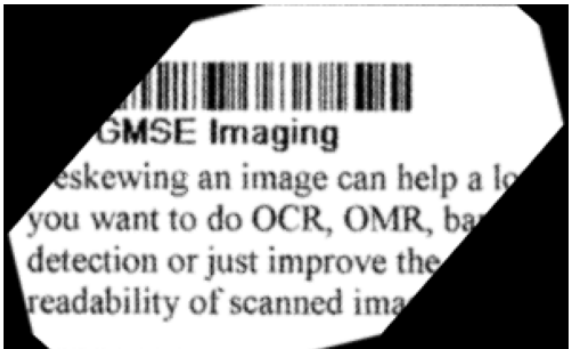
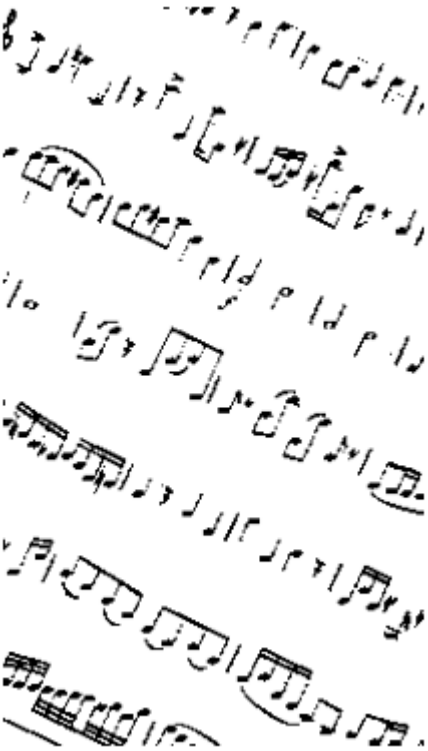
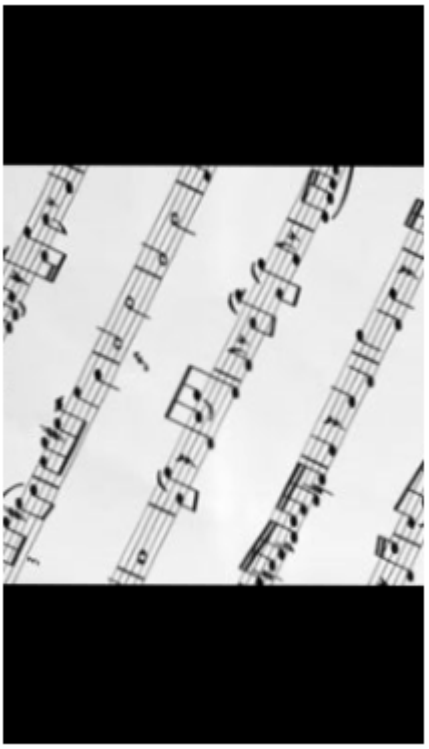
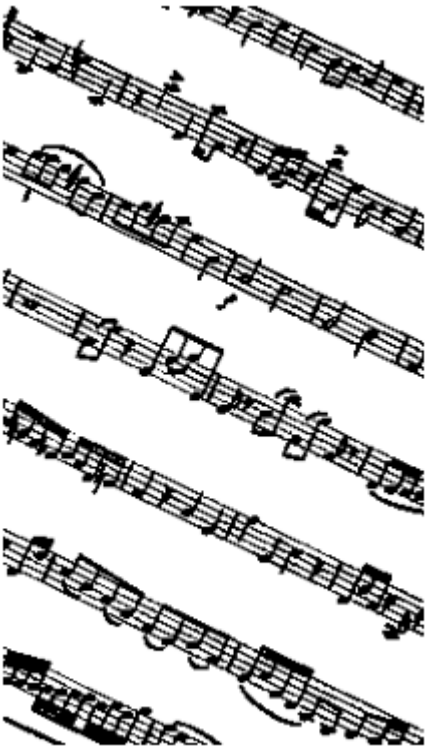
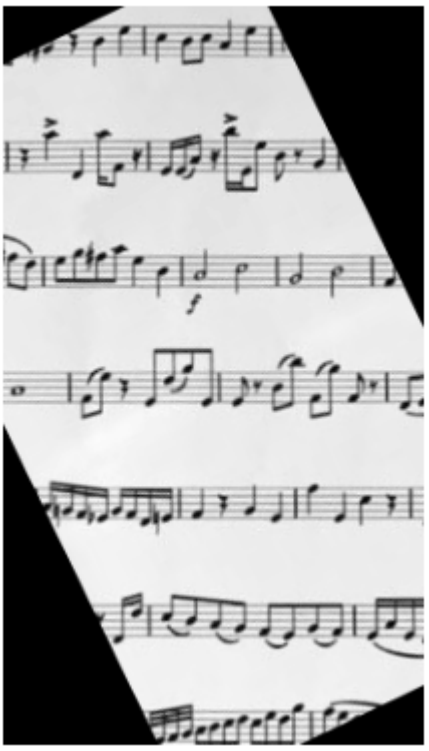


Imagem sample1.png alinhada

Um caso específico interessante em que foi necessário o pré-processamento binarizando a imagem a partir de um limiar testado empiricamente foi o caso da imagem **partitura.png**. Neste caso, foi preciso aumentar bem o limiar de binarização da imagem para que as linhas da partitura pudessem ser evidentes o suficiente. Assim a projeção horizontal pôde usá-las para que houvesse a correta identificação do melhor ângulo para alinhamento.

	
Partitura binarizada com limiar 127	*Resultado alinhamento com limiar 127*
	
Partitura binarizada com limiar 200	*Resultado alinhamento com limiar 200*

Transformada de Hough

Para o alinhamento a partir da transformada de Hough, realizou-se a binarização com limiar de 200. Aplicou-se a detecção de bordas de Sobel em cada eixo a partir da função *Sobel* do pacote OpenCV e tal resultado foi passado como entrada para a função *HoughLines* também do pacote OpenCV. O resultado retornado da execução de *HoughLines* provê uma lista de valores de *rho* e *theta*, com os quais as linhas de Hough podem ser desenhadas na imagem original para visualização. Finalmente, para determinação do ângulo de rotação foi considerado empiricamente como suficiente a extração da média dos valores de *theta* retornados pela *HoughLines*. Com o valor médio obtido a imagem é rotacionada para que nela o alinhamento seja realizado.

Apesar do uso da média para efetuar o alinhamento, os resultados variavam muito em diferentes imagens dependendo do limiar passado para *HoughLines*.

Imagem sample1.png utilizando limiar em HoughLines de 200

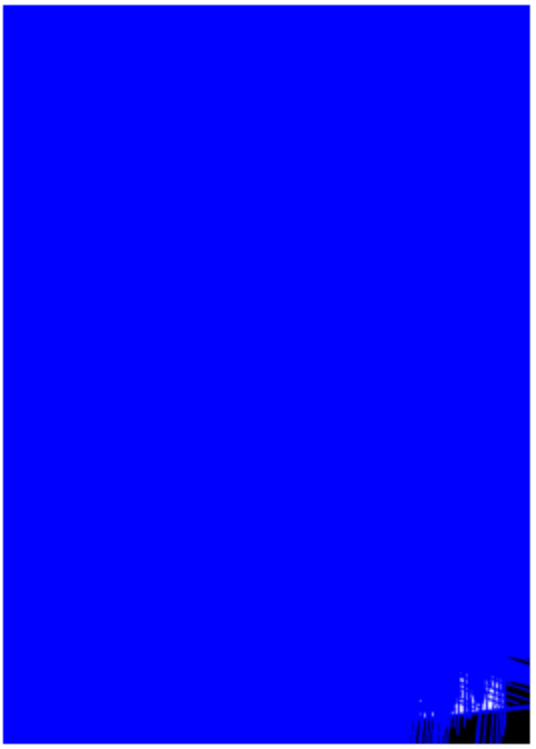


Imagem sample2.png utilizando mesmo limiar de 200 em HoughLines

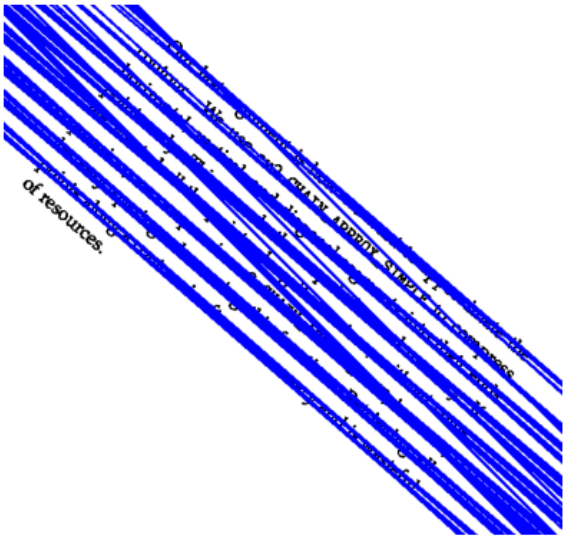
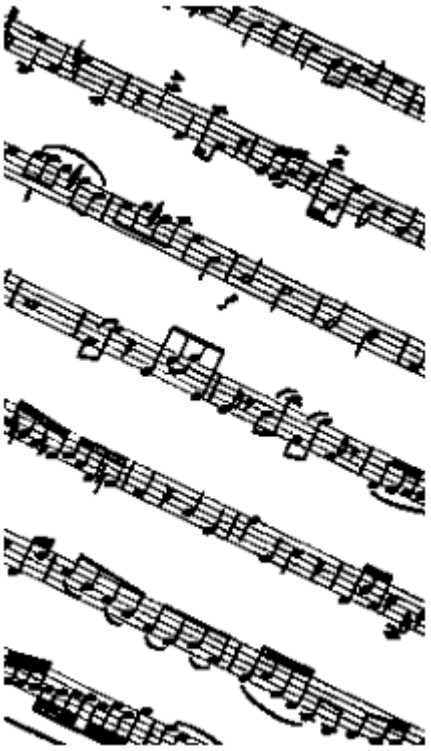
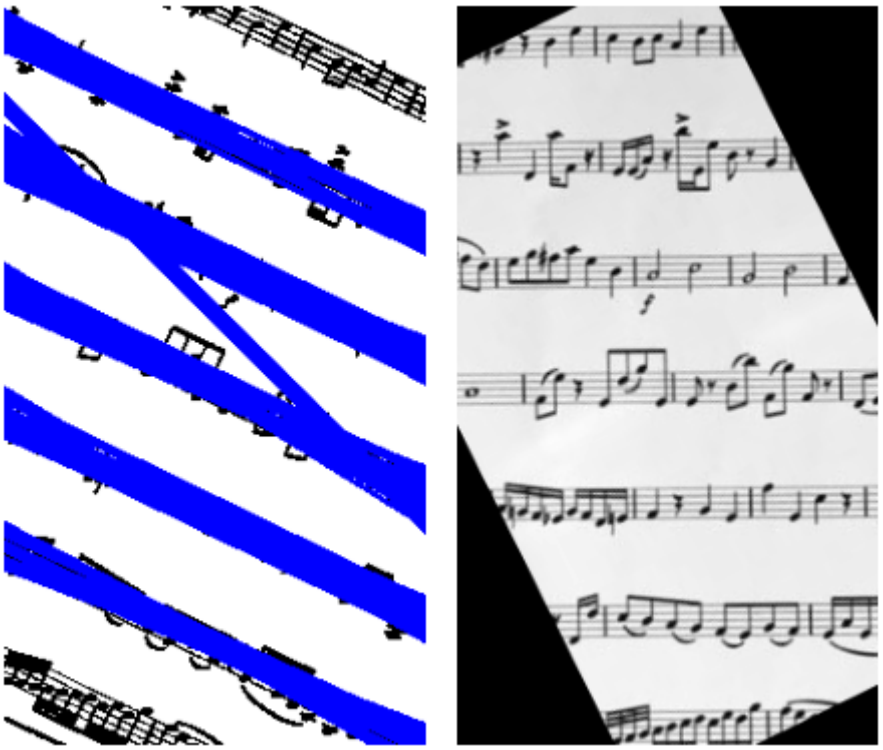


Imagem pos_41.png utilizando limiar em HoughLines de 400



Na imagem partitura.png, utilizando-se limiar 400, nenhuma linha é encontrada

Com o intuito de permitir um certo nível de calibração da execução da transformada de Hough, foi considerado criar como entrada do script de execução de alinhamento o parâmetro ***--houghThreshold***. Assim, para diferentes imagens foi possível ajustar a execução do alinhamento final.

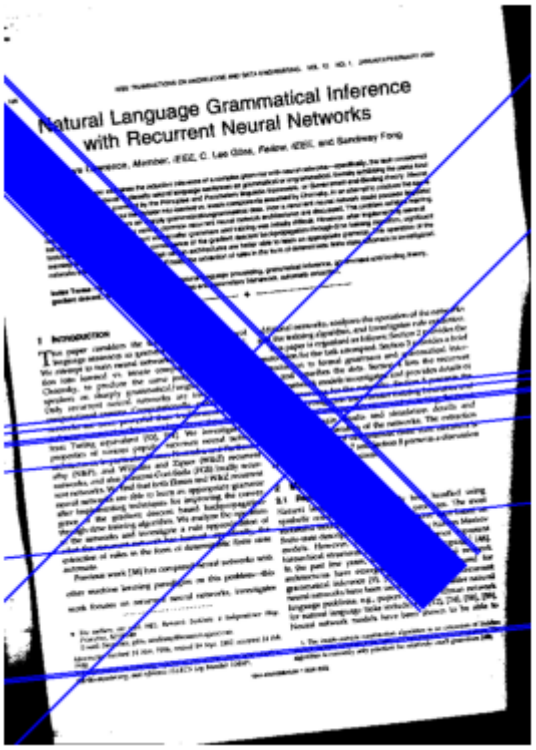


Linhas de Hough desenhadas a partir de limiar de 200 na função HoughLines; Alinhamento resultante após identificação do ângulo de rotação da imagem original dado pela média dos ângulos retornados pela função HoughLines

Comparação entre os métodos

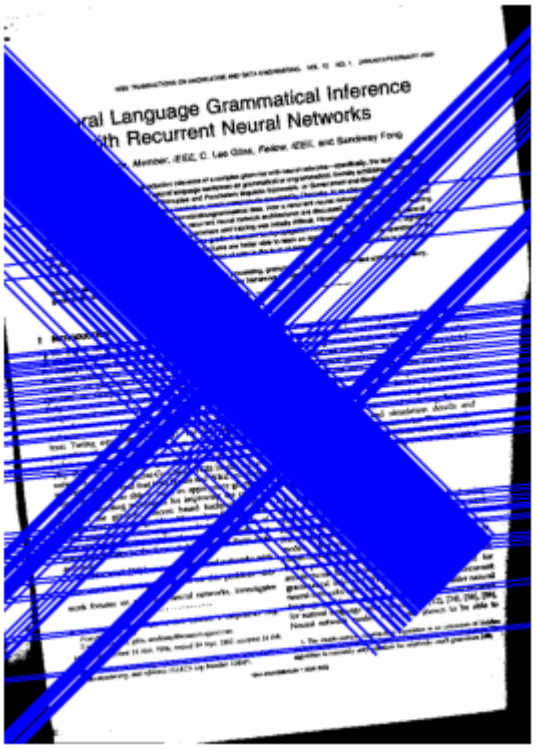
Em geral, os dois métodos apresentaram um bom resultado para o alinhamento. Uma diferença notável é o fato de a projeção horizontal, quando aplicada às imagens cujos nomes já continham suas respectivas rotações, conseguir achar o ângulo exato de rotação, enquanto o resultado da transformada de Hough (por ser obtido da média dos ângulos retornados) se limitava a chegar a um valor próximo da rotação original.

O único caso onde não foi possível por Hough realizar um alinhamento adequado foi na imagem *sample2.png*. Nela a maior parte das linhas obtidas não tinham nenhuma relação com o alinhamento do texto, mesmo se variando o ***--houghThreshold.***



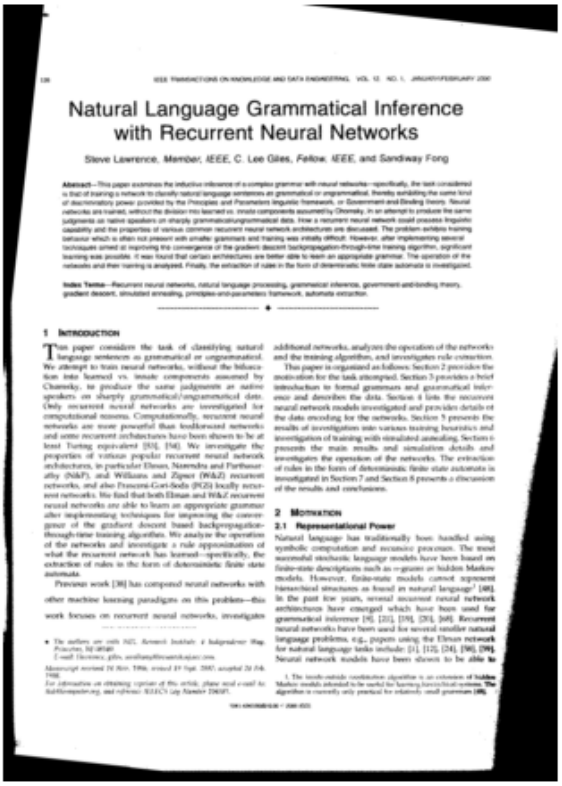
HoughLines com limiar 520

Resultado alinhamento



HoughLines com limiar 480

Resultado alinhamento



Natural Language Grammatical Inference with Recurrent Neural Networks

Steve Lawrence, *Member, IEEE*, C. Lee Giles, *Fellow, IEEE*, and Sandiway Fong

Alinhamento correto a partir da projeção horizontal da imagem sample2.png

Testes com Tesseract OCR

Com o uso do Tesseract OCR, obtido com o import do módulo *pytesseract* (instalação prévia via conda foi necessária), é possível através do método *image_to_string* passar como argumento uma imagem contendo algum texto e obter como saída a string que representa o texto na imagem. Durante a execução do script *alinhar.py* há dois momentos de uso dessa função: antes da realização do alinhamento pelo método escolhido e após o alinhamento. Com isso podemos perceber a diferença no reconhecimento do texto.

Foi claramente perceptível a diferença que o alinhamento faz no reconhecimento do texto. Em algumas imagens, tais como *pos_41.png* e *neg_28.png*, antes do alinhamento o tesseract não reconheceu nenhum texto. Após o alinhamento, todo o texto contido nas imagens mencionadas foi reconhecido.

```
(/home/pedrobastos/repositories/mc920/conda-env) pedrobastos@ubuntu:trabalho3$ python3 alinhar.py --inputImg images/pos_41.png --mode PROJECTI
ON --outputImg result.png --houghThreshold 200
--------------------------------
Detected text before processing:

--------------------------------
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
libGL error: MESA-LOADER: failed to open vmwgfx: /usr/lib/dri/vmwgfx_dri.so: cannot open shared object file: No such file or directory (search
 paths /usr/lib/x86_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)
libGL error: failed to load driver: vmwgfx
libGL error: MESA-LOADER: failed to open swrast: /usr/lib/dri/swrast_dri.so: cannot open shared object file: No such file or directory (search
 paths /usr/lib/x86_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)
libGL error: failed to load driver: swrast
Melhor angulo para rotação clockwise (negative degress) ->  -0.4
Melhor angulo para rotação non-clockwise (positive degress) ->  41.00000000000004
Angulo final é  41.00000000000004
--------------------------------
Detected text after alignment processing:
Our last argument is how we want to approximate the
contour. We use cv2.CHAIN_APPROX_SIMPLE to compress
horizontal, vertical, and diagonal segments into their end-
points only. This saves both computation and memory. If
we wanted all the points along the contour, without com-
pression, we can pass in cv2.CHAIN_APPROX_NONE; however,
be very sparing when using this function. Retrieving all
points along a contour is often unnecessary and is wasteful
of resources.
```

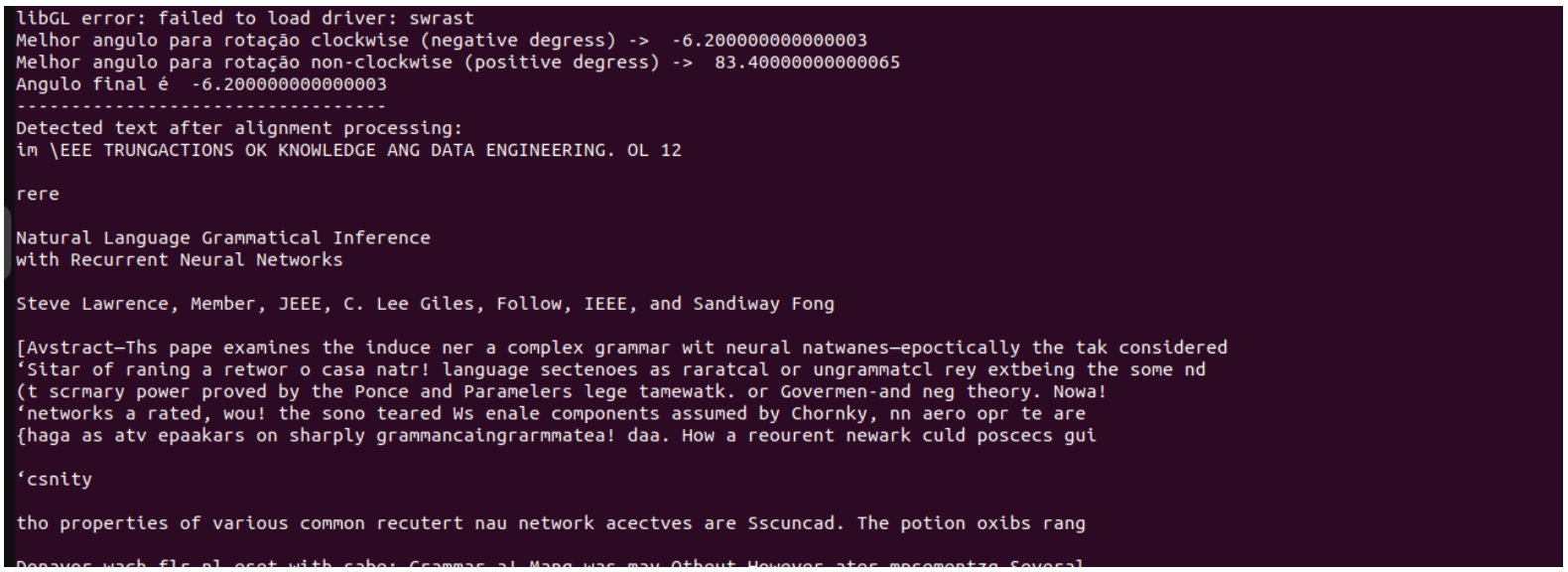
Já em algumas imagens como *sample2.png*, que possui uma grande quantidade de texto, no início alguns caracteres foram reconhecidos mas a maior parte do texto não pode ser corretamente identificado, tão pouco pôde ser classificado como um texto com algum sentido. Após o alinhamento pela projeção horizontal (único método que funciona para *sample2.png*), uma grande parte do texto pode ser reconhecido.

```
--------------------------------
(/home/pedrobastos/repositories/mc920/conda-env) pedrobastos@ubuntu:trabalho3$
(/home/pedrobastos/repositories/mc920/conda-env) pedrobastos@ubuntu:trabalho3$ python3 alinhar.py --inputImg images/sample2.png --mode PROJECT
ION --outputImg result.png --houghThreshold 200
--------------------------------
Detected text before processing:
cee mnerrg0 ROME M AN ow, NON SAAROFEDRIATY 298

Natural Language Grammatical Inference
| Networks

rent Neural

with Recut
Steve Lawrence, Member. IEEE, C. Lee Giles, Fellow. 1EEE, and Sandiway Fong:
ernment tt eee Te canvas
song et ence ba sae es ect Sie saat
a a en ee ae cre ae or
Screens er a ee
wc pee oa roa
var ra aon
Sm ss oe wh or tower gen
ene ee npr Set
Sa ett ce > ae
ag caw mes seem sat
sn 2 sovormtrerrre
```



Um bom uso do tesseract que poderia ter sido feito neste trabalho seria para detectar se a imagem de saída alinhada estava de cabeça para baixo ou não. Poderia ter sido feito um reconhecimento via tesseract para checar se o texto reconhecido era o texto da imagem original. Se a imagem de saída alinhada estivesse de cabeça para baixo, o reconhecimento de texto via tesseract obteria uma resposta totalmente incoerente com o texto da imagem original.