PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

# Trabalho Prático: A Implementação de um OCR simplificado

André Lucas Ribeiro Costa[1]
Pedro Oliveira Simões[2]
Sandor Borges Scoggin[3]

**Resumo**

Esse trabalho irá descrever o processo e a implementação de um reconhecedor óptico de caracteres. O objetivo é ler uma sequência da imagem e identificar quais são os algarismos nessa imagem. O trabalho implementa técnicas de pré-processamento para encontrar os dígitos e após esse processo ele se utiliza de dois classificadores para analisar e definir qual era o algarismo escrito. Esses classificadores incluem uma máquina de vetores de suporte e uma rede neural de feed-forward. Com a implementação dessas técnicas foi usada a base de dados para algarismos escritos a mão do mnist. Ambos os classificadores foram treinados e testados com a base, e em ambos os casos foi possível atingir acurácia superior a 85%.

[1]Aluno do Programa de Graduação em Ciência da Computação, Brasil – andrelucassribeiro@gmail.com.
[2]Aluno do Programa de Graduação em Ciência da Computação, Brasil – pedroppsimoes@gmail.com.
[3]Aluno do Programa de Graduação em Ciência da Computação, Brasil – sandorscog@gmail.com.

# 1 INTRODUÇÃO

Neste documento será apresentado o projeto de implementação de um OCR simples, capaz de ler uma imagem com uma sequência de dígitos e identificar seu valor. É possível dividir a apresentação das implementações em duas partes principais, sendo a primeira constituída do pré-processamento após a obtenção da imagem, no qual ocorre os processos de localização e de obtenção dos números, e a segunda de reconhecimento, onde os dígitos são fornecidos à duas inteligências artificiais, o Support Vector Machine(SVM) e uma rede neural feed-forward, ambas treinadas com a base de dados MNIST. Em conjunto com as descrições das implementações, serão exibidos os problemas encontrados pelos autores a respeito dos resultados obtidos.

# 2 DESCRIÇÃO DO PROBLEMA E DAS TÉCNICAS IMPLEMENTADAS

O trabalho tem o dois principais objetivos que ao final se unem em um único software. Esses objetivos são: gerar projeções de algarismos de uma sequência e o reconhecimento desses algarismos a partir dessas projeções.

## 2.1 Pré-processamento e segmentação

Inicialmente é feito carregamento de uma sequência de algarismos contida em uma imagem, nela são feitas múltiplas operações para filtrar e identificar a sequência. O primeiro procedimento é a execução do algoritmo de Otsu, nesse caso a imagem, que já está em escala de cinza, será limiarizada e como resultado uma nova imagem é gerada. Com essa nova imagem serão aplicados métodos para encontrar os contornos de objetos da imagem, para isso é necessário diferenciar os contornos das áreas claras e compreender quais delas realmente definem os objetos. Ademais, ao encontrar os objetos é feito um alinhamento da sequência rotacionando a mesma, ou seja, os algarismos escritos ficam em uma posição melhor para gerar as projeções e uma nova imagem é gerada. Contudo, é necessário recalcular as posições das caixas de contorno para se adequar ao novo posicionamento dos objetos. Com os dígitos definidos dentro de suas caixas de contorno é possível fazer um recorte de cada um e salvá-los em uma estrutura de dados, também é necessário fazer uma ordenação deles a partir de suas posições na imagem rotacionada uma vez que a função de encontrar contornos do OpenCV, biblioteca utilizada para esse fim, retorna os contornos fora de ordem. Por fim, é necessário gerar uma projeção de cada um dos dígitos recortados, para isso são feitas as projeções verticais e horizontais, e elas são unidas em uma única projeção. Para encontrar essa projeção final é necessário fazer algumas operações a fim de padronizar as projeções, primeiramente a projeção horizontal é transposta, em seguida ambas as projeções são refatoradas e postas em um formato único que mantém as proporções originais, ao final, as duas projeções são concatenadas.

### 2.2 Reconhecimento e interpretação

A segunda parte do software trabalha com o reconhecimento dos dígitos, para isso é feito uma análise da projeção gerada na etapa anterior por um dos classificadores implementados. Ambos os classificadores devem ser inicialmente treinados para gerar um modelo e, de maneira posterior a esses treinos, podem ser feitas consultas que geram como resposta um valor derivado da análise da projeção passada para o teste e comparada com o modelo. O primeiro classificador é uma rede neural feed-forward, o formato de rede utilizado é a implementação disponibilizada pela biblioteca tensorflow, a base de dado que foi utilizada para construção do modelo é a do mnist. Para a implementação da rede foram definidas três camadas intermediárias de 128 neurônios e uma camada final com 10 saídas. A rede então é treinada com 16 épocas e a base do mnist, contudo, a base não é passada em seu formato original, são geradas projeções para cada elemento da base, essa projeção passa pelas mesmas operações aplicadas na etapa de pré-processamento para padronização como o algoritmo de otsu, a criação de projeções, a interpolação da projeção horizontal, e a refatoração respeitando as proporções da imagem original. Esse treino como resultado foi capaz de gerar um modelo com 91% de acurácia quando testado com a base de testes do mnist. O segundo modelo implementado foi uma Support Vector Machine (SVM), que passa por processo similar. Foi usado a SVM implementada pela biblioteca do OpenCV, assim como no treino do modelo anterior é necessário criar a SVM com parâmetros específicos, nesse caso foi usado uma função da própria biblioteca que busca os melhores valores para os parâmetros para definir como seria o treino. A base de treino passou pelos mesmos processos aplicados no modelo anterior e como resultado do treino foi possível alcançar 86% de acurácia ao testar com a base de testes do mnist.

## 3 COMPLEXIDADE DOS MÉTODOS

O cálculo das complexidades foi limitado à complexidade dos métodos implementados para o software em questão devido a indisponibilidade da informação de complexidade por parte das bibliotecas. Por exemplo, operações feitas pelo OpenCV são assumidas complexidades de $O(1)$.

Não houve grande variação na complexidade dos métodos implementados, com a maioria variando em três categorias. Os métodos: findBoundingBox, exNeuralNet, executeSVM, processForSVM, getTrainData e removeZeros possuem ordem de complexidade de $O(n)$. Os métodos: trainSVM, testSVM, processSamples e sort possuem ordem de complexidade de $O(n^2)$. Algumas exceções são os métodos: standarize e executeOtsu que possuem complexidades $O(V + H)$ e $O(LN)$ respectivamente. Os demais métodos foram considerados $O(1)$ devido às razões já descritas ou por serem métodos de interface gráfica.

## 4 BIBLIOTECAS UTILIZADAS E INSTRUÇÕES PARA USO

Para implementação do software foi utilizada a linguagem Python(PYTHON..., ) que pode ser instalada a partir do site da linguagem. Foram utilizadas cinco bibliotecas externas à linguagem. O OpenCV(OPENCV..., ), o Tensorflow(TENSORFLOW..., ), o Numpy(NUMPY..., ), o Keras(KERAS..., ) e Qt(QT..., ). Ambos o Tensorflow e o Keras podem ser instalados pelo comando "pip install tensorflow", essas bibliotecas foram usadas para acessar o mnist e para a implementação da rede neural. Para as interfaces gráficas foi usado o Qt, que pode ser instalado pelo comando "pip install pyside6". Para processamento das imagens e manipulação delas, além da implementação da SVM, foi utilizado o OpenCV, que pode ser instalado pelo comando "pip install opencv-python". Para algumas funções matemáticas e integração com o OpenCV foi usado o Numpy que pode ser instalado pelo comando "pip install numpy".

Para a interpretação do programa basta usar o comando "python main.py", esse processo inicia a aplicação e abre o OCR. Porém é necessário executar o treino dos modelos antes, para isso é necessário executá-los de maneira independente pelos comandos "python nnTrain.py" e "python svmTrain.py". Alternativamente é possível apenas clicar nos ícones dos arquivos uma vez que o python estiver instalado.

A utilização é descrita em grande parte pelos botões nas interfaces gráficas, ou seja, para o teste basta carregar a imagem e clicar os botões seguintes para limiarizar a imagem, aplicar a detecção dos dígitos e por fim classifica-los de acordo com o classificador desejado. Para os treinos dos modelos basta pressionar os botões de treino, para uso posterior da SVM e seus teste é necessário salvar o modelo antes usando o botão na interface.

## 5 MEDIDAS DE TEMPO DE EXECUÇÃO

Para a medida dos tempos foram definidas quatro seções, os treinos de cada modelo e os testes de cada modelo. Para a SVM temos uma média de tempo de 26 segundos e para a predição de 1.8 segundos. Já para a rede neural o seu treino ocorre com uma média de 21 segundos e a sua predição em 0.2 segundos.

## 6 EXEMPLOS DE ERROS E ACERTOS

Alguns erros ocorrem na fase de pré-processamento que impossibilitam parcialmente ou completamente uma análise de qualidade. Casos que comumente vão impossibilitar a análise da sequência incluem: rotação de alto grau em imagens em que a sequência se encontra próxima às bordas o que pode retirar os caracteres do domínio da imagem, caracteres com pouca largura o que causa fragmentação do dígito e caracteres que estão em contato com vizinhos o que impossibilita definir a caixa de limites e, por consequência, suas projeções.

Os classificadores possuem comportamentos distintos em suas avaliações. A SVM, apesar de ter 86% de acurácia nos testes do mnist, possui acurácia muito baixa quando exposta à outra entrada de dados, e não passa de 15%, o erro é em geral uma avaliação de quase qualquer projeção ser de um “5”. Já a rede neural tem uma taxa de acerto muito similar para os testes do mnist e de outras entradas, contudo existem erros comuns como interpretar o “1” como um “7” ou a troca entre “0”, “8” e “3”, além disso, algumas variações de “9” e “4” podem se confundir e gerar respostas equivocadas.

## 7 DISCUSSÃO COMPARATIVA DOS RESULTADOS

Apesar de ambos os classificadores terem conseguido resultados de qualidade nos treinos específicos e ao serem testados com a base de treino do próprio mnist, a rede neural se apresentou como uma solução mais eficiente. Para atingir esse modelo final com 91% de acurácia foram feitas múltiplas tentativas variando o número de neurônios e camadas intermediárias. Para o número de neurônios foram testados valores em potência de dois iniciando em 32 e indo até 512, mas após 128 os resultados não tinham ganhos reais e variam pouco. Algo semelhante ocorreu no número de camadas em que até 3 camadas havia grande avanço, porém um maior número de camadas não apresentava melhoria. Para os parâmetros da SVM foram testados vários parâmetros mas sem grande sucesso o que resultava em modelos que não passavam de 76% ou 79% de precisão quando testados com o mnist, então foi optado utilizar a função “trainAuto” que gera aproximações do que seriam os melhores parâmetros, e esses valores levaram a acurácia para níveis mais altos. Contudo, é importante ressaltar os resultados dos classificadores quando usados com dados vindos fora do mnist, a rede neural ainda possui grande taxa de acerto na avaliação, e consegue acertar 87%, próximo dos resultados com o mnist. Já a SVM não teve tanto sucesso não passando de 15%, o que foi surpreendente uma vez que isso não está próximo dos testes com a base do mnist.

## 8 CONCLUSÃO

Em suma, foi possível compreender melhor como ocorre cada etapa de um software de processamento de imagem. Neste trabalho foi feita a implementação de um OCR simples que se utiliza de dois classificadores, uma SVM e uma rede neural, para identificar algarismos escritos em sequência em uma imagem. Esses classificadores geram modelos treinados na base de dígitos escritos à mão do mnist, e ao testar com a base de testes dele foi possível alcançar mais de 85% de acurácia em ambos. Já em testes práticos a SVM não apresentou resultados satisfatórios, ficando abaixo de 15%, porém, a rede neural obteve saídas excelentes mantendo a média de acerto acima dos 85%. Esses resultados mostram uma boa fundação mas que ainda possui espaço para melhorias, especialmente no uso prático do classificador com SVM.

## REFERÊNCIAS

KERAS. Home - Keras Documentation. [S.l.: s.n.]. Disponível em: <https://keras.io/>. Acesso em: 30 outubro. 2021.

NUMPY — NumPy. [S.l.: s.n.]. Disponível em: <https://numpy.org/>. Acesso em: 30 outubro. 2021.

OPENCV: OpenCV modules. [S.l.: s.n.]. Disponível em: <https://docs.opencv.org/4.5.4/>. Acesso em: 30 outubro. 2021.

PYTHON. Welcome to Python.org. [S.l.: s.n.]. Disponível em: <https://www.python.org/>. Acesso em: 30 outubro. 2021.

QT Documentation. [S.l.: s.n.]. Disponível em: <https://doc.qt.io/>. Acesso em: 30 outubro. 2021.

TENSORFLOW. TensorFlow. [S.l.: s.n.]. Disponível em: <https://www.tensorflow.org/>. Acesso em: 30 outubro. 2021.