Detecção de Defeitos em Tecidos Através de Redes Neurais Convolucionais

Lucas Candiani Souza [a] e Celso Ap.de França [b]

Departamento de Engenharia Elétrica da Universidade Federal de São Carlos

***Resumo* – Este trabalho visa aplicar *transfer learning* a um detector baseado em redes neurais convolucionais para identificação de defeitos e estampas através de imagens de tecidos. Com a Indústria 4.0, sistemas inteligentes estão sendo cada vez mais aplicados de forma integrada na produção; sendo assim o objetivo deste trabalho é desenvolver um modelo para realizar a identificação automática de estampas e defeitos, visto que um detector eficaz aplicado em uma linha de produção pode reduzir custos e fornecer dados sobre a produção no geral. O dataset utilizado foi o ZJU-Leaper, o *backbone* utilizado foi da RetinanetR101 e para execução do código várias bibliotecas do Python foram utilizadas, como Pytorch, OpenCV e numpy, além da API do Detectron2. O mAP (mean average precision) do modelo desenvolvido, calculado para todas as classes (6 estampas e "defeito") no conjunto de validação, é de 92,4%.**

Palavras-Chave – Reconhecimento de Textura, Aprendizado de Máquina, Aprendizado por Transferência

1. Introdução

O crescente poder computacional atingido na última década alavancou ainda mais os estudos sobre redes neurais artificiais, dado que problemas que antes pareciam insolucionáveis, como processamento de linguagem natural, agora apresentam resultados similares (ou às vezes até melhores) ao desempenho humano. Com este rápido desenvolvimento de métodos de deep learning, vários campos como medicina, robótica, comércio e biologia atingiram consideráveis resultados em aplicações de aprendizado de máquina. Redes neurais convolucionais em específico têm mostrado alta performance em tarefas como classificação de imagens, segmentação e detecção de objetos [1].

Com o advento da indústria 4.0, o uso de componentes inteligentes, robôs e IoT é cada vez mais comum em fábricas [2]. Essa integração entre o ambiente digital e físico na indústria permite não apenas um desenvolvimento mais rápido de produtos, mas também personalização em massa (pedidos personalizados dos clientes vão direto para o sistema inteligente da linha de produção), flexibilização da produção e redução de custos.

[a] E-mail autor: lucascandiani@estudante.ufscar.br

[b] E-mail orientador:celsofr@ufscar.br

Para a indústria têxtil, por exemplo, o controle de qualidade é vital, visto que um defeito na malha tende a reduzir seu preço de venda de 45% a 65% [3]. Sendo assim, uma grande quantidade de defeitos de fabricação pode reduzir significativamente a receita de uma produtora de tecidos, de forma que é sumamente importante que tais defeitos sejam identificados durante a produção, assim como suas causas [4].

Apesar de vital, o processo de inspeção dos tecidos tende a se tornar extremamente cansativo para o funcionário (quando feito manualmente), dado o volume de produção das indústrias têxteis e o caráter repetitivo do trabalho. Além disso, o processo de inspeção manual é sujeito a falhas humanas e a queda de produtividade ao longo do turno de trabalho.

Nota-se então um gargalo de produtividade, dado que um inspetor humano, quando desempenha de forma plena, consegue inspecionar cerca de 20 metros por minuto de um rolo de tecido [5]. Naturalmente, dadas as características da tarefa, uma automação deste processo poderia não apenas aumentar a produtividade, como também reduzir custos ou liberar mão-de-obra humana para outras tarefas.

Portanto o objetivo desse trabalho de conclusão de curso é aplicar os conceitos de *machine learning* aqui descritos para a solução de um problema real: detecção de defeitos - como os mostrados na Figura 1 - em tecidos em uma linha de produção, além da classificação da estampa detectada.

Figura 1 - Exemplo de tecido com defeitos de fabricação

Fonte: *dataset* ZJU-Leaper

Para tal, o modelo proposto utiliza como entrada imagens de tecidos em linhas de produção. A fonte das imagens é o dataset público ZJU-Leaper [6], que contém mais de 90 mil imagens.

2. Fundamentação Teórica

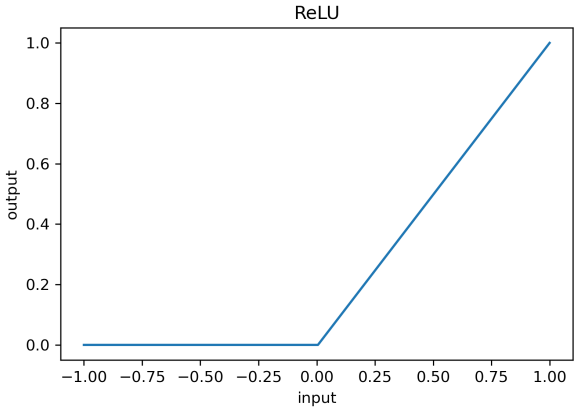
Redes neurais artificiais são fortemente inspiradas no funcionamento biológico de sistemas nervosos [7]. Os dados de entrada são alimentados a um grupo de nós computacionais

interconectados, chamados de neurônios, onde cada *input* X é multiplicado por um peso ω e então somado as outras entradas e aplicado à função de ativação do neurônio. Essa função determina se o neurônio vai "disparar" ou não, produzindo um *output* que será alimentado às camadas subsequentes da rede. O funcionamento de um neurônio pode ser expressado de forma geral pela Equação 1, onde *f* representa a função de ativação.

$$Output = f(\sum \omega_i X_i) \quad (1)$$

Funções de ativação determinam se o neurônio em questão será ativado ou não, além de serem as responsáveis por adicionar não-linearidade ao modelo (visto que o somatório ponderado de inputs é puramente linear). Um exemplo de função de ativação muito utilizada é a ReLU [7](Rectified Linear Unit), que retorna 0 para valores menores ou iguais a zero e retorna o próprio input para valores maiores que 0. A Figura 2 a seguir representa o comportamento dessa função de ativação.

Figura 2 - Comportamento da função ReLU

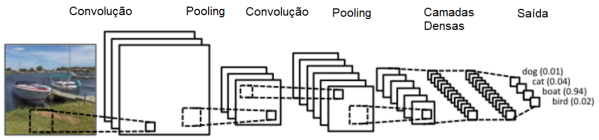


Fonte: autor

A concatenação de camadas de neurônios - onde a primeira recebe os *inputs*, e as subsequentes recebem as saídas das camadas anteriores - permite ao modelo ter uma grande capacidade de abstração, permitindo que ele aprenda representações dos dados e identifique padrões. Essas camadas de neurônios totalmente conectados são chamadas de camadas densas e esse processo de aprendizado com múltiplas camadas concatenadas é chamado de *Deep Learning* [8].

CNNs (Redes neurais convolucionais) possuem algumas significativas diferenças: como utilizam imagens como entrada, os *inputs* possuem 3 dimensões: largura, altura e profundidade - esta última representa a quantidade de canais de cor (RGB) da imagem, não a profundidade da rede - e três tipos de camadas: camadas densas, camadas convolucionais e camadas de *pooling*. Camadas convolucionais, diferentemente das camadas densas, recebem como entrada apenas uma parcela dos *inputs* da camada anterior e retornam um *output* de menor dimensionalidade em largura e altura, porém com maior profundidade (maior número de canais). Camadas de *pooling* visam reduzir a dimensionalidade da imagem sem perdas significativas de informação, a fim de acelerar o processo computacional. O processo de *pooling* consiste em representar uma área de n x n pixels em um único pixel, e repetindo-se esse processo por toda a imagem, deslizando essa "janela" de dimensão n x n . Essa representação pode ser feita por meio da média dos valores numéricos ou utilizando o máximo dentre estes valores, por exemplo. Essas camadas aplicadas em conjunto permitem ao modelo reconhecer padrões nas imagens e definir relações espaciais entre os elementos da imagem. Uma representação do funcionamento geral de uma CNN pode ser vista na Figura 3.

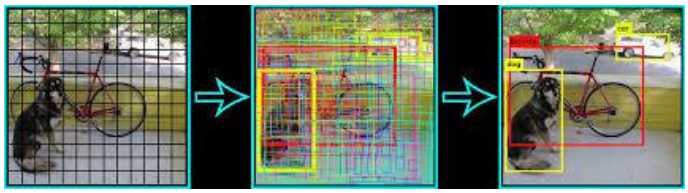
Figura 3 - Representação visual das camadas de uma CNN



Fonte: traduzido de https://matheusfacure.github.io/2017/03/12/cnn-captcha/

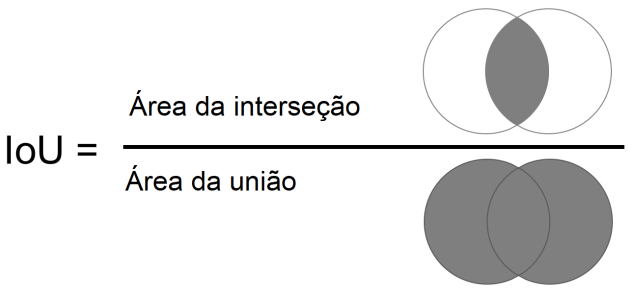
Detectores são um tipo especial de CNN que buscam identificar todas as instâncias de objetos - de classes conhecidas pelo modelo - presentes na imagem de entrada. Esse processo é feito através da proposição de regiões de interesse, chamadas âncoras, que são feitas para cada um dos pixels da imagem [9]. O modelo então dá uma pontuação (chamada valor de confiança) para cada uma das classes na região proposta. As dimensões m x n dessa janela são então ajustadas, baseado em regressão, e o processo é repetido. Naturalmente, esse processo retorna múltiplas detecções de um mesmo objeto, dado que várias âncoras detectam o mesmo objeto na imagem - como pode ser visto na Figura 3. Para solucionar este problema é utilizado *non-max suppression* [10], um método que analisa detecções com um IOU (intersection over union, representado visualmente na Figura 4) elevado - ou seja, significativamente sobrepostas - e descarta todas as detecções exceto a que possui o maior valor de confiança.

Figura 3 - âncoras antes e após *non-max suppression*



Fonte: http://cs231n.stanford.edu/slides/2020/section_7_detection.pdf

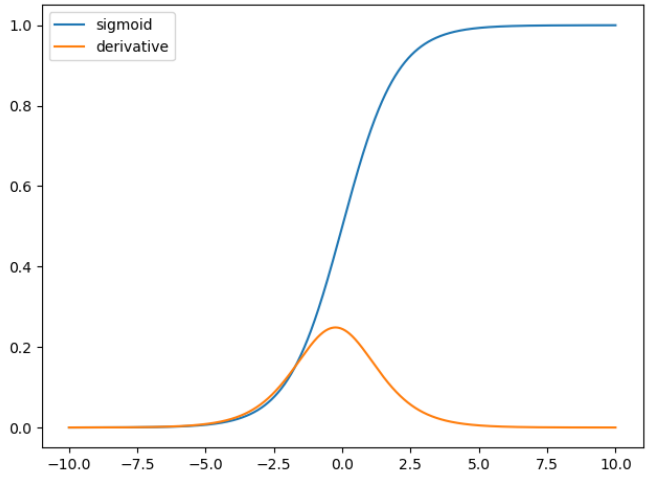
Figura 4 - Representação visual do IOU



Fonte: traduzido de https://www.oreilly.com/library/view/python-advanced-guide/9781789957211/5ff13e9b-f8db-445f-9a39-742ebdf61587.xhtml

Normalmente, espera-se que aumentar o tamanho (em número de neurônios) e profundidade (número de camadas) resulte em um melhor desempenho da rede neural; porém isso não é verdade em todos os casos. O erro do modelo pode aumentar devido a alguns fatores, como *overfitting* (pesos ajustados demais ao conjunto de treinamento, de forma que o modelo perde a capacidade de generalizar bem para novos dados) e *gradient vanishing* (desaparecimento do gradiente, cujo valor é utilizado para reajustar os pesos da rede durante o treinamento). O desaparecimento do gradiente é causado pelo fato de muitas funções de ativação (como a sigmóide, representada na Figura 5) achatam o espaço de *input* para valores entre 0 e 1, de forma que uma variação na entrada tende a gerar uma variação muito pequena na saída, logo a derivada (utilizada para o cálculo do gradiente) será muito pequena também. Esse efeito é potencializado com a concatenação de camadas, prejudicando o treinamento [11].

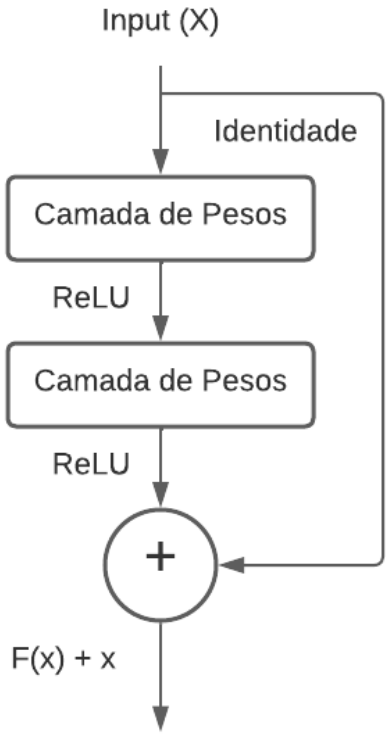
Figura 5 - Função de ativação sigmóide e sua derivada



Fonte: https://suzyahyah.github.io/machine%20learning/2020/01/17/Sigmoid.html

Muitas abordagens foram sugeridas para contornar este problema, e para este trabalho a abordagem escolhida foi utilizar uma *ResNet* [12](rede neural residual). Redes residuais atenuam significativamente o impacto do desaparecimento do gradiente, devido ao fato do input ser alimentado novamente após um número determinado de camadas, como mostra a Figura 6. O bloco lógico representado na figura é então concatenado com outros blocos similares, de forma que indiferentemente da profundidade da rede o *input* vai ser sempre alimentado novamente a cada N camadas (definidas na arquitetura do modelo).

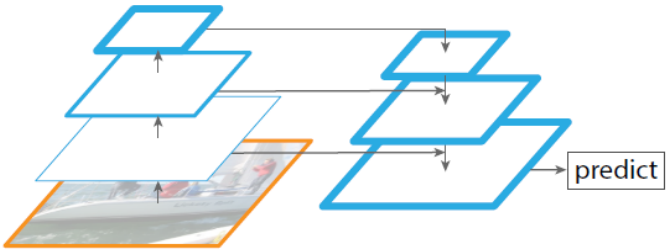
Figura 6 - representação visual de um bloco lógico da ResNet



Fonte: autor

Mais uma questão relevante é que os objetos podem aparecer em diferentes escalas em relação ao tamanho total da imagem. Para melhorar o desempenho para sistemas de reconhecimento uma abordagem comumente usada é o uso de *Feature Pyramids* [13] (Pirâmides de características). Este método consiste no uso de uma pirâmide hierárquica de regiões de predição, como ilustrado no Figura 7. Cada nível da pirâmide tem seus pesos ajustados separadamente, de forma que todos os níveis são semanticamente fortes. Os *outputs* do nível mais alto, semanticamente mais fortes porém espacialmente mais escassos, são então redimensionados e fundidos com a saída do segundo nível hierárquico da pirâmide através de uma conexão lateral. Esse processo é repetido até a dimensionalidade se tornar novamente igual à base da pirâmide. Esse processo é mais custoso computacionalmente e aumenta significativamente o número de parâmetros, porém quando utilizado em conjunto com CNNs se mostra eficaz para reconhecer objetos em diferentes escalas.

Figura 7 - Representação visual da pirâmide de características



Fonte: LIN, Tsung-Yi et al.[13]

Além da qualidade da arquitetura e tamanho (quantidade de neurônios e camadas) do próprio modelo, outro fator determinante no desempenho de um algoritmo de *deep learning* é a quantidade e qualidade dos dados utilizados para treinamento, visto que são deles que o modelo vai tirar o aprendizado.

Uma forma de aumentar a quantidade de dados de treinamento sem fazer coleta é utilizar *data augmentation*[14], um método que gera novos dados de entrada aplicando determinadas transformações aos mesmos, como *flips* horizontais e verticais, rotações, *crops* (recortes) de seções das imagens, entre outros. Além de gerar mais dados, essa abordagem aprimora o desempenho do modelo dado que um objeto rotacionado ou espelhado continua sendo o mesmo objeto, entretanto os valores numéricos dos pixels são diferentes, auxiliando o modelo a generalizar.

Outra forma de melhorar o desempenho do modelo é utilizar *transfer learning* [15] (aprendizado por transferência), que consiste na inicialização do modelo já com pesos de um treinamento em outro conjunto de dados, ao invés de zeros ou aleatórios, e então fazer o *fine-tuning* através de um novo treinamento do modelo nos novos dados. Esse processo, além de acelerar o treinamento, produz melhores resultados e é mais efetivo quanto maior for a semelhança entre o conjunto de dados de treinamento atual e prévio.

3. Trabalhos Relacionados

Várias abordagens são possíveis para detecção de defeitos em imagens de tecidos através de redes neurais convolucionais. Jing et al. [16] propõe separar a imagem de *input* em 81 entradas, um *grid* (grade) de 9x9, de menores dimensionalidades, e então classificar cada segmento da imagem. Esta abordagem eleva a acurácia do modelo, porém diminui a precisão espacial da detecção, dado o *grid* fixo.

Zhao et al. [17] apresenta um modelo que também subdivide a imagem de entrada em múltiplas partes, porém aplicando um auto-codificador em cada parte para classificar a amostra entre normal ou defeituosa, e segmentar o defeito quando presente. Uma vantagem desta abordagem é o bom desempenho mesmo com poucas amostras da classe defeituosa, visto que é a proporção esperada durante o funcionamento em uma linha de produção.

O modelo proposto por Huang et al. [18] entretanto separa o processo entre segmentação e decisão, isto é, uma rede neural faz a segmentação das regiões de interesse e uma segunda classifica (decide) se a região segmentada é um defeito.

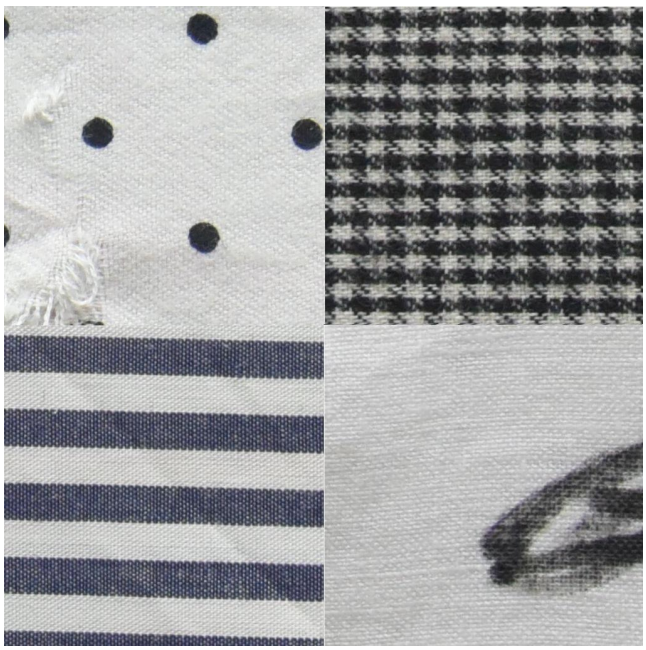
Estas abordagens porém sofrem de um mesmo problema: os modelos só desempenham bem para imagens que tem estampas representadas no conjunto de treinamento. Para lidar com esta limitação, Yapi et al. [19] propõe um modelo mais robusto, que separa o processo em duas etapas: aplicar a transformada de Contourlet recursiva por toda a imagem a fim de reconhecer o padrão que se repete na estampa, e então utilizar essa informação para detectar defeitos na imagem através de um classificador bayesiano.

Outra questão que deve ser levada em conta para utilizar um detector baseado em redes neurais em uma linha de produção é o tempo de inferência, ou seja, o tempo que o algoritmo leva para realizar a predição na imagem de entrada. Como isso pode ser um fator limitante, a proposta de Jing et al. [20] para lidar com o problema foi utilizar uma rede convolucional chamada Mobile-Unet. Visando o melhor desempenho computacional, este modelo realiza uma convolução separável em profundidade, que reduz a complexidade do modelo.

4. Metodologia

O *dataset* (conjunto de dados) utilizado foi uma parcela do ZJU-Leaper [6], que contém 98777 imagens de 512x512 pixels divididos em 19 tipos de estampas, com todas as classes contendo exemplos defeituosos. Para o experimento foram utilizadas 756 imagens divididas entre 6 classes (estampas): branco liso, listras finas, listras grossas, dot, houndstooth e gingham. Das 126 amostras de cada classe, 72 amostras eram normais e 54 defeituosas. A Figura 8 a seguir apresenta alguns exemplos de estampas com e sem defeitos.

Figura 8 - Exemplos de imagens presentes no ZJU-Leaper



Fonte: dataset ZJU-Leaper [6]

Para tarefas de visão computacional, anotações são fundamentais para fornecer metadados ao modelo. Neste contexto, anotações são um conjunto de informações (normalmente 4 coordenadas posicionais da *bounding box* e um rótulo/classe) que são utilizadas como *ground truth* (referência) pelo modelo para o aprendizado. Uma *bounding box* é um retângulo alinhado com os eixos X e Y que descreve a posição do objeto na imagem.

As anotações foram realizadas manualmente utilizando o software CVAT [21] (Computer Vision Annotation Tool) e seguindo o seguinte método: foi anotada uma *bounding box* que compreende toda a imagem, com a classe correspondente à estampa do tecido e uma *bounding box* para cada defeito, quando presentes, como uma sétima classe. A Figura 9 a seguir mostra um exemplo de anotação utilizando esta ferramenta e método.

Figura 9 - Exemplo de anotação na interface do CVAT

Fonte: autor

Fazer as anotações é uma tarefa que consome tempo e de alto custo, dado que é necessário mão-de-obra humana - por vezes especializada na tarefa que o modelo pretende desempenhar - para realizá-la. Porém, a qualidade das anotações está diretamente ligada ao desempenho do algoritmo [22].

As 756 imagens foram então divididas em 2 conjuntos: 80 % das amostras para treinamento e 20 % para validação. O conjunto de treinamento é aquele que o modelo recebe não somente as entradas, mas também as saídas esperadas; através do treino o modelo ajusta os pesos através de *backpropagation,* corrigindo os pesos de cada camada retroativamente, baseado nos erros de suas previsões. O conjunto de validação é útil para analisar o desempenho do modelo em novos casos, dado que no treinamento o modelo não teve acesso às saídas esperadas destes dados. As métricas no conjunto de validação são portanto o real medidor de desempenho do modelo, dado que a proposta de operação consiste em analisar dados novos.

A métrica utilizada para comparação entre o modelo proposto e os modelos apresentados pelos criadores do dataset foi o *F1 score* (Equação 2), que consiste na média harmônica da precisão (Equação 3) e do *recall* (Equação 4). Nas equações abaixo, a variável TP se refere a verdadeiros positivos, FP a falsos positivos e FN a falsos negativos.

$$F1\ Score = 2\ \frac{precisão * recall}{precisão + recall} \quad (2)$$

$$Precisão = \frac{TP}{TP + FP} \quad (3)$$

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

O modelo utilizado no experimento foi a RetinanetR101 FPN3x [23], que foi implementado utilizando a API (Application Programming Interface) do *Detectron2* [24], biblioteca do *Python* que permite fácil implementação de algoritmos do estado-da-arte de detecção de objetos. Esta biblioteca também calcula automaticamente o mAP (mean Average Precision), métrica amplamente utilizada em detecção de objetos. *Average Precision* (AP) é definido como a área abaixo da curva de *precision-recall* para determinado limite de IOU [25](para um limite mínimo de IOU de 75%, a métrica é chamada de AP 75, por exemplo); já o *mean Average Precision* é a média entre APs calculados em diferentes limites. Estas métricas permitem analisar não somente se o modelo consegue detectar ou não o objeto, mas também quão boas são as regiões de interesse propostas pelo modelo - comparando o mAP, AP 90 e AP 75 por exemplo.

A RetinanetR101 FPN3x é implementada utilizando um *backbone* da RetinanetR101, que possui 100 camadas ocultas, porém adicionando as *Feature Pyramids*, totalizando mais de 57 milhões de parâmetros treináveis.

O transfer learning foi feito através dos pesos da RetinaNet quando treinada no *imagenet* [26], um dos maiores *datasets* públicos de imagens, contendo mais de 14 milhões de amostras. Mesmo com o conteúdo dos dois conjuntos de dados sendo bem diferente, essa inicialização de pesos auxilia a identificação de objetos em geral dada a variedade de classes presentes na *imagenet*.

5. Resultados e Discussões

Todas as 152 imagens do conjunto de validação foram classificadas corretamente em relação a estampa, porém algumas poucas *bounding boxes* preditas não preenchiam a totalidade da imagem. Isso, porém, não gera um impacto negativo real, dado que essas 6 classes são apenas para classificação. A Figura 10 mostra um exemplo de output esperado quando não há defeitos presentes na imagem.

Figura 10 - Exemplo de predição sem defeitos

Fonte: autor

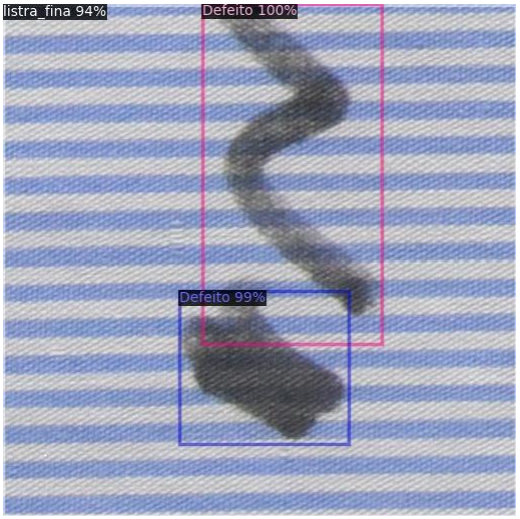
Em todo o conjunto de validação há 93 defeitos anotados em diferentes imagens - esse valor é correspondente a *N° Total de Positivos* nas Equaçãoes (3) e (4). O modelo detectou 77 instâncias da classe "Defeito" em todo o conjunto (valor correspondente à *Positivos Detectados*), dos quais 72 foram considerados *"Positivos Verdadeiros"*, dado que seus valores de IOU eram superiores a 0,8. A Tabela 1 a seguir apresenta as métricas calculadas a partir destes valores.

Tabela 1 - Métricas do modelo no conjunto de validação

Métrica	Valor
Precisão	0,77
Recall	0,94
F1 Score	0,83

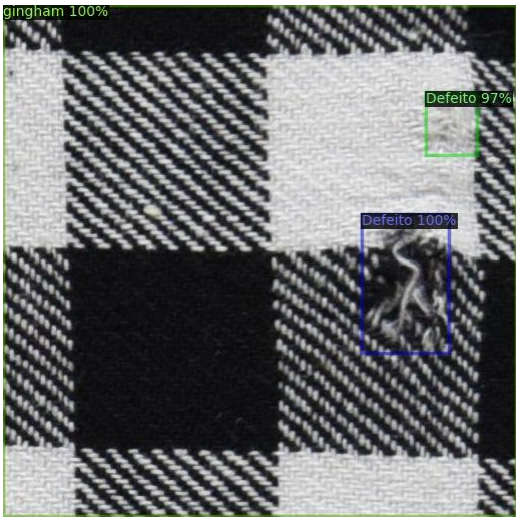
Reduzindo o limite de IOU para se considerar uma detecção como *Positivo Verdadeiro* para 0,5, o recall sobe para 1. Esses dados mostram que, por mais que alguns defeitos não tenham sido detectados, a taxa de falsos positivos é muito baixa. As Figuras 11 e 12 mostram dois exemplos bons de predições, e as figuras 13 e 14 são exemplos de falhas.

Figura 11 - Exemplo 1 de boa predição



Fonte: autor

Figura 12 - Exemplo 2 de boa predição

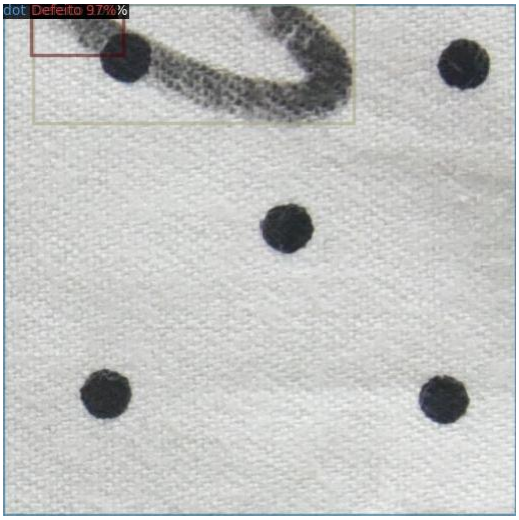


Fonte: autor

Figura 13 - Exemplo 1 de má predição

Fonte: autor

Figura 14 - Exemplo 2 de má predição



Fonte: autor

A Tabela 2 a seguir apresenta a comparação entre os F1 scores dos *baselines* propostos pelos autores do *dataset* ZJU-Leaper [6] e do modelo aqui proposto.

Tabela 2 - Comparação de F1 scores entre o modelo proposto e os *baselines* do ZJU-Leaper

Modelo	F1 Score
Sparse Coding	0,169
Convolutional auto encoder	0,187
One-class SVM	0,118
U-Net	0,558
U-Net(Transfer Learning)	0,603
U-Net(Data augmentation)	0,588
U-Net(label annotations)	0,224
U-Net (bounding box ann.)	0,683
U-Net(all masks annotations)	0,906
RetinanetR101FPN3x (proposto)	0,826

Percebe-se que o modelo proposto atingiu o 2º maior *F1 score,* atrás apenas do U-Net com masks annotations. Porém, esse modelo é custoso em termos de processamento de acordo com os próprios autores.

Entre os trabalhos relacionados, Jing et al. [19] apenas fornece os valores de precisão do modelo nos datasets testados (que é em média de 87%), não sendo possível calcular o *F1 score* pois não há informação sobre o recall. Sendo assim, a Tabela 3 apresenta a comparação entre os *F1 scores* médios (entre todas as classes/estampas) dos trabalhos relacionados no conjunto de validação dos seus respectivos *datasets*.

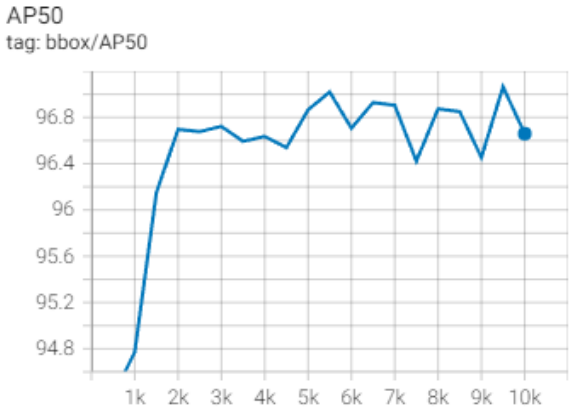
Tabela 3 - Comparação de F1 scores entre o modelo proposto e os trabalhos relacionados

Modelo	F1 Score
Jing et al. [16]	0,973
FCSDA [17]	0,416
RCT-MoGG [18]	0,953
Mobile-Unet [20]	0,820
RetinanetR101FPN3x (proposto)	0,826

Como o modelo de Jing et al. [16] apenas classifica uma das 81 subdivisões da imagem, essa alta pontuação vem em detrimento de uma boa localização do defeito. Percebe-se também que a pontuação do RCT-MoGG é significativamente maior que o restante, mostrando a eficiência do método de detectar padrões de estampa e então segmentar seus ruídos.

A API do *Detectron2* também gera gráficos dos mAPs ao longo das iterações de treino do modelo, como pode ser visto na Figura 15. Mesmo sem uma base de comparação com outros modelos, esses dados indicam novamente uma taxa muito baixa de falsos positivos, dado o AP 50 (mean average precision com o limite de IOU para verdadeiros positivos de 0,5) de 96,6% ser ainda mais alto que o mAP (média entre diferentes limites) de 92,4%, indicando que algumas bounding boxes consideradas como falsos positivos na verdade apenas não cobriram a área necessária do defeito.

Figura 15 - AP50 ao longo das iterações de treinamento



Fonte: autor

O tempo de execução da predição do modelo treinado foi de 19,07 segundos para as 152 imagens do conjunto de validação utilizando como *hardware* um computador pessoal com 16 Gb de memória RAM e utilizando GPU (RTX 3060), o que equivale a uma taxa de 8 imagens processadas por segundo.

Dois dos trabalhos relacionados informaram o tempo de execução da inferência, e estes estão representados na Tabela 4 em comparação com o modelo proposto. Entretanto, ambos modelos utilizaram como *inputs* imagens de 256x256 pixels, enquanto neste projeto os *inputs* utilizados eram de 512x512. Sendo assim, cada imagem de 512x512 tem quatro vezes mais

dados a serem processados. Portanto, a quantidade de inferências por segundo do modelo proposto será multiplicada por 4, como uma estimativa para a quantidade de inferências por segundo na mesma dimensão dos outros modelos, para tornar a comparação mais justa.

Tabela 4 - Comparação de inferências por segundo dos modelos quando executados em GPU

Modelo	Inferências por segundo
RCT-MoGG [18]	25
Mobile-Unet [20]	130
RetinanetR101FPN3x (proposto)	32

Percebe-se que de fato a Mobile-Unet é mais eficiente computacionalmente por uma larga margem. Porém, o modelo proposto ainda possui uma taxa de inferência maior que o RCT-MoGG.

6. Conclusões

O modelo aqui sugerido se mostrou capaz de identificar facilmente todas as estampas e a maioria dos defeitos. Entretanto, como pode-se perceber na Tabela 3, existem modelos mais robustos para segmentar com mais precisão os defeitos, como o RCT-MoGG. Como pode ser visto na Tabela 4, o modelo proposto tem uma taxa de inferência maior que o RCT-MoGG, porém bem inferior à taxa da Mobile-Unet, que possui um desempenho bem parecido. Apesar disso, existem formas de aprimorar o resultado obtido com o modelo proposto, como: reduzir a dimensionalidade das entradas a fim de reduzir o gasto computacional (mas não a ponto de prejudicar o desempenho), utilizar mais imagens para treinamento (afinal das 98777 imagens disponíveis, apenas 604 foram utilizadas para treinamento), melhorar a qualidade das anotações utilizando segmentações ao invés de bounding boxes, ajustes nos hiperparâmetros, etc. Ademais, o tempo e esforço humano gastos para realizar as anotações na abordagem proposta são significativamente menores que os necessários para anotações de segmentação. Sendo assim, este modelo pode ser aperfeiçoado e se tornar solução plausível para o problema proposto.

6 Referências

[1] DHILLON, Anamika; VERMA, Gyanendra K. Convolutional neural network: a review of models, methodologies and applications to object detection. **Progress in Artificial Intelligence**, v. 9, n. 2, p. 85-112, 2020.

[2] DAMIANI, Lorenzo et al. Augmented and virtual reality applications in industrial systems: A qualitative review towards the industry 4.0 era. **IFAC-PapersOnLine**, v. 51, n. 11, p. 624-630, 2018.

[3] DASTOOR, P. H. et al. SDAS: a knowledge-based framework for analyzing defects in apparel manufacturing. **Journal of the Textile Institute**, v. 85, n. 4, p. 542-560, 1994.

[4] GAJANAN, G. K. Review of defect detection and classification methods for fabric. International Journal of Scientific & Engineering Research, 2014.

[5] KUMAR, Ajay. Computer-vision-based fabric defect detection: A survey. **IEEE transactions on industrial electronics**, v. 55, n. 1, p. 348-363, 2008.

[6] ZHANG, Chenkai et al. ZJU-Leaper: A Benchmark Dataset for Fabric Defect Detection and a Comparative Study. **IEEE Transactions on Artificial Intelligence**, v. 1, n. 3, p. 219-232, 2020.

[7] AGARAP, Abien Fred. Deep learning using rectified linear units (relu). **arXiv preprint arXiv:1803.08375**, 2018.

[8] LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **nature**, v. 521, n. 7553, p. 436-444, 2015.v.

[9] ZHANG, Xiaosong et al. Freeanchor: Learning to match anchors for visual object detection. **Advances in neural information processing systems**, v. 32, 2019.

[10] NEUBECK, Alexander; VAN GOOL, Luc. Efficient non-maximum suppression. In: **18th International Conference on Pattern Recognition (ICPR'06)**. IEEE, 2006. p. 850-855.

[11] HU, Yuhuang et al. Overcoming the vanishing gradient problem in plain recurrent networks. **arXiv preprint arXiv:1801.06105**, 2018.

[12] HE, Kaiming et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2016. p. 770-778.

[13] LIN, Tsung-Yi et al. Feature pyramid networks for object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2017. p. 2117-2125.

[14] SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on image data augmentation for deep learning. **Journal of big data**, v. 6, n. 1, p. 1-48, 2019.

[15] WEISS, Karl; KHOSHGOFTAAR, Taghi M.; WANG, DingDing. A survey of transfer learning. **Journal of Big data**, v. 3, n. 1, p. 1-40, 2016.

[16] JING, Jun‐Feng; MA, Hao; ZHANG, Huan‐Huan. Automatic fabric defect detection using a deep convolutional neural network. **Coloration Technology**, v. 135, n. 3, p. 213-223, 2019.

[17] LI, Yundong; ZHAO, Weigang; PAN, Jiahao. Deformable patterned fabric defect detection with fisher criterion-based deep learning. **IEEE Transactions on Automation Science and Engineering**, v. 14, n. 2, p. 1256-1264, 2016.

[18] HUANG, Yanqing; JING, Junfeng; WANG, Zhen. Fabric defect segmentation method based on deep learning. **IEEE Transactions on Instrumentation and Measurement**, v. 70, p. 1-15, 2021.

[19] YAPI, Daniel; ALLILI, Mohand Saïd; BAAZIZ, Nadia. Automatic fabric defect detection using

learning-based local textural distributions in the contourlet domain. **IEEE Transactions on Automation Science and Engineering**, v. 15, n. 3, p. 1014-1026, 2017.

[20] JING, Junfeng et al. Mobile-Unet: An efficient convolutional neural network for fabric defect detection. **Textile Research Journal**, v. 92, n. 1-2, p. 30-42, 2022.

[21] MANOVICH, Nikita. **Computer Vision Annotation Tool**. [S. l.], 29 jun. 2018. Disponível em: https://github.com/openvinotoolkit/cvat. Acesso em: 20 abr. 2022.

[22] TARAN, Vlad et al. Impact of ground truth annotation quality on performance of semantic image segmentation of traffic conditions. In: **International Conference on Computer Science, Engineering and Education Applications**. Springer, Cham, 2019. p. 183-193.

[23] LIN, Tsung-Yi et al. Focal loss for dense object detection. In: **Proceedings of the IEEE international conference on computer vision**. 2017. p. 2980-2988.

[24] WU, Yuxin. **Detectron2** [S. l.], 8 set. 2020. Disponível em: https://github.com/facebookresearch/detectron2. Acesso em: 20 abr. 2022.

[25] HENDERSON, Paul; FERRARI, Vittorio. End-to-end training of object class detectors for mean average precision. In: **Asian conference on computer vision**. Springer, Cham, 2016. p. 198-213.

[26] DENG, Jia et al. Imagenet: A large-scale hierarchical image database. In: **2009 IEEE conference on computer vision and pattern recognition**. Ieee, 2009. p. 248-255.