UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Henrique Roos Baltezan

Validação da abordagem de segmentação via superpixel seguida de uma classificação via redes neurais convolucionais

Florianópolis
2021

Henrique Roos Baltezan

Validação da abordagem de segmentação via superpixel seguida de uma classificação via redes neurais convolucionais

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Eric Aislan Antonelo, Dr.
Supervisor: Renato Simão, MSc.

Florianópolis
2021

Henrique Roos Baltezan

Validação da abordagem de segmentação via superpixel seguida de uma classificação via redes neurais convolucionais

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 03 de Dezembro de 2021.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Eric Aislan Antonelo, Dr.
Orientador
UFSC/CTC/DAS

Renato Simão, MSc.
Supervisor
Instituto SENAI de Inovação em Sistemas Embarcados

Prof. Jomi Hubner, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

ACORDO DE NÃO DIVULGAÇÃO

O documento original deste Projeto de Fim de Curso contém dados confidenciais do Instituto SENAI de Inovação em Sistemas Embarcados. Não é permitida a publicação ou reprodução da obra original. O trabalho original só poderá ser disponibilizado aos revisores, à banca examinadora e aos responsáveis pelo arquivamento na universidade.

Desta maneira, o relatório aqui apresentado foi elaborado para conter apenas dados não confidenciais, a serem disponibilizados ao público em geral, conforme exigido pela Universidade Federal de Santa Catarina.

Florianópolis, dezembro de 2021

AGRADECIMENTOS

Especificamente sobre o Projeto de Fim de Curso, gostaria de agradecer o apoio, companheirismo e ensinamentos de toda a equipe do Instituto SENAI de Inovação em Sistemas Embarcados. Especialmente, gostaria de agradecer a: Patryk Gonçalves, Renato Simão, Alisson Furlani e Sidney de Carvalho.

Também com respeito ao Projeto de Fim de Curso, agradeço ao professor Eric Aislan Antonelo, por toda a ajuda, mas principalmente, pela ajuda na escrita da monografia.

Com relação à graduação, gostaria de agradecer a meus colegas, amigos e professores, por todo o auxílio durante minha formação.

Ademais, gostaria de agradecer a todo o apoio de meus pais, namorada e amigos, durante este período, não só no projeto de fim de curso e na graduação, mas também em todo o resto.

RESUMO

A tarefa principal a ser realizada pelo sistema de visão computacional e aprendizado de máquina, é receber uma série de imagens, capturadas por uma câmera, e retornar, a porcentagem de cada uma das classes em cada imagem. Para realizar tal tarefa, foi utilizado um método de segmentação, com objetivo de separar as classes na imagem, seguido por um método de classificação, para classificar cada região segmentada como sendo de uma classe ou de outra. O método utilizado na segmentação, foi o de segmentação por superpixel, que consiste, basicamente, no agrupamento de pixeis, com base em suas posições e cores. Para a classificação, diversos métodos foram utilizados e comparados ao longo deste documento. Constatou-se que o método de classificação por redes neurais convolucionais, supera, na precisão, de forma consistente, as demais técnicas consideradas no estudo.

Palavras-chave: Superpixel. Redes neurais convolucionais.

ABSTRACT

The main task to be performed by the computer vision and machine learning system is to receive a series of images, captured by a camera, and return the percentage of each of the classes in each image. To accomplish this task, a segmentation method was used, with the objective of separating the classes in the image, followed by a classification method, to classify each segmented region as being of one class or another. The method used in segmentation was superpixel segmentation, which basically consists of grouping pixels based on their positions and colors. For classification, several methods were used and compared throughout this document. It was found that the classification method by convolutional neural networks consistently outperforms the other techniques considered in the study.

Keywords: Superpixel. Convolutional neural networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
API	Application Programming Interface
CNN	Rede Neural Convolucional
CPU	Central Processing Unit
ET	Extra Trees
GPU	Graphics Processing Unit
HSV	Hue Saturation Value
ISI-SE	Instituto SENAI de Inovação em Sistemas Embarcados
MLP	Multi-Layer Perceprton
PFC	Projeto de Fim de Curso
RF	Random Forest
RGB	Red Green Blue
SLIC	Simple Linear Iterative Clustering
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
UDP	User Datagram Protocol
VC	Visão Computacional

SUMÁRIO

1 INTRODUÇÃO

Como mencionado no acordo de não divulgação, a área na qual o projeto foi feita não pode ser divulgada por motivos de sigilo. Portanto, este documento será baseado na validação dos métodos de Visão Computacional (VC) e Aprendizado de Máquina (AM) abordados durante o Projeto de Fim de Curso (PFC).

1.1 ESCOPO DO PROJETO

1.1.1 Equipes do projeto

O projeto da empresa Instituto SENAI de Inovação em Sistemas Embarcados (ISI-SE) é dividido em diversas áreas, cada uma atribuída a uma equipe responsável. As equipes são dividias em:

- **Equipe de VC e AM:** Responsável por toda a parte de VC e AM. O trabalho realizado por essa equipe, pode ser dividido em duas grandes áreas, a área responsável pelo aquisição e a área responsável pelo sistema VC e AM.

  O sistema de aquisição, é responsável por adquirir as imagens e envia-las ao sistema de VC e AM.

  Já o sistema de VC e AM consiste, basicamente, em algoritmos de VC e AM, que recebem imagens do sistema de aquisição e retornam variáveis utilizadas por outros sistemas (i.g. sistema de controle).

- **Equipe de controle:** Responsável pelo desenvolvimento do sistema de controle.

- **Equipe de mecânica e instrumentação:** Responsável pelo dimensionamento dos sensores e instrumentação das peças utilizadas por todas as equipes.

- **Equipe de *firmware*:** Responsável por toda a parte de comunicação.

O autor deste documento, faz parte da equipe de VC e AM e desenvolveu seu PFC justamente nessa área.

1.2 OBJETIVOS

Como objetivos deste projeto, tem-se:

- Analisar o conjunto de dados para decidir a abordagem utilizada.
- Validar a abordagem de segmentação via superpixeis.
- Criar um *framework* para rotular o conjunto de dados de superpixeis.

- Para classificar os superpixeis, validar a abordagem de extração de *feature* manual, seguida de uma classificação.
- Ainda para classificação de superpixeis, validar a utilização de uma Rede Neural Convolucional (CNN).
- Comparar os resultados obtidos pela CNN com a abordagem de extração de *feature* manual.
- Realizar a análise estatística dos dados obtidos.
- Realizar melhorias no tempo de processamento do algoritmo.
- Comparar o algoritmo antes e depois das melhorias no tempo de processamento.

1.3 METODOLOGIA

O PFC, teve sua metodologia planejada com base nas atividades designadas ao autor deste documento e seguiram os padrões de atividades da empresa (i.g. viagens de trabalho e reuniões). Ademais, a organização das atividades é feita por meio do método *Kanban*, no aplicativo *Microsoft Teams*.

A primeira etapa, consiste na análise do conjunto de dados, para que assim seja possível tomar uma decisão sobre quais abordagens serão utilizadas. Toda a análise é realizada utilizando a linguagem de programação *Python*.

Em um segundo momento, após analisar o conjunto de dados e decidir para que caminho seguir, faz-se necessário o estudo das possíveis técnicas utilizadas. Diversas técnicas avançadas de VC e AM foram estudadas, contudo, destaca-se a CNN e a técnica de segmentação via superpixel.

Uma vez finalizado o estudo das técnicas de VC e AM, foi dado início a etapa de desenvolvimento dos algoritmos e *frameworks* utilizados. O algoritmo principal, pode ser dividido em duas partes, uma segmentação da imagem, seguida por um processo de classificação. Ambas foram realizadas durante esse período. Nesta etapa, também ocorreu o processo de rotulação das classes utilizadas no aprendizado supervisionado.

Após um certo tempo, também foram iniciados os testes dos algoritmos, tanto em laboratório, quanto em campo. Durante o PFC, foram feitas diversas viagens, nas quais, foram testadas todas as funcionalidades desenvolvidas pela equipe do ISI-SE, incluindo os algoritmos, de VC e AM, elucidados neste documento. Para todos esses testes, a equipe seguiu a seguinte metodologia:

1. Preparar os algoritmos e testá-los em laboratório (nos mesmos *hardwares* que vão para campo).

2. Quando em campo, fazer a montagem de todo o equipamento (placas, cabos, sensores, etc.).
3. Realizar os testes programados e exigidos pela empresa, em reuniões prévias.
4. Reportar todos os problemas e eventuais possíveis soluções.
5. De volta ao laboratório, realizar os ajustes necessários e implementar as soluções, dando início a um novo ciclo (voltando ao item 1).

Por fim, vale destacar que, os resultados são entregues e discutidos, com o supervisor geral e todos os outros colegas de projeto, em reuniões semanais.

1.4 ORGANIZAÇÃO DO DOCUMENTO

O documento é organizado em cinco capítulos. No primeiro capítulo, é feita uma introdução, apresentada a divisão das equipes, discutidos os objetivos e apresentados a metodologia do PFC e organização do documento.

No segundo capítulo, uma fundamentação teórica é realizada, com intuito de apresentar os temas e métodos estudados ao leitor.

O terceiro capítulo, apresenta a análise do conjunto de dados, bem como a abordagem de segmentação utilizada. É importante salientar que, essa separação, ainda não é a classificação, os elementos são segmentados, porém, ainda não se sabe a que classes pertencem. Ao fim do capítulo, é apresentada a aplicação, no conjunto de dados, do método de segmentação escolhido.

Com a segmentação realizada na imagem, é possível separar (com um erro associado) as classes desejadas. Essa separação, pode então ser alimentada a um classificador, que classifica cada elemento e torna possível o cálculo da porcentagem de cada uma das classes na imagem. Todo o processo relacionado com a escolha do método de classificação é discutido no capítulo quatro.

O quarto capítulo, descreve os métodos testados, resultados obtidos e modificações realizadas, seguindo uma ordem cronológica conforme a realização do PFC, desde o primeiro método testado, até o método escolhido. Os métodos são divididos em duas principais abordagens, com e sem extração de *feature* manual, para uma posterior comparação. Ainda neste capítulo, o método escolhido tem sua aplicação no conjunto de dados apresentada. Por fim, é discutida a comunicação com outros sistemas e as APIs utilizadas.

O quinto capítulo finaliza o desenvolvimento do documento com as validações experimentais. Neste capítulo é feita a análise estatística referente ao tempo de processamento do algoritmo, bem como suas melhorias implementadas.

No sexto capítulo, é fornecida, uma conclusão sobre todo o PFC e com relação aos trabalhos futuros e suas possíveis soluções.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é feita uma fundamentação teórica dos principais temas abordados ao decorrer do documento. Ressalta-se também, que esses temas foram estudados durante a etapa de revisão bibliográfica do PFC, como descrito em 1.3.

2.1 APRENDIZADO DE MÁQUINA

O aprendizado de máquina é uma área da inteligência artificial que tem como princípio, fornecer uma série de dados a um algoritmo ou modelo, que através de técnicas estatísticas, tenta aprender a interpretar os dados para predizer um valor cada vez mais preciso. A palavra aprendizado vem do fato de que o algoritmo começa com uma predição ruim, justamente por ainda não *conhecer* o conjunto de dados, e iterativamente vai *aprendendo* sobre o conjunto de dados e melhorando sua predição através do ajuste de parâmetros, buscando sempre se aproximar ao máximo do *ground truth*.

2.2 APRENDIZADO SUPERVISIONADO

O aprendizado supervisionado, é uma categoria do aprendizado de máquina, onde os modelos recebem um conjunto de dados previamente rotulados e, ao passar por uma etapa de treinamento, tentam aprender, através de um *supervisionamento* dos dados rotulados. Este *supervisionamento*, ocorre iterativamente até que o modelo possa predizer o rótulo de uma dada entrada com precisão.

Os problemas tratados pelos algoritmos de aprendizado supervisionado, podem ser divididos em duas grandes categorias:

- **Problemas de regressão:** São os problemas que assumem valores contínuos, quando se deseja, por exemplo, aprender o padrão de uma dada curva de uma função ou de um conjunto de pontos.
- **Problemas de classificação:** São os problemas relacionados a classificação dos elementos em diferentes categorias, por exemplo, a classificação de um superpixel como sendo de uma classe ou de outra.

O caso do classificador de superpixeis, se encaixa na categoria de classificação do aprendizado supervisionado. Na seção 4.3, são utilizados os seguintes modelos de aprendizado supervisionado:

- **Support Vector Machine (SVM)** (CRAMMER; SINGER, 2002).
- **Random Forest (RF)** (BREIMAN, 2001).

- **Extra Trees (ET)** (GEURTS; ERNST; WEHENKEL, 2006).
- **Multi-Layer Perceprton (MLP)** (PEDREGOSA *et al.*, 2011).
- ***XGBoost*** (CHEN; GUESTRIN, 2016).

Contudo, como estes métodos foram utilizados, neste documento, apenas a título de comparação com o método principal (redes neurais convolucionais), suas fundamentações não serão elucidadas.

2.3 MÉTODO SLIC

Um superpixel é um conjunto de pixeis que possuem características em comum (i.e. posição espacial na imagem e cor). O número de superpixeis de uma dada imagem representa em quantos segmentos esta será dividida, com o valor mínimo de um (toda a imagem da mesma cor) aumentando até o número de pixeis da imagem (neste valor, a imagem segmentada se torna idêntica a original). A técnica de segmentação via superpixel vem aparecendo cada vez mais em trabalhos acadêmicos e em algoritmos de VC como detecção de objeto, segmentação de imagens, reconhecimento de imagens, esqueletização de imagens, entre outros. Um dos métodos de segmentação via superpixel mais famosos e utilizados, é o método Simple Linear Iterative Clustering (SLIC).

O método de segmentação SLIC, recebe como parâmetro o número desejado de superpixeis (representado por K) e, com base no número de pixeis da imagem (representado por N), cria aproximadamente K pontos ao longo de toda a imagem, todos igualmente espaçados a uma distância de $S = \sqrt{N/K}$ pixeis. Cada um destes pontos é o centro de um superpixel e, neste primeiro momento, cada superpixel é representado por um retângulo, portanto, o centro do superpixel coincide com o centro geométrico do respectivo retângulo. Dessa forma a área de cada um dos superpixeis é aproximadamente igual a N/K pixeis, ou seja, S^2 pixeis.

Além da informação espacial do centro de cada superpixel, o método SLIC leva em consideração também o espaço de cores CIELAB[1] de cada píxel. O espaço de cores CIELAB, foi desenvolvido para representar a percepção humana das cores. Assim como o RGB ou HSV, o CIELAB é um espaço de cores definido por três canais, o canal l, representando a percepção luminosa, o canal a, representando a percepção do verde ao vermelho e o canal b, representando a percepção do azul ao amarelo.

Assim sendo, o centro de cada superpixel depende não só de sua posição espacial mas também de sua cor e pode ser definido da seguinte maneira:

[1] O espaço CIELAB é utilizado aqui, pois, para cálculos de distâncias euclidianas de cores, retorna valores perceptivelmente significativos, mesmo para pequenas distâncias. Ou seja, distâncias euclidianas no espaço CIELAB, retornam valores mais próximos da percepção humana de cores quando comparados a outros espaços como Hue Saturation Value (HSV) e Red Green Blue (RGB).

$$C_k = [l_k,\ a_k,\ b_k,\ x_k,\ y_k]^T \tag{1}$$

com, l, a e b sendo os valores CIELAB do píxel central do superpixel, x e y a localização espacial do centro do superpixel na imagem (em pixeis) e k variando dentre o intervalo $[1, K]$, com seus centros distando S pixeis um do outro.

Como próximo passo, o algoritmo move o centro C_k de cada superpixel dentre uma vizinhança quadrada de nxn pixeis (comumente $n = 3$), até a posição que tenha o mais baixo gradiente (menor valor para $G(x, y)$, fórmula 2). A fórmula do gradiente, para um único píxel, é dada pela seguinte expressão:

$$G(x, y) = \| \mathbf{I}(x+1, y) - \mathbf{I}(x-1, y) \|^2 + \| \mathbf{I}(x, y+1) - \mathbf{I}(x, y-1) \|^2 \tag{2}$$

com, $G(x, y)$ sendo o gradiente do píxel presente na posição x e y, $\mathbf{I}(x, y)$ sendo o vetor no espaço CIELAB do píxel presente na posição x e y e $||.||$ a norma L_2. É importante lembrar que x e y são localizados no centro geométrico do superpixel.

Assim, o algoritmo consegue medir qual dos n^2 pixeis tem o menor $G(x, y)$ e então, utiliza-lo como novo centro do superpixel. Este procedimento é realizado para evitar que o centro do superpixel seja escolhido em alguma borda (i.e. posições da imagem que representem uma transição brusca de cores) ou em algum píxel ruidoso.

Uma vez com os centros C_k de cada superpixel definidos, o algoritmo calcula a distância D_s (baseada, tanto na localização, quanto na cor do píxel) do centro C_k de cada um dos superpixeis com relação aos outros pixeis presentes em uma vizinhança quadrada (também com centro em C_k) de tamanho $2Sx2S$ pixeis. Como a área de cada superpixel é aproximadamente S^2, pixeis localizados fora da vizinhança quadrada, teriam sua componente da distância espacial de D_s sendo praticamente desprezada, portanto, não seriam desejados. Além disso, essa limitação também diminui consideravelmente o tempo de execução do algoritmo.

Visto que a área da região de busca ($4S^2$) é maior que a área média de cada superpixel (S^2), cada píxel tem sua distância calculada com relação a mais de um centro de superpixel e é com o superpixel que possuir a menor dessas distâncias que o método atribui o píxel, gerando assim novos agrupamentos de pixeis (i.e. novos superpixeis, com diferentes formatos).

As componentes de cor e espaço de D_s são calculadas através das distâncias euclidianas tanto com relação ao espaço de cores CIELAB (d_{lab}) quanto com relação à localização espacial (d_{xy}). Ambas as fórmulas são apresentadas em 3 e 4:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{3}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{4}$$

onde, o subscrito k representa o pixel do centro do superpixel k e o subscrito i representa um pixel qualquer da região quadrada $2Sx2S$ em torno de C_k.

As distâncias d_{lab} e d_{xy} são calculadas separadas, pois, ao invés de ser feita a simples norma euclidiana no espaço 5D, os autores do método propõem a seguinte fórmula para a distância D_s:

$$D_s = d_{xy} + \frac{m}{S} d_{lab} \tag{5}$$

onde, m é uma variável com a finalidade de ditar o peso da distância d_{lab} na equação e varia ente $[1, 20]$. Isto é, quanto maior o valor de m, mais os superpixeis serão influenciados pelas cores dos pixeis da imagem e quanto menor o valor de m mais os superpixeis serão influenciados pela distância espacial dos pixeis na imagem.

Assim que todos os superpixeis tenham seus formatos recalculados, um novo centro C_k é calculado a partir do vetor *labxy* médio com relação a todos os pixeis presentes no agrupamento. Então, com este novo centro, novas distâncias D_s são calculadas dando, mais uma vez, inicio ao mesmo procedimento de associação dos pixeis com os centros C_k próximos e consequentemente recalculando os próximos valores para C_k. Este procedimento é repetido um número pré-estabelecido de vezes ou até que o erro residual[2] entre o centro C_k antigo e o atual seja menor que um limiar estabelecido.

Dependendo da distribuição de cores da imagem, raramente, alguns superpixeis são formados por espaços desconexos e, por este motivo, o último passo do algoritmo é o de reforçar a conectividade dos superpixeis. Isso é feito atribuindo as regiões desconexas dos superpixeis ao superpixel vizinho de maior área.

2.4 MATRIZ DE CONFUSÃO

Encontrar uma boa métrica de avaliação é umas partes mais importantes na construção de um algoritmo de AM. Dado que, com uma métrica inadequada, um modelo ruim pode ser escolhido ou até pode levar a uma errada análise dos fatos. Neste PFC, a métrica de avaliação escolhida foi a matriz de confusão. A matriz de confusão é uma matriz que relaciona as classes preditas com as classes do *ground truth* (neste caso, manualmente rotuladas).

A matriz de confusão tem sua configuração descrita na Tabela 1:
onde, n_{ij} é o número de superpixeis que esperava-se estar na classe i e foram classificados como a classe j, 1 representa a *Classe 1*, 2 representa a *Classe 2* e 3 representa a *Classe 3*. Para auxiliar a explicação, a palavra *Reais* marca que, as linhas da matriz de confusão, representam as classes reais (*ground truth*) do conjunto de dados e a palavra *Preditos* marca que, as colunas, representam as classes preditas pelo modelo.

[2] O erro residual é calculado a partir da norma L_1 entre o centro C_k antigo e o atual.

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	n_{11}	n_{12}	n_{13}
	Classe 2	n_{21}	n_{22}	n_{23}
	Classe 3	n_{31}	n_{32}	n_{33}

Tabela 1 – Exemplo ilustrativo de uma matriz de confusão

Obs.: Vale a pena lembrar que, além das informações específicas de cada classe, na matriz de confusão, também é possível obter a precisão geral do modelo (ou acurácia), através da divisão entre a soma da diagonal principal (número total de superpixeis classificados corretamente) e o número total de elementos (número total de superpixeis).

O lado bom da matriz de confusão, para este caso, é que se tem apenas três classes (*Classe 1*, *Classe 2* e *Classe 3*) e, por este motivo, é fácil e rápido de verificar quais classes estão sendo classificadas incorretamente. Fato o qual é escondido em métricas de avaliação mais tradicionais, como a acurácia. Ainda, é importante ressaltar que, uma maneira fácil e rápida de verificar os elementos corretamente classificados, é olhar para a diagonal principal da matriz de confusão.

2.5 REDES NEURAIS CONVOLUCIONAIS

Hoje em dia, tanto no âmbito acadêmico como industrial, uma das abordagens mais utilizadas, para desafios de VC, são as CNNs. A CNN é uma técnica baseada em aprendizado profundo que utiliza o *array* multidimensional[3] de uma imagem como entrada, em seguida, a partir de um conjunto de camadas, extrai diferentes *features* e, com elas, pode classificar uma dada imagem. Para uma compreensão em maiores detalhes, uma breve descrição sera apresentada.

A CNN recebe o *array* tridimensional de uma imagem, de dimensões iguais a *HxWxC*, onde *H* é a altura da imagem, *W* é a largura e *C* é o número de canais de cores (ou profundidade[4]). Ao entrar na CNN, o *array* passa por uma série de camadas (arranjadas nas mais diversas arquiteturas) até chegar a sua classificação. Essas camadas podem ser separadas em três principais grupos:

- Camadas de Convolução
- Camadas de Subamostragem (ou *max pooling*, em inglês)

[3] Na prática, comumente, o *array* da imagem de entrada é de 4 dimensões, sendo a primeira o número de imagens por lote (por iteração), a segunda a altura da imagem, a terceira a largura e a quarta o número de canais de cores. Porém, neste documento, para simplificar, serão utilizadas apenas as 3 últimas dimensões: altura, largura e número de canais de cores da imagem.

[4] Neste documento, durante a contextualização sobre as CNNs, será utilizada a palavra profundidade para se referir a terceira dimensão do *array* tridimensional.

- Camadas de Rede Neural Totalmente Conectada (ou *fully connected layers*, em inglês)

2.5.1 Camadas de Convolução

Antes de entender as camadas de convolução, é preciso entender a operação de convolução[5], uma das operações mais importantes e a base das camadas de convolução. A convolução entre uma matriz M, de dimensões nxm, e um filtro F de dimensões fxf é representada pela seguinte fórmula:

$$G(u,v) = \sum_{i=1}^{f} \sum_{j=1}^{f} F(i,j)M(u+i-1, v+j-1) \tag{6}$$

onde, G é a matriz resultante da convolução, u pertence ao intervalo $[1,\ n-f+1]$ e v pertence ao intervalo $[1,\ m-f+1]$. Ou seja, G tem sempre as dimensões $(n-f+1)\ x\ (m-f+1)$. A matriz resultante G também é chamada de mapa de *features*.

Como visto em 6, cada elemento da matriz G é obtido da soma de todos os elementos da multiplicação, elemento a elemento, do filtro F com a matriz M, para uma dada posição (rede localmente conectada). Para exemplificar, na Figura 1, assume-se uma matriz $5x5$ $(n = 5,\ m = 5)$ e um filtro $3x3$ $(f = 3)$ portanto, o mapa de *features* terá dimensão $3x3$ $(n-f+1 = 3,\ m-f+1 = 3)$.

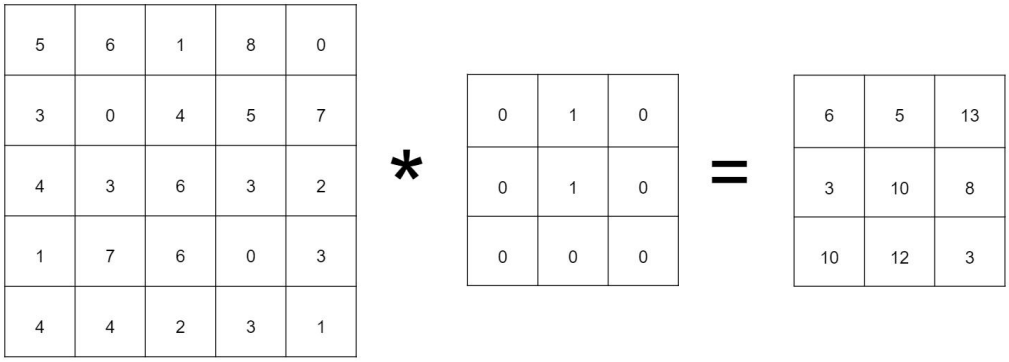

Figura 1 – Exemplo ilustrativo de uma operação de convolução em duas dimensões.

Uma analogia que pode ser feita aqui é que o filtro F *desliza* ao longo da matriz M, percorrendo todas as posições possíveis (sem que uma linha ou coluna fique para *fora* da matriz).

A operação de convolução pode ser realizada, não só por *arrays* bidimensionais (matrizes), mas também por *arrays* tridimensionais. Neste caso, obrigatoriamente, a

[5] Não só neste documento, mas também na maioria das literaturas de VC, a operação chamada de convolução refere-se a correlação cruzada da literatura do processamento de sinais. Portanto, a analogia deve ser feita com a correlação cruzada e não com a convolução da literatura do processamento de sinais.

profundidade (terceira dimensão) do filtro *F* deve ser igual à profundidade da matriz *M*. Assim, a equação do mapa de *features* é descrita por:

$$G(u,v) = \sum_{k=1}^{K} \sum_{i=1}^{f} \sum_{j=1}^{f} F(i,j,k) M(u+i-1, v+j-1, k) \tag{7}$$

onde *K* é a profundidade tanto de *F* quanto de *M* e *k* varia de [1, *K*].

A Figura 2, mostra a convolução de um *array* 5*x*5*x*3 (*n* = 5, *m* = 5 e profundidade = 3) e um filtro 3*x*3*x*3 (*f* = 3 e profundidade = 3) resultando no mapa de *features*, de dimensão 3*x*3 ($n - f + 1 = 3$, $m - f + 1 = 3$). Nota-se que, a dimensão de *G* não depende da profundidade de *M* e *F*, o mapa de *features* é sempre uma matriz (*array* bidimensional).

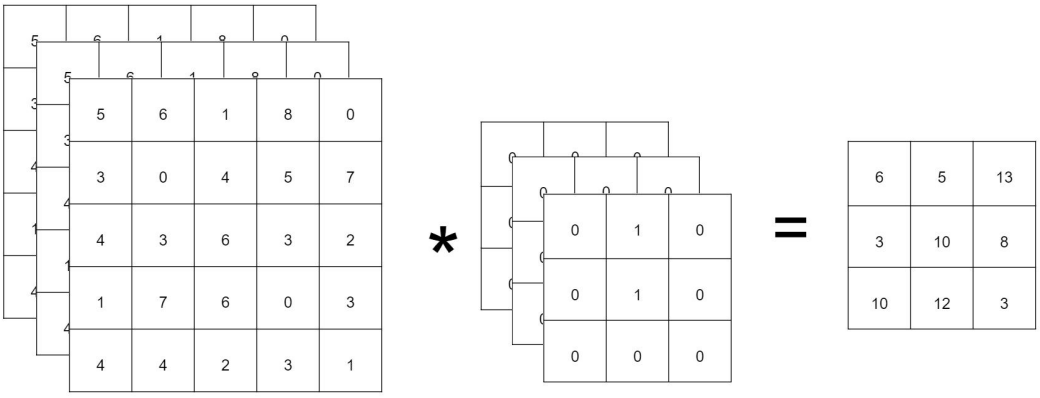

Figura 2 – Exemplo ilustrativo de uma operação de convolução em três dimensões.

Em uma camada de convolução, um mesmo *array M* é convolucionado com diversos filtros (F_1, F_2, ...) de mesma profundidade (i.g. diversas vezes a operação da Figura 2). Portanto, para um dado número de filtros n_f, essa operação resulta em um conjunto de n_f mapas de *features*. Assim, o *array* bidimensional, da Figura 2 (apenas um mapa de *features*), se torna um *array* tridimensional, cuja profundidade é justamente n_f, resultando na dimensão $(n - f + 1)\ x\ (m - f + 1)\ x\ n_f$.

Obs.: Em uma CNN, os *arrays M* são imagens ou mapas de *features*, logo, seus elementos, são os pixeis da respectiva imagem ou mapa de *features*.

Assim, a dimensão e o número de filtros da camada de convolução se tornam dois dos seus principais hiperparâmetros. As dimensões mais utilizadas, na literatura, para os filtros são: 3*x*3, 5*x*5 e 7*x*7. Já para o número de filtros costuma de utilizar um valor cada vez maior conforme a profundidade[6] da CNN vá aumentando.

Tão importante quanto a dimensão e o número de filtros, é saber quais características da imagem se deseja capturar e qual arranjo dos valores do filtro resultará

[6] Nota-se que aqui, a palavra profundidade se refere ao número de camadas da CNN e não a terceira dimensão de uma determinada camada.

na captura de tais características. Por exemplo, para detectar bordas verticais em uma certa imagem, uma opção é utilizar o filtro Sobel detector de bordas verticais que consiste na seguinte configuração:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{8}$$

Para ilustrar, o filtro Sobel detector de bordas verticais é aplicado em uma certa imagem na Figura 3. Vale a pena destacar que, como é utilizada uma imagem com três canais de profundidade (RGB), o filtro tem o mesmo arranjo de valores da equação 8 para seus três canais.

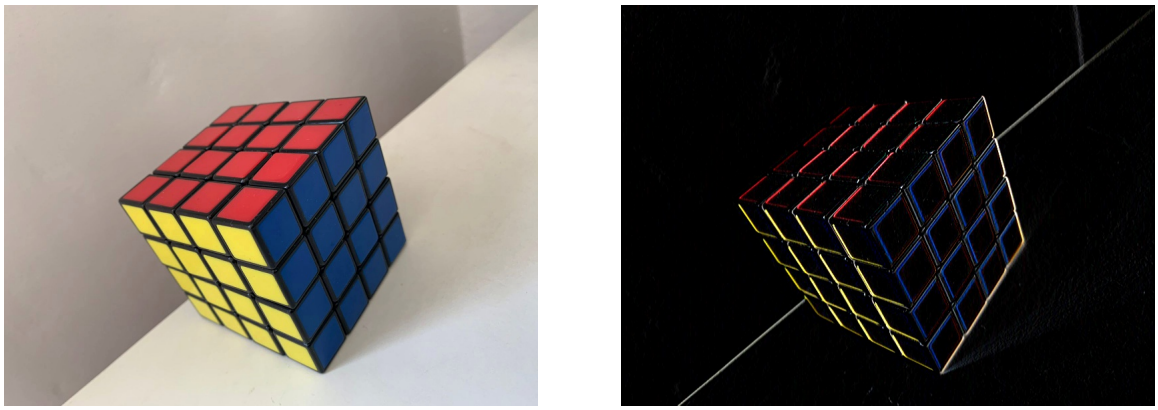

Figura 3 – Efeito do filtro Sobel detector de bordas verticais para uma dada imagem.

No entanto, é necessário saber, não só qual arranjo dos valores dos filtros utilizar, mas também qual é a característica mais adequada a ser identificada na imagem (i.g. para detecção de objetos) é uma tarefa desafiadora. A solução que a CNN traz é justamente, tratar os valores do filtro como parâmetros e utilizar técnicas de AM para que a CNN aprenda (i.g. por retro-propagação com descenso do gradiente), quais características são mais interessantes para definir um dado objeto e quais valores dos filtros capturariam tais características.

Assim como nas redes neurais tradicionais, uma não linearidade[7] (i.g. tangente hiperbólica, função sigmoide, ReLU) é aplicada. Nas CNNs, a não linearidade é aplicada na soma de cada um dos parâmetros do filtro com um *bias* (mesmo valor de *bias* para todos os parâmetros do respectivo filtro). Na Figura 4, têm-se os mapas de *features*, obtidos da convolução dos filtros aprendidos por uma CNN treinada em reconhecimento facial (LEE *et al.*, 2009).

[7] Muitas literaturas trazem a camada não linear separada da camada de convolução. Isso porque, não necessariamente é aplicada uma não linearidade ao final de uma camada de convolução. Neste caso, é utilizada uma função de ativação linear.

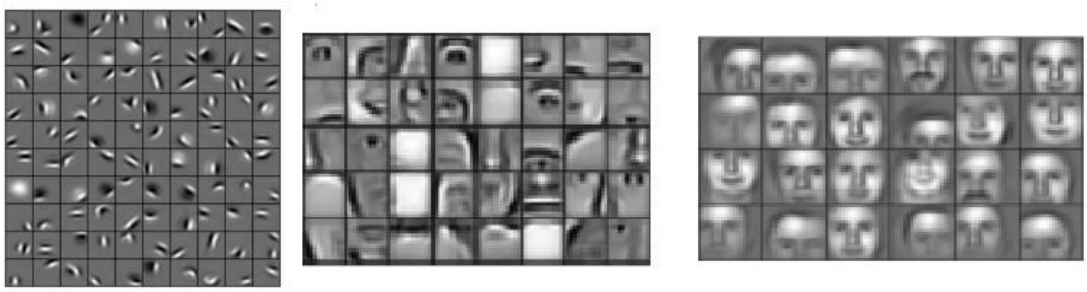

Figura 4 – Três diferentes conjuntos de mapas de *features* de uma CNN de reconhecimento facial (LEE *et al.*, 2009)

Obs.: Devido aos mesmos valores dos filtros, serem utilizados para todas as posições de altura e largura da imagem (compartilhamento de parâmetros), uma mesma característica da imagem, pode ser capturada mesmo estando em diferentes posições.

Um filtro tem então, $f * f * n_c + 1$ parâmetros sendo, $f * f * n_c$ parâmetros do filtro (onde, n_c é a profundidade do filtro) e um *bias*. No caso de um conjunto de filtros, o número de parâmetros aumenta para $(f * f * n_c + 1) * n_f$.

Por fim, é de suma importância falar ainda de dois outros hiperparâmetros da camada de convolução: o preenchimento da imagem e o passo de convolução.

O preenchimento da imagem é a adição de uma camada de zeros, na borda da imagem. Simbolizado pela letra p, o preenchimento pode assumir qualquer valor inteiro maior ou igual a zero, contudo, na prática, costuma assumir valores pequenos. Nos exemplos das imagens 1 e 2, o preenchimento da imagem não foi utilizado ($p = 0$).

Com a utilização do preenchimento, é possível obter um mapa de *features* que tenha as mesmas dimensões do *array* de entrada. Caso as dimensões não se mantenham, após sequenciais camadas de convolução, algumas importantes informações podem ser perdidas. Ademais, caso não seja utilizado preenchimento, os pixeis da borda da imagem serão muito menos utilizados que os demais e informações importantes presentes neles podem não estar sendo consideradas. Dito isso, o preenchimento muda a dimensão do *array* de saída para $(n + 2p - f + 1)\ x\ (m + 2p - f + 1)\ x\ n_f$

O passo de convolução nada mais é que, de quantos em quantos pixeis o filtro irá se movimentar, ao longo da imagem de entrada, para calcular o mapa de *features*. Representado pela letra s, o passo é um número inteiro maior ou igual a um, mas também costuma assumir valores pequenos. Nas explicações feitas até aqui, foi utilizado $s = 1$. O passo de convolução muda a dimensão do mapa de *features* para $(\frac{n+2p-f}{s} + 1)\ x\ (\frac{m+2p-f}{s} + 1)\ x\ n_f$.

Obs.: Nota-se que, para manter as dimensões do mapa de *features* como números inteiros, se utiliza a expressão $\lfloor \frac{n+2p-f}{s} + 1 \rfloor\ x\ \lfloor \frac{m+2p-f}{s} + 1 \rfloor\ x\ n_f$. Onde, $\lfloor . \rfloor$ é a função chão. Assim, fica garantido que todos os elementos do filtro (caso devidamente dimensionado) sempre irão multiplicar algum píxel da imagem, durante a convolução.

2.5.2 Camadas de Subamostragem

Por mais que seja possível construir uma CNN com apenas camadas de convolução, ao longo dos anos, foi evidenciado que resultados ainda melhores eram obtidos quando adicionadas camadas de subamostragem e de redes neurais totalmente conectadas.

Assim como as camadas de convolução, as camadas de subamostragem também aplicam um filtro no *array* de entrada resultando em um *array* de saída. A grande diferença é na operação em si (não mais utilizada a convolução) e nas dimensões do filtro e do *array* de saída. As camadas de subamostragem tem como principal objetivo, reduzir as dimensões do *array* de entrada para que assim, a próxima camada, tenha menos parâmetros a serem treinados. Além de controlar o *overfitting*, essa redução da imagem tem como finalidade, reduzir a complexidade e custo computacional da CNN e, ao mesmo tempo, focando em minimizar a perda de informações.

Os filtros da camada de subamostragem, tem dimensão igual a $f \, x \, f$, com apenas um canal de profundidade (bidimensional). Semelhante às camadas de convolução, o filtro é aplicado ao longo de todo o *array* de entrada (com um dado preenchimento p e passo s), porém, como o filtro tem apenas duas dimensões, o mesmo filtro é aplicado para cada uma das camadas de profundidade do *array* de entrada. Consequentemente, um *array* de dimensão $n \, x \, m \, x \, n_c$ resulta em um *array* de saída com dimensão igual a $\lfloor \frac{n+2p-f}{s} + 1 \rfloor \, x \, \lfloor \frac{m+2p-f}{s} + 1 \rfloor \, x \, n_c$.

Em cada uma das posições do filtro, os elementos do *array* de entrada, que estiverem limitados por este filtro, são submetidos a uma operação de subamostragem. Essas operações podem ser diversas, porém, as mais utilizadas são as subamostragens de valor máximo (calcula o maior valor da região) e de valor médio (calcula a média da região). Um exemplo é fornecido na Figura 5, com uma subamostragem de valor máximo e com um filtro 2x2 com $s = 2$ e $p = 0$.

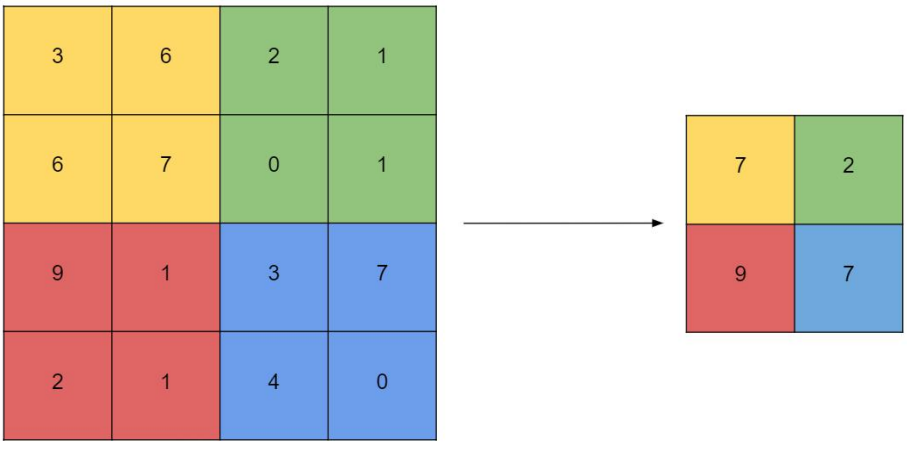

Figura 5 – Exemplo ilustrativo de uma operação de subamostragem.

Cada uma das quatro cores da Figura 5 representa cada uma das posições do filtro no *array* de entrada e seu respectivo valor máximo, no *array* de saída.

Hoje em dia, a operação de subamostragem mais utilizada é a de valor máximo e sem preenchimento ($p = 0$) pois, na prática, é a que apresenta os melhores resultados. Ademais, a grande maioria dos filtros são de dimensões 2 x 2 ou 3 x 3, com passo $s = 2$ pois, subamostragens aplicadas com um maior passo ou dimensão, tem grandes chances de causar perda de informação da imagem.

Uma importante característica das camadas de subamostragem é que elas não adicionam nenhum parâmetro a ser aprendido pelo AM, elas apenas reduzem o número de parâmetros na próxima camada com relação à camada anterior.

2.5.3 Camadas de Rede Neural Totalmente Conectada

As camadas de rede neural totalmente conectadas são as conhecidas camadas ocultas das redes neurais tradicionais, cujos neurônios são conectados com todas as funções de ativações dos neurônios da camada anterior. Cada ligação, entre neurônios de camadas adjacentes (incluindo o *bias*), tem um parâmetro que será também aprendido durante a retro-propagação, totalizando assim $N[l-1] * N[l] + 1$ parâmetros. Onde $N[l]$ é o número de neurônios da respectiva camada e $N[l-1]$ o número de neurônios da camada anterior. O número de neurônios da camada de rede neural totalmente conectada pode ser então considerado seu mais importante (e muitas vezes único) hiperparâmetro.

2.5.4 Arquitetura de uma CNN

Por mais criativas que as arquiteturas das CNNs possam ser, geralmente, acabam seguindo alguns padrões que, ao longo dos anos, tem se mostrado mais efetivos para as dadas aplicações.

Um dos mais comuns padrões de uma CNN, é ter uma camada de convolução como sua primeira camada[8], transformando a imagem de entrada em uma série de mapas de *features*.

Normalmente, após uma camada de convolução vem uma camada de subamostragem ou outra camada de convolução. No caso da camada de subamostragem, ela transforma o mapa de *features* em outro de mesma profundidade, porém de menor altura e largura. Esse padrão (camada de convolução seguida por camada de subamostragem ou de convolução) se repete uma série de vezes dependendo da arquitetura da CNN.

[8] Na literatura, frequentemente, o *array* tridimensional da imagem é chamado de camada de entrada e tratado como primeira camada da CNN. Contudo, neste documento, será como se fosse uma camada *zero* ou *inicial*.

Após essa série de camadas, o *array* tridimensional resultante (da última camada), é transformado em um vetor (*array* unidimensional), cujo comprimento é justamente, igual à multiplicação das dimensões do *array* tridimensional. Logo, o vetor obtido é o conjunto de neurônios, que representa a primeira camada de rede neural totalmente conectada da CNN. Essa camada, é conectada, ou não (dependendo da arquitetura da CNN), a uma sequência de outras camadas de rede neural totalmente conectada (preferencialmente, cada vez com menor número de neurônios).

Por fim, uma camada de classificação (i.g. *softmax*[9]) é adicionada como camada de saída, que resulta na classificação final da CNN.

Para ilustrar a explicação, na Figura 6, é apresentada a arquitetura de uma das primeiras e mais famosas CNNs, a *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2017). Nota-se os padrões mencionados: camada de convolução no início, camadas de convoluções seguidas por camadas de subamostragem ou outras de convoluções, camadas de rede totalmente conectadas ao final e camada de classificação como camada de saída da CNN.

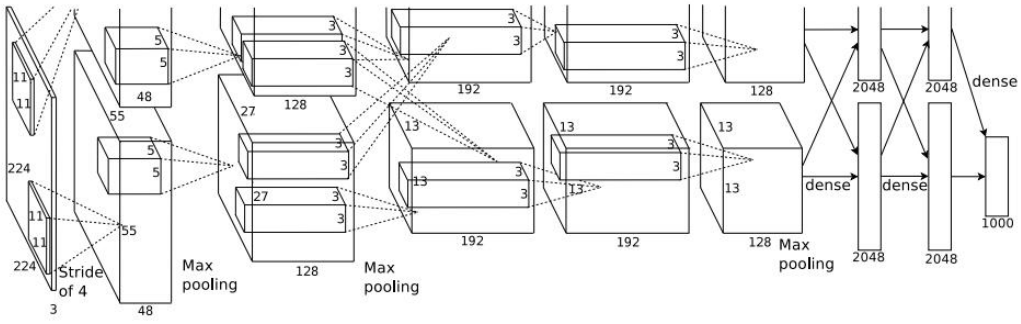

Figura 6 – Arquitetura da *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

[9] A *softmax* é uma função que, normaliza, uma série de K valores de entrada, em uma distribuição de probabilidades. Por meio da seguinte fórmula: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$.

3 SEGMENTAÇÃO DA REGIÃO DE INTERESSE

3.1 ANÁLISE DO CONJUNTO DE DADOS

Também com relação ao acordo de não divulgação, as imagens do conjunto de dados, bem como suas informações, foram retiradas deste documento, a versão não confidencial.

O primeiro passo, de VC e AM, a ser realizado é a seleção da região de interesse, através de uma segmentação semântica da imagem. Dessa forma, o sistema VC e AM, recebe a máscara da imagem (calculada pela segmentação semântica, em outro módulo), possibilitando que a região de interesse seja selecionada do resto da imagem (Figura 7).

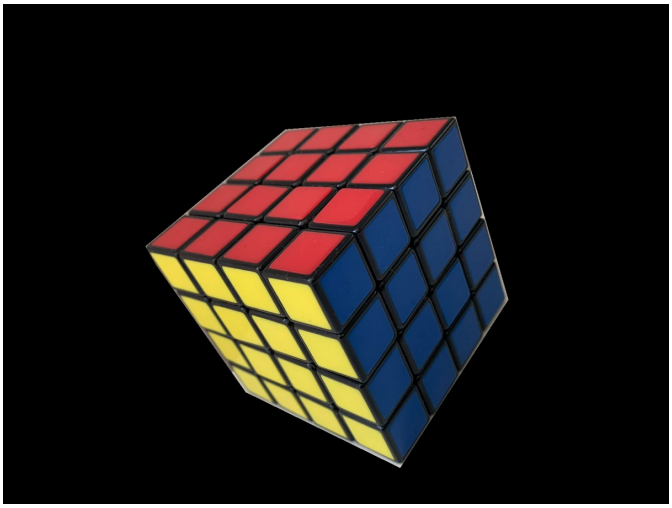

Figura 7 – Exemplo ilustrativo do resultado da segmentação semântica para uma dada imagem.

3.1.1 Análise do espectro HSV

Uma análise mais detalhada das imagens é necessária, para que assim, a mais adequada técnica seja escolhida. Para tanto, a relação de cores entre os elementos de cada uma das imagens foi estudada através da análise dos seus respectivos canais de cores (i.g. HSV e RGB).

Como o sistema estudado tem apenas três classes, primeiramente, foi tentado separar as classes apenas aplicando filtros em seus canais HSV e RGB. Para tanto, foi desenvolvido um framework, criado justamente para visualizar o efeito de diferentes filtros de maneira fácil e prática.

Este processo foi feito com diversas imagens, mas nenhuma conseguiu uma boa separação entre as classes.

Este resultado é explicado pelo fato de que, ambas as classes tem espectros de cores muito semelhantes (impedindo a seleção de apenas uma das classes pelo filtro) e a imagem possui uma alta variabilidade de brilho (impedindo com que todos os elementos de uma classe sejam selecionados pelo mesmo filtro).

Mesmo não obtendo uma separação bem sucedida, esta análise foi fundamental para a escolha da próxima abordagem a ser utilizada, a segmentação via Superpixel. Posto que, foi dela que foi tirada a ideia de realizar uma segmentação para separar as classes e, posteriormente, utilizar seus histogramas para a classificação.

3.2 SEGMENTAÇÃO VIA SUPERPIXEL

Acredita-se que a técnica de segmentação via superpixel, aliada a uma posterior classificação, seja uma técnica adequada para este caso. Uma vez que, técnicas mais convencionais, como a detecção de objetos ou segmentação semântica, teriam dificuldade dada a alta variabilidade tanto de iluminação quanto da posição dos objetos na imagem.

3.2.1 Métodos de segmentação via Superpixel

Agora precisa-se de um método de segmentação que gere os superpixeis de maneira rápida e eficiente. Em um primeiro momento, foram testados quatro métodos:

- **Método de Felzenszwalb** (FELZENSZWALB; HUTTENLOCHER, 2004).
- **SLIC** (ACHANTA *et al.*, 2012).
- **Quickshift** (VEDALDI; SOATTO, 2008).
- **Compact Watershed** (NEUBERT; PROTZEL, 2014).

Para que a separação entre as classes seja bem sucedida, idealmente, cada superpixel deve apresentar apenas uma das classes em seu interior. Todavia, na prática, muitos superpixeis contém mais de uma classe e encontrar uma boa segmentação se torna uma tarefa árdua. Dentre os métodos estudados, foi escolhido, empiricamente, aquele com o menor número de superpixeis contendo ambas as classes ao mesmo tempo, o método SLIC.

3.2.2 Aplicação do método estudado

Com os superpixeis segmentados, tem-se uma separação (de posição espacial e cor) dos elementos da imagem. Todavia, para ocorrer uma boa separação de classes, cada superpixel deve conter apenas uma das classes em seu interior. A mais eficiente

maneira, para garantir que isso aconteça, é aumentar o número de superpixeis (hiperparâmetro de entrada do algoritmo), uma vez que, um maior número de segmentos, resulta em uma segmentação mais precisa e tem uma menor probabilidade de gerar superpixeis com superposição de ambas classes. Por outro lado, o número de superpixeis não pode ser muito grande, pois aumenta não só o tempo de processamento e custo computacional do algoritmo como também o processo de rotulação. Assim, achar um bom *trade-off* entre essas duas situações se torna uma difícil tarefa.

Como o algoritmo recebe, do módulo anterior, além da imagem, sua respectiva máscara (Figura 7), notou-se que muitos dos segmentos formados (imagem da esquerda da Figura 8) eram apenas pixeis pretos (parte da imagem fora da região de interesse).

Felizmente, a Application Programming Interface (API) Sci-kit Image (WALT *et al.*, 2014), fornece uma excelente implementação do método SLIC que conta inclusive com uma ferramenta adicional, a máscara de fundo. Com a máscara de fundo, é possível alcançar uma mesma precisão com um menor número de superpixeis, uma vez que a região de fora da máscara (fundo) não é segmentada (imagem da direita da Figura 8).

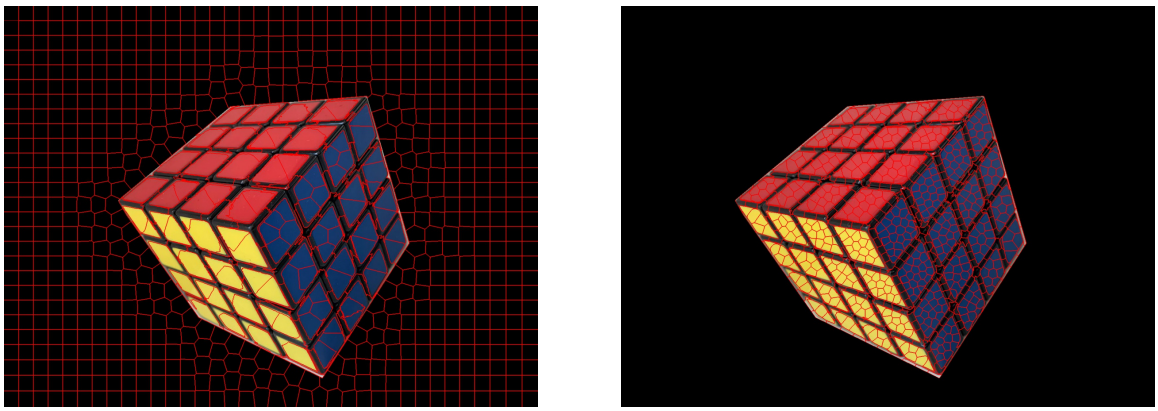

Figura 8 – Exemplo ilustrativo para a segmentação SLIC com 1000 superpixeis por imagem sem a máscara de fundo (esquerda) e com a máscara de fundo (direita).

Agora, com a máscara de fundo, uma maior precisão é alcançada com um menor número de segmentos, facilitando a tarefa de escolha do número de superpixeis. Analisando a segmentação de diversas imagens para diferentes valores de número de segmentos, empiricamente, foi escolhido trabalhar com três diferentes casos, baseados em três diferentes tamanhos para a área da região de interesse:

- **Segmentar a imagem em 100 segmentos (tamanho pequeno):** Imagens cujas áreas das máscaras são de 2 a 15 % da área total da imagem.

- **Segmentar a imagem em 200 segmentos (tamanho médio):** Imagens cujas áreas das máscaras são de 15 a 25 % da área total da imagem.
- **Segmentar a imagem em 300 segmentos (tamanho grande):** Imagens cujas áreas das máscaras estão acima de 25 % da área total da imagem.

Como a imagem tem uma altíssima variabilidade, tanto da posição, quanto da proporção de seus elementos ao longo da região de interesse, é extremamente difícil avaliar a precisão do método utilizado. Consequentemente, a métrica de avaliação aqui utilizada, foi a avaliação empírica das imagens, sempre visando avaliar positivamente as imagens com poucos superpixeis contendo duas ou mais classes e negativamente as imagens com muitos.
Obs.: Uma opção, que poderia ser implementada como métrica, é a contagem (manual) do número de superpixeis contendo ambas as classes na segmentação da imagem.

Ao fim da segmentação, cada superpixel é recortado e separado em uma imagem, de largura, altura e canais de cores iguais às próprias largura, altura e canais de cores do respectivo superpixel.

Uma vez separados (cada um designado a uma imagem diferente), os superpixeis estão prontos para passar pelo processo de classificação, designando-os para suas respectivas classes.

4 CLASSIFICAÇÃO DOS SUPERPIXEIS

Com a segmentação do método SLIC, é possível então, encontrar um *array* tridimensional para cada superpixel (i.e. imagem do superpixel recortado). A primeira dimensão do *array*, tem *W* elementos, a segunda *H* elementos e a terceira 3 elementos, com *W* e *H* sendo, a largura e a altura da imagem, respectivamente, e os 3 elementos da terceira dimensão sendo os canais RGB da imagem. Com isso, a intenção agora é fazer um classificador (baseado em técnicas de inteligência artificial), para classificar cada um dos superpixeis.

O classificador, receberá como entrada cada um dos superpixeis e os classificará como sendo da classe 1, 2 ou 3.

4.1 EXTRAÇÃO DE *FEATURE*

É conhecido que, cada imagem, tem seu espectro de cores RGB (ou qualquer outro modelo de cores) que pode ser representado por um histograma, onde a incidência de cada píxel (eixo das ordenadas), para cada range de cores (eixo das abscissas), é traçada com seus canais separados ou juntos. Um exemplo é fornecido na Figura 9.

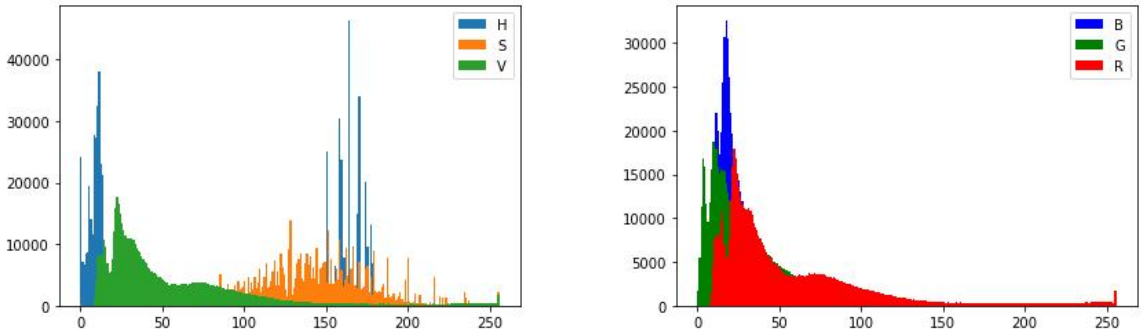

Figura 9 – Exemplo de histograma dos espectros HSV (esquerda) e RGB (direita), com seus canais separados, para uma dada imagem.

A ideia é então, utilizar os valores do histograma não para cada imagem, mas sim para cada superpixel de cada uma das imagens (i.e. imagem do superpixel recortado). Dessa forma, um algoritmo de AM recebe um vetor representando o histograma do superpixel e o classifica dentre as classes 1, 2 e 3. Este processo é repetido para todos os superpixeis da imagem.

Utilizando o histograma com os canais de cores juntos[1], cada superpixel pode ser representado por um *array* unidimensional de 256 elementos, sendo cada elemento, a incidência do respectivo espectro de cor na imagem (i.e. eixo das abscissas do

[1] Foi optado por trabalhar com os canais de cores juntos, pois separados gerariam três vezes mais elementos no *array* e como o número de superpixeis, por imagem, é alto (de 100 a 300 superpixeis) o número de elementos gerados seria extremamente alto.

histograma de cores). Esses 256 elementos, seriam as *features* (características) do conjunto de dados, descritas abaixo:

$$\begin{aligned} X_1 &= [x_{1,1};\ x_{1,2}; x_{1,3};\ \cdots\ ;\ x_{1,255}] \\ X_2 &= [x_{2,1};\ x_{2,2}; x_{2,3};\ \cdots\ ;\ x_{2,255}] \\ &\vdots \\ X_n &= [x_{n,1};\ x_{n,2}; x_{n,3};\ \cdots\ ;\ x_{n,255}] \end{aligned} \tag{9}$$

onde, n é o número de superpixeis da imagem, X_i é o i-ésimo superpixel da imagem e $x_{i,j}$ é a j-ésima *feature* do superpixel X_i.

Obs.: É importante salientar que, $x_{n,0}$ foi removido da equação 9 pois (após o método SLIC), os superpixeis são separados em imagens de fundo preto e os pixeis com o valor 0 (i.e. $x_{n,0}$) seriam justamente o fundo da imagem (pixeis completamente pretos). Isso não danifica a imagem, dado que nenhum superpixel (não completamente preto) tem um píxel completamente preto em seu interior.

Cada X_n da equação 9, será classificado dente três diferentes classes: classes 1, 2 e 3. Logo, com a aplicação de um classificador, obtém-se um *array* de saída y, que representa a classe de cada um dos superpixeis:

$$y = [y_1;\ y_2;\ y_3;\ \cdots\ ;\ y_n] \tag{10}$$

onde, n é o número de superpixeis da imagem.

Essa é uma abordagem de extração de *feature* manual, visto que as *features* são extraídas antes da classificação. Neste documento, a abordagem será comparada com outra, dessa vez, sem extração de *feature* manual.

4.2 PROCESSO DE ROTULAÇÃO

Agora, o *array* y precisa ser devidamente rotulado, seja para o treinamento do modelo, seja para verificar se o modelo esta atingindo as métricas de avaliação desejadas. Com tal finalidade, o autor deste documento, desenvolveu um *framework* de rotulação de superpixeis, que permite ao usuário (de forma automatizada e rápida) rotular cada um dos superpixeis do conjunto de dados, criando assim um novo conjunto de dados, dessa vez, composto por cada um dos superpixeis (rotulados) de cada uma das imagens.

No *framework*, notou-se a necessidade da adição de uma quarta classe, a classe *não identificado*, composta basicamente dos superpixeis que tem mais de uma das outras 3 classes em seu interior. Essa classe, pode ser utilizada sem problemas apenas na fase de rotulação (não enviada ao algoritmo de AM), pois, o que se quer é mostrar ao classificador (através de diferentes posições e iluminações dos objetos) o que é cada uma das classes. Assim, espera-se que o algoritmo (quando bem treinado)

possa inferir a qual classe o superpixel *não identificado* (o qual o ser humano não soube rotular) pertence.

Obs.: Novamente, com relação ao acordo de não divulgação, não serão mostrados os dados de cada uma das classes do conjunto de dados. Porém, evidenciou-se que, o conjunto de dados tem um pequeno desequilíbrio em suas classes, com a mais numerosa classe tendo três vezes mais elementos do que a menos numerosa.

Uma vez devidamente rotulado, para poder ser utilizado pelos algoritmos de AM, o conjunto de dados é dividido de forma estratificada em conjuntos de treinamento, teste e validação (70 % para o treinamento, 18 % para a validação e 12 % para o teste). O conjunto de treinamento é utilizado para treinar o modelo, o conjunto de validação é utilizado para avaliar o modelo ao fim de cada época de treinamento e o conjunto de teste é utilizado para testar o modelo, ao fim de todas as épocas de treinamento.

4.3 ALGORITMOS DE APRENDIZADO SUPERVISIONADO

Como se tem um rótulo para cada superpixel, um número definido de classes (três) e a facilidade de rotular as classes de cada elemento, a utilização de um algoritmo de aprendizado supervisionado se torna imprescindível.

Na fase de treinamento, o algoritmo, recebe o conjunto de *features* com suas classes pré-rotuladas. Uma vez treinado, o modelo pode ser utilizado para predizer as classes para uma dado conjunto de teste. As classes preditas são então comparadas com as manualmente rotuladas e avaliadas por uma dada métrica de avaliação.

4.3.1 Abordagens com extração de *feature* manual

A abordagem de extração de *feature* manual, é utilizada para alimentar os modelos utilizados em 4.3.1.1, 4.3.1.2 e 4.3.1.3.

4.3.1.1 Modelos com ajuste em seus hiperparâmetros

Primeiramente, os modelos de aprendizado supervisionado, foram testados com os hiperparâmetros padrão e, em um segundo momento, uma busca exaustiva[2] foi feita, com a finalidade de encontrar os melhores valores para seus hiperparâmetros. Assim, os hiperparâmetros padrão serviriam de base para uma comparação com os valores encontrados pela busca. Ao todo, cinco modelos de aprendizado supervisionado foram testados:

- **Support Vector Machine (SVM)**
- **Random Forest (RF)**

[2] A técnica de busca utilizada foi: *Grid search* (PEDREGOSA *et al.*, 2011) com validação cruzada.

- **Extra Trees (ET)**
- **Multi-Layer Perceprton (MLP)**
- ***XGBoost***

As matrizes de confusão, tanto para os hiperparâmetros padrão, quanto para os hiperparâmetros ajustados, são apresentadas na sequência:

- ET

		Preditos (padrão)			Preditos (ajustados)		
		Classe 1	Classe 2	Classe 3	Classe 1	Classe 2	Classe 3
Reais	Classe 1	40.42 %	1.60 %	57.99 %	41.37 %	1.60 %	57.03 %
	Classe 2	0.00 %	74.16 %	25.84 %	0.00 %	74.50 %	25.50 %
	Classe 3	5.54 %	9.73 %	84.73 %	5.29 %	9.58 %	85.13 %

Tabela 2 – Matriz de confusão para o modelo ET, com os parâmetros padrão (esquerda) e com os parâmetros ajustados (direita).

- MLP

		Preditos (padrão)			Preditos (ajustados)		
		Classe 1	Classe 2	Classe 3	Classe 1	Classe 2	Classe 3
Reais	Classe 1	33.71 %	3.04 %	63.26 %	47.28 %	0.96 %	51.76 %
	Classe 2	4.45 %	67.59 %	27.95 %	0.11 %	69.60 %	30.29 %
	Classe 3	10.58 %	12.23 %	77.20 %	10.08 %	10.38 %	79.54 %

Tabela 3 – Matriz de confusão para o modelo MLP, com os parâmetros padrão (esquerda) e com os parâmetros ajustados (direita).

- *XGBoost*

		Preditos (padrão)			Preditos (ajustados)		
		Classe 1	Classe 2	Classe 3	Classe 1	Classe 2	Classe 3
Reais	Classe 1	42.81 %	1.60 %	55.59 %	42.01 %	1.60 %	56.39 %
	Classe 2	0.22 %	74.72 %	25.06 %	0.00 %	74.05 %	25.95 %
	Classe 3	7.04 %	9.83 %	83.13 %	5.09 %	9.58 %	85.33 %

Tabela 4 – Matriz de confusão para o modelo *XGBoost*, com os parâmetros padrão (esquerda) e com os parâmetros ajustados (direita).

As colunas delimitadas por *Preditos (padrão)* e *Preditos (ajustados)*, representam os superpixeis pertencentes as classes preditas, pelo modelo com seus hiperparâmetros padrões e ajustados, respectivamente.
Obs.: Aqui, apenas os três métodos com melhor resultado foram apresentados, devido a questões de sigilo.

As matrizes de confusão, são fornecidas em porcentagem, com relação ao número total de elementos do *ground truth* de cada classe (ou seja, cada linha deve somar 100 %). Desta análise, pode-se extrair, tanto a precisão de cada classe (i.e. quantidade de superpixeis, de uma classe, corretamente classificados, dividido pelo total de superpixeis dessa mesma classe), quanto os erros de classificação, ou seja, todos os elementos da matriz de confusão que não estejam na diagonal principal.

Com os resultados das precisões de classe (elementos da diagonal principal das matrizes de confusão, que não são mostrados aqui por questões de sigilo), em uma primeira análise, é possível verificar que as precisões mais altas, para cada uma das classes, aconteceram cada uma em um modelo diferente. Isso demonstra que, por mais diferente que a abordagem seja, nenhuma consegue elevar muito as precisões de uma classe, sem diminuir em outra.

Ainda, é possível analisar que não houve uma melhoria expressiva (maior que 10 %) nas precisões de classes dos modelos, quando aplicado o ajuste em seus hiper parâmetros, pois, a maioria, tem como padrão hiperparâmetros já próximos aos ótimos, para este caso. O único modelo que teve uma melhoria considerável, foi o MLP, justamente por ser uma rede neural e então, fortemente sensível a mudanças em seus hiperparâmetros.

Além disso, nota-se, nas Tabelas 2, 4 e 3, que nenhum algoritmo teve problemas em confundir a classe 1 com a classe 2 (apenas 0.96 % e 0.11 % de erro de classificação, para o método MLP com parâmetros ajustados, por exemplo) e todos os algoritmos apresentaram boa precisão (acima de 70 %) para a classe 3 (79.54 % de precisão, para o método MLP com parâmetros ajustados, por exemplo). Por outro lado, as classes 1 e 2, tiveram muitos superpixeis confundidos com a classe 3 (principalmente, superpixeis das classes 1 e 2 classificados como 3; 51.76 % e 30.29 % de erro de classificação, para o método MLP com parâmetros ajustados, por exemplo).

Essa confusão, das classes 1 e 2 com a classe 3, ocorre devido a três principais fatores:

- Desequilíbrio das classes (classe 3 representa 60 % do conjunto de dados).
- Baixa qualidade das imagens.
- Método de classificação/extração de *features* não é o mais adequado.

4.3.1.2 Técnica de sobre-amostragem

Para lidar com o desequilíbrio das classes, foi utilizada a técnica de sobre-amostragem, Synthetic Minority Over-sampling Technique (SMOTE) (CHAWLA *et al.*, 2002), que consiste, basicamente, em sintetizar novos exemplos de treinamento para a(s) classe(s) minoritária(s). Essa síntese, ocorre a partir da seleção aleatória de

um exemplo de treinamento, da classe minoritária; uma posterior seleção, também aleatória, de um de seus *k* vizinhos; em seguida é realizada a conexão entre o exemplo e seu vizinho, através de uma linha no espaço de *features*; e, finalmente, o dado sintético é criado como uma combinação convexa entre o exemplo e seu vizinho.

A técnica SMOTE, depende de dois principais hiperparâmetros: a porcentagem de elementos que se deseja que a classe minoritária tenha em relação à classe majoritária e o número de vizinhos. Para encontrar os melhores valores para tais hiperparâmetros, foi feita uma busca exaustiva com validação cruzada. Ademais, não satisfeito somente com os resultados do SMOTE, foi decidido testar também extensões da técnica[3], são elas:

- **SMOTE seguido de um *undersample*** (técnica semelhante ao SMOTE, porém com o intuito de diminuir a classe majoritária).
- ***Borderline SMOTE*** (HAN; WANG; MAO, 2005).
- ***Borderline SMOTE SVM*** (NGUYEN; COOPER; KAMEI, 2011).
- ***ADASYN*** (HE, H. *et al.*, 2008).

Nas Tabelas 5, 6 e 7, são apresentadas as matrizes de confusão para cada modelo.

Obs.: Visto que, 5 diferentes técnicas de sobre-amostragem são testadas em 5 diferentes modelos, tem-se 25 diferentes configurações de modelos, que passam pela busca exaustiva. Por este motivo, são mostradas apenas as matrizes de confusão com as maiores precisões de classe.

- SMOTE seguido de um *undersample* com o modelo RF

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	68.21 %	3.35 %	28.43 %
	Classe 2	0.56 %	81.40 %	18.04 %
	Classe 3	17.51 %	16.07 %	66.42 %

Tabela 5 – Matriz de confusão para o modelo RF (com parâmetros ajustados) utilizando a técnica SMOTE, seguida de um *undersample*, para sobre-amostragem.

- *Borderline SMOTE* com o modelo SVM
- *Borderline SMOTE SVM* com o modelo SVM

[3] As técnicas não são descritas em detalhes, pois, como dito no parágrafo seguinte, o desequilíbrio das classes não era um dos principais problemas e as técnicas foram aplicadas a título de comparação com as redes neurais convolucionais. Informações mais aprofundadas sobre o assunto podem ser encontradas em suas respectivas bibliografias fornecidas.

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	66.13 %	3.51 %	30.35 %
	Classe 2	0.89 %	80.51 %	18.60 %
	Classe 3	17.17 %	14.97 %	67.86 %

Tabela 6 – Matriz de confusão para o modelo SVM (com parâmetros ajustados) utilizando a técnica *Borderline SMOTE*, para sobre-amostragem.

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	68.53 %	3.51 %	27.96 %
	Classe 2	1.00 %	80.51 %	18.49 %
	Classe 3	19.01 %	14.92 %	66.07 %

Tabela 7 – Matriz de confusão para o modelo SVM (com parâmetros ajustados) utilizando a técnica *Borderline SMOTE SVM*, para sobre-amostragem.

Quando comparado com os modelos das Tabelas 2, 4 e 3, houve uma melhoria considerável (maior que 10 %) na precisão da classe 1 e, não tão significativa, na precisão da classe 2. Já a classe 3, teve uma alta diminuição (cerca de 20 %, no pior dos casos) em suas precisões de classe. Isso ocorreu, porque as técnicas de sobre-amostragem visam, justamente, dar mais importância para as classes minoritárias aumentando sua quantidade de elementos no conjunto de treinamento.

O aumento da precisão das classes minoritárias, acompanhado da diminuição da precisão da classe 3, mostra que, neste caso, por mais que o conjunto de dados esteja equilibrado, um dado modelo não consegue aumentar a média ponderada das precisões de classe (i.e. acurácia). Este fator, aliado a circunstância de que o desequilíbrio entre as classes é relativamente baixo (classe majoritária apenas 3 vezes maior que a minoritária), leva a conclusão de que este não é um dos principais causadores das baixas precisões de classe.

Obs.: Outra alternativa, para resolver o desbalanço das classes, seria enviar, na fase de treinamento, uma quantia balanceada de cada classe do conjunto de dados. Essa alternativa seria inclusive mais simples do que a sobre-amostragem implementada, porém teria menos exemplos de treinamento.

4.3.1.3 Modelos com novo conjunto de dados

Com a intenção de sanar o problema da baixa qualidade das imagens, foi utilizada uma nova câmera. A imagem obtida com a câmera nova possui uma resolução de muita maior qualidade quando comparada a câmera antiga.

Com essa nova câmera, um novo conjunto de dados foi criado, utilizado o mesmo *framework* desenvolvido para o antigo conjunto de dados (discutido em 4.2).

É importante lembrar que, a divisão do conjunto de dados entre treinamento,

validação e teste, continua igual à utilizada no conjunto de dados anterior. Divisão estratificada de 70 % para o treinamento, 18 % para a validação e 12 % para o teste. O conjunto de treinamento é utilizado para treinar o modelo, o conjunto de validação é utilizado para avaliar o modelo ao fim de cada época de treinamento e o conjunto de teste é utilizado para testar o modelo, ao fim de todas as épocas de treinamento.
Obs.: Mais uma vez, com relação ao acordo de não divulgação, não serão mostrados os dados de cada uma das classes do conjunto de dados.

As matrizes de confusão para o novo conjunto de dados, já com os hiperparâmetros ótimos (obtidos pelo mesmo processo de busca exaustiva com validação cruzada), são apresentadas nas Tabelas 8, 9 e 10.

- RF

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	74.41 %	2.87 %	22.72 %
	Classe 2	5.47 %	71.88 %	22.66 %
	Classe 3	20.87 %	5.85 %	73.28 %

Tabela 8 – Matriz de confusão para o modelo RF, com os parâmetros ajustados (novo conjunto de dados).

- ET

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	73.63 %	1.31 %	25.07 %
	Classe 2	6.25 %	71.09 %	22.66 %
	Classe 3	21.12 %	4.33 %	74.55 %

Tabela 9 – Matriz de confusão para o modelo ET, com os parâmetros ajustados (novo conjunto de dados).

- *XGBoost*

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	72.85 %	2.35 %	24.80 %
	Classe 2	3.91 %	72.66 %	23.44 %
	Classe 3	23.92 %	6.62 %	69.47 %

Tabela 10 – Matriz de confusão para o modelo *XGBoost*, com os parâmetros ajustados (novo conjunto de dados).

Como visto nas Tabelas 8, 9 e 10, assim como com o conjunto de dados antigo, nenhum algoritmo teve problemas em confundir a classe 1 com a classe 2 (apenas 2.87

% e 5.47 % de erro de classificação, para o método RF com parâmetros ajustados, por exemplo) e todos os algoritmos apresentaram boa precisão (aproximadamente, acima de 70 %) para a classe 3 (73.28 % de precisão, para o método RF com parâmetros ajustados, por exemplo). A grande diferença é que, agora, as classes 1 e 2, tiveram um número muito menor de superpixeis confundidos com a classe 3 (no máximo, cerca de 22 % de erro de classificação, para o método RF com parâmetros ajustados, por exemplo). Por consequência, a diagonal principal da matriz de confusão tem, para todas as classes, uma precisão maior ou igual a aproximadamente 70 %, estatística a qual, não foi atingida por nenhum modelo visto até então.

Essa análise, evidencia que este era um dos principais problemas e que, com o aumento do tamanho do conjunto de dados, ainda mais superpixeis serão classificados corretamente, tornando as classificações ainda mais precisas.

4.3.2 Abordagem sem extração de *feature* manual

4.3.2.1 CNN com aprendizado por transferência

Por fim, foi decidido testar uma nova abordagem, tanto na etapa da classificação como na extração de *features*. Para isso, decidiu-se trabalhar com uma CNN, que, se tratando de imagens, além de extrair as *features* de maneira muito mais eficiente, possui muito menos parâmetros a ser treinados, quando comparada, por exemplo, a uma rede neural tradicional, reduzindo assim tempo de treinamento, tempo de execução, custo computacional, *overfitting*, etc. A CNN, tem um menor número de parâmetros, sobretudo, por possuir técnicas de compartilhamento de parâmetros e por ser constituída, principalmente, de conexões locais.

Em aplicações de VC, ao invés de construir uma CNN do zero, na grande maioria das vezes, é mais vantajoso utilizar uma arquitetura já existente, com parâmetros já treinados, mesmo que para uma aplicação completamente diferente. Isso pois, os modelos utilizados no aprendizado por transferência, não só tem uma arquitetura mais complexa e bem estruturada, mas também foram treinados em imensos conjuntos de dados, portanto, tem um maior poder de generalização.

No aprendizado por transferência, comumente, um modelo já treinado é selecionado e seus parâmetros são congelados, ou seja, seus parâmetros não mudarão mais independente se modelo estiver realizando uma inferência ou sendo treinado. Dependendo do tamanho do conjunto de dados da aplicação, mais ou menos parâmetros podem ser congelados. Feito isso, a saída do modelo (porventura, algumas das últimas camadas também) é alterada para uma mais condizente com a aplicação desejada, deixando assim, relativamente poucos parâmetros a serem treinados.

Para o caso estudado, o modelo que será utilizado no aprendizado por transferência, foi escolhido baseado, especialmente, no *trade-off* entre tempo de execução e

precisão. Os principais modelos testados[4] foram:

- ***Resnet*** (HE, K. *et al.*, 2015).
- ***MobileNet v2*** (SANDLER *et al.*, 2018).
- ***ResNeXt*** (XIE *et al.*, 2017).

Dentre estes três, o modelo *MobileNet v2*, foi o escolhido. Pois, tem uma baixa complexidade computacional (quando comparado com os demais), um número de parâmetros relativamente baixo (aproximadamente 3 milhões) e, para o conjunto de dados estudado, apresentou uma ótima precisão.

Os valores do histograma HSV da imagem (equação 9), não serão mais utilizados como *features*. Dessa vez, a extração de *features* ocorre nas camadas de convolução (como explicado em 2.5.1).

Antes de o conjunto de dados ser alimentado na CNN, ele passa por um simples pré-processamento, onde, as dimensões de altura e largura da imagem de entrada são redimensionadas para as específicas da *MobileNet v2* (224 *x* 224) e seus três canais de cores (RGB), são normalizados utilizando valores de média e desvio padrão, também específicos da *MobileNet v2*.

A CNN, é constituída, em suma, pelo modelo *MobileNet v2*, com todos os seus parâmetros congelados. A única exceção é, sua camada de classificação final (*softmax* para 1000 classes), a qual é substituída por outra semelhante, porém, para o número de classes da aplicação (*softmax* para 3 classes).

Na última camada do modelo (antes da *softmax*), foi adicionada a funcionalidade de *dropout*[5], com o intuito de diminuir o *overfitting*. O *dropout*, nada mais é que um método de regularização que, basicamente, seleciona aleatoriamente uma quantidade pré-estabelecida de neurônios de uma camada e impede que a informação de suas respectivas funções de ativação sejam passadas adiante (durante a retro-propagação).

Durante o treinamento, como é um problema de classificação, é utilizada, como função custo, a entropia cruzada e, com o intuito de minimizar o seu valor, é utilizado, como otimizador, o descenso do gradiente estocástico, com taxa de aprendizado de 0.001. Após treinada por 24 épocas, a CNN é testada, com o conjunto de testes, e os resultados (em forma de matriz de confusão) podem ser vistos na Tabela 11.

Não só a precisão de cada uma das classes aumentou drasticamente, como também, não se tem mais o problema de confusão das classes 1 e 2 com a classe 3. Como todas as classes têm precisão muito próxima, a acurácia do modelo (93.73 %) também acaba sendo próxima. Ainda, é importante destacar que, por mais que

[4] Destaca-se que, foi feita uma escolha prévia de modelo. No futuro, deve ser feito um estudo, de modo a escolher um modelo ainda mais adequado.

[5] Mais informações, podem ser encontradas em (SRIVASTAVA *et al.*, 2014).

		Preditos		
		Classe 1	Classe 2	Classe 3
Reais	Classe 1	93.91 %	0.43 %	5.65 %
	Classe 2	0.00 %	94.81 %	5.19 %
	Classe 3	2.98 %	3.83 %	93.19 %

Tabela 11 – Matriz de confusão para a CNN, com aprendizado por transferência da arquitetura *MobileNet v2*.

seja um bom resultado, a alta acurácia pode evidenciar *overfitting* e uma análise mais aprofundada deve ser feita futuramente. Não obstante, a abordagem da CNN, com aprendizado por transferência, é claramente muito mais precisa que todas as outras abordagens anteriormente analisadas.

Um dos principais motivos, que leva a tão melhor precisão da CNN, é a extração de *features*, que, agora, é feita pela CNN (como explicado em 2.5.1) e antes, era feita através da extração do histograma HSV de cada superpixel. O histograma HSV, perde informação, não só pelo fato de ser apenas um canal (não foi utilizado os 3 canais HSV separados, pois aumentava muito o custo computacional), mas também, por levar em consideração apenas a localização no espaço de cores de cada píxel e não sua localização espacial e possível formas e características, que um conjunto (de pixeis) possa gerar.

4.4 COMUNICAÇÃO COM OUTROS MÓDULOS

A informação está então pronta para ser enviada ao sistemas de controle. A comunicação ocorre através do protocolo *Ethernet* User Datagram Protocol (UDP) e é feita e garantida por outros integrantes da equipe do ISI-SE.
Obs.: Novamente, com relação ao acordo de não divulgação, não serão apresentadas em maiores detalhes as comunicações do sistema de VC e AM com os outros módulos.

4.5 APIS UTILIZADAS

Inicialmente, o algoritmo de VC e AM, tinha suas principais funções escritas na API *TensorFlow* (ABADI *et al.*, 2015), porém notou-se que, quando embarcada na plataforma VC/AM, o algoritmo ocupava um grande espaço na memória e tinha um tempo de processamento muito grande. Não foram feitos estudos aprofundados sobre os motivos de tais inconvenientes, porém acredita-se que a principal razão seja a complexidade dos modelos carregados, especialmente o modelo da CNN. Foi tentado utilizar modelos mais simples e com menos parâmetros, porém não se obteve uma diferença significativa.

Por estes motivos, foi decidido migrar para uma outra API, o *PyTorch* (PASZKE *et al.*, 2019), utilizando os mesmos modelos e técnicas. O espaço de memória ocupado

pelo algoritmo, com a utilização dessa nova API, foi muito menor que o anterior, não apresentando nenhum inconveniente (relacionado a espaço de memória) para a aplicação. Também em comparação a API anterior, o tempo de processamento diminuiu consideravelmente.

Com o intuito de diminuir ainda mais o tempo de processamento, sem perder no desempenho do algoritmo, foi decidido converter os modelos para a API *TensorRT* (NVIDIA *et al.*, 2016), específica para trabalhar com modelos de inteligência artificial nas placas da *NVIDIA*.

5 VALIDAÇÃO EXPERIMENTAL

Como este capítulo visa apresentar e discutir todos os resultados referentes a todas as variáveis da análise do sistema, a grande maioria teve de ser retirada do documento não sigiloso, seguindo, mais uma vez, o acordo de não divulgação de informações confidenciais da empresa.

A variável que será apresentada e discutida neste documento público, será o tempo de processamento do algoritmo como um todo.

5.1 TEMPO DE PROCESSAMENTO

No que diz respeito ao tempo de processamento, foi contado o tempo desde que, o sistema VC/AM, recebe uma imagem (proveniente do sistema de aquisição) até o momento em que as informações são enviadas para os outros sistemas. As principais ações realizadas, durante esse período, são: aquisição das imagens, execução da segmentação semântica, execução da segmentação via superpixel e classificação dos superpixeis pela CNN. Na Figura 10, é apresentado o histograma de frequência do tempo de processamento do sistema VC/AM. O experimento teve média 4.19 s e desvio padrão de 1.33 s.

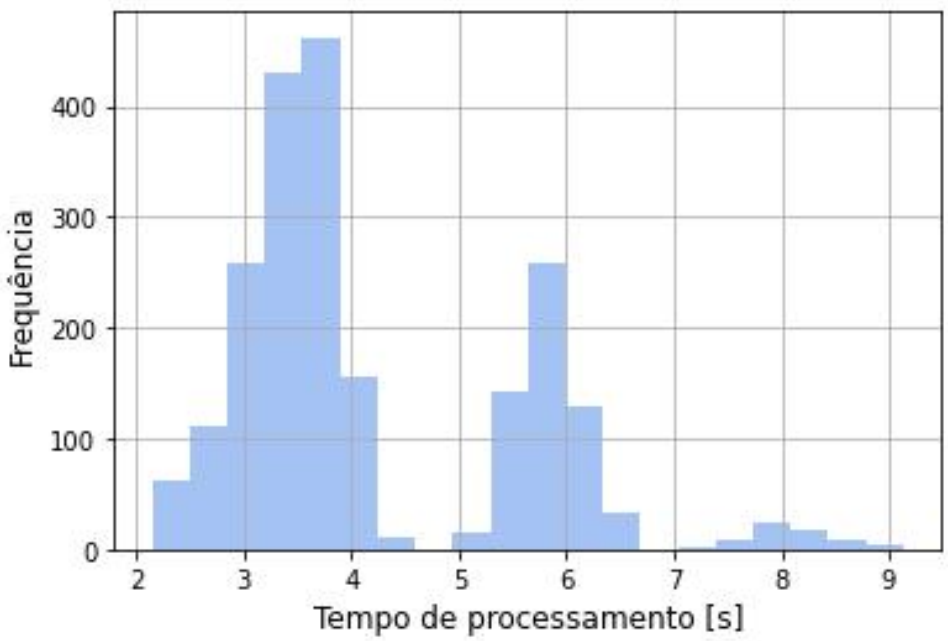

Figura 10 – Histograma do tempo de processamento do Sistema VC/AM.

Como a distribuição apresentada é trimodal, separando cada um dos seus três picos (três diferentes modas) obtêm-se os seguintes valores:

- Para o primeiro pico, média de 3.40 s e desvio padrão de 0.439 s.
- Para o segundo pico, média de 5.83 s e desvio padrão de 0.289 s.

- Para o terceiro pico, média de 8.09 s e desvio padrão de 0.387 s.

Assim, conclui-se que, o primeiro pico tem cerca de 70 % dos elementos do conjunto e, por este motivo, faz com que a média geral do experimento (4.19 s) se aproxime da sua (3.4 s). Já o segundo pico, como pode ser representado pela distribuição normal, tem seus elementos variando entre 4.9 s e 6.7 s ($5.83 \pm 3(0.289)$). Por sua vez, o terceiro pico, tem apenas 3 % dos elementos do conjunto, portanto, por mais que sua média seja muito alta, ele tem pouco impacto na média geral do experimento.

Para entender melhor o que cada um desses três picos representa, na Figura 11, é traçado o gráfico da área da região de interesse com relação ao tempo de processamento do sistema.

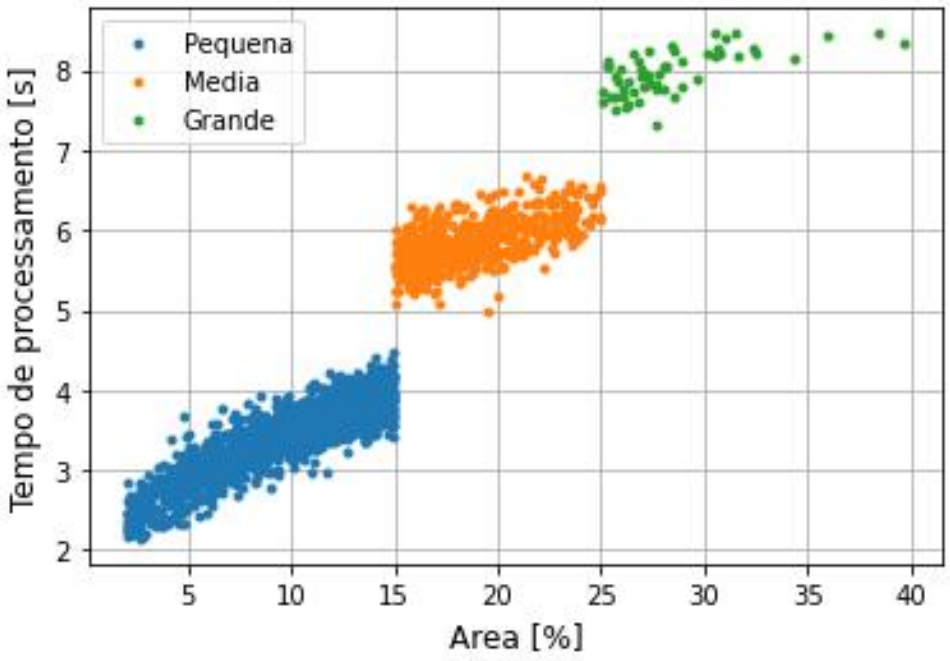

Figura 11 – Relação entre a área da região de interesse (dividia em grupos) e o tempo de processamento do Sistema VC/AM.

Na Figura 11, nota-se a divisão em três grupos, que representam, não só cada um dos três picos do histograma da Figura 10, como também cada um dos três diferentes tamanhos de região de interesse: regiões de área pequena, média e grande (como visto em 3.2.2). Dentro de cada uma das regiões, o tempo de processamento, tem um aumento aproximadamente linear e diretamente proporcional a área da região de interesse. No entanto, de um tipo de região para outro, observa-se uma não linearidade, explicada justamente pela mudança do número de superpixeis de uma região para outra (100 superpixeis para a região pequena, 200 para a média e 300 para a grande).

No intuito de, localizar quais são os principais causadores do aumento no tempo de processamento, em laboratório, foi analisado quais das ações do sistema de VC/AM

levavam mais tempo. Em um primeiro momento, foi evidenciado que a aquisição das imagens e a execução da segmentação semântica, representavam apenas cerca de 0.2 s do tempo total. Portanto, todo o resto do tempo de processamento, vem parte da segmentação via superpixel e parte da classificação feita pela CNN.

Através de uma análise ainda mais minuciosa, foi evidenciado que, praticamente todo o tempo que o algoritmo levava para executar a segmentação via superpixel, vinha da função que aplicava o método SLIC. O método SLIC, tem complexidade $O(N)$, onde N é o número de pixeis da imagem, logo, como todas as imagens são de mesmo tamanho, seu tempo de execução não seria muito afetado por um maior número de superpixeis. No entanto, quando é utilizada a máscara de fundo (como em 3.2.2), a complexidade começa a depender também do número de superpixeis, se tornando $O(N + N_S)$, onde N_S seria o número de superpixeis.

De maneira semelhante, para a classificação feita pela CNN, foi evidenciado que seu tempo de execução só era tão alto, pois a classificação para cada superpixel era feita em série (por meio de um laço de repetição). Ou seja, quanto maior o número de superpixeis, mais tempo o classificador levará por imagem.

Em suma, o sistema VC/AM, no que diz respeito a tempo de processamento, tem dois principais gargalos, o método SLIC e a classificação sequencial da CNN. Logo, optou-se trabalhar nas melhorias, primeiramente, do tempo de processamento.

5.1.1 Melhorias no tempo de processamento do método SLIC

Começando pelo gargalo do método SLIC, o principal causador do alto tempo de execução, é a utilização da máscara. Porém, sem a máscara, o número de superpixeis teria que aumentar muito, desencadeando em um aumento no tempo de execução da classificação da CNN. Além disso, o ganho que se tem ao utilizar o SLIC sem a máscara, não é tão relevante ao ponto de compensar esse tempo a mais na classificação da CNN.

Por estes motivos, ao invés de modificar a função ou criar uma do zero, foi buscado tentar trabalhar com outro método de segmentação de superpixeis. Como, o sistema VC/AM é embarcado em uma placa da *NVIDIA*, que tem uma excelente Graphics Processing Unit (GPU) e para cálculos relacionados a imagem, a GPU costuma ser muito mais rápida do que a Central Processing Unit (CPU), procurou-se por um método de segmentação de superpixel que utilizasse o poder da GPU. Assim, foi então encontrado, um método que utiliza a GPU justamente para realizar a implementação do método SLIC.

Testes comparando a velocidade dos métodos, foram realizados, sendo evidenciado que o método SLIC com GPU é cerca de 5 vezes mais rápido do que o SLIC convencional. Ainda, dentro da região de interesse, existe uma alta similaridade nas bordas dos superpixeis geradas por ambos os métodos. Concluindo assim, que

o método SLIC com GPU, fornece uma resposta muito mais rápida, sem perder na qualidade da segmentação. Ademais, o SLIC com GPU, por não ter máscara de fundo, tem agora um número constante de superpixeis N_s. Inicialmente utiliza-se $N_s = 800$, porém, como este valor foi determinado empiricamente, estudos mais apurados devem ser feitos, futuramente, com a finalidade de encontrar um valor mais adequado para N_s. Por fim, para remover os pixeis pretos (referentes ao fundo da imagem, antes retirado pela máscara) um filtro é adicionado antes que os superpixeis sejam enviados a CNN. Esse filtro remove todos os superpixeis que não estão na região de interesse, para que assim, classificações desnecessárias sejam evitadas.

5.1.2 Melhorias no tempo de processamento da classificação feita pela CNN

O outro gargalo do aumento do tempo de processamento é a classificação sequencial feita pela CNN. Como a classificação é feita sequencialmente, superpixel por superpixel, é de se esperar que o tempo de execução, desse laço de repetição, seja grande. Para tanto, foi optado em executar tal laço em paralelo, uma vez que, a ordem em que os superpixeis são classificados, não altera o resultado da proporção de cada uma das classes.

Quando se deseja diminuir o tempo de execução de um algoritmo, através da paralelização de suas tarefas, as abordagens mais utilizadas e que tem se demonstrado mais efetivas (tanto no âmbito acadêmico quanto industrial), são o *Multithreading* e o *Multiprocessing*.

Multithreading, é a criação de múltiplas *threads* (ramificações) do código principal, que dividem recursos e memória entre si, com o intuito de paralelizar uma ou mais tarefas. No *Multithreading*, essa paralelização ocorre por meio da execução alternada de diferentes *threads* em uma alta frequência, assim, dando a impressão de uma execução simultânea.

Já o *Multiprocessing*, representa a criação de múltiplos processos, que podem (*Multiprocessing* Síncrono) ou não (*Multiprocessing* Assíncrono) executar uma mesma funcionalidade ou código. No *Multiprocessing*, os processos não compartilham memória e recursos, dependendo de técnicas de serialização de objetos para interagir uns com os outros. Assim como no *Multithreading*, o *Multiprocessing* alterna a execução de seu código para se aproximar cada vez mais de uma execução simultânea, porém, aqui, a frequência com que essa alternância ocorre é muito mais alta, justamente por ser gerenciada pelo sistema operacional. Por ter essa maior frequência, o *Multiprocessing*, leva menos tempo para executar um determinado processo, mas, ao mesmo tempo, tem um maior custo computacional.

Dito isso, para trabalhar com o laço de repetição da classificação feita pela CNN, decidiu-se utilizar o *Multithreading* ao invés do *Multiprocessing*, pois com o *Multiprocessing*, muitos processos seriam criados, aumentando significativamente o

uso do processador. Ademais, a remoção dos pixeis pretos também é paralelizada, por meio do *Multithreading*.

Contudo, para trabalhar com o sistema VC/AM rodando em paralelo com o sistema de aquisição, utilizou-se o *Multiprocessing*.

5.1.3 Testes em laboratório, após as melhorias no tempo de processamento

Para testar, em laboratório, as melhorias realizadas no algoritmo, foi realizado um teste nas mesmas condições de 5.1.

Na Figura 12, são comparados os histogramas de frequência do tempo de processamento do sistema VC/AM, antes e depois das melhorias com relação ao tempo de processamento. O experimento teve sua média abaixada de 4.19 s para 2.75 s e seu desvio padrão de 1.33 s para 0.50 s.

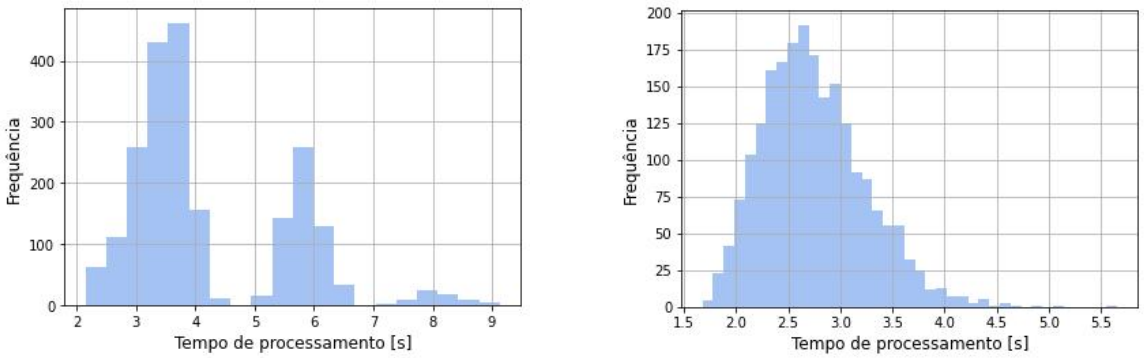

Figura 12 – Comparação dos histogramas do tempo de processamento do Sistema VC/AM, antes (esquerda) e depois (direita) das melhorias no tempo de processamento.

Nota-se, que a distribuição deixou de ser trimodal, para se tornar unimodal, pois agora as regiões não são mais divididas em grupos. Além disso, agora, a maior parte dos elementos estão situados abaixo de 3.75 s, valor o qual é ainda menor que a média do conjunto nos testes anteriores (4.19 s), mostrando assim a excelente melhora no tempo de processamento.

Na Figura 13, é traçado o gráfico da área da região de interesse com relação ao tempo de processamento do sistema VC/AM, para os casos de antes e depois das melhorias no tempo de processamento.

Agora, o gráfico da área da região de interesse com relação ao tempo de processamento, não é mais dividido em três regiões e seu comportamento aproxima-se do linear. Nota-se também, que as imagens continuam distribuídas, em sua maioria, com uma área da região de interesse menor que 25 % da imagem.

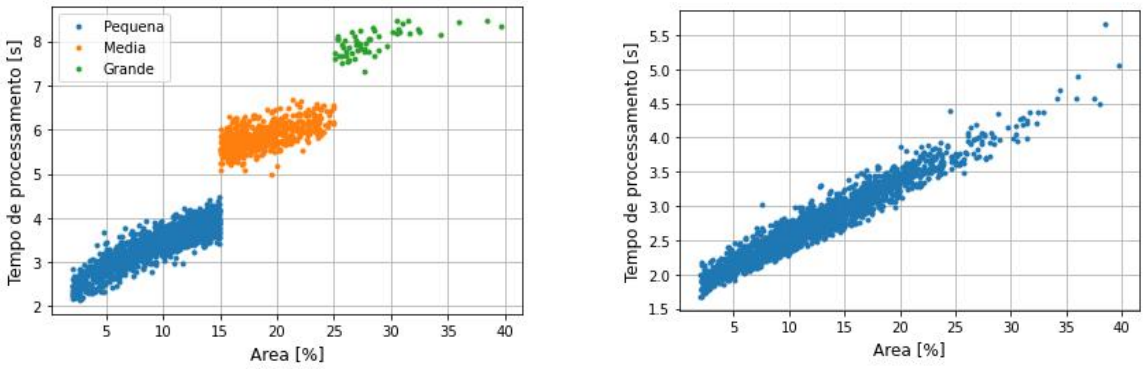

Figura 13 – Comparação dos gráficos da área da região de interesse e tempo de processamento do Sistema VC/AM, antes (esquerda) e depois (direita) das melhorias no tempo de processamento.

6 CONCLUSÃO

É importante lembrar que, assim como nos outros capítulos, a conclusão também possui informações confidenciais da empresa ISI-SE. Dito isso, neste documento, consta apenas um resumo de toda a conclusão elucidada no documento oficial.

Neste PFC, não só técnicas de visão computacional e aprendizado de máquina são discutidas e implementadas, como também *frameworks* específicos para a realização de necessárias tarefas são criados e ainda, são feitas análises e melhorias com relação ao tempo de processamento do algoritmo.

Todo o algoritmo é embarcado na plataforma VC/AM, com o sistema de aquisição. Após adquirir uma imagem, o sistema de aquisição a envia para o sistema VC/AM, que começa com uma segmentação semântica da imagem recebida, selecionando a região de interesse. A região de interesse é então segmentada em superpixeis, que são, posteriormente, separados em um vetor, cujo tamanho é igual ao número de superpixeis da imagem. Por fim, esse vetor de superpixeis é enviado para a CNN que, classifica cada um conforme sua respectiva classe, finalizando o processo e deixando as informações prontas para serem enviadas aos outros módulos.

Além de toda implementação do algoritmo embarcado na plataforma VC/AM, também foram desenvolvidos *frameworks* auxiliares, como os de análise HSV, de criação/rotulação de conjuntos de dados de superpixeis e de análise estatística. O *framework* de análise HSV, foi utilizado principalmente para a análise inicial das imagens do conjunto de dados (vide 3.1.1), porém pretende-se utiliza-lo no futuro. Já o *framework* de criação/rotulação de conjuntos de dados de superpixeis, é justamente responsável pela criação de todo o conjunto de dados que alimenta a CNN. Este *framework*, foi utilizado na criação da primeira versão do conjunto de dados e será utilizado na tarefa contínua de atualização e expansão do mesmo. Por fim, o *framework* de análise estatística, gera estatísticas não só das varáveis de saída, mas também do tempo de processamento do sistema VC/AM, sendo de extrema importância para a análise dos dados e para as tomadas de decisões ao longo do projeto.

Em um primeiro momento, para avaliar o desempenho da segmentação via superpixel, é utilizada a avaliação empírica para uma série de imagens. Optou-se pela avaliação empírica como métrica de avaliação, principalmente devido à alta variabilidade da posição e proporção dos elementos ao longo da região de interesse. Por outro lado, a classificação dos superpixeis, tem inúmeras métricas que podem servir como métrica de avaliação. A escolhida, foi a matriz de confusão com porcentagem de classe, justamente por possibilitar a análise precisa da porcentagem de elementos, de cada classe, que estão sendo classificados, correta e incorretamente. Com a utilização da CNN, obteve-se uma precisão de cerca de 93 % para cada uma das três classes.

Dentre as principais dificultardes encontradas, ao longo do desenvolvimento do

projeto, ressalta-se a alta variabilidade, não só de disposição e posição de cada uma das classes ao longo da imagem, como também de iluminação das mesmas. Essa dificuldade, levou justamente a utilização da abordagem dos superpixeis, que, neste caso, possibilitam a segmentação de um número qualquer de elementos de cada classe. Ainda como dificuldade, ressalta-se a não tão alta qualidade das imagens obtidas com a primeira câmera, a qual foi solucionada através da troca de câmeras. Por fim, também vale salientar a dificuldade no que se diz respeito a tempo de processamento do sistema VC/AM. Esta tarefa ainda está em desenvolvimento, contudo, muito se melhorou ao longo do projeto (como descrito em 5.1.3).

6.1 PRÓXIMOS PASSOS E TRABALHOS FUTUROS

Como o fim do período do PFC não coincide com o fim do cronograma de atividades do projeto da empresa ISI-SE, alguns desenvolvimentos e atividades ainda não foram concluídos ou até realizados. Contudo, o autor deste documento, irá, futuramente, voltar a trabalhar na empresa ISI-SE e poderá dar continuidade ao trabalho feito.

Dentre todas as pendências, limitações e melhorias que precisam ser tratadas e implementadas vale a pena citar aqui as principais e mais importantes:

- **Aumentar o conjunto de dados:** A precisão para cada uma das classes, de cerca de 93 %, indica que o modelo e algoritmo desenvolvido, consegue classificar corretamente a maior parte dos superpixeis de um dado conjuntos de dados. A única limitação que fica aqui, é a alta variabilidade dos elementos da imagem, que pode diminuir essa precisão. Para suprir toda essa variabilidade, faz-se necessário então, um aumento no conjunto de dados.
- **Otimização do tempo de processamento:** Por mais que o tempo de processamento tenha sido consideravelmente reduzido, ainda se procura por uma redução.
- **Funções de pre-processamento:** Uma análise mais apurada no que diz respeito às funções de processamento da CNN deve ser feita. Visto que, foi utilizada apenas a função de pré-processamento padrão da *Mobilenet v2*.

REFERÊNCIAS

ABADI, Martın *et al.* **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. [*S.l.*: *s.n.*], 2015. Software available from tensorflow.org. Disponível em: `https://www.tensorflow.org/`.

ACHANTA, Radhakrishna; SHAJI, Appu; SMITH, Kevin; LUCCHI, Aurelien; FUA, Pascal; SÜSSTRUNK, Sabine. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 11, p. 2274–2282, 2012. DOI: `10.1109/TPAMI.2012.120`.

BREIMAN, Leo. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

CHAWLA, Nitesh; BOWYER, Kevin; HALL, Lawrence; KEGELMEYER, W. SMOTE: Synthetic Minority Over-sampling Technique. **J. Artif. Intell. Res. (JAIR)**, v. 16, p. 321–357, jun. 2002. DOI: `10.1613/jair.953`.

CHEN, Tianqi; GUESTRIN, Carlos. XGBoost: A Scalable Tree Boosting System. *In*: PROCEEDINGS of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, California, USA: ACM, 2016. (KDD '16), p. 785–794. DOI: `10.1145/2939672.2939785`. Disponível em: `http://doi.acm.org/10.1145/2939672.2939785`.

CRAMMER, Koby; SINGER, Yoram. On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines. **J. Mach. Learn. Res.**, JMLR.org, v. 2, p. 265–292, mar. 2002. ISSN 1532-4435.

FELZENSZWALB, Pedro F; HUTTENLOCHER, Daniel P. Efficient graph-based image segmentation. **International journal of computer vision**, Springer, v. 59, n. 2, p. 167–181, 2004.

GEURTS, Pierre; ERNST, Damien; WEHENKEL, Louis. Extremely randomized trees. **Machine learning**, Springer, v. 63, n. 1, p. 3–42, 2006.

HAN, Hui; WANG, Wen-Yuan; MAO, Bing-Huan. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. *In*: HUANG, De-Shuang; ZHANG, Xiao-Ping; HUANG, Guang-Bin (Ed.). **Advances in Intelligent Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. P. 878–887.

HE, Haibo; BAI, Yang; GARCIA, Edwardo A.; LI, Shutao. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *In*: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). [*S.l.*: *s.n.*], 2008. P. 1322–1328. DOI: 10.1109/IJCNN.2008.4633969.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. **CoRR**, abs/1512.03385, 2015. arXiv: 1512.03385. Disponível em: http://arxiv.org/abs/1512.03385.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 60, n. 6, p. 84–90, mai. 2017. ISSN 0001-0782. DOI: 10.1145/3065386. Disponível em: https://doi.org/10.1145/3065386.

LEE, Honglak; GROSSE, Roger; RANGANATH, Rajesh; NG, Andrew Y. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. *In*: PROCEEDINGS of the 26th Annual International Conference on Machine Learning. Montreal, Quebec, Canada: Association for Computing Machinery, 2009. (ICML '09), p. 609–616. DOI: 10.1145/1553374.1553453. Disponível em: https://doi.org/10.1145/1553374.1553453.

NEUBERT, Peer; PROTZEL, Peter. Compact Watershed and Preemptive SLIC: On Improving Trade-offs of Superpixel Segmentation Algorithms. *In*: 2014 22nd International Conference on Pattern Recognition. [*S.l.*: *s.n.*], 2014. P. 996–1001. DOI: 10.1109/ICPR.2014.181.

NGUYEN, Hien M; COOPER, Eric W; KAMEI, Katsuari. Borderline over-sampling for imbalanced data classification. **International Journal of Knowledge Engineering and Soft Data Paradigms**, Inderscience Publishers, v. 3, n. 1, p. 4–21, 2011.

NVIDIA; GRAY, Allison; GOTTBRATH, Chris; OLSON, Ryan; PRASANNA, Shashank. **Production Deep Learning with NVIDIA GPU Inference Engine**. [*S.l.*: *s.n.*], 2016. Disponível em: https://developer.nvidia.com/tensorrt.

PASZKE, Adam *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *In*: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; D'ALCHÉ-BUC, F.; FOX, E.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 32**. [*S.l.*]: Curran Associates, Inc., 2019. P. 8024–8035. Disponível em:

http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

PEDREGOSA, F. *et al.* Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey; CHEN, Liang-Chieh. Mobilenetv2: Inverted residuals and linear bottlenecks. *In*: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [*S.l.*: *s.n.*], 2018. P. 4510–4520.

SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. **The journal of machine learning research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

VEDALDI, Andrea; SOATTO, Stefano. Quick shift and kernel methods for mode seeking. *In*: SPRINGER. EUROPEAN conference on computer vision. [*S.l.*: *s.n.*], 2008. P. 705–718.

WALT, Stéfan van der; SCHÖNBERGER, Johannes L.; NUNEZ-IGLESIAS, Juan; BOULOGNE, François; WARNER, Joshua D.; YAGER, Neil; GOUILLART, Emmanuelle; YU, Tony; CONTRIBUTORS, the scikit-image. scikit-image: image processing in Python. **PeerJ**, v. 2, e453, jun. 2014. ISSN 2167-8359. DOI: 10.7717/peerj.453. Disponível em: https://doi.org/10.7717/peerj.453.

XIE, Saining; GIRSHICK, Ross; DOLLÁR, Piotr; TU, Zhuowen; HE, Kaiming. Aggregated residual transformations for deep neural networks. *In*: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [*S.l.*: *s.n.*], 2017. P. 1492–1500.