UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS GIANOGLIO VINHAS

CLASSIFICAÇÃO DE PINTURAS COM APRENDIZADO DE MÁQUINA

BAURU
Julho/2021

LUCAS GIANOGLIO VINHAS

CLASSIFICAÇÃO DE PINTURAS COM APRENDIZADO DE MÁQUINA

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Rafael Gonçalves Pires

BAURU
Julho/2021

Lucas Gianoglio Vinhas Classificação de pinturas com Aprendizado de Máquina/ Lucas Gianoglio Vinhas. – Bauru, Julho/2021- 44 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Rafael Gonçalves Pires
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Julho/2021.
1. Aprendizado de Máquina 2. Convolutional Neural Networks 3. Reconhecimento de Imagens 4. Classificação

Lucas Gianoglio Vinhas

Classificação de pinturas com Aprendizado de Máquina

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Rafael Gonçalves Pires
Orientador
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Profa. Dra. Simone das Graças Domingues Prado
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Profa. Dra. Andréa Carla Gonçalves Vianna
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, 21 de Julho de 2021

Dedico esse trabalho a minha família, que proporcionou tudo o que foi necessário para que eu pudesse chegar até aqui

Agradecimentos

Agradeço a minha família por ter me acompanhado até aqui em todos os momentos da minha vida. A UNESP e o corpo docente por tudo que aprendi no curso e ao meu orientador por mais do que me orientar, me guiar e ajudar nesse trabalho.

"I visualise a time when we will be to robots what dogs are to humans, and I'm rooting for the machines."

Stephen Hawking

Resumo

Este trabalho explorara diferentes arquiteturas de Redes Neurais Convolucionais para classificar pinturas de acordo com seus autores. Tal a complexidade do problema foi necessário utilizar a técnica que consiste em treinar exaustivamente a rede com milhões de amostras em uma determinada base e depois transferir esse conhecimento para o problema proposto. Foi feito o uso cinco arquiteturas de redes: VGG16, *Inception*, *Xception*, *ResNet* e *InceptionResnet*. A base utilizada foi a *WikiArt Dataset*, em que todos os pintores com mais de 500 obras foram selecionados, totalizando 23 classes para categorização, sendo que as classes foram balanceadas conforme o número de imagens presentes. Ficou evidente nos experimentos a eficácia da rede *Inception* e suas variantes. Todo o treinamento foi realizado na plataforma *Colab* do Google, de modo a otimizar o tempo de treinamento, que não é breve em computadores domésticos.

Palavras-chave: Aprendizado de Máquina, Convolutional Neural Network, Reconhecimento Imagens, Classificação.

Abstract

This work intends to explore different architectures of Convolutional Neural Networks to classify paintings according to their authors. Such is the complexity of the problem, it was necessary to use the technique in which consists of exhaustively training the network with millions of samples on a given basis and then transfer this knowledge to the proposed problem. We made use of five network architectures: VGG16, Inception, Xception, ResNet and InceptionResnet. The dataset used was the WikiArt Dataset, in which all painters with more than 19.000 works were selected, totaling 23 classes for categorization. were balanced according to the number of images present. It was evident in the experiments the effectiveness of *Inception* network and its variants. All training was carried out on Google's Colab platform, in order to optimize training time, which is not short on home computers.

Keywords: Machine Learning, Convolutional Neural Network, Image Recognition, Classification.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

AM	Aprendizado de Máquina
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphical Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ML	Machine Learning

Sumário

1 Introdução

Desde pinturas rupestres feitas em cavernas, até obras produzidas na atualidade, várias formas de artes plásticas acrescentaram à vida humana, algumas focadas no aspecto estético, outras, na reflexão propiciada. As diferenças entre as obras de arte são demarcadas hoje por meio de categorização feita conforme o movimento artístico ao qual pertencem. Classicismo, modernismo, impressionismo e cubismo são exemplos de categorias. A análise de tais obras era feita até bem pouco tempo por meio de curadores, que examinavam o trabalho manualmente, observando suas características. Foi só no início dos anos 80 que projetos como ACHOIR, CRISATEL, VASARI, NARCISSE, VISEUM, ARTISTE, entre outros, surgiram para implementar métodos computacionais ao processo de curadoria dos museus. E, no fim dessa década, a computação já havia avançado a ponto de tornar possível armazenar e analisar um volume grande de obras de arte com alta resolução (BARTOLINI et al., 2003).

Uma das consequências imediatas da revolução digital nesta área é o crescimento considerável do acesso a pinturas que, muitas vezes, não estão devidamente identificadas, nem categorizadas, criando um número grande de dados ainda sem a devida análise.

Estudos recentes destacam que técnicas de aprendizado de máquina se tornaram uma das abordagens mais eficazes para a resolução de muitos problemas do mundo real, tais como detectar ataques a sistema de reconhecimento facial (SOUZA, 2019) e remoção de ruído em imagens (RAFAEL, 2018).

No problema de classificação de pinturas utilizando Aprendizado de Máquina podemos destacar os trabalhos de (SALEH; ELGAMMAL, 2015), em que a ideia consiste em extrair características das pinturas utilizando o método *Classeme features*, que proporcionou resultados adequados. Já no trabalho de (CETINIC; LIPIC; GRGIC, 2018) utilizam a rede neural CNN com *fine-tunning* obtendo resultados satisfatórios.

Classificar pinturas pode ser muito mais complexo que classificar imagens naturais. Considerando o uso de Redes Neurais de Convolução (*Convolutional Neural Networks* – CNN), podemos destacar os seguintes desafios:

- Existem muitas características semelhantes em categorias diferentes como é possível observar na Figura 1 (TAN et al., 2016).
- É difícil encontrar uma arquitetura baseada em CNN satisfatória, pois as características extraídas entre as camadas inferiores e superiores podem ser muito diferentes na mesma categoria. (TAN et al., 2016).

Desta forma, este trabalho tem como objetivo:

- Investigar e viabilizar diferentes arquiteturas baseadas em Redes Neurais de Convolução (CNN).
- Explorar o uso da técnica transferência de aprendizado, do inglês *Transfer Learning*, na tarefa de classificar pinturas de acordo com seus respectivos artistas.
- Contribuir com mais estudos na área proposta.

Figura 1 – Características semelhantes em categorias diferentes.

Marilungo (1998)

Um classificador capaz de distinguir diversas pinturas de diferentes artistas certamente interessaria a organizadores de coleções de arte, a museus etc. Um especialista humano não mais precisaria rotular cada pintura. A obtenção de bons resultados nesta tarefa também beneficiaria outras tarefas complexas, como reconhecimento facial, reconhecimento de cena, enfim, qualquer tarefa que requeira categorização com base em critérios complexos.

Este trabalho está organizado conforme segue: O Capítulo 2 introduz a fundamentação teórica do trabalho. O Capitulo 3 apresenta a metodologia do trabalho. O Capitulo 4 descreve os resultados dos experimentos e o Capítulo 5 contém a conclusão da pesquisa.

.

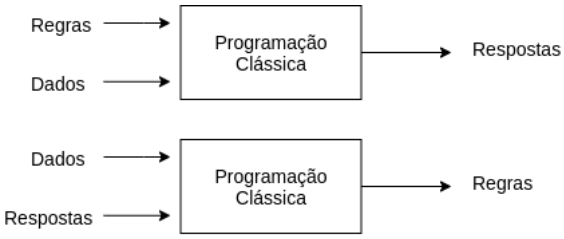
2 Fundamentação Teórica

Este capítulo introduz o referencial teórico de aprendizado de máquina com enfoque em redes neurais convolucionais.

2.1 Aprendizado de Máquina

A programação convencional é marcada por uma sequência de passos descritos pelo programador para atingir um determinado objetivo, os dados e as ações são resultados da programação feita pelo programador. Um sistema de Aprendizado Profundo é treinado ao invés de programado, o programador fornece um conjunto de dados e o sistema o analisa até achar padrões estatísticos que permitem inferir regras e modelos sobre aquele conjunto de dados (CHOLLET, 2017a). A figura 2 exemplifica essa ideia.

Figura 2 – Diagrama Aprendizado Máquina.



Fonte: Elaborada pelo Autor

2.2 Reconhecimento de Padrões

Logo após o processo descrito no tópico 2.1, é necessário analisar os dados e criar métricas para verificar se o modelo corresponde às expectativas. Nesse trabalho, serão utilizados quatro métricas distintas. Considerando a classe X como o resultado positivo, tem-se:

- Verdadeiro Positivo: O modelo previu X e a resposta é X;
- Verdadeiro Negativo: O modelo previu que não é X e a resposta não é X;
- Falso Positivo: O modelo previu X e a resposta não é X;
- Falso Negativo: O modelo previu que não é X e a resposta é X ;

2.2.1 Acurácia

A ideia intuitiva é quanto de verdadeiros e falsos o modelo consegue inferir.

$$A = \frac{\sum Verdadeiros\ Positivos + \sum Verdadeiros\ Negativos}{Numero\ de\ amostras\ Teste} \tag{2.1}$$

2.2.2 Precisão

A ideia é a proporção de positivos classificados corretamente, ou seja, dos classificados como positivos quantos são positivos realmente.

$$P = \frac{\sum Verdadeiros\ Positivos}{\sum Verdadeiros\ Positivos + \sum Falsos\ Positivos} \tag{2.2}$$

2.2.3 Recall

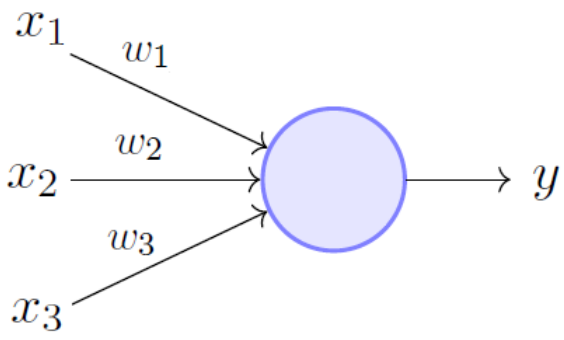
A ideia intuitiva é proporção de positivos identificados corretamente, ou seja, o quão bom o modelo é para detectar positivos.

$$R = \frac{\sum Verdadeiros\ Positivos}{\sum Verdadeiros\ Positivos + \sum Falsos\ Negativos} \tag{2.3}$$

2.3 Redes Neurais Artificiais

O *Perceptron* (ROSENBLATT, 1962) foi o primeiro modelo de rede Neural Artificial que surgiu nos anos 50.

Figura 3 – Exemplo *Perceptron* básico.



Fonte: Minsky e Papert (1969)

.

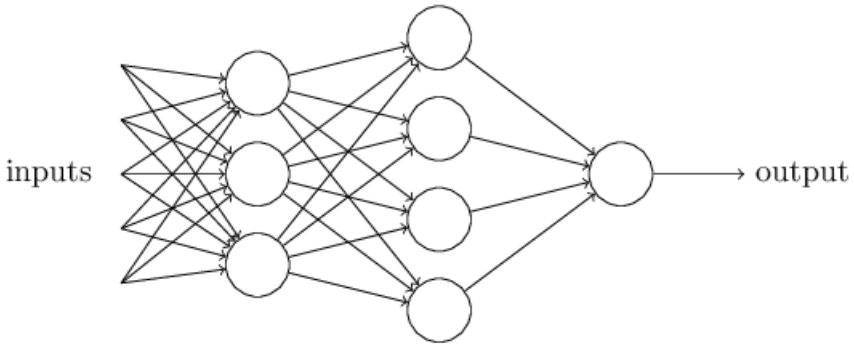
A Figura 3 ilustra o modelo básico de *Perceptron*. Nela, existem três entradas *x1, x2, x3* e uma saída. Para computar a saída são utilizados pesos *w1,w2,w3*, que são aprendidos

pela rede utilizando a técnica *Backpropagation*. O Neurônio tem como a saída *y* como o valor 0 ou 1, de acordo com a Equação 2.4.

$$saída = \begin{cases} 0 \text{ se } \sum_j w_j x_j \leq \textit{limiar} \\ 1 \text{ se } \sum_j w_j x_j \geq \textit{limiar} \end{cases} \tag{2.4}$$

De acordo com Nielsen (2015), o *Perceptron* é um dispositivo que toma decisões balanceando evidências. Variando os pesos e os parâmetros, é possível obter diferentes modelos de tomada de decisão.

Figura 4 – Exemplo *Perceptron* mais sofisticado.



Fonte: Nielsen (2015)

.

A Figura 4 mostra um exemplo mais complexo de *Perceptron*. Quanto maior o número de camadas e neurônios, maior o poder de extração de características do dado de entrada. As últimas camadas são responsáveis por extrair detalhes finos (exemplo de uma imagem: bordas e outras transições abruptas) do dado de entrada que correspondem a altas frequências. A decisão é tomada a partir das características extraídas das camadas anteriores.

2.4 Convolutional Neural Network (CNN)

As Redes Neurais de Convolução (CNN, do inglês *Convolutional Neural Networks*) (LECUN et al., 1998a) são estruturas tradicionais de *Deep Learning*. Uma CNN é constituída por uma ou mais camadas onde espécies de filtros (operações de convolução e amostragem), são aplicados aos dados de entrada (imagens bidimensionais, por exemplo) (CHELLAPILLA; PURI; SIMARD, 2006). O resultado de uma camada inferior serve de entrada para a camada imediatamente superior. Em contraste com as redes neurais totalmente conectadas, de complexa construção e uso, as CNNs compreendem redes de topologia simplificadas, com camadas de

convolução e amostragem, como dito, e camadas opcionais ao topo de nós completamente conectados (CHELLAPILLA; PURI; SIMARD, 2006).

Dada uma imagem bidimensional de entrada, em cada camada, um conjunto de n filtros de convolução K_i, com $i = 1, 2, ..., n$, podem ser aplicados, de modo a obter várias bandas B_i da imagem original (também chamadas de *feature maps*):

$$B_i(p) = \sum_{\forall w \in N(p)} I(w) \cdot K_i(w) \tag{2.5}$$

onde w corresponde a um pixel pertencente à vizinhança de p considerada, isto é, $N(p)$. $I(w)$ corresponde ao valor da imagem na posição w e $K_i(w)$ corresponde ao valor na posição respectiva a w no *kernel* i.

Na realidade, o resultado das operações de convolução pode passar ainda por funções de ativações como a de retificação linear dada por:

$$B_i(p) = max\{B_i(p); 0\} \tag{2.6}$$

Operações de *pooling* espacial (amostragem) também são realizadas a partir dos *feature maps* da imagem original a fim de se obter invariância translacional. Em geral, emprega-se operações de *pooling* máximo:

$$M_i(r) = max_{\forall p \in \beta(r)}\{B_i(p)\} \tag{2.7}$$

onde $\beta(r)$ corresponde à região de *pooling* referente a r, no *feature map* B_i.

Mais operações similares podem ser agregadas. Os resultados (estruturas de dados geradas, isto é, *feature maps*) são passados para camadas superiores da rede. Ao final, tem-se uma representação de alto nível dos dados originais, codificada em um vetor de características numérico Chellapilla, Puri e Simard (2006), Pinto et al. (2009), Bergstra, Yamins e Cox (2013), Menotti et al. (2015).Tal representação pode então alimentar um classificador a fim de identificar o padrão sob análise.

Vale ressaltar, entretanto, que muitas vezes a própria rede CNN pode ser transformada num classificador ao acoplar-se camadas completamente conectadas no topo de neurônios especiais, como unidades *softmax*, por exemplo. Basicamente tais unidades servem para indicar a qual classe pertence um sinal de entrada e, no caso de unidades *softmax*, seguem a equação:

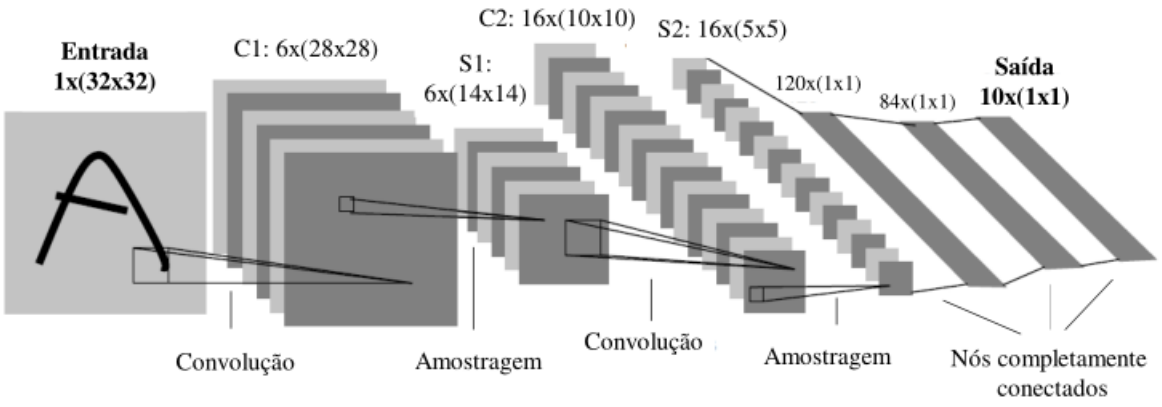
$$S_{max}(u_i) = \frac{e^{u_i}}{\sum_{j=1}^{n} e^{u_j}} \tag{2.8}$$

onde u_i corresponde à soma ponderada dos sinais de entrada no neurônio *softmax* i da rede e n corresponde à quantidade de neurônios *softmax* existentes (classes do problema sendo

tratado). Devido ao denominador ser uma função de partição, a saída dos neurônios *softmax* estará sempre no intervalo $[0; 1]$ e o neurônio cuja ativação for a máxima indica a classe do sinal de entrada.

A Figura 5 ilustra um exemplo de arquitetura CNN de duas camadas (sem a camada de classificação). Conforme pode-se observar, dada uma imagem inicial, operações de convolução e amostragem são aplicadas seguidas por camadas de nós completamente conectados, obtendo-se deste modo um vetor de característica reduzido de alto nível para a mesma.

Figura 5 – Exemplo CNN.



Fonte: (LECUN et al., 1998b)

Conforme observado por (LECUN et al., 1998a), as redes convolucionais apresentam grande robustez a distorções, variações na escala e translação dos objetos nas imagens, justamente devido às operações com que trabalham. Os neurônios que respondem às operações de convolução, por exemplo, por estarem espacialmente distribuídos e compartilharem os mesmos *kernels* na geração de cada *feature map*, acabam detectando as características importantes na imagem mesmo que estas estejam deslocadas (a característica pode ficar transladada, mas continua presente no respectivo *feature map*). Os nós que respondem às operações de *pooling* espacial, acabam por atenuar diferenças de escala e pequenas distorções nos objetos, preservando apenas os aspectos principais do mesmo e o posicionamento relativo das características detectadas na operação de convolução.

Como em outras redes neurais, as CNNs também aprendem por *backpropagation* (LECUN et al., 1998a). Os pesos dos *kernels* de convolução, por exemplo, podem ser vistos como os pesos sinápticos entre os neurônios das camadas da rede, os quais são inicializados e atualizados durante a aprendizagem da rede. Em (MENOTTI et al., 2015), por exemplo, os autores inicializam tais pesos com valores amostrados de uma distribuição uniforme. Uma grande vantagem de tais redes em relação às redes tradicionais completamente conectadas reside no fato de que, devido aos nós compartilharem pesos sinápticos (pesos dos *kernels*), a

quantidade de parâmetros livres do sistema acaba ficando menor, viabilizando o treinamento mesmo com menor quantidade de amostras disponíveis.

2.5 Arquiteturas Baseadas em CNN

Serão utilizadas cinco arquiteturas baseadas em CNN: a *VGG16*, *Inception*, *Xception*, *Resnet* e *InceptionResnet*. Todas utilizarão pesos pré-treinados da base de dados *Imagenet*.

2.5.1 ILSVRC

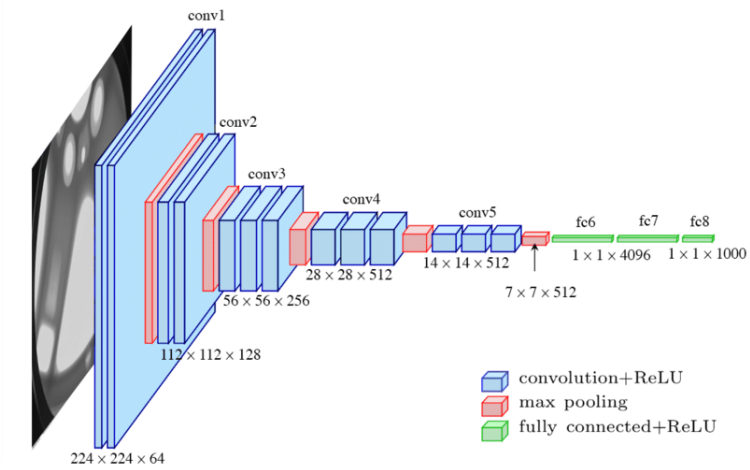
O *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* (RUSSAKOVSKY et al., 2015) é um campeonato de reconhecimento de imagem que existe desde 2010. Em 2012, (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) conseguiram atingir um novo patamar de reconhecimento com suas redes profundas. Desde então, esse campeonato passou a ser utilizado para medir a capacidade de arquiteturas de redes neurais novas. Ele será mencionado mais vezes nessa seção, já que todos os artigos originais das arquiteturas empregadas neste projeto o utilizam para medir sua *performance*.

2.5.2 VGG16

É uma rede neural profunda (muitas camadas) proposta por (SIMONYAN; ZISSERMAN, 2015). A ascensão das técnicas de aprendizado profundo e das redes neurais convolucionais teve como início a arquitetura proposta por (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) durante a competição *ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)*. Os autores tentaram aprimorar a versão proposta em 2012 para garantir resultados melhores de classificação. A entrada possui imagem de tamanho fixo em $224 \times 224 \times 3$ onde respectivamente cada parâmetro representa altura, largura e número de canais. Tal imagem passa por várias camadas de convolução com filtros de pequeno campo de recepção (3×3). Na Figura 6 segue a arquitetura mencionada.

Para avaliar sua robustez, os autores treinaram a rede em diversas configurações, como mostra a Figura 7. As melhores arquiteturas foram a VGG16, composta de 16 camadas, e a VGG19, composta de 19 camadas. Nesse trabalho, a arquitetura utilizada será a VGG16.

Figura 6 – Exemplo da arquitetura VGG16.



Fonte: Simonyan e Zisserman (2015)

Figura 7 – Diferentes configurações da VGG16.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fonte: Simonyan e Zisserman (2015)

2.5.3 Resnet

Uma das ideias centrais da CNN é extrair *features* a cada camada. Logo, uma conclusão obtida a partir da colocação anterior é que, quanto maior o número de camadas, melhor será o resultado. O problema é que, em arquiteturas convencionais, cada camada adicionada

aumenta o percentual de erro na validação, fazendo com que arquiteturas com muitas camadas percam acurácia. Simonyan e Zisserman (2015) criou uma solução para esse problema usando uma rede residual. Cada camada tem uma ligação direta com a próxima, mas também pode "pular" camadas para aumentar o grau de aprendizado. Isso permitiu treinar uma rede com um número muito maior de camadas e conseguir, ainda assim, um resultado melhor. Nesse trabalho utilizamos a *Resnet152V2* (HE et al., 2016).

Nesse trabalho será usada a *Resnet152V2* (HE et al., 2016), uma versão mais nova da *Resnet*. A figura 8 compara as arquiteturas da *VGG16* e da *Resnet*.

Figura 8 – Comparação entre uma rede convencional, *VGG16* e a *Resnet*.

Fonte: He et al. (2016)

2.5.4 Inception

(SZEGEDY et al., 2015) Esta seção será relacionada a rede Inception. Serão descritos a *inception* v1,v2,v3,v4. No trabalho foi utilizada a Inception v3. Isso aconteceu pois a Inception v4 não é suportada pela api Keras. Uma alternativa a isso seria criar o modelo de acordo como o paper e realizar o treinamento nele, contudo as redes comparadas nesse trabalho utilizaram transfer-learning com os pesos pré treinados do ImageNet. O ImageNet é uma base muito

grande e treinar a Inception v4 requer recursos computacionais que não estão disponíveis para o autor desse trabalho. Além disso, foi constatado nos experimentos dos autores da InceptionV4 que ela possui a mesma performance de reconhecimento do que a InceptionResnet, utilizada aqui

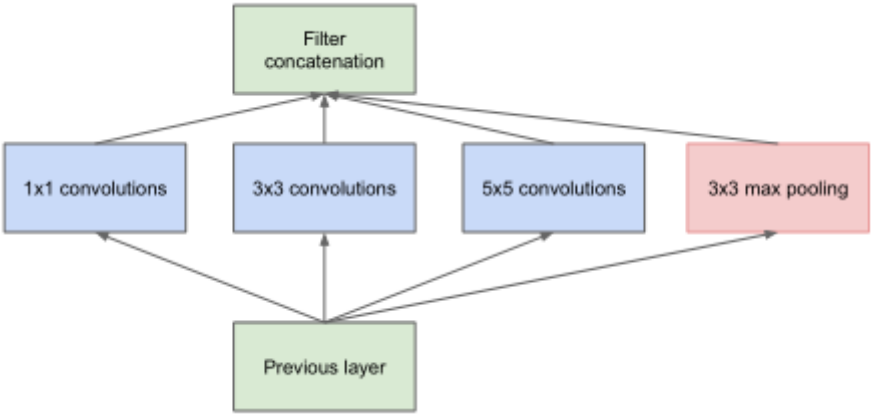
2.5.4.1 Inception V1

A Inception é uma rede desenvolvida em 2014. (SZEGEDY et al., 2014). Essa rede foi um novo avanço no desenvolvimento de redes CNN. Antes dela, a única forma de melhorar a performance de uma rede era aumentando o tamanho, a profundidade ou a largura das camadas. Essa perspective tem dois problemas:

- O aumento de tamanho normalmente significa que também vai aumentar o numero de parâmetros. O que torna a rede mais propicia a overfitting, especialmente em bancos de dados com poucas amostras
- Outro problema é que ao aumentar o tamanho da rede, também aumenta o custo computacional de treinar a rede. A solução para esses dois problemas seria mudar de *full connected layers* para *sparsed connected layers*, até mesmo dentro das convoluções. O problema dessa solução é que os algoritmos numéricos são muito ineficientes para calcular operações em matrizes esparsas
- É difícil definir um tamanho do *Kernel* eficiente, pois a informação que se pretende encontrar não tem sempre o mesmo tamanho

A solução inicial para esse problema seria colocar mais de um filtro na mesma camada, fazendo a rede crescer em largura. A figura 9 mostra a implementação simples dessa ideia que foi chamada de *Inception Module* Nela há três filtros: um de 1×1, um do 3×3 e um de 5×5 com um *max pooling* de 3×3 , executados na mesma camada. Seu resultado é concatenado em um filtro final do módulo.

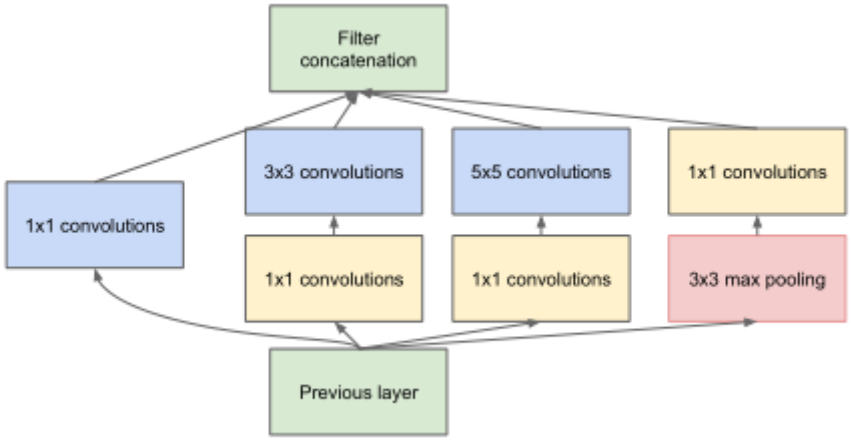
Figura 9 – Inseption Básica 1



Fonte: Szegedy et al. (2014)

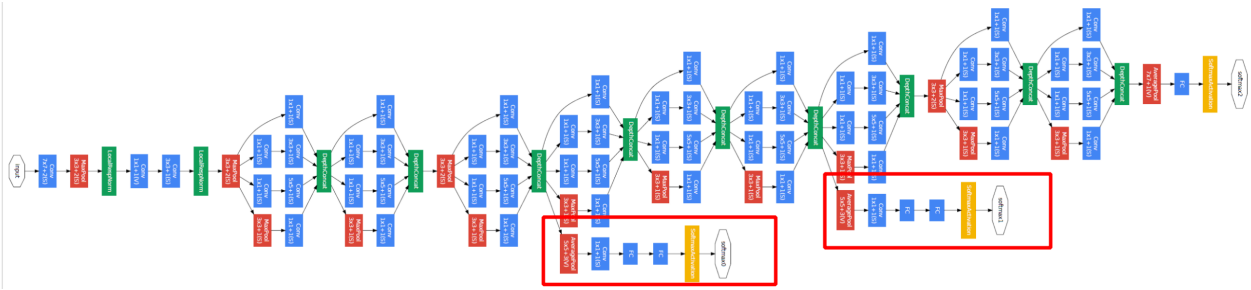
Como mencionado anteriormente, uma das ideias da *Inception* é gerar uma rede cujo custo computacional não seja muito alto. Juntar filtros de 3×3 , 5×5 e uma *max pooling* de 5×5 na mesma linha da camada gera um custo alto de processamento. Para otimizar isso os autores sugerem uma arquitetura que aplica filtros 1×1 antes dos filtros 3×3 e de 5×3 e depois do *max pooling*. Isso aumenta o desempenho da rede ao diminuir o número de canais de entrada que vão chegar aos filtros. A figura 10 mostra essa ideia.

Figura 10 – Inseption Básica 2



Fonte: Szegedy et al. (2014)

Figura 11 – Google Lenet



Fonte: Szegedy et al. (2014)

Com os *Inceptions modules*, a equipe responsável criou a rede que ficou conhecida como Google Lenet. Como mostra a figura 11. Os autores previram que a rede poderia não produzir mais resultados depois da metade e, para esses casos, eles adicionaram dois classificadores auxiliares para a saída de dois dos *Inceptions Modules*, atualizando a função de custo final para contemplar esses classificadores

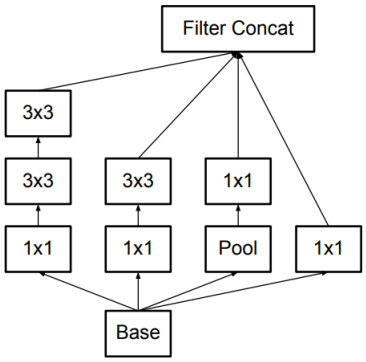
2.5.4.2 Inception V2

A *InceptionV2* (SZEGEDY et al., 2015) é uma nova versão da *Inception* projetada para aumentar a acurácia e diminuir o custo computacional da *InceptionV1*. Os autores perceberam que podiam melhorar a *Inception* de duas formas:

- Redes neurais possuem melhor desempenho quando as convoluções não alteram seu tamanho original drasticamente, pois isso pode causar parda de informação.
- Poderiam ser feitas fatorações pontuais que tornariam as convoluções mais performáticas em relação ao custo computacional

A proposta solução desenvolvida em etapas: a primeira foi fatorar a convolução de 5×5 para duas convoluções de 3×3. Os autores argumentam que uma convolução de 5×5 tem o custo computacional 2,78 vezes maior do que uma de 3×3, o que significa que duas de 3×3 seriam mais rápidas do que uma de 5×5

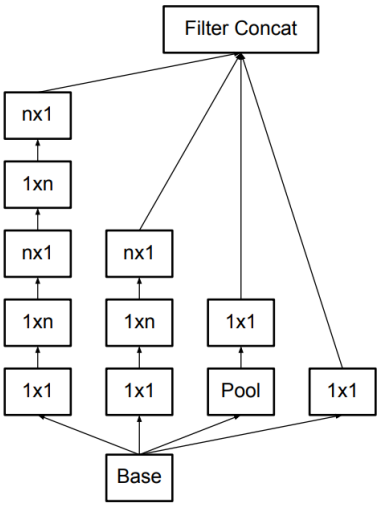
Figura 12 – Inception v2 1



Fonte: Szegedy et al. (2015)

Depois disso, os autores expandiram a ideia e fatoraram todas as convoluções de tamanho $n \times n$ para convoluções de tamanho $1 \times n$ e $n \times 1$. Por exemplo: uma convolução de 3×3 seria equivalente a primeiro realizar uma convolução de 1×3 e depois uma de 3×1 na saída da primeira. Eles descobriram que isso poderia diminuir em até 33% o custo computacional dessas convoluções, a figura 12 ilustra essa ideia.

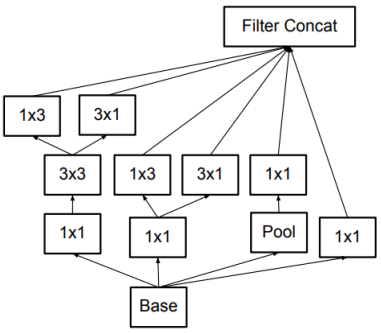
Figura 13 – Inception v2 2



Fonte: Szegedy et al. (2015)

A terceira ideia foi aumentar a largura e diminuir a profundidade da rede. Como mencionado anteriormente, aumentar a profundidade da rede pode culminar em perda de informações. A figura 13 mostra essa ideia.

Figura 14 – Inception v2 3



Fonte: Szegedy et al. (2015)

2.5.4.3 Inception V3

Os autores da *Inception* perceberam que os classificadores auxiliares, adicionados no meio da rede, sendo mencionados na seção da *InceptionV1*, não contribuíam muito até o fim do processo de treinamento, quando a acurácia estava prestes a atingir o limite. Eles dizem que esses classificadores funcionam como reguladores. A solução para esse problema é, além das melhorias da *v2*, adicionar:

- Usar o *RMSProp* como otimizador
- Adicionar convoluções fatoráveis de 7×7
- Adicionar normalização de lote nos classificadores auxiliares
- Adicionar um regularizador na fórmula de perda para impedir a rede de atingir *overfitting*

2.5.5 InceptionResnet

(SZEGEDY et al., 2016) A ideia central da *InceptionResnet* (SZEGEDY et al., 2016) é adicionar um módulo residual no fim das convoluções dos módulos *Inceptions*. Para isso funcionar, é necessário que as convoluções possuam a mesma dimensão de entrada e saída. Sendo assim, foi adicionado uma convolução *point wise* que consiste no uso de *kernels* de tamanho 1×1 no final das convoluções. Além disso, a operação de *pooling* foi substituída pelas conexões residuais.

Conforme os autores, redes neurais profundas com mais de mil camadas não são eficientes. Sendo assim, os autores aumentaram a ativação residual de 0,1 para 0,3.

Os experimentos demonstraram que os módulos residuais não conseguem aumentar a acurácia final de treinamento, porém, aumentou a velocidade de treinamento, garantindo acurácias mais altas em menos épocas.

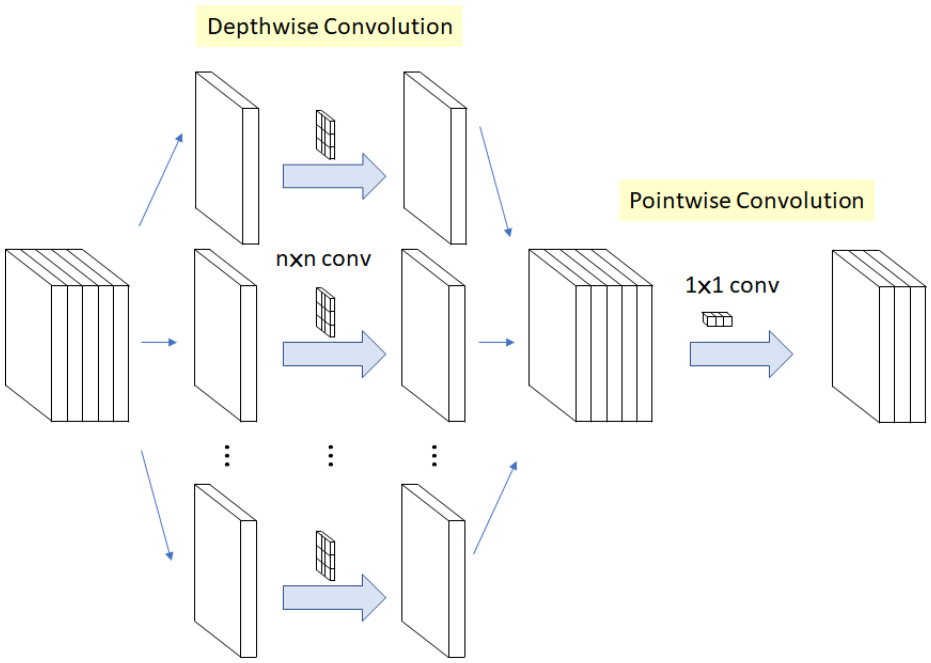
2.5.6 Xception

O Nome *Xception* vem de *Extreme Inception* (CHOLLET, 2017b) e a grande inovação foi utilizar uma *depthwise separable convolution*, garantindo melhores acurácias. Uma *depthwise separable convolution* é uma *depthwise convolution* seguida de uma *pointwise convolution*.

Depthwise convolution é uma convolução orientada a *channels*. Dado uma CNN de dimensão $n \times n$ com cinco *channels*, serão feitas cinco convoluções $n \times n$. Na Figura 15, há cinco *channels*, logo, teremos cinco convoluções $n \times n$

A *pointwise convolution* é uma convolução de 1×1 que muda a dimensão. A diferença entre esse tipo de convolução para a convolução normal é que não há necessidade de aplicar a convolução em todos os canais de uma vez, cada convolução é aplicada separadamente nos *channels*, fazendo com que a rede a fique com menos conexões, o que gera um modelo mais leve.

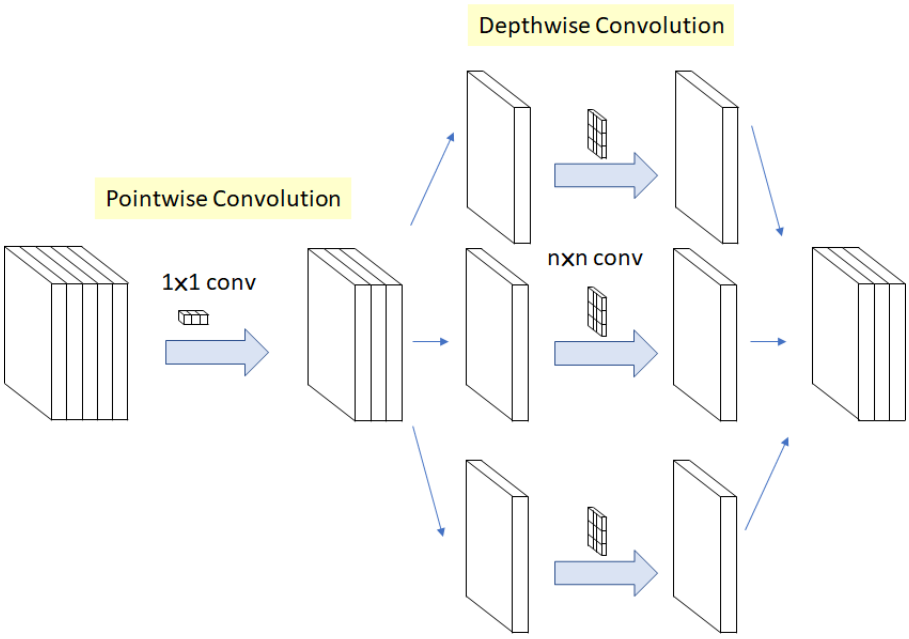
Figura 15 – *Depthwise Separable Convolution*.



Fonte: Tsang (2021)

Na *Xception* o modelo é diferente. Primeiro, é feita a convolução *pointwise*. Depois é feita a *Depthwise*. Isso com base na *InceptionV3*, em que é aplicado uma convolução de 1x1 antes de qualquer convolução $n \times$n . A figura 16 exemplifica essa ideia:

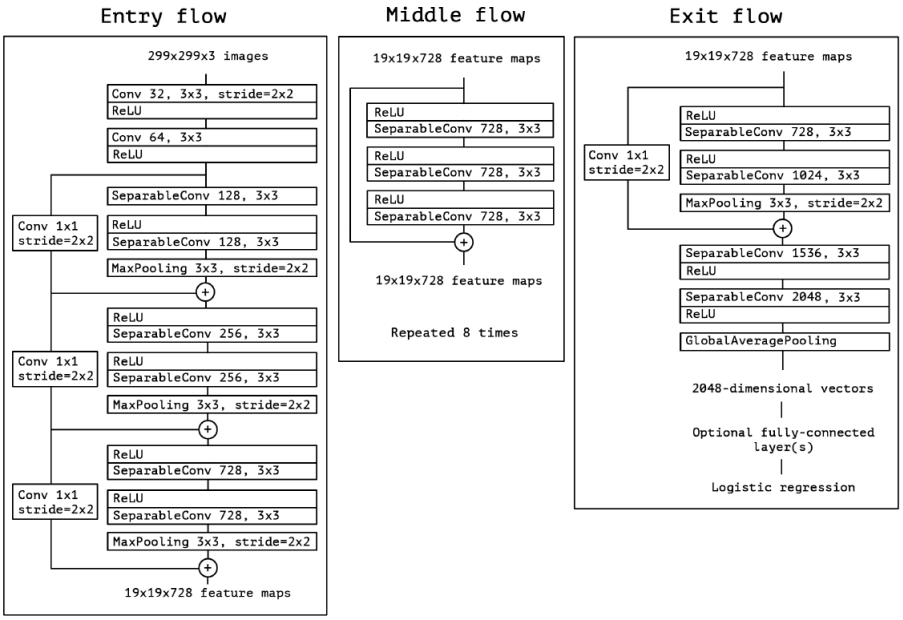
Figura 16 – *Depthwise Separable Convolution* Modificada.



Fonte: Tsang (2021)

Uma outra diferença entre a *Xception* e a *Inception* é que os *Inception Modules* possuem um ativador não-linear *ReLU* depois da primeira operação, e a *Xception* não possui nenhum ativador intermediário.

Figura 17 – Arquitetura *Xception*.



Fonte: Chollet (2017b)

A Figura 17 mostra a arquitetura da *Xception*. *SeparableConv* é a *DepthwiseSeparable*

Convolution. Nessa arquitetura, eles são tratados como *Inception Modules*. Além disso, também são encontradas conexões residuais.

3 Metodologia

3.1 Ferramentas

Nesta etapa será descrita as ferramentas utilizadas nesse trabalho.

3.1.1 Python

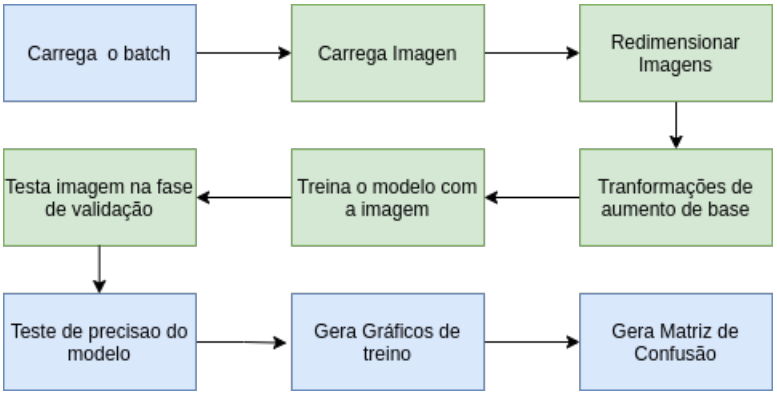
O *Python* é uma linguagem de programação dinâmica, criada por Guido van Rossum. Sua principal característica é ter uma alta legibilidade de código e facilidade de aprendizado de uso, possibilitando a criação de uma vasta gama de bibliotecas. Esses atributos fizeram com que o *Python* se tornasse a linguagem mais usada e recomendada para Aprendizado de Máquina e computação científica. (AYER; MIGUEZ; TOBY, 2014)

3.1.2 Tensorflow

O *Tensorflow* (TENSORFLOW,) é uma plataforma completa para o desenvolvimento de projetos com aprendizado de máquina, criada pela área de desenvolvimento e pesquisa do *Google Brain*. Foi lançada oficialmente em 2017, e hoje é uma das plataformas de AM mais utilizadas. A escolha do *Tensorflow* para esse projeto se deu pela rápida curva de aprendizado e pela quantidade de recursos disponíveis. Dentro da plataforma existe uma sub-biblioteca chamada *Keras* (KERAS. . . , 2018), responsável por tornar o desenvolvimento ainda mais dinâmico. A arquitetura do *Tensorflow* é pensada em *pipeline* de acordo com a Figura 18.

A Figura 18 descreve todas as etapas realizadas neste trabalho. A primeira etapa é o carregamento do lote (do inglês *batch*) de imagens que serão treinadas. Na Figura 18, os elementos caracterizados em verde demonstram a fase de *pipeline* dessa biblioteca. As figuras em azul representam as fases em que todas as imagens são utilizadas de uma vez. Na fase verde o processo é diferente. Cada imagem vai de um estágio para o outro por vez até chegar na fase de validação. Depois, as outras imagens passam pelo mesmo processo até todas as imagens acabarem.

Figura 18 – Etapas *Tensorflow* .



Fonte: Elaborada pelo autor.

3.1.3 Jypter Notebooks

Os *Jypter Notebooks* (JUPYTER,) são uma aplicação *web* que permite a criação de documentos com códigos de programação em tempo real. Neles, é possível visualizar imagens, equações e gráficos em tempo real, enquanto o código é interpretado. Normalmente, já possui bibliotecas de computação numérica instaladas para agilizar o desenvolvimento. São muito populares para projetos de aprendizado de máquina porque permitem ao desenvolvedor acompanhar o treinamento do modelo em tempo real.

3.1.4 Google Colab

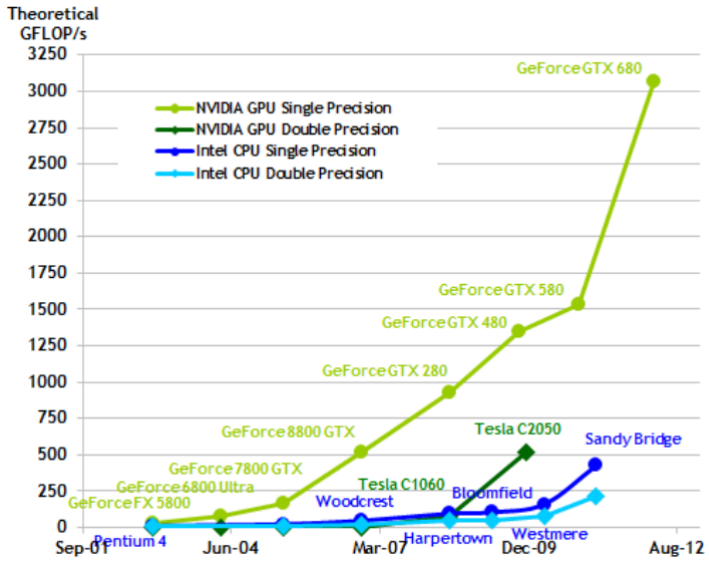
Treinamentos com um alto número de imagens demandam muito tempo de processamento para serem efetivados. Para otimizar o processo, são usadas GPU. Devido ao alto custo de *Hardware* envolvido com o treinamento usando GPU ou mesmo CPU, é difícil obter um treinamento rápido e eficiente com computadores domésticos. As placas de vídeo especializadas para AM são extremante caras e consomem bastante energia elétrica. Para minimizar esse problema, o Google desenvolveu a plataforma *Colabs* (GOOGLE. . . ,). Com apenas uma conta Google, qualquer pessoa pode ter acesso a um *Notebook* com acesso a uma GPU de alta potência. Os experimentos foram realizados nesta plataforma.

3.1.5 Cuda

O Cuda é uma plataforma desenvolvida pela Nvidia para o uso de GPU no treinamento de AM. A plataforma foi desenvolvida com o objetivo de tornar a programação com GPU simples a qualquer programador (GARLAND et al., 2008). Problemas de AM são altamente paralelizáveis, o que significa que GPUs conseguem ter uma performance de treinamento consideravelmente superior a CPU. Com o Cuda, qualquer computador com placa de vídeo Nvidia pode usar GPU para todo tipo de processamento. O *Python* e o *Tensorflow* têm suporte

para essa tecnologia. Pesquisadores também puderam observar que GPUs conseguem entregar mais potência de processamento do que CPUs atualmente, e isso vem aumentando nos últimos anos (NAVARRO; HITSCHFELD-KAHLER; MATEU, 2014). Como mostra a Figura 19. Ela mostra a *performance* de CPUs e GPUs ao lungo dos anos, e , mostra que as GPUs estão proporcionando muito mais poder do que as GPUs

Figura 19 – CPU vs GPU.



Fonte: Navarro, Hitschfeld-Kahler e Mateu (2014)

3.1.6 Git

O *Git* é uma ferramenta desenvolvida por Linus Trovalds para controlar e versionar arquivos. Foi originalmente proposta para ser utilizada no desenvolvimento do Núcleo do *Linux*, mas hoje é amplamente usada no mercado e tornou-se uma das ferramentas mais importantes no mundo.

3.2 Experimentos

A base de dados utilizada foi a *WikiArt Dataset* (SALEH; ELGAMMAL, 2015). Até o momento, é a maior base de dados de pinturas de arte do mundo (TAN et al., 2016). Contém em torno de 80,000 pinturas de arte, variando do século quinze aos tempos modernos. Esta base de dados contém 27 estilos e 45 gêneros diferentes. Para esse trabalho, foram escolhidos os artistas com mais de 500 imagens, o que resultou em 23 artistas e 19,006 imagens A base de treinamento foi composta por 70%, totalizando 13,294 amostras. Já para validação e teste utilizamos 15%, totalizando 2,856 amostras para cada etapa.

Foi utilizado aumento de base nas amostras para evitar o problema de sobreajuste (do

inglês *overfitting*). Este termo é utilizado em AM para descrever quando uma rede neural se ajusta muito bem às amostras da base de treinamento e se mostra ineficaz em predizer resultados quando considerado amostras não observadas pela rede como do conjunto de teste. Foram utilizados 5 tipos de aumento de bases: redimensionamento, rotação, cisalhamento, giro horizontal, giro vertical. A ideia é que a mesma imagem pode ser treinada de formas diferentes, causando a modificação de pesos diferente e melhorando o resultado da rede. No Tensorflow, o *framework* usa uma imagem diferentes para cada iteração de treinamento. O número de imagens continua o mesmo, mas elas são treinadas de outra forma.

Também foi utilizado o método de aprendizado por transferência (do inglês *Transfer Learning)*, que é uma técnica de aprendizado de máquina em que um modelo treinado em uma tarefa é reaproveitado em uma segunda tarefa diferente. O algoritmo de otimização utilizado foi o Adam. Redes neurais convolucionais aprendem utilizando equações como a equação 2.4. A cada iteração o mínimo dessa equação é calculado utilizando o método do gradiente. O problema é que o número de pesos normalmente é muito grande, causando um custo computacional muito grande para se achar o mínimo. Para lidar com esse problema, o método do gradiente estocástico é utilizado. O otimizador é como esse algoritmo é implementado, cada um varia conforme as otimizações e os processamentos vetoriais realizados.

Os treinamentos foram realizados com for tamanho do *batch* foi 64, *learning rate* inicial de 10^{-4} e ajustado pelo *framework* durante o treinamento. Os experimentos foram executados três vezes para extrair a média e desvio padrão, já que a rede neural é estocástica. A função de custo foi a *Categorical Cross-Entropy*, que consiste em aplicar um ativador *SoftMax* antes da função de custo *Cross-Entropy*.

4 Resultados

São apresentados os resultados obtidos compostos da acurácia, a média das acurácias e gráficos de progresso da etapa de treinamento.

Aqui são mostrados os resultados encontrados nos treinamentos. Estão presentes: o resultado do teste de acurácia de cada iteração, a média final de acurácia e os desvios padrão de cada arquitetura. Os gráficos e o resultado por classe exibidos referem-se à melhor iteração de cada categoria de rede.

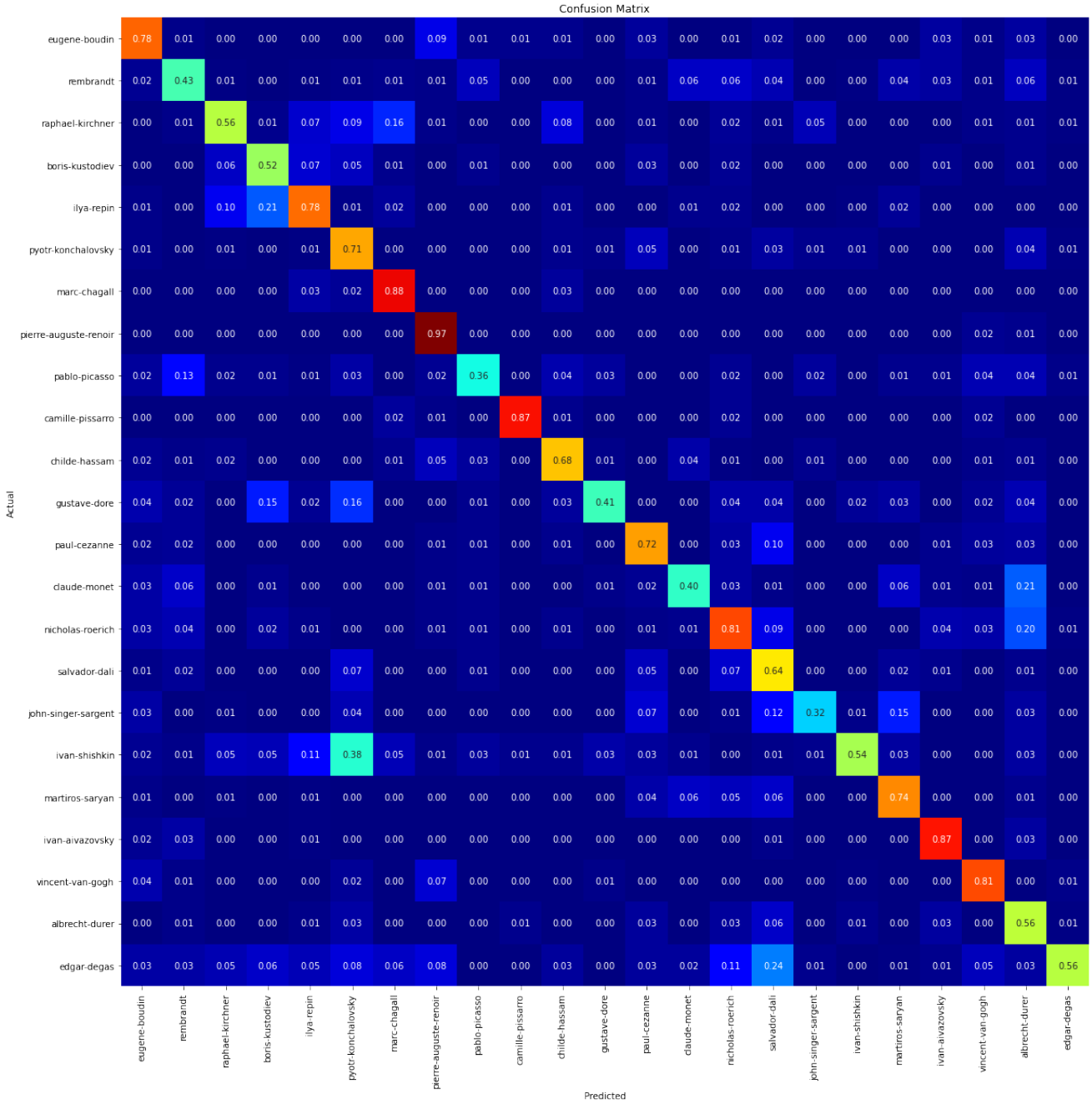
Tabela 1 – Resultados dos experimentos por acurácia.

Critério	1ª Iteração	2ª Iteração	3ª Iteração	Média	Desvio Padrao
VGG16	79,86%	81,28%	76,45%	79.20%	2.02
Resnet	82.06%	88.17%	93.99%	88.07%	4,877
Inception	98.15%	98.22%	97.83%	98.07%	0.16
Inception Resnet	97.05%	96.62%	97.15%	96.94%	0.22
Xception	91.43%	89.25%	96.87%	92.52%	3.2

Fonte: Elaborado pelo autor

O resultado da VGG16 é positivo, considerando o fato de ser a rede mais simples utilizada nesse trabalho e que possui apenas 16 camadas. Pode-se destacar alguns detalhes do resultado. O primeiro é que a precisão do Albrecht-Durer foi muito baixa (40%). Isso pode ser devido ao fato que, de todos os pintores analisados, ele é um dos que contêm pinturas com mais detalhes. A Figura 22 exemplifica esse argumento. Uma rede com filtros maiores poderia trazer resultados mais satisfatórios para esse pintor. Outro destaque é que a rede se confundiu bastante entre os pintores Ivan-Shishkin e Pyotr-Konchalovsky, como é mostrado na Figura 20. Isso pode ser atribuído ao fato de o banco de dados utilizado nesse trabalho possuir diversas pinturas de campos e paisagens dos dois pintores. A VGG16 precisaria de fotos mais diversificadas desses dois pintores para aumentar a diferença entre eles. O Gráfico de treinamento mostra que a VGG16 levou por volta de 20 épocas para atingir o resultado. O desvio padrão da *VGG16* foi baixo, considerando que foi a rede que teve a menor média de acurácia.

Figura 20 – Matriz Confusão VGG16.



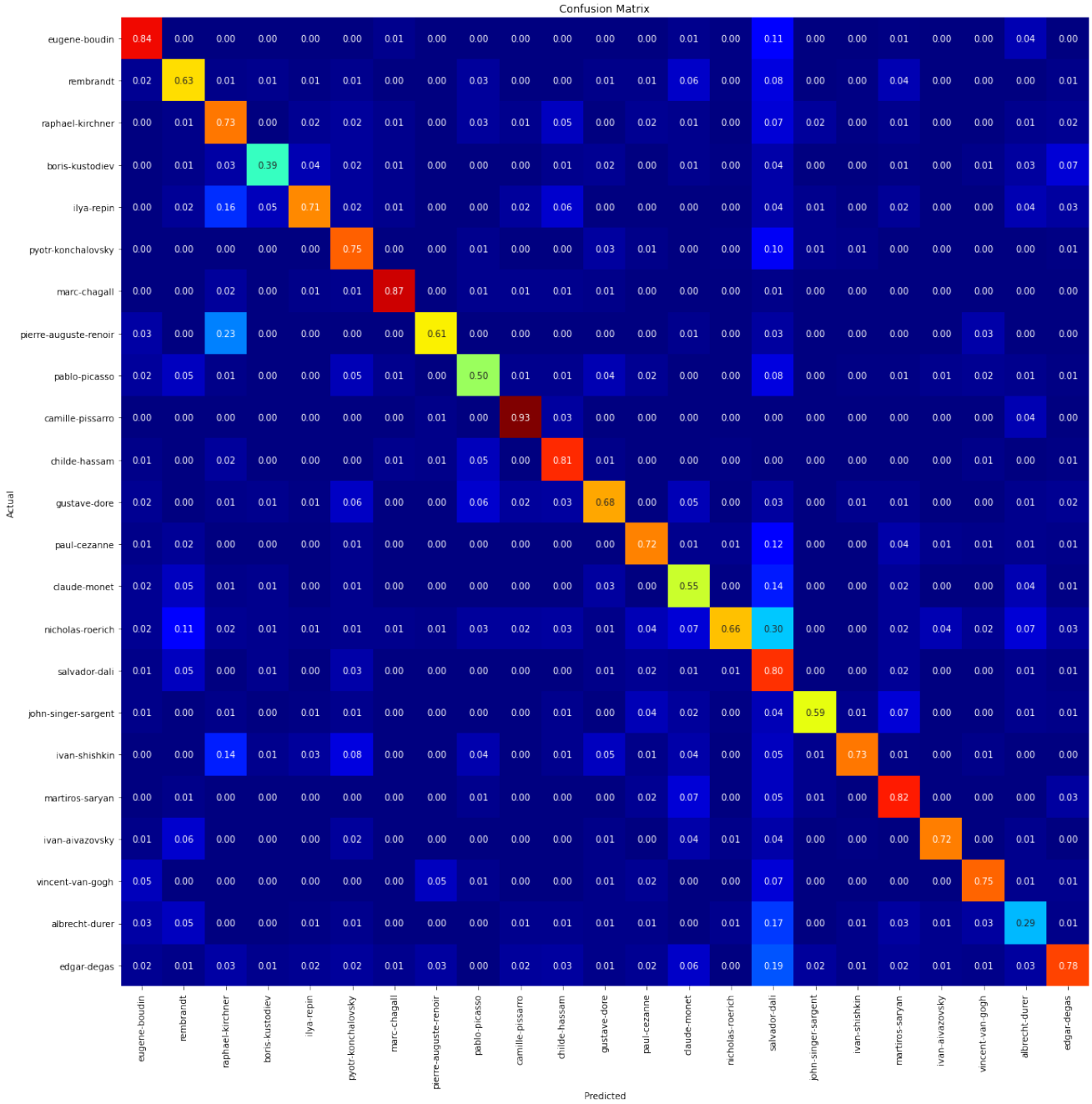
Fonte: Elaborado pelo autor.

A Figura 20 ilustra a matriz de confuão da VGG16. Nela, pode-se observar a confusão mencionada anteriormente e também varias outras confusões com os pintores: Salvador Dali, Monet e Nicholas Roerich. A quantidade de confusão ilustrada na Figura mostra o resultado inferior da VGG16 em relação as outras redes. É nítido que a VGG16 gerou mais confusão que as outras redes.

A *Resnet* apresentou um resultado melhor do que a VGG16. Os módulos residuais, junto

com o aumento do número de camadas, permitiram aumentar por volta de 10% o resultado da *Resnet*. Observando a matriz de confusão e os resultados, pode-se observar os mesmos padrões do que a VGG16 de forma menos acentuada. Albrecht-Durer continua sendo a classe com menos precisão. Nessa rede, pode-se observar que a maioria dos falsos-positivos ocorreu com o pintor Salvador Dali. O alto número de camadas da rede pode ter feito o modelo ver partes do surrealismo em todos os pintores. Em especial, o maior número de confusões ocorreu com o pintor Nicholas Roerich, que possui um estilo de pinturas semelhantes ao Salvador Dali. A *Resnet* também levou por volta de 20 épocas para treinar o modelo. A *Resnet* teve o maior desvio de todas as redes, mostrando que os módulos residuais nem sempre conseguem atingir valores altos de acurácia, principalmente ao comparar o resultado da *InceptionResnet* com a *Inception*, onde a *InceptionResnet* teve um resultado um pouco pior que o da *Inception*.

Figura 21 – Matriz Confusão *Resnet*.



Fonte: Elaborado pelo autor.

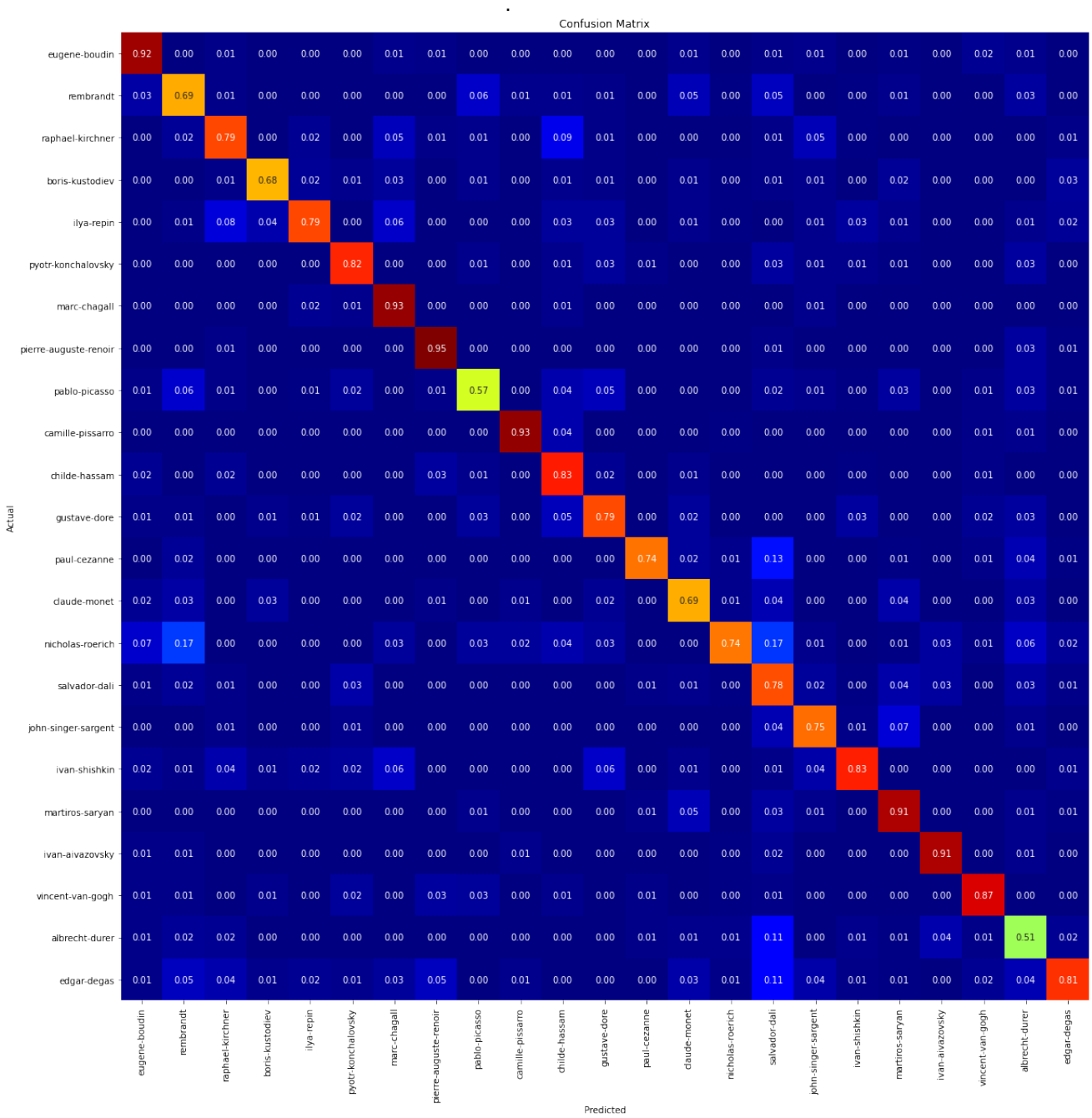
Observando a matriz de confusão da *Resnet* representada pela Figura 21, fica nítido que a rede se confundiu muito com a classe representada por Salvador Dali. Mesmo assim, pode-se observar menos confusão do que a VGG16.

Figura 22 – Pintura Albrecht-Durer.

Fonte: Banco de dados *Wiki Art*

A *Inception* foi a arquitetura que obteve os melhores resultados. Uma possível interpretação é que filtros com tamanho estático não são adequados para analisar obras de arte. O padrão buscado para se extrair a *feature* de uma pintura tem diversos tamanhos diferentes. Diferente de um classificador que procura objetos em uma imagem e que os objetos normalmente possuem tamanhos semelhantes. A grande diferença da *Inception* é que ela possui vários tamanhos de filtros, podendo extrair *features* de diversos tamanhos. Até mesmo a classe Albrecht-Durer, que foi o pior resultado das outras redes, teve um resultado melhor do que 50%. Além disso, a *Inception* foi uma rede que aprendeu em menos épocas, por volta de 15, do que as redes anteriores. Isso mostra o aumento de performance que os autores estavam buscando ao criar a arquitetura. No desvio, a *Inception* também mostrou o melhor resultado, confirmando ser a rede mais adequada para o problema.

Figura 23 – Matriz Confusão *Inception*.

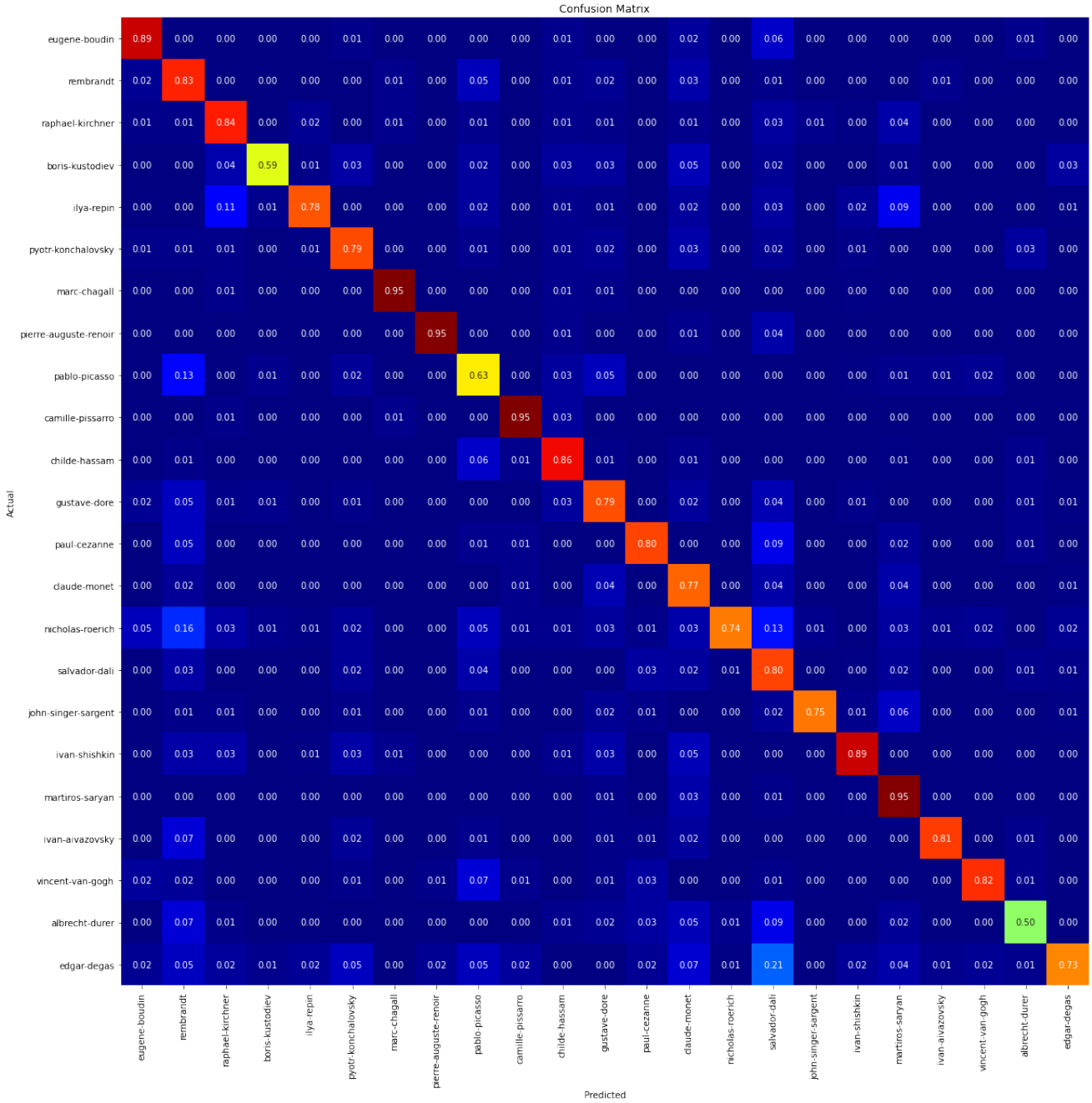


Fonte: Elaborado pelo autor.

A Figura 23 mostra a matriz de confusão da *Inception*. Pode-se observar uma confusão mínima com a classe representada pelo pintor Salvador Dali. Além disso, fica evidente que é a rede que propiciou o melhor resultado com o mínimo de confusão.

A *InceptionResnet* teve um resultado bom, mas não tão bom quanto o da *Inception*. Isso mostra que os módulos residuais não são tão adequados para esse problema e que a arquitetura da *Inception* é a melhor opção.

Figura 24 – Matriz Confusão *InceptionResnet*.



Fonte: Elaborado pelo autor.

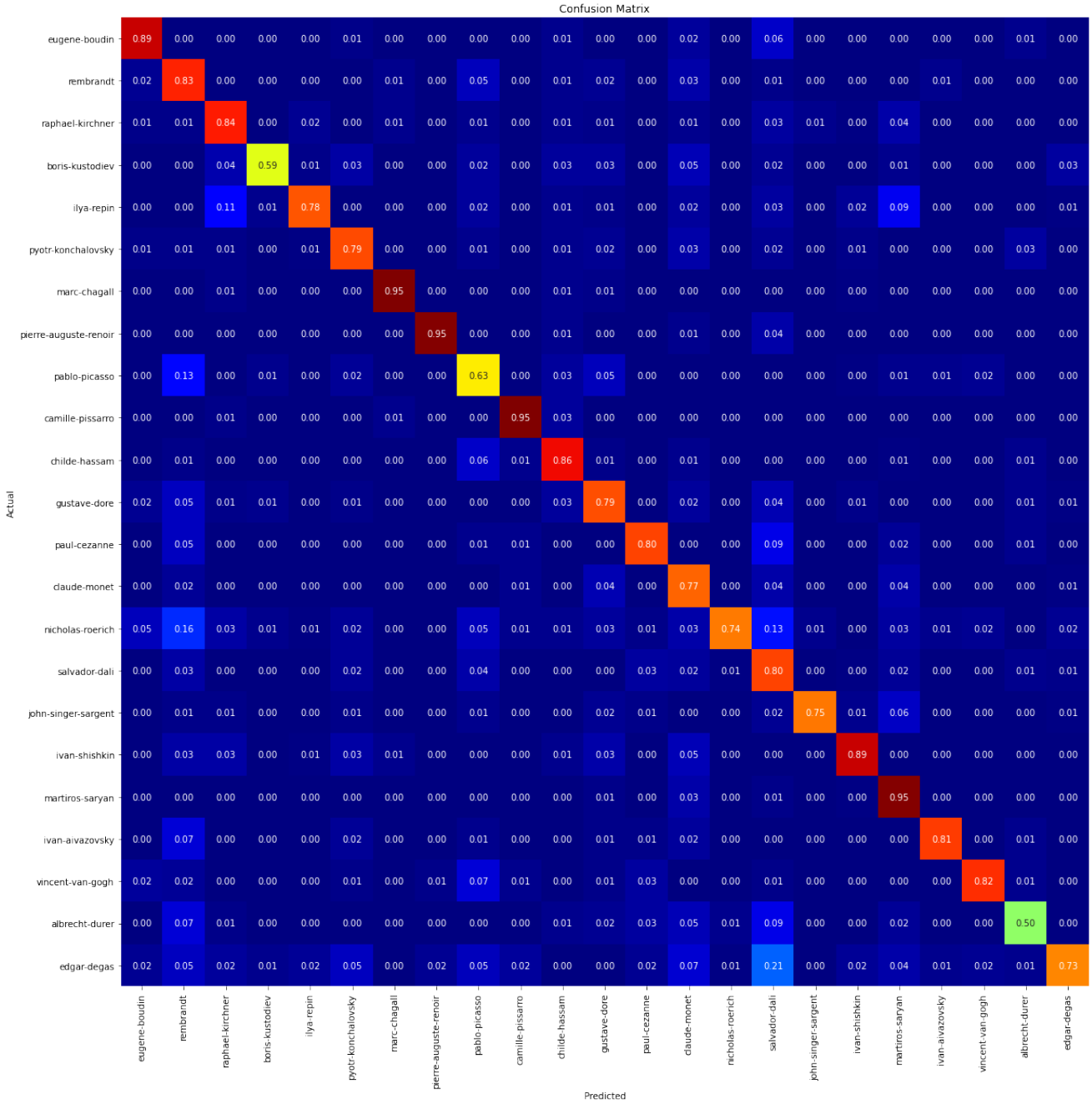
A Figura 24 representa a matriz de confusão do *InceptionResnet*. Nela, também pode-se observar que a classe representada pelo pintor Salvador-Dali foi a classe com mais confusão. Contudo, diferente da *Resnet* convencional, a classe representada pelo pintor Rembrandt também mostrou bastante confusão. Apesar disso, a Figura também mostra que foi uma rede com um resultado muito bom.

O resultado da *Xception* é um resultado bom, se olhado isoladamente, mas ruim se

comparado às outras redes. Do ponto de vista de desempenho, essa rede conseguiu atingir o mínimo de *loss* no *validation* antes da 15.ª época. O resultado mostra que, apesar de filtros de múltiplos tamanhos propiciarem resultados muito bons, há um limite para isso. Ao criar muitos filtros de tamanho $n \times n$, a *Xception* não conseguiu trazer os mesmos resultados que a *Inception* ou a *Inception Resnet*.

Como modelo geral, ela conseguiu trazer um resultado satisfatório para quase todas as classes. Mesmo a classe Albrecht-Durer, que não teve resultados muito altos nas redes mencionadas anteriormente, conseguiu chegar perto do 67% nesta rede.

Figura 25 – Matriz Confusão *Xception*.



Fonte: Elaborado pelo autor.

A Figura 25 representa a matriz de confusão da *Xception*. Nela, pode-se observar que a classe representada pelo pintor Rembrandt foi a classe que a rede mais se confundiu. Além disso, também pode-se observar que a classe representada pelo Salvador Dali, como nas outras redes, também gerou confusão.

5 Conclusão

Esse trabalho teve como proposta classificar obras de arte. Seu objetivo foi analisar e implementar diferentes arquiteturas de Redes Neurais Convolucionais, analisar a viabilidade da técnica de transferência de aprendizado e contribuir para o assunto. Para isso, cinco diferentes arquiteturas de CNN foram utilizadas: *VGG16, Resnet, Inception, InceptionResnet* e *Xception*. Nos resultados, pode-se observar que o método de transferência de aprendizado combinado com a *Inception* obteve os melhores resultados 98.07% de acurácia. Isso demostra a capacidade da *Inception* em extrair características das diferentes classes devido aos filtros de vários tamanhos diferentes, que permitem extrair diferentes tipos de características. Classificar obras de arte não é uma tarefa simples e usar filtros com tamanho fixo não proporciona a mesma acurácia que o modelo da *Inception* pode garantir. No desvio padrão, a *Inception* também foi a rede com melhor desempenho, próximo de zero, mostrando mais uma vez que é a rede mais adequada para o problema proposto.

Com o resultado da Inception, é possível auxiliar e ajudar curadores do mundo todo. Pode diminuir consideravelmente o tempo de catalogação de exposições e acervos artísticos. 98% de precisão de *features* extraídas em uma base de dados de mais de 19,000 imagens e pode gerar nível precisão de classificação que nem mesmo especialistas em obras de arte conseguiriam ter. O mesmo modelo pode inferir vários artistas diferentes, o que precisaria de vários especialistas diferentes para saber de quem é a obra.

5.1 Trabalhos Futuros

Devido à falta de recursos computacionais, foram escolhidos os artistas com mais obras de arte da base a fim de garantir o melhor resultado. Contudo, poderia ser feito os mesmos tipos de treinamento com a base inteira, com suas mais de 80,000 obras de arte. Além disso, poderiam ser feitas novas iterações de experimentos para garantir um resultado estatisticamente mais preciso.

Referências

AYER, V. M.; MIGUEZ, S.; TOBY, B. H. Why scientists should learn to program in python. *Powder Diffraction*, Cambridge University Press, v. 29, n. S2, p. S48–S64, 2014.

BARTOLINI, F.; CAPPELLINI, V.; MASTIO, A. D.; PIVA, A. Applications of image processing technologies to fine arts. In: SALIMBENI, R. (Ed.). *Optical Metrology for Arts and Multimedia*. SPIE, 2003. v. 5146, p. 12 – 23. Acesso: 07-06-2021. Disponível em: <https://doi.org/10.1117/12.504630>.

BERGSTRA, J.; YAMINS, D.; COX, D. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: *Anais da International Conference on Machine Learning*. Etados Unidos: JMLR, 2013.

CETINIC, E.; LIPIC, T.; GRGIC, S. Fine-tuning convolutional neural networks for fine art classification. *Expert Systems with Applications*, v. 114, p. 107–118, 2018. ISSN 0957-4174. Acesso: 07-06-2021. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417418304421>.

CHELLAPILLA, K.; PURI, S.; SIMARD, P. High performance convolutional neural networks for document processing. In: *Anais da International Workshop on Frontiers in Handwriting Recognition*. França: CCSD, 2006.

CHOLLET, F. *Deep Learning with Python*. [S.l.]: Manning, 2017. Acesso: 07-06-2021. ISBN 9781617294433.

CHOLLET, F. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017.

GARLAND, M.; GRAND, S. L.; NICKOLLS, J.; ANDERSON, J.; HARDWICK, J.; MORTON, S.; PHILLIPS, E.; ZHANG, Y.; VOLKOV, V. Parallel computing experiences with cuda. *IEEE Micro*, v. 28, n. 4, p. 13–27, 2008.

GOOGLE COLAB. Acesso: 22-04-2021. Disponível em: <https://colab.research.google.com/>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. *Identity Mappings in Deep Residual Networks*. 2016.

JUPYTER. Acesso: 22-04-2021. Disponível em: <https://jupyter.org>.

KERAS Documentation. 2018. Disponível em: <<https://keras.io/>>. Acesso: 07-06-2021.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Acesso: 07-06-2021. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Anais da IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [s.n.], 1998. v. 86, n. 11, p. 2278–2324. Acesso: 07-06-2021. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665>.

MARILUNGO. *Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)*. 1998. Acesso: 07-06-2021. Disponível em: <https://write365project.wordpress.com/2018/06/08/day-159-history-of-art-explained-by-the-avant-garde-fly-killer>.

MENOTTI, D.; CHIACHIA, G.; PINTO, A.; SCHWARTZ, W. R.; PEDRINI, H.; FALCAO, A. X.; ROCHA, A. Deep representations for iris, face, and fingerprint spoofing detection. *IEEE Transactions on Information Forensics and Security*, v. 10, n. 4, p. 864–879, 2015.

MINSKY, M.; PAPERT, S. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969. Acesso: 07-06-2021.

NAVARRO, C. A.; HITSCHFELD-KAHLER, N.; MATEU, L. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, Cambridge University Press, v. 15, n. 2, p. 285–329, 2014.

NIELSEN, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. Acesso: 07-06-2021. Disponível em: <http://neuralnetworksanddeeplearning.com/>.

PINTO, N.; DOUKHAN, D.; DICARLO, J. J.; COX, D. A high-throughput screening approach to discovering good forms of biologically-inspired visual representation. *PLoS Computational Biology*, v. 5, n. 11, p. e1000579, 2009.

RAFAEL, G. P. *Restauração de imagens utilizando aprendizado de máquina*. 2018.

ROSENBLATT, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962. (Cornell Aeronautical Laboratory. Report no. VG-1196-G-8). Acesso: 07-06-2021. Disponível em: <https://books.google.ca/books?id=7FhRAAAAMAAJ>.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. *ImageNet Large Scale Visual Recognition Challenge*. 2015.

SALEH, B.; ELGAMMAL, A. *Large-scale Classification of Fine-Art Paintings: Learning The Right Metric on The Right Feature*. 2015.

SIMONYAN, K.; ZISSERMAN, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015.

SOUZA, G. *Detecção de ataques a sistemas de reconhecimento facial utilizando abordagens eficientes de aprendizado de máquina em profundidade*. 2019.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. *Going Deeper with Convolutions*. 2014.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. *Rethinking the Inception Architecture for Computer Vision*. 2015.

TAN, W. R.; CHAN, C. S.; AGUIRRE, H. E.; TANAKA, K. Ceci n'est pas une pipe: A deep convolutional network for fine-art paintings classification. In: *2016 IEEE International Conference on Image Processing (ICIP)*. [S.l.: s.n.], 2016. p. 3703–3707.

TENSORFLOW. Acesso: 22-04-2021. Disponível em: <ttps://www.tensorflow.org>.

TSANG, S. ho. *Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)*. 2021. Acesso: 07-06-2021. Disponível em: <https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568/>.