



# Classificação de Imagens de Flores

## Aplicando Modelos Pré-Treinados de Redes Neurais Convolucionais

Allan de Souza Batista  
Thábata Cristina Giglio Lourenço Maciel

Projeto de Bloco  
Bloco D Mineração de Texto

Dezembro de 2018

Resumo do Projeto de Bloco apresentado a Escola Superior da Tecnologia da Informação como parte dos requisitos necessários para aprovação no Bloco D do curso de MIT em Big Data.

## Classificação de Imagens de Flores Aplicando Modelos Pré-Treinados de Redes Neurais Convolucionais

Allan de Souza Batista

Thábata Cristina Giglio Lourenço Maciel

Dezembro/2018

Desde 2012, Redes Neurais Convolucionais começaram a se destacar para solução de problemas de classificação de imagem, e hoje em dia já se consolidaram como a arquitetura preferida dos modelos de alta performance neste contexto. O presente trabalho teve como objetivo explorar o universo de Redes Neurais Artificiais e *deep learning* através da aplicação de Redes Neurais Convolucionais para problemas deste tipo, mais especificamente, fotografias de flores e seus tipos.

Neste estudo, foi utilizado um *dataset* de imagens de flores coletado por outros autores através da API de uma rede social de fotografias chamada Flickr. 5111 imagens, distribuídas de forma desbalanceada em 17 categorias de flores, foram então pré-tratados, antes de serem carregadas para um *storage* no Google Cloud Platform (GCP), onde foram realizados todos os treinamentos. Para a construção e treinamento de modelos, utilizou-se uma técnica de Transferência de Conhecimento, onde as camadas de extração de recursos de outras Redes Neurais Convolucionais de alta performance pré-treinadas são utilizadas.

Foi desenvolvido um algoritmo generalizado para a implementação flexível de treinamentos no GCP, e, nos experimentos, foram avaliadas 4 frações de *dropout*, 3 quantidades de neurônios na camada oculta, 2 quantidades de camadas ocultas, 3 funções de ativação em camada de saída, 3 funções de ativação em camada oculta, 2 funções de custo, 3 algoritmos de otimização, e 5 modelos pré-treinados para *feature extraction*. Segundo uma estratégia sequencial de treinamento, foram 18 experimentos realizados para avaliação da influência de cada um destes parâmetros na performance do modelo.

Dos resultados obtidos, destacam-se a superioridade de performance dos experimentos com as maiores frações de *dropout*, função de ativação Softmax na camada de saída, função de ativação Sigmóide na camada oculta, Gradiente Descendente Estocástico como otimizador, e a CNN pré-treinada ResNet50 para as camadas de extração de recurso. A CNN com melhor performance obtida apresentou 95% de F1-score e 95% de acurácia, resultados extremamente satisfatórios, considerando a pequena quantidade de imagens para treinamento.

Por fim, este trabalho ainda pode ser facilmente expandido com desenvolvimentos futuros tais como avaliação de *data augmentation*, re-treinamento das camadas pré-treinadas, e um *fine-tuning* mais completo, como outros hiperparâmetros, e uma maior quantidade de experimentos.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Motivação e Objetivo . . . . .	3
1.2	Estrutura do Texto . . . . .	4
<b>2</b>	<b>Fundamentos Técnicos e Literatura</b>	<b>5</b>
2.1	Redes Neurais Artificiais . . . . .	7
2.1.1	Histórico . . . . .	8
2.1.2	Conceitos Básicos . . . . .	10
2.1.2.1	Perceptron . . . . .	10
2.1.2.2	Funções de Ativação . . . . .	11
2.1.2.3	Teorema da Universalidade . . . . .	11
2.1.2.4	Redes de Propagação Direta e Redes Recorrentes . .	12
2.1.2.5	Retropropagação . . . . .	13
2.1.3	Arquiteturas e Camadas de Redes Neurais Artificiais . . . . .	15
2.1.3.1	Redes Neurais Convolucionais . . . . .	17
2.1.3.2	Camada de Convolução . . . . .	19
2.1.3.3	Camada de Agrupamento . . . . .	21
2.1.3.4	Camada de Classificação . . . . .	22
2.1.3.5	Camada de <i>Dropout</i> . . . . .	24
2.2	Transferência de Conhecimento . . . . .	25
2.2.1	ImageNet . . . . .	27
2.2.2	VGG . . . . .	28
2.2.3	Inception V3 . . . . .	30
2.2.4	ResNet50 . . . . .	31
2.3	Tecnologias . . . . .	32
2.3.1	Python . . . . .	32
2.3.2	Google Cloud Platform . . . . .	33
2.3.2.1	Compute Engine . . . . .	33
2.3.2.2	Cloud Storage . . . . .	34
2.3.2.3	ML Engine . . . . .	34
2.3.3	TensorFlow . . . . .	34
2.3.3.1	Keras . . . . .	35
2.3.4	Jupyter Notebook . . . . .	35

<b>3 Desenvolvimento e Experimentos</b>	<b>37</b>
3.1 Descrição do Problema . . . . .	38
3.2 Aquisição de Dados . . . . .	39
3.3 Preparação dos dados . . . . .	41
3.3.1 Exploração . . . . .	41
3.3.2 Pré-Processamento . . . . .	43
3.4 Análise de modelos . . . . .	47
3.4.1 Experimentos . . . . .	50
3.4.1.1 Configuração Inicial . . . . .	52
3.4.1.2 Variando <i>Dropout</i> . . . . .	53
3.4.1.3 Variando Quantidade de Neurônios . . . . .	53
3.4.1.4 Variando Quantidade de Camadas Ocultas . . . . .	54
3.4.1.5 Variando Funções de Ativação da Camada de Saída . .	54
3.4.1.6 Variando Funções de Ativação da Camada Oculta . .	55
3.4.1.7 Variando Funções de Custo . . . . .	56
3.4.1.8 Variando Algoritmos de Otimização . . . . .	56
3.4.1.9 Variando Modelos de <i>Feature Extraction</i> . . . . .	57
3.5 Reportar e Agir . . . . .	57
<b>4 Resultados</b>	<b>60</b>
4.1 <i>Dropout</i> . . . . .	60
4.2 Quantidade de Neurônios . . . . .	65
4.3 Quantidade de Camadas Ocultas . . . . .	67
4.4 Funções de Ativação da Camada de Saída . . . . .	69
4.5 Funções de Ativação da Camada Oculta . . . . .	70
4.6 Funções de Custo . . . . .	74
4.7 Algoritmos de Otimização . . . . .	75
4.8 Modelos de <i>Feature Extraction</i> . . . . .	77
<b>5 Conclusões</b>	<b>85</b>
5.1 Sugestão de Próximos Passos . . . . .	87
<b>A Resultados</b>	<b>89</b>
A.1 <i>Dropout</i> . . . . .	89
A.2 Quantidade de Neurônios . . . . .	97
A.3 Quantidade de Camadas Ocultas . . . . .	103
A.4 Funções de Ativação da Camada de Saída . . . . .	108
A.5 Funções de Ativação da Camada Oculta . . . . .	114
A.6 Funções de Custo . . . . .	120
A.7 Algoritmos de Otimização . . . . .	125
A.8 Modelos de <i>Feature Extraction</i> . . . . .	131
<b>Referências Bibliográficas</b>	<b>143</b>

# Capítulo 1

## Introdução

Aprendizagem é o processo de transformação de dados e experiência em conhecimento e inteligência, uma atividade natural e, muitas vezes, realizada inconscientemente pelos seres humanos. O conceito de Aprendizagem de Máquina (do inglês, *Machine Learning*), estende este processo para algoritmos em computação, com o objetivo de reconhecer padrões e produzir resultados baseados em um histórico de dados fornecidos [28]. Matthew Kinsey define *Machine Learning* como (tradução livre):

Um método de análise de dados que incorpora o uso de algoritmos com habilidades de aprender com os dados e produzir resultados sem a necessidade de programar explicitamente para obter os mesmos.

reforçando a capacidade poderosa destes algoritmos de realizar estas tarefas automaticamente, sem a limitação introduzida pela programação manual, e a uma velocidade infinitamente superior à capacidade de processamento do cérebro humano [20].

Como um subgrupo da área de Inteligência Artificial, a Aprendizagem de Máquina vem recebendo uma atenção crescente nas últimas duas décadas, culminando em um crescimento exponencial da sofisticação dos algoritmos na atualidade. Já presente em uma grande diversidade de setores, como mercado financeiro, varejo, saúde, educação e serviços diversos, *Machine Learning* pode ser utilizada na solução de um número também crescente de aplicações: processamento de linguagem natu-

ral, detecção de anomalias, reconhecimento de imagens, predição de estados futuros, otimização de recomendações, categorização e agrupamentos (do inglês, *clustering*), redução de ruídos, entre outros [14, 27].

Uma destas aplicações que evoluiu de uma forma impressionante nos últimos seis anos é a classificação de imagens. Uma competição anual é organizada pelo projeto ImageNet, desde 2010, com o objetivo de acelerar a pesquisa e desenvolvimento de algoritmos de classificação e detecção de objetos pela comunidade mundial, e a Figura 1.1 apresenta a evolução das taxas de erro das equipes vencedoras dos desafios em cada ano. O ano de 2012 se caracterizou como um marco nesta área quando uma Rede Neural Convolucional, um tipo de algoritmo de *Machine Learning*, foi utilizada por Alex Krizhevsky e sua equipe pela primeira vez em larga escala para imagens, garantindo o primeiro lugar na competição com uma margem de quase de 11% para o segundo lugar [19, 26].

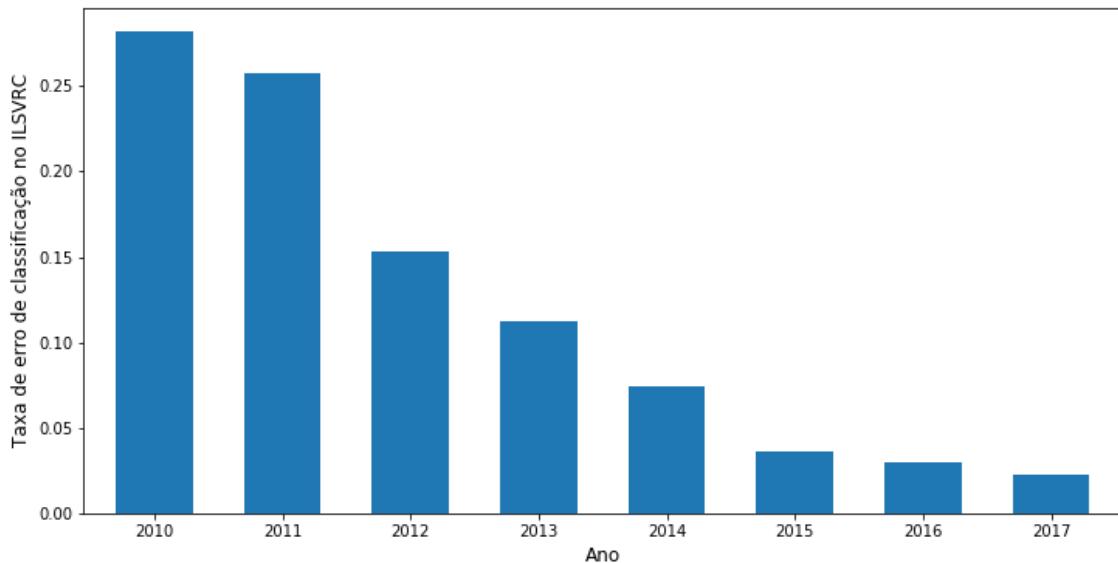


Figura 1.1: Taxa de erro de classificação de imagens das equipes vencedoras nas competições anuais *ImageNet Large Scale Visual Recognition Competition* (ILSVRC).

Desde 2012, todos os vencedores da tarefa de classificação de imagens nesta competição aplicaram Redes Neurais Convolucionais (do inglês, *Convolutional Neural Networks*, ou CNN), e os melhores resultados encontrados nos últimos anos indicam

que este tipo de algoritmo representa o estado-da-arte nesta área.

## 1.1 Motivação e Objetivo

O presente trabalho tem como principal objetivo explorar as características, aplicar e comparar modelos de Redes Neurais Convolucionais, entre outros parâmetros relacionados a algoritmos de Aprendizagem de Máquina, para a solução de problemas de classificação de imagens.

Estando este tipo de algoritmo no topo da performance para esta tarefa, a motivação para se entender e explora-lo é clara. No entanto, a arquitetura de uma CNN pode se tornar bastante complexa, e requer a definição de diferentes propriedades de construção de modelo e de otimização, tais como: quantidade e tipos de camadas de extração de recursos (do inglês, *feature extraction*), quantidade e tamanho de camadas de classificação, funções de ativação, otimizadores de hiperparâmetros, etc. Assim, a comparação e exploração da influência destes parâmetros na performance de uma CNN traz um valor adicional à motivação aqui apresentada.

Este trabalho, então, aplicou uma base de dados de imagens de flores, obtida através de algoritmo aberto [6] de busca e obtenção de imagens pela API do Flickr - uma rede social de fotografias -, a biblioteca Keras com o *framework* TensorFlow por trás, para o desenvolvimento do algoritmo, e a plataforma em nuvem da Google (*Google Cloud Platform*, ou GCP) para a infraestrutura necessária no treinamento dos modelos e execução de experimentos.

Nota-se também que, para viabilizar a execução deste trabalho, optou-se por utilizar modelos CNN pré-treinados na competição do ImageNet, ou seja, aplicando Transferência de Conhecimento. Conhecidamente, o treinamento de redes neurais exige enorme poder computacional para o tratamento de milhares, até milhões, de imagens, necessárias para obter as performances observadas anteriormente. A aplicação de modelos pré-treinados em outras bases de imagens facilita o processo

de treinamento, já que requer menor poder computacional e uma menor quantidade de imagens das classes pretendidas no objetivo.

Por fim, todos os algoritmos desenvolvidos e resultados obtidos neste trabalho estão disponibilizado em repositório público no GitHub [5].

## 1.2 Estrutura do Texto

O presente trabalho encontra-se estruturado em 5 capítulos de forma a abordar desde os fundamentos técnicos de redes neurais até as conclusões finais baseadas nos resultados dos experimentos.

- **Capítulo 1:** É a presente introdução.
- **Capítulo 2:** Uma apresentação técnica dos conceitos de redes neurais, transferência de conhecimento e as ferramentas aplicadas, além de apresentar os principais trabalhos da literatura sobre o assunto.
- **Capítulo 3:** Apresenta o trabalho realizado, incluindo um detalhamento sobre a base de dados aplicada, desenvolvimento do algoritmo, e experimentados investigados.
- **Capítulo 4:** Demonstra os resultados obtidos com os experimentos, desde a performance da utilização do *Google Cloud Platform*, até a acurácia dos modelos treinados.
- **Capítulo 5:** Resume e conclui sobre o projeto, resultados e dificuldades, além de providenciar algumas sugestões sobre possíveis trabalhos a serem desenvolvidos.

# Capítulo 2

## Fundamentos Técnicos e Literatura

O tema de Aprendizagem de Máquina é bastante extenso, e novos modelos são desenvolvidos e aperfeiçoados todos os anos pela comunidade mundial. A fórmula básica de construção de quase todos os algoritmos de *Machine Learning* compreende quatro elementos: uma base de dados especificada para o treinamento, uma função de custo para o direcionamento do treinamento, um procedimento de otimização para minimização da função de custo, e, finalmente, o modelo, que representa o formato da função final a ser treinada.

Segundo esta fórmula, modelos tradicionais de *Machine Learning*, tais como Regressão Logística, Árvores de Decisão, SVM (*Support Vector Machine*), entre outros, são amplamente utilizados, e podem ser bastante eficientes quando aplicados em situações apropriadas, e em conjunto com técnicas de tratamento de dados, redução de dimensionalidade e avaliação de importância de variáveis. No entanto, estes métodos apresentam grandes deficiências em áreas centrais da Inteligência Artificial, como reconhecimento de fala e classificação de imagem.

A capacidade de generalização de um modelo de Aprendizagem de Máquina é a habilidade de responder corretamente a dados de entrada não observados previa-

mente durante o treinamento, e os modelos tradicionais costumam apresentar baixa performance de generalização para funções complexas em um espaço de alta dimensão. Muitas vezes, a experiência na área de conhecimento do problema é suficiente para, em conjunto com técnicas de redução de complexidade e dimensionalidade, tornar a identificação de padrões mais fácil para o algoritmo. Para os casos em que tais técnicas não surtem efeito, ou não são aplicáveis, surge a motivação de se aplicar uma arquitetura diferente de computação, chamada de Aprendizagem Profunda (do inglês, *Deep Learning*) [10].

O principal desafio da área de Inteligência Artificial é a realização de tarefas que os seres humanos resolvem de forma intuitiva, as quais são difíceis de descrever em etapas formais. Por exemplo, reconhecer uma pessoa em uma foto, ler uma carta escrita à mão, e entender o que é dito por alguém parecem atividades triviais, mas, como não são formalmente bem definidas, se tornam extremamente complexas de serem aprendidas por um algoritmo. *Deep Learning* resolve este problema com uma arquitetura de hierarquização de conceitos, onde uma definição é abstraída em conceitos mais simples, que por sua vez também são descritos por outros conceitos mais simples, e assim sucessivamente, através de diversas camadas de abstração. A aprendizagem é então realizada passando por cada uma destas camadas e, como o grafo final do relacionamento de conceitos é formado por inúmeras destas, trata-se de um grafo profundo, dando origem ao nome desta arquitetura [9, 10].

Um exemplo bastante comum do funcionamento de um algoritmo de Aprendizagem Profunda, e bastante relevante para o presente trabalho, é a classificação de imagens. A Figura 2.1 apresenta uma representação simplificada de um processo de Aprendizagem Profunda para classificação de um rosto humano em uma foto. No exemplo, o reconhecimento de um rosto humano é realizado através da identificação de elementos básicos, como nariz e boca, os quais, por sua vez, são abstraídos em formas mais simples e bordas, que, por fim, são compostas pelos *pixels* originais.

Este formato de arquitetura apresentado na Figura 2.1 é conhecido como Rede

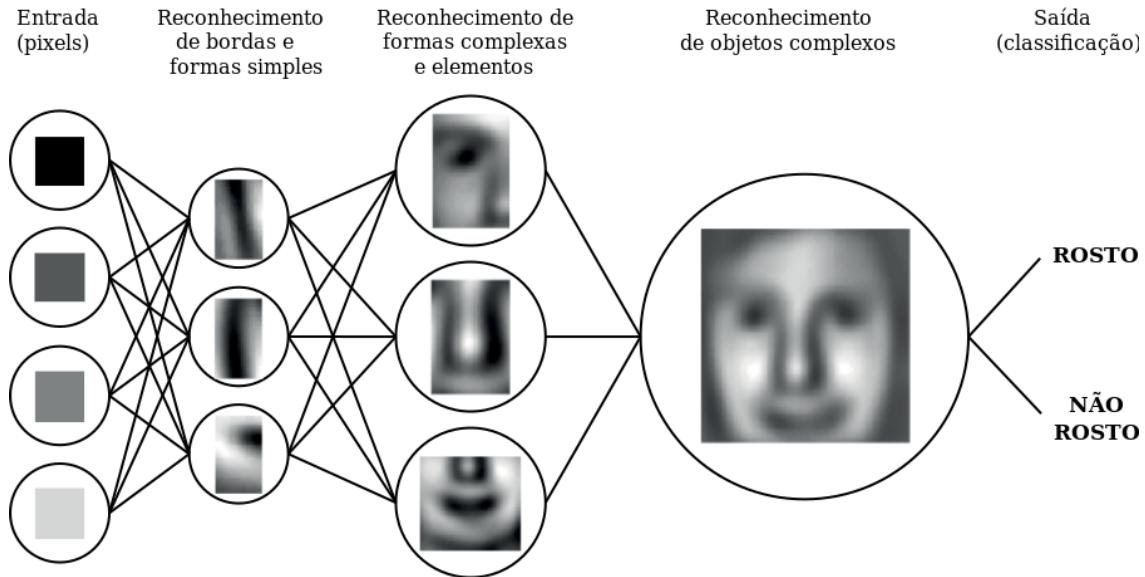


Figura 2.1: Representação do formato de hierarquização de conceitos em um algoritmo de Aprendizagem Profunda para classificação de imagens. Adaptado de [21].

Neural Artificial, podendo ou não ser uma Rede Neural Profunda (*Deep Neural Network*, ou DNN), dependendo da quantidade de camadas de abstração que compõe a estrutura. Desta forma, Aprendizagem Profunda pode ser definida como um subgrupo dos algoritmos de Aprendizagem de Máquina que aplicam Redes Neurais Profundas como o modelo de aprendizado [23].

## 2.1 Redes Neurais Artificiais

Redes Neurais Artificiais, do inglês, *Artificial Neural Network*, ou ANN, são arquiteturas de computação e algoritmos inspiradas na biologia do sistema nervoso, onde elementos básicos de processamento (nêurons) operam em paralelo e se relacionam em uma rede que pode ser composta por milhares ou milhões destas células [8]. Este formato de programação permite, então, a solução de problemas complexos, não-lineares e matematicamente mal definidos através da computação de diversas operações matemáticas simples que se relacionam entre si. A Figura 2.2 apresenta uma arquitetura genérica de uma Rede Neural Artificial composta por diversos neurônios estruturados em quatro camadas.

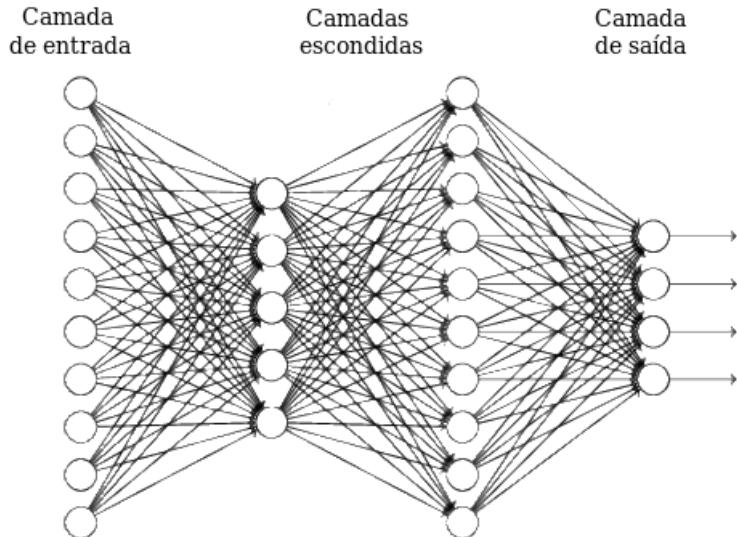


Figura 2.2: Arquitetura genérica de uma Rede Neural Artificial. Adaptado de [23].

### 2.1.1 Histórico

ANN não é um conceito novo, e seus primeiros princípios básicos foram descritos em 1943 por Warren McCulloch e Walter Pitts, os quais inspiraram o desenvolvimento da primeira Rede Neural Artificial funcional em 1958, por Frank Rosenblatt, chamada de Perceptron. Hoje, Rosenblatt é referido como o “pai da neurocomputação”, mas, ainda nas décadas de 50 e 60, outras estruturas de Redes Neurais também foram desenvolvidas, como a Adaline, por Bernard Widrow em 1960, cuja metodologia de aprendizagem ainda é utilizada nos dias de hoje [7, 22, 24, 37].

A década de 80 foi um época de interesse renovado no tópico, também conhecida como a segunda geração de ANNs, com o surgimento de técnicas e estruturas fundamentais de Redes Neurais, como a Rede de Hopfield, desenvolvida em 1982 por John Hopfield, e o algoritmo de Retropropagação (do inglês, *Backpropagation*) por David Rumelhart e colaboradores em 1986. Também foi a década da primeira conferência internacional de Redes Neurais, da criação da Sociedade Internacional de Redes Neurais, INSS, e fundação do primeiro jornal acadêmico dedicado ao tópico, *Neural Networks* [12, 18, 25].

As técnicas desenvolvidas nas décadas de 80 e 90 aumentaram significativamente o interesse da comunidade científica pelo assunto. Técnicas como a Gradiente Des-

cendente Estocásticos (do inglês, *Stochastic Gradient Descent*) e a Retropropagação são utilizadas em Redes Neurais modernas, mas, na época, resultavam em um processo de aprendizagem muito lento e muitas vezes ineficientes para Redes Profundas, com muitas camadas. Esta dificuldade só foi superada a partir de 2006, quando novas ideias começaram a ressurgir na literatura para resolver esse problema. Geoffrey Hinton, por exemplo, introduziu o conceito de Redes de Crenças Profundas (do inglês, *Deep Belief Networks*), que permitiu uma inicialização mais eficiente do processo iterativo de aprendizado de ANNs [9, 17, 23].

Desde então, o interesse e o desenvolvimento científico em Redes Neurais Artificiais e, principalmente, Redes Neurais Artificiais Profundas, vem crescendo a uma taxa impressionante. Em 2009, DNNs foram aplicadas com êxito pela primeira vez em larga escala no reconhecimento de fala, e, em 2012, na classificação de imagens. Ainda assim, a Figura 2.3 demonstra que o interesse mundial no tópico continua crescendo nos últimos anos, e novas técnicas e modelos continuam surgindo [9].

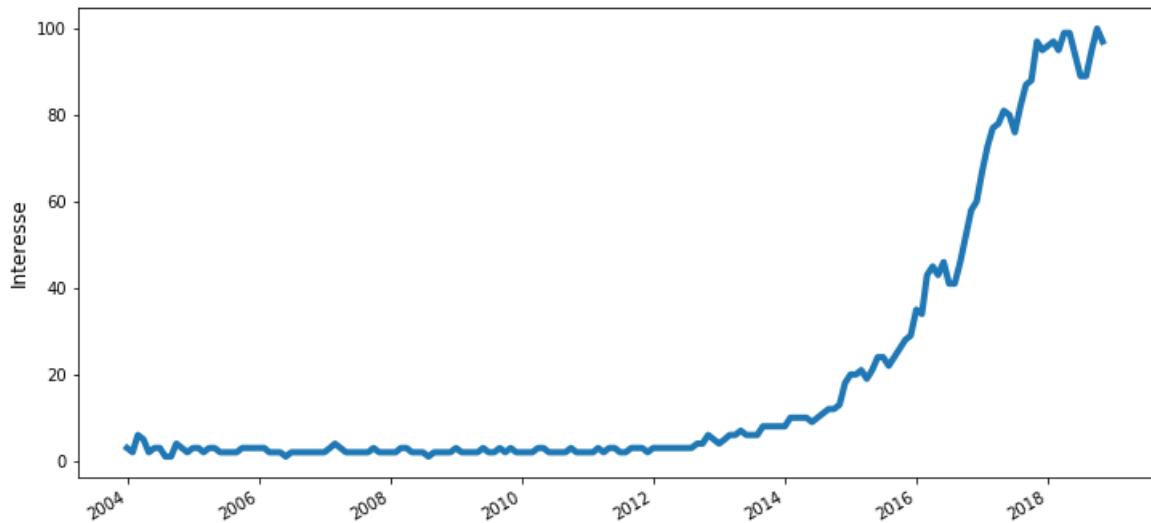


Figura 2.3: Interesse mundial no termo de busca “*deep learning*” no Google ao longo do tempo, de acordo com Google Trends [1].

## 2.1.2 Conceitos Básicos

O conteúdo apresentado nesta seção tem como objetivo resumir a teoria de alguns conceitos básicos de Redes Neurais Artificiais, fundamentais ao entendimento do escopo deste trabalho.

### 2.1.2.1 Perceptron

Perceptron foi o primeiro modelo computacional neural, e é considerado o alicerce de construção de muitos outros modelos de Redes Neurais. Trata-se da definição da estrutura do elemento básico de uma ANN, um nêuron (Figura 2.4).

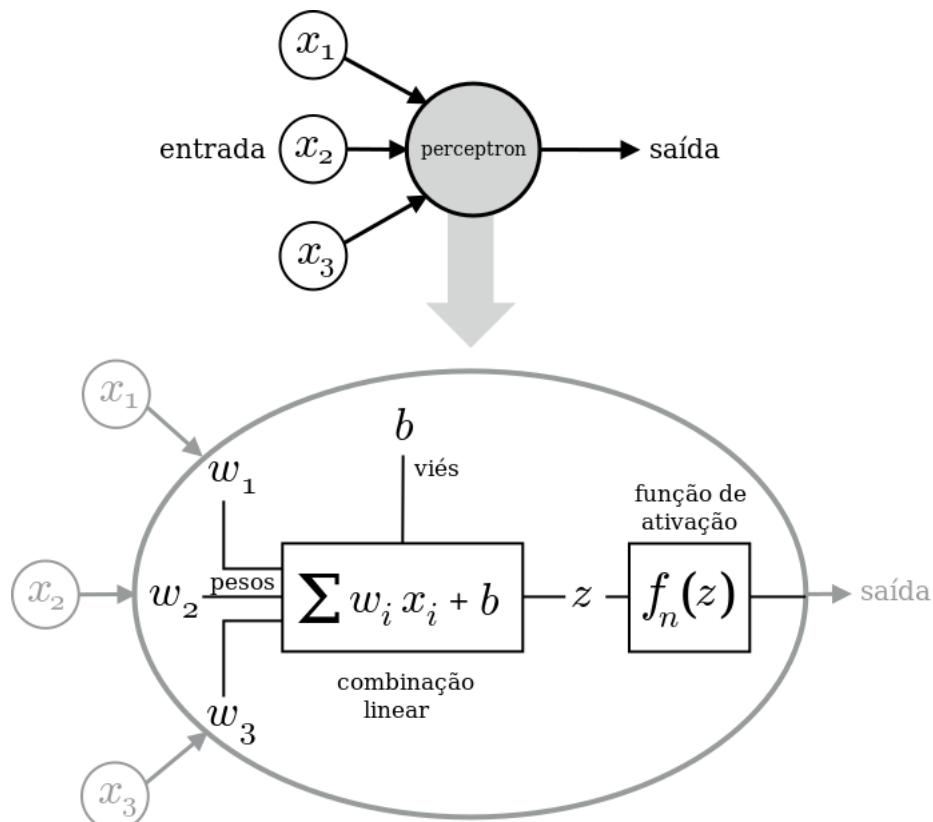


Figura 2.4: Estrutura genérica básica de um Perceptron com uma camada de entrada de três dimensões. Adaptado de [12].

Um Perceptron é composto por cinco elementos: um vetor de dados de entrada, um vetor de pesos, uma combinação linear, uma função de ativação, e um valor de saída. A Figura 2.4 apresenta a estrutura genérica de um Perceptron com uma

camada de entrada de três dimensões. Nesta estrutura, todos os valores numéricos de entrada  $x_i$  do modelo são multiplicados pelos seus respectivos pesos  $w_i$  e somados entre si e a um coeficiente linear chamado viés,  $b$  ( $\sum_i w_i x_i + b$ ), resultando em um valor intermediário  $z$ , que é convertido por uma função não-linear  $f_n(z)$  no valor final de saída do neurônio,  $y$  [12].

O Perceptron é um classificador linear, o que significa que é matematicamente comprovado que, com os pesos certos, pode simular qualquer problema de separação linear, independente da dimensão do espaço.

### 2.1.2.2 Funções de Ativação

Para se obter os pesos e viés corretos de uma rede neural são normalmente utilizados algoritmos iterativos, que modificam os pesos de um Perceptron gradualmente, com base na magnitude do erro dos seus resultados. Uma das principais aplicações da função de ativação de um neurônio é permitir que esta modificação gradual dos pesos não resulte em grandes variações nos resultados, já que é responsável por limitar os valores finais. Por esta razão, também são chamadas de funções de esmagamento (do inglês, *squashing functions*), e uma das mais utilizadas é a função sigmoide, que possui um formato suavizado de uma função de degrau (Equação 2.1 e Figura 2.5) [12, 23]. Outros tipos de funções de ativação também são apresentados na seção 2.1.3.4.

$$f_n(z) = \frac{1}{1 + e^{-z}} \quad (2.1)$$

### 2.1.2.3 Teorema da Universalidade

Para problemas de classificação não-lineares, como o próprio operador lógico XOR, se faz necessária a combinação de múltiplos Perceptrons em Redes Neurais com mais de uma camada, tal como esquematizado anteriormente na Figura 2.2. Na realidade, através do Teorema da Universalidade, ou também conhecido como

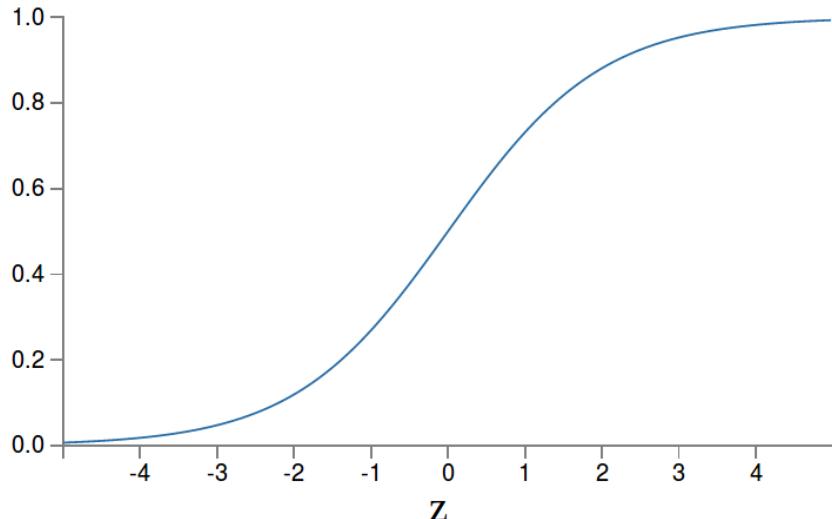


Figura 2.5: Função de ativação do tipo sigmoide.

Teorema da Aproximação Universal, é possível comprovar matematicamente que qualquer função pode ser representada por uma Rede Neural Artificial com duas ou mais camadas (excluindo a camada de entrada de dados) [10, 23].

Nota-se, no entanto, que este Teorema não garante a viabilidade do treinamento da Rede Neural, mas apenas que existe uma configuração de neurons que, com os pesos e viés corretos, é capaz de representar qualquer função.

#### 2.1.2.4 Redes de Propagação Direta e Redes Recorrentes

Grande parte das Redes Neurais Artificiais modernas podem ser separadas em dois grandes grupos: Redes de Propagação Direta e Redes Neurais Recorrentes.

Aplicando o conceito de Perceptron para o desenvolvimento de uma Rede Neural com múltiplas camadas, a estrutura formada segue uma lógica de movimentação direta dos dados, ou seja, desde a camada de entrada, passando por camadas ocultas onde são realizadas as transformações lineares e não-lineares, até a camada de saída, após a qual o resultado é obtido. Para uma ANN construída desta forma, dá-se o nome de Rede de Propagação Direta (do inglês, *Feedforward Network*), e que também é muitas vezes referenciada na literatura como Perceptrons de Múltiplas Camadas (do inglês, *Multilayer Perceptrons*, ou MLP), onde a saída de uma camada é utilizada

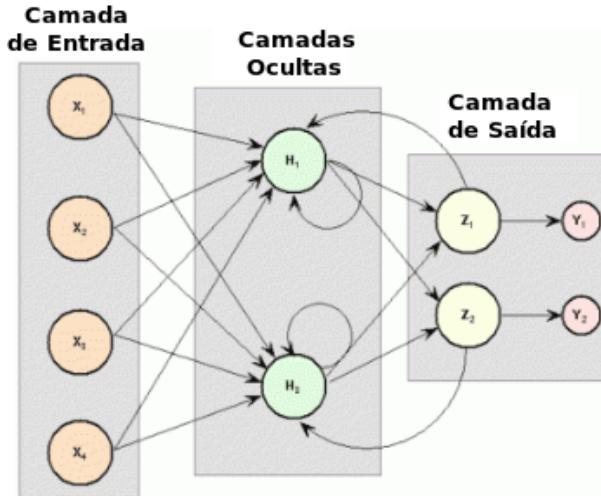


Figura 2.6: Estrutura genérica de uma Rede Neural Recorrente, esquematizando estruturas cíclicas de propagação de dados. Traduzido de [4].

somente na alimentação de camadas seguintes [10].

Paralelas às Redes *Feedforward*, as Redes Neurais Recorrentes (do inglês *Recurrent Neural Networks*, ou RNN) são um outro tipo de disposição de neurons que permite o feedback de informações através de relacionamentos cílicos entre os mesmos (Figura 2.6). Estas estruturas se tornam possíveis através de ativações de neurons em índices de tempo diferentes, permitindo que os resultados de uma camada sejam re-alimentados à mesma camada ou a camadas anteriores em um momento posterior, de forma que influencie o resultado final de uma camada subsequente [4].

Redes Neurais Recorrentes possuem um formato muito natural de se modelar dados sequenciais, que podem ser representados através de um eixo temporal. Através desta estratégia, são construídos modelos dinâmicos com “memória”, que melhoram a acurácia de uma classificação através de análise de contexto. Infelizmente, uma das grandes desvantagens de Redes Neurais Recorrentes é a complexidade de seus processos de treinamento.

#### 2.1.2.5 Retropropagação

O algoritmo de Retropropagação (do inglês, *Backpropagation*) é a técnica mais importante da história das Redes Neurais Artificiais, sendo introduzido pela primeira

em 1970, mas ganhou popularidade a partir de 1986, com os trabalhos de David Rumelhart, Geoffrey Hinton e Ronald Williams [25]. De forma simplificada, este algoritmo utiliza a regra da cadeia em derivadas parciais das funções de custo, aplicada em toda a extensão da rede neural. [23].

O objetivo do *Backpropagation* é otimizar os pesos e viés de uma rede de forma iterativa para minimizar o erro de seus resultados finais. Em algoritmos de Aprendizagem de Máquina tradicionais, técnicas diferenciais, como o famoso Gradiente Descendente, aplicam as derivadas das funções de custo para avaliar se a evolução do erro do modelo segue na direção do mínimo durante o treinamento. A aplicação do gradiente, no entanto, não é tão simples para Redes Neurais, pois existe uma dependência funcional entre os neurônios, ou seja, o cálculo de como o resultado final varia com modificações de um certo parâmetro também depende de como o resultado varia com a alteração de todos os outros parâmetros em camadas subsequentes da rede [4].

A Retropropagação, então, consiste em dois conceitos principais:

1. aplicar uma estratégia de diferenciação no modo reverso para todos os pesos da Rede Neural, i.e. utilizando o operador  $\frac{\partial \text{erro}}{\partial}[\cdot]$  nos pesos, ao invés de derivadas diretas do tipo  $\frac{\partial \text{erro}}{\partial x_i}$ , onde  $x_i$  corresponde ao vetor de dados de entrada;
2. aplicar a regra da cadeia de cálculo diferencial para se obter os valores de  $\frac{\partial \text{erro}}{\partial w_i}$ , e as derivadas nesta decomposição são calculadas individualmente a partir da diferenciação das funções de ativação aplicadas às combinações lineares em cada neurônio. Como exemplo, a Equação 2.2 abaixo demonstra esta aplicação a um peso posicionado duas camadas ocultas antes da camada de saída de uma rede:

$$\frac{\partial \text{erro}}{\partial w} = \frac{\partial \text{erro}}{\partial z_{final}} \cdot \frac{\partial z_{final}}{\partial z_n} \cdot \frac{\partial z_n}{\partial z_{n-1}} \cdot \frac{\partial z_{n-1}}{\partial w} \quad (2.2)$$

### 2.1.3 Arquiteturas e Camadas de Redes Neurais Artificiais

Redes Neurais Artificiais é um tópico muito extenso, principalmente pelo fato de ser uma arquitetura de computação de alta flexibilidade. A construção e incorporação de camadas com funcionalidades e objetivos diferentes proporciona características únicas à rede e, muitas vezes, podem os tipos de problema que são melhor resolvidos por cada uma. O desenvolvimento constante de novas técnicas que facilitam o treinamento e utilização de Redes Neurais possibilita o desenvolvimento de arquiteturas cada vez mais sofisticadas e mais eficientes dentro de suas respectivas áreas de especialidade.

A Figura 2.7 esquematiza as principais arquiteturas de ANNs da literatura, como compilado pelo Instituto Asimov em 2016 [36]. Nesta figura é possível observar as arquiteturas clássicas como o Perceptron, Redes de Propagação Direta e Redes Recorrentes, além de diversas outras variações e casos especiais.

Por exemplo, *Autoencoder* consiste em um uso diferente de uma Rede Neural de Propagação Direta, onde o objetivo é copiar os dados entrada nos dados de saída passando por uma primeira etapa de compactação (*encode*) para redução de tamanho, e uma segunda etapa de descompressão (*decode*) para o retorno ao tamanho original. Muitas vezes, a vantagem trazida por um modelo treinado para este propósito não está exatamente nos dados de saída, mas sim na representação dos dados de entrada no centro da Rede Neural, onde é armazenado um formato codificado que pode, por exemplo, ocupar menos espaço ao armazenar somente os detalhes mais importantes dos dados. No entanto, outras variações de *Autoencoders*, são úteis pela representação não perfeita dos dados de entrada na saída, como é o caso do *Denoising Autoencoder*. Neste, o objetivo da rede é ser treinada para re-apresentar os dados de entrada de uma forma aprimorada na saída, como a remoção de ruído de um trecho sonoro, e a reprodução de uma imagem com melhor resolução [10].

Outras variantes de ANN são as Redes de Hopfield (HN), Cadeias de Markov (MC), e Máquinas de Boltzmann (BM), as quais são casos muito especiais de Re-

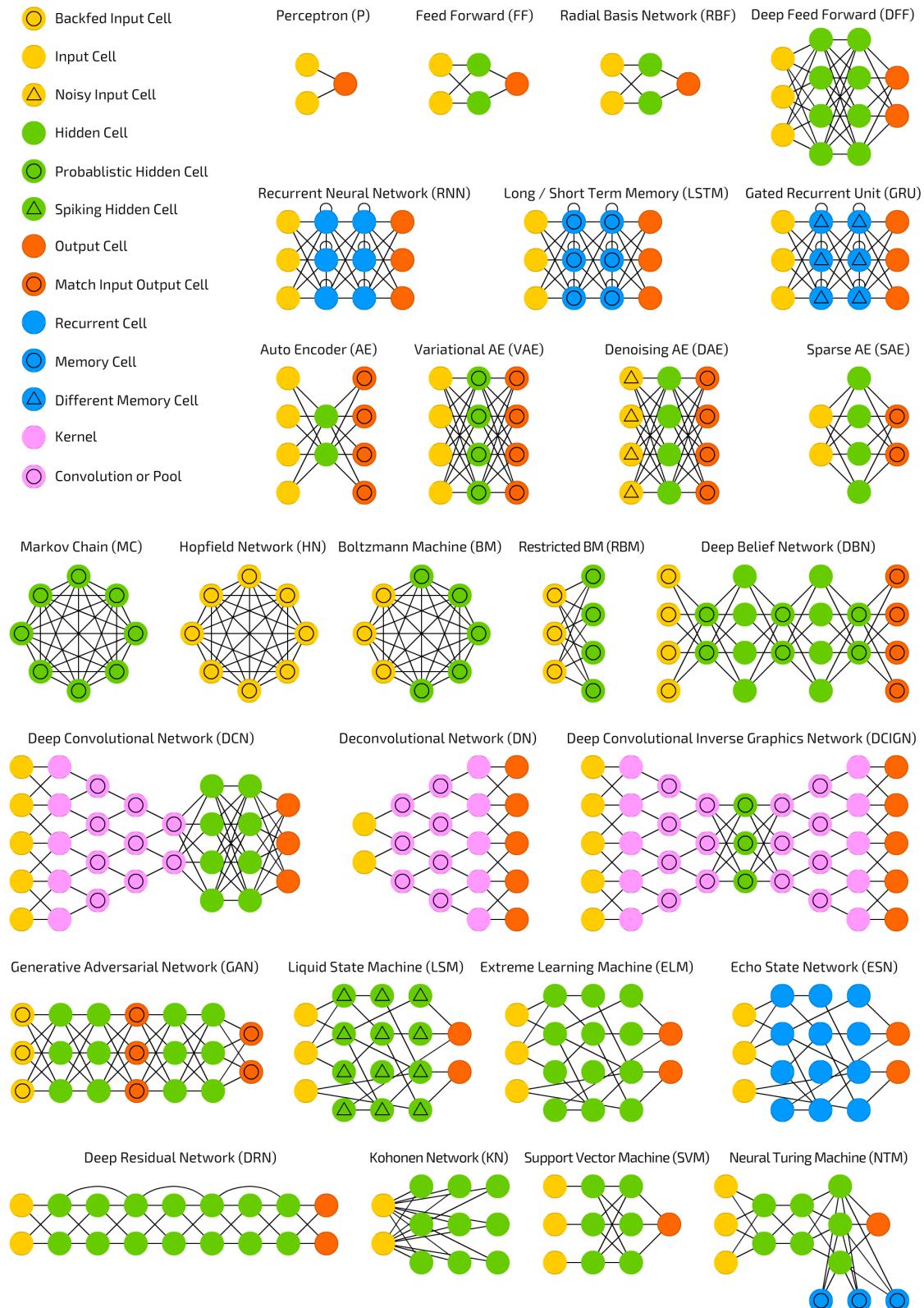


Figura 2.7: Esquemas das principais arquiteturas de Redes Neurais Artificiais. Adaptado de [36].

des Neurais Recorrentes, chamadas de Redes Simétricas, e possuem casos de uso e características muito particulares. As Redes de Hopfield, por exemplo, são construídas de forma que todos os neurônios cumprem exatamente o mesmo papel, servindo como uma camada de entrada no início do treinamento, camada oculta durante o treinamento, e camada de saída ao final, [36].

Um conceito quase oposto às Redes Simétricas acima são as chamadas Redes Generativas Adversárias (tradução livre do inglês *Generative Adversarial Networks*, ou GAN), uma arquitetura bastante recente, e introduzida pela primeira vez em 2014 [11]. Primeiramente as GANs são Redes Neurais generativas e não discriminatórias, o que significa que ao invés de ser treinada para predizer a que categoria um dado de entrada pertence, é treinada para gerar um dado que pertence a uma determinada categoria [4]. O formato da arquitetura de uma GAN é composto por dois tipos diferentes de redes que possuem o objetivo básico de competir entre si durante o treinamento. A primeira, chamada de geradora (*generator*) é responsável por criar amostras de dados, as quais são avaliadas pela a segunda, a rede discriminadora (*discriminator*), que calcula a probabilidade do dado pertencer ao conjunto de dados de entrada para o treinamento ou ter sido criado pela geradora [10].

Como ser observado na Figura 2.7, existem diversos outros tipos de arquiteturas de Redes Neurais, formadas por diferentes camadas de neurons. Para o propósito deste trabalho, no entanto, a teoria por trás das Redes Neurais Convolucionais se faz mais importante, e é tratada nas seções a seguir.

### 2.1.3.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*, ou CNN) possuem uma arquitetura bastante diferente da maioria das demais redes, e são primariamente utilizadas no processamento de imagens, onde um uso típico é a classificação de imagens em categorias definidas. Uma definição formal de Redes Convolucionais diz (tradução livre de [10]):

Redes Neurais Convolucionais são Redes Neurais Artificiais que aplicam a operação de convolução em pelo menos uma de suas camadas.

onde a operação de convolução, descrita pela Equação 2.3, é uma forma de medição da extensão que duas funções  $f(t)$  e  $g(t)$  se sobrepõem em diferentes momentos  $t$  à medida que  $g(t)$  é varrida através de  $f(t)$ .

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau \quad (2.3)$$

Além de camadas de convolução, CNNs também são comumente construídas com algumas camadas de agrupamento (tradução do livre do inglês, *pooling*), formando um conjunto de camadas referido como camadas de extração de recurso. Este conjunto possui uma característica importante, onde os neurônios de uma camada são conectados somente a um subgrupo dos neurônios da camada subsequente (conectividade esparsa), o que traz uma propriedade de valorização de dados próximos entre si, ao invés de uma dependência mais rígida da configuração global como em camadas completamente conectadas (*fully-connected*). O objetivo das camadas de extração de recursos em uma CNN é providenciar um pré-processamento dos dados de entrada através da remoção de detalhes não interessantes e do destaque de características relevantes.

Em seguida às camadas de *feature extraction*, modelos de Redes Neurais Convolucionais possuem um segundo grupo de camadas efetivamente responsável pela classificação dos dados após o pré-processamento. Estas camadas de classificação já apresentam conectividade completa entre os neurônios e englobam a camada de saída da Rede Neural, podendo apresentar funções de ativação diferentes. Normalmente, entre diferentes camadas de classificação, é prática comum incluir uma camada de *Dropout*, mas a configuração pode variar bastante em diversos modelos. A Figura 2.8 apresenta um esquema de composição das camadas de uma CNN, e um detalhamento teórico destas camadas é realizado a seguir.

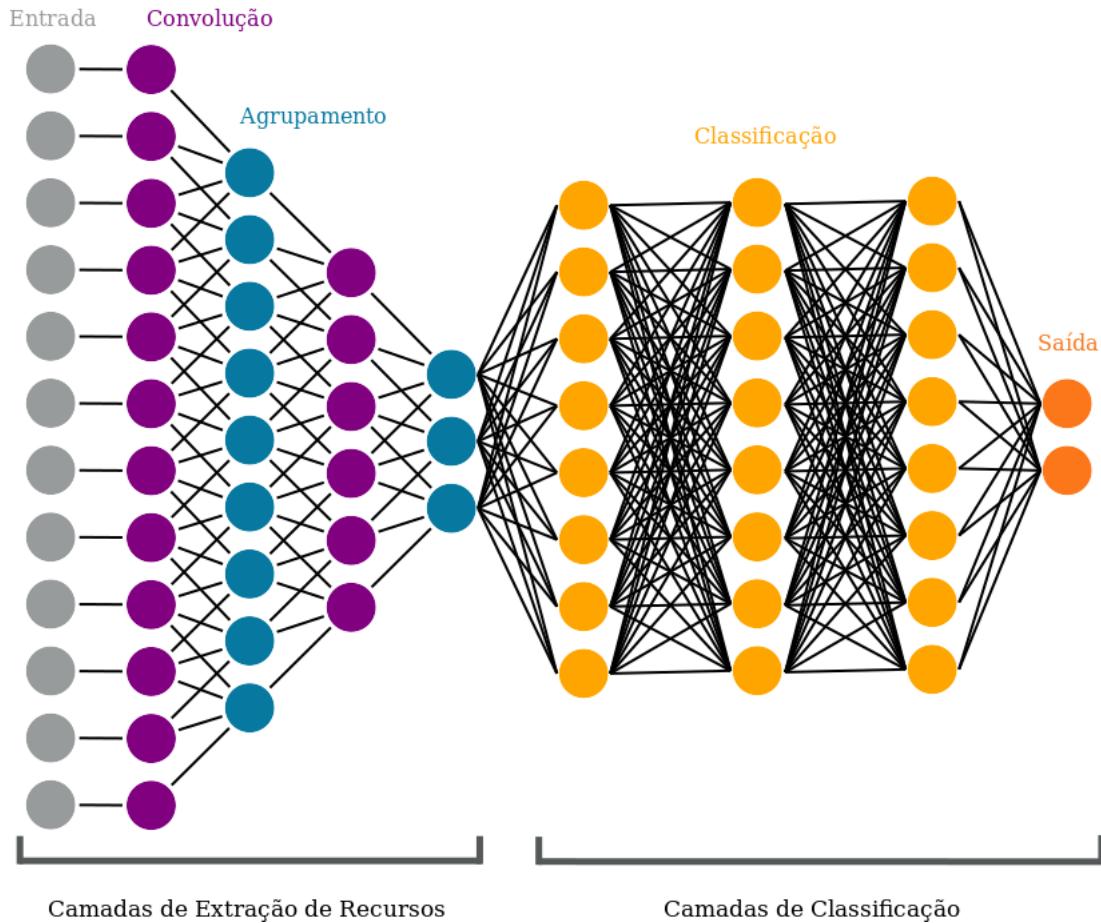


Figura 2.8: Esquema gráfico representando um formato genérico de uma Rede Neural Convolucional, enfatizando os dois conjuntos de camadas de extração de recurso e de classificação. Fonte própria.

### 2.1.3.2 Camada de Convolução

Camadas de Convolução (*Convolution Layers*) são grupos de neurônios que realizam operações de convolução nos dados de entrada da camada. Em um exemplo de processamento de imagem, a operação é discretizada para aplicação em *arrays* bidimensionais dos pixels, como apresentado na Equação 2.4 [10].

$$I(i, j) * K(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n) \quad (2.4)$$

Nesta equação,  $I(i, j)$  representa a matriz dos pixels da imagem, e  $K(i, j)$  é chamado de Kernel da operação, ou filtro da convolução, e corresponde aos pesos da combinação linear que serão treinados na camada. O que esta operação efetivamente

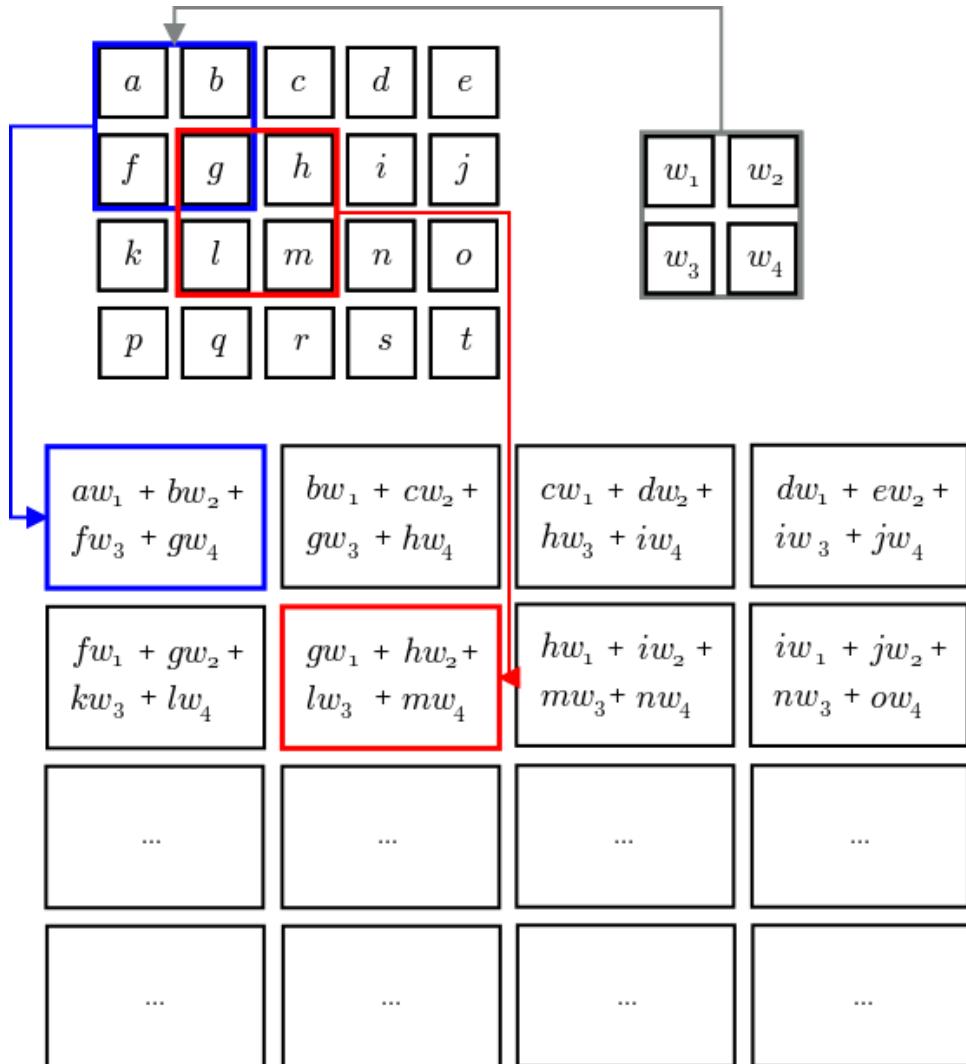


Figura 2.9: Esquema de operação de convolução entre dois *arrays* bidimensionais. À direita, o Kernel da operação é “deslizado” sobre a função principal, à esquerda, e uma combinação linear entre os elementos de ambos é realizada para resultar no mapa de ativação. Baseado em [10].

realiza é esquematizado na Figura 2.9, onde o Kernel, de tamanho arbitrário, é “deslizado” através da imagem para selecionar parte dos pixels sobre os quais é aplicada a combinação linear com os pesos do Kernel. O *array* resultante desta operação é comumente chamado de Mapa de Ativação (do inglês, *activation map*).

Nota-se que existem alguns parâmetros importantes que compõe uma camada de convolução em uma Rede Neural. Estes são:

- **Dimensão do Kernel:** Dimensão do *array* de pesos que compõe o Kernel da camada.

- **Quantidade de Kernels:** Uma camada de convolução pode possuir diversos Kernels distintos, independentes entre si. Os mapas de ativação de cada Kernel são agregados em dimensões diferentes do mapa de ativação final da camada (e.g. dez Kernels que resultam em mapas de 256x256 cada formam um mapa de ativação final de 256x256x10).
- **Passo:** Do inglês *Stride*, é o tamanho do passo dado durante a varredura do Kernel, e influencia diretamente no tamanho final do mapa de ativação. Por exemplo, a Figura 2.9 apresenta um Kernel com passo de 1.
- **Preenchimento:** Do inglês, *Padding*, é a prática de se adicionar elementos nulos nas extremidades do *array* de entrada de forma a evitar a redução de dimensão do mapa de ativação final.

### 2.1.3.3 Camada de Agrupamento

Do inglês, *pooling*, operações de agrupamento são métodos estatísticos para representar o resultado de uma camada da Rede Neural em um formato mais resumido. Por exemplo, agrupamento por máximo (*max pooling*) resume os valores de um conjunto de dados com apenas o valor do máximo entre eles, como na Figura 2.10. Agrupamento pela média (*average pooling*) calcula a média de um conjunto de dados para representá-lo.

Além de aumentar a eficiência do algoritmo através da redução da quantidade de neurônios na Rede Neural, operações de agrupamento adicionam um grau de invariância a pequenas diferenças nos dados de entrada. Por exemplo, uma imagem com alguns pixels levemente transladados terá um resultado bastante parecido ao da imagem original quando processado através de uma camada de agrupamento por máximo, já que esta operação não é sensível a pequenas modificações, mas sim ao valor máximo de um vizinhança.

De forma semelhante às camadas de convolução, camadas de agrupamento pos-

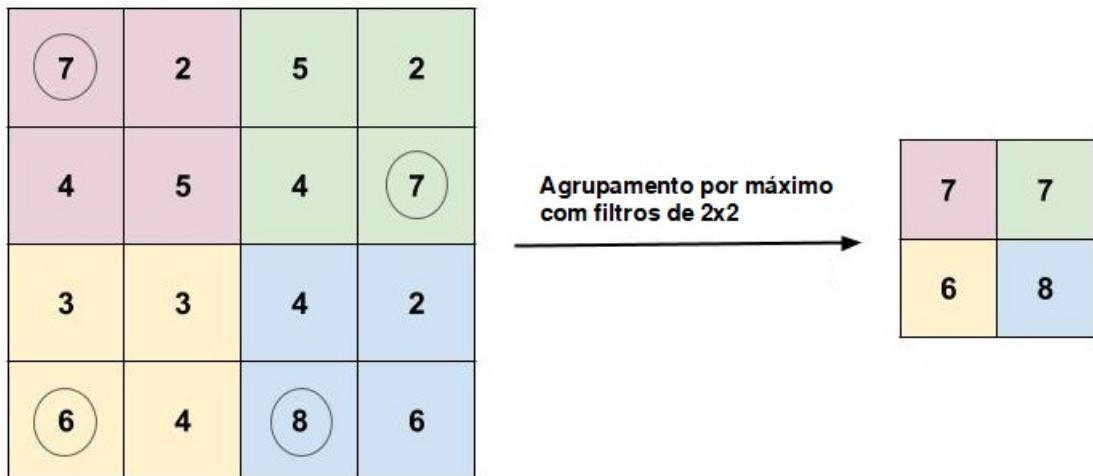


Figura 2.10: Representação de uma camada de agrupamento por máximo em uma Rede Neural aplicada no contexto de pré-processamento de imagens. Adaptado de [13].

suem características associadas ao tamanho do filtro (dimensão do Kernel, na convolução) e do passo. Normalmente é do interesse do algoritmo reduzir a dimensão dos dados após a camada de agrupamento, então o tamanho do passo costuma ser superior a 1, e não são adicionados *paddings* como em camadas de convolução.

#### 2.1.3.4 Camada de Classificação

Como mostrado anteriormente, funções de ativação adicionam não-linearidade aos modelos de Redes Neurais e possibilitam a modelagem de funções complexas pela redes. Isoladas, funções de ativação costumam apresentar um formato simples, como a função sigmóide da Figura 2.5 onde a saída do neurônio é “ativada” (1) ou não (0) dependendo do resultado da combinação linear que, por sua vez, depende dos pesos e viés. As funções assumem formatos mais complexos à medida que a quantidade de neurônios é aumentada.

A Figura 2.11, por exemplo, apresenta o formato final da combinação de duas funções ativações de dois neurônios. No exemplo, cada neurônio possui funções de ativação do tipo degrau com limites de ativação diferentes (0,4 e 0,6) que, quando linearmente combinados na camada subsequente, produzem um formato de função

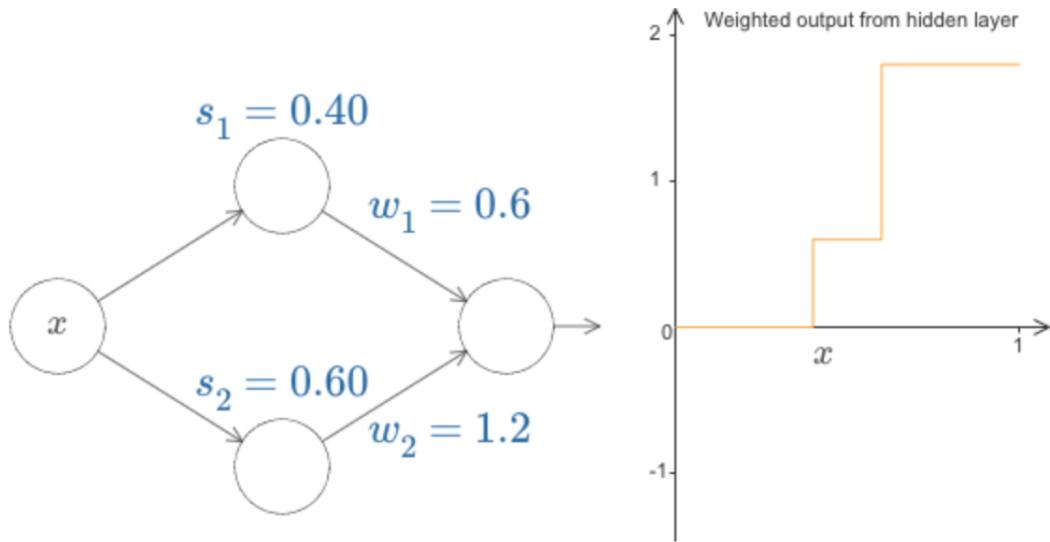


Figura 2.11: Representação da combinação de dois neurônios com funções de ativação do tipo degrau, resultando em uma modelagem mais complexa [4].

mais complexo. Assim, é possível também concluir que outros formatos de funções também podem ser modelados através da combinação de outros tipos de funções ativação.

Este é o principal objetivo do grupo de camadas de classificação em uma Rede Neural Convolucional, onde camadas com funções de ativação são combinadas para representar um classificador complexo não-linear. Existem diversos tipos de funções de ativação, e abaixo são listados alguns dos tipos mais comuns utilizados em CNNs.

- **Função Degrau:** Função discreta que representa dois estados separados por um limite discreto  $a$ .

$$\begin{cases} f(x) = 0, & x \geq a \\ f(x) = 1, & x \leq a \end{cases} \quad (2.5)$$

- **Função Sigmóide:** Função contínua com um formato de degrau suavizado, variando de 0 a 1.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.6)$$

- **Função Tanh:** Tangente Hiperbólica, uma versão escalonada da função Sig-

móide, variando de -1 a 1.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (2.7)$$

- **Função ReLU:** Unidade linear rectificada (do inglês, *Rectified Linear Unit*), ativa somente neurônios que recebem valores positivos de entrada.

$$f(x) = \max(0, x) \quad (2.8)$$

- **Função Softmax:** Tipo de função Sigmóide aplicada para operações de classificação com  $K$  classes.

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}, j = 1, \dots, K \quad (2.9)$$

### 2.1.3.5 Camada de *Dropout*

Uma boa capacidade de generalização, ou seja, a habilidade de previsão de resultados corretos em dados novos, é o principal objetivo dos algoritmos de *machine learning*, como as Redes Neurais. *Overfitting*, ou sobreajuste, é o termo utilizado quando os modelos desenvolvidos possuem alta acurácia para os dados de treinamento, mas alto índice de erro na validação; e regularização (*regularization*) é o nome dado às técnicas utilizadas para prevenção de *overfitting* através da redução do erro de generalização [10].

Em Redes Neurais Articiais, uma destas técnicas é a operação de *dropout*. Baseada na técnica de *bagging*, onde diversos modelos diferentes são treinados e avaliados, *dropout* é uma alternativa mais barata em termos de poder computacional, já que consiste somente na eliminação temporária e aleatória de neurônios em camadas ocultas da rede durante o treinamento (Figura 2.12) [31].

Em termos práticos, *dropout* desabilita de forma aleatória alguns neurônios du-

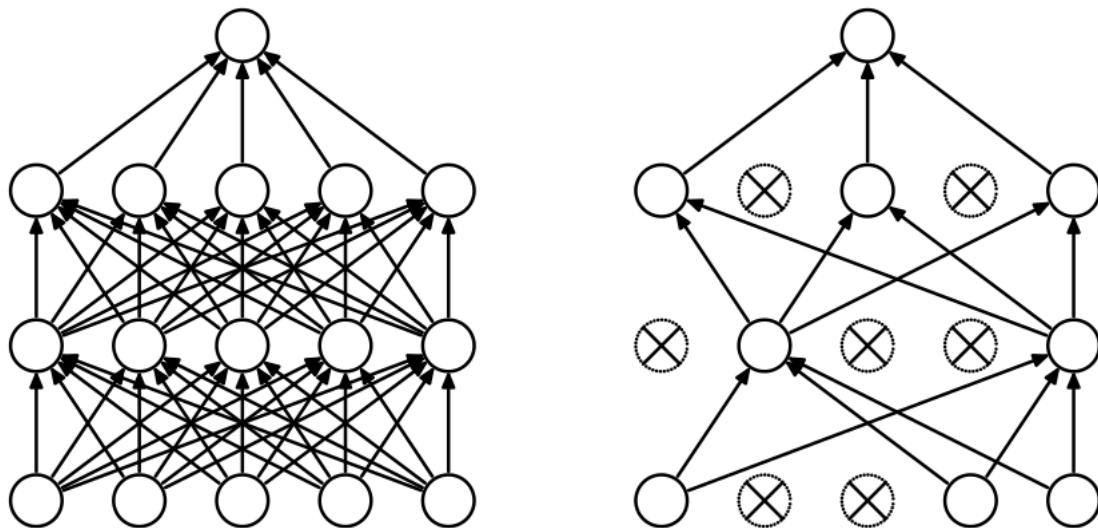


Figura 2.12: Representação da operação de *dropout*. À esquerda, Rede Neural padrão com duas camadas ocultas, à direita, aplicação de *dropout* no treinamento, com remoção aleatória de neurônios para prevenir o *overfitting* [31].

rante as etapas de propagação direta e retropropagação para uma certa quantidade de dados de treino. Em seguida, estes são reabilitados e um outro subgrupo de neurônios é desativado para o treinamento em cima de outros dados de treino. Esse processo continua aleatoriamente até que todos os dados sejam utilizados no treinamento, por quantas *epochs* necessárias [4].

Esta desativação aleatória de neurônios previne que seus pesos sejam modificados para se ajustarem perfeitamente aos dados de treino mas não necessariamente a dados novos. Como Redes Neurais Artificiais são modelos muito complexos que podem facilmente sofrer *overfitting*, é bastante comum adicionar camadas de *dropout* entre diferentes camadas ocultas.

## 2.2 Transferência de Conhecimento

Redes Neurais Artificiais podem ser modelos extremamente complexos, alcançando facilmente milhões de parâmetros no processo de aprendizagem. Para um treinamento eficiente destes modelos, são também usualmente necessários milhões de dados, e grande capacidade computacional. Uma técnica bastante utilizada para

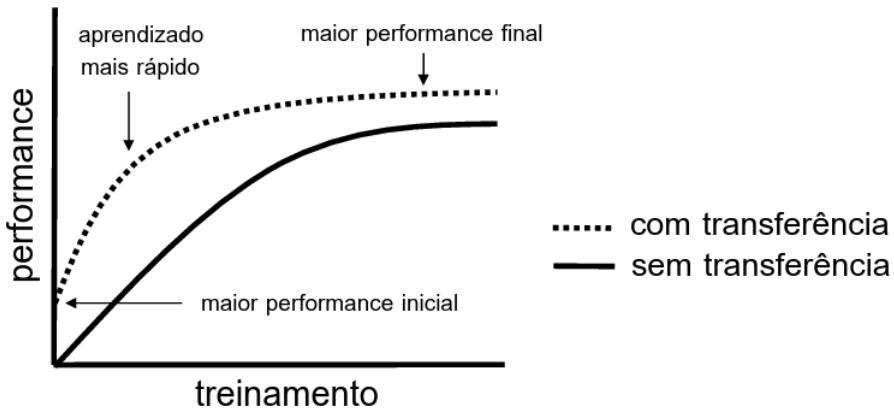


Figura 2.13: Benefícios da técnica de Transferência de Conhecimento no processo de aprendizado de um modelo de *machine learning*. Traduzido de [34].

contornar esta situação é chamada de Transferência de Conhecimento (do inglês, *Transfer Learning*), e se baseia em aplicar os pesos de Redes Neurais pré-treinadas em um novo treinamento com dados diferentes e específicos ao problema a ser resolvido.

Transferência de Conhecimento pode ser definida como uma técnica na área de Aprendizagem de Máquina em que o treinamento de modelos é realizado utilizando não somente os dados de entrada, mas uma outra fonte de conhecimento, adquirido em um treinamento prévio sob outras circunstâncias e dados [34]. O objetivo desta técnica é aperfeiçoar o processo de aprendizado alavancando um conhecimento prévio, o que pode aumentar a performance inicial, acelerar o processo de treinamento, e também aumentar a acurácia do modelo final (Figura 2.13).

Em Redes Neurais Convolucionais, este conceito é particularmente útil, já que uma grande parte das camadas de extração de recursos possuem características genéricas para o pré-processamento de imagens, e podem ser utilizadas inclusive sem um novo treinamento. Camadas de baixo nível, por exemplo, como as primeiras camadas de *feature extractions*, podem ser utilizadas diretamente, e as demais são retrainadas a partir de um estado inicial já otimizado, reduzindo significativamente o tamanho da base de dados e poder computacional necessário para um treinamento completo.

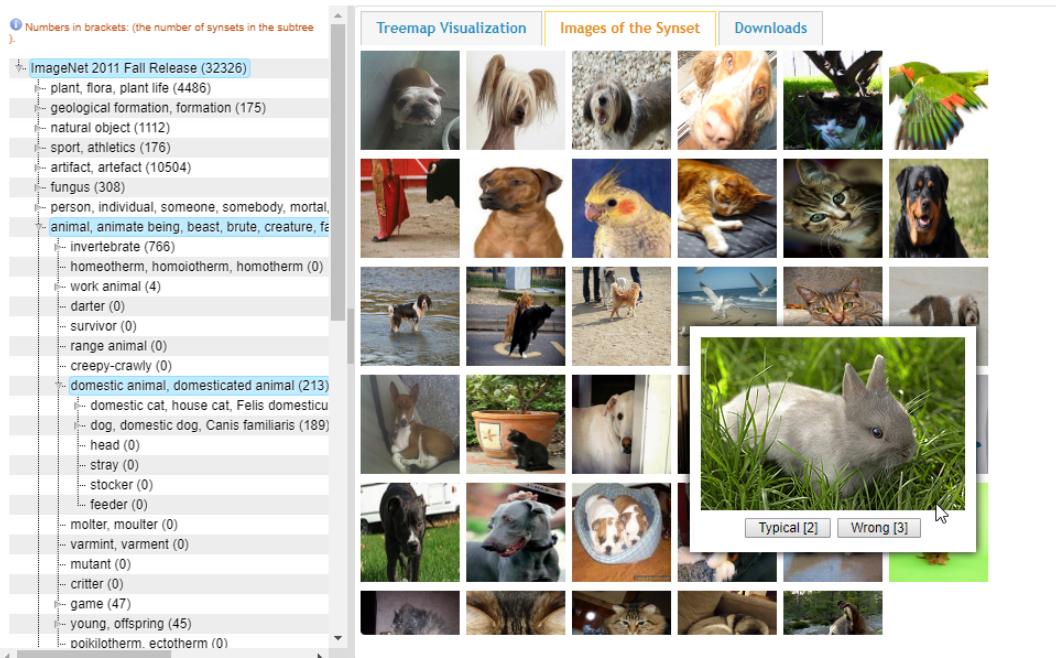


Figura 2.14: Exemplo de uma janela de exploração das imagens presentes na base de dados do ImageNet, mostrando um trecho do subgrupo de animais domésticos [2].

Certamente, a Transferência de Conhecimento deve ser realizada de forma consciente e cuidadosa, para que seja possível se aproveitar de características de outras Redes Neurais que sejam benéficas ao problema em questão. Na área de classificação de imagens, uma grande fonte de Redes Neurais pré-treinadas são os modelos disponibilizados ao público durante às competições do ImageNet.

### 2.2.1 ImageNet

ImageNet é um projeto de pesquisa para o desenvolvimento de uma base de dados de larga escala para imagens (Figura 2.14) [2]. Coordenado por um grupo de professores e pesquisadores de diferentes universidades, possui atualmente um conjunto de mais de 14 milhões de imagens estruturadas de acordo com a hierarquia do WordNet, um banco de dados léxico de substantivos, verbos, adjetivos e advérbios em inglês [3].

Desde 2010, o projeto ImageNet coordena uma competição anual, chamada *ImageNet Large Scale Visual Recognition Challenge* (ou ILSVRC), com o objetivo de

acelerar a pesquisa e desenvolvimento de algoritmos de classificação de imagens e detecção de objetos pela comunidade científica mundial. Alguns dos principais modelos de Redes Neurais Convolucionais pré-treinados utilizados em estratégias de transferência de conhecimento são de participantes bem colocados em diferentes anos desta competição, tais como VGG16, Inception V3 e ResNet50.

### 2.2.2 VGG

O VGG é um modelo desenvolvido pelo grupo Oxford Visual Geometry Group, que obteve segundo lugar na competição do ImageNet de 2014 na categoria de classificação de objetos com uma acurácia de 25,3231% e uma taxa de erro 0,7405% [19,29]. O primeiro lugar deste ano da competição foi alcançado pelo modelo GoogLeNet, discutido na seção a seguir, e os resultados podem ser encontrados na íntegra em <http://image-net.org/challenges/LSVRC/2014/results>.

Para a competição deste ano, o *dataset* utilizado foi o ILSVRC-2012, com 1000 categorias de objetos e contou com 1,3 milhões de imagens no treinamento, 50 mil imagens de validação e 100 mil imagens para testar o modelo.

A arquitetura do VGG contempla duas versões principais, com 16 e 19 camadas de profundidade (*hidden layers*), filtros convolucionais de 3x3 e com uma entrada fixa de 224x224 RGB, o que representa uma matriz com 50.176 características (*features*). O RGB é composto por 3 cores (vermelho, verde e azul), para fazer a redução desse vetor de 3 para uma em seu treinamento, foi utilizada a média das cores durante o pré-processamento, o que permitiu a redução das características para 1/3.

A estrutura deste modelo é esquematizada a seguir na Figura 2.15 e também na Tabela 2.1.

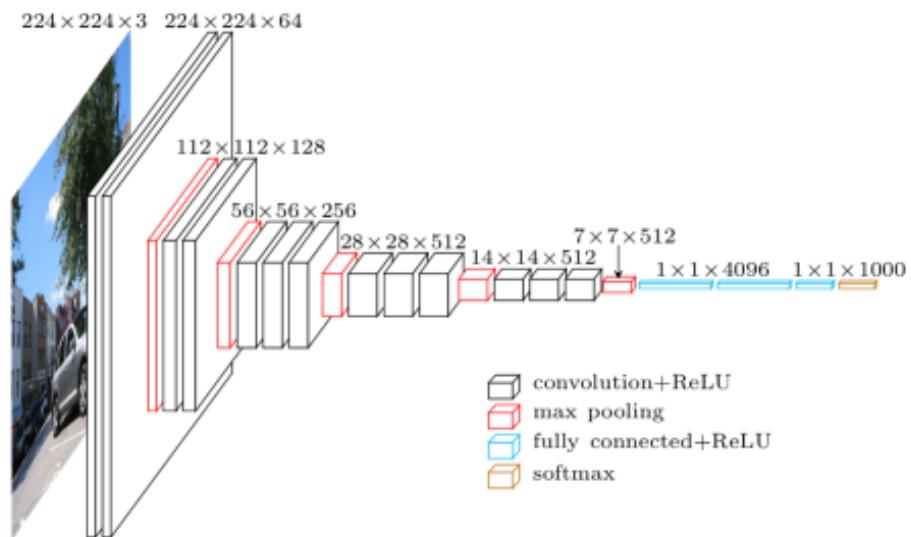


Figura 2.15: Arquitetura macro do VGG com 16 e 19 camadas.

Tabela 2.1: VGG com 16 e 19 camadas

<b>16 camadas</b>	<b>19 camadas</b>
entrada (224×224 RGB)	
conv3-64	conv3-64
conv3-64	conv3-64
Maxpool	
conv3-128	conv3-128
conv3-128	conv3-128
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
Maxpool	
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
Maxpool	
FC-4096	
FC-4096	
FC-1000	
softmax	

### 2.2.3 Inception V3

Inception V3 é a terceira versão do modelo GoogLeNet, arquitetura campeã do desafio de classificação de imagem do ILSVRC de 2014 [19], com erro em 6,656%. Desenvolvido pela Google, é um modelo que se aproveita de camadas de convoluções fatorizadas, e regularização agressiva. O principal objetivo deste formato de arquitetura é agir como um extrator de recursos “multi-nível”, onde operações de convolução são conduzidas em paralelo, com tamanhos de Kernel distintos (e.g. 1x1, 3x3, e 5x5), e estratégias de redução de dimensionalidade, como mostra a Figura 2.16 [32, 33].

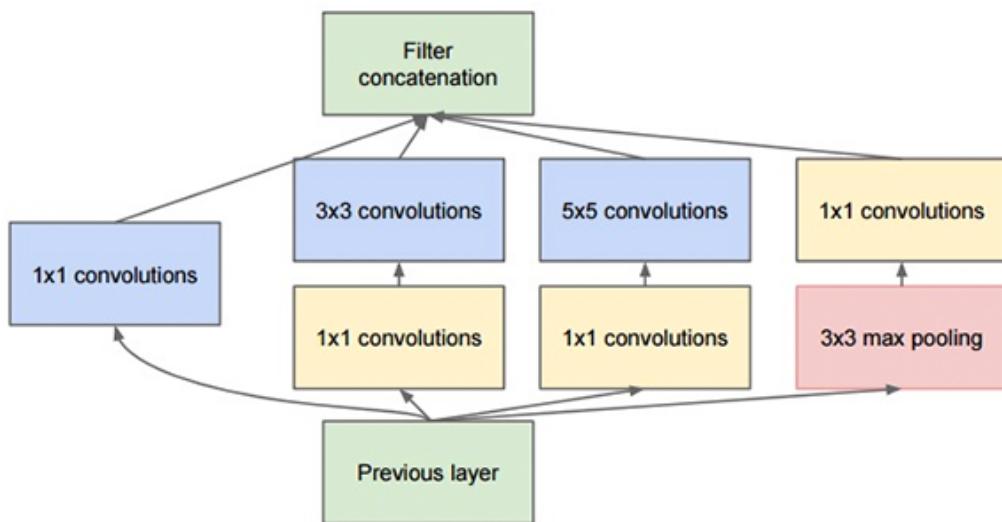


Figura 2.16: Módulo chamado *inception* na arquitetura original do InceptionV3, representando operações de convolução fatorizadas e redução de dimensionalidade.

Na sua última versão, a arquitetura do InceptionV3 proposta possui 6 camadas de convolução isoladas, duas camadas de pooling, três módulos de *Inception*, variantes da Figura 2.16, uma camada completamente conectada com função de ativação linear, e uma camada de saída com função de ativação *softmax*.

## 2.2.4 ResNet50

A ResNet50 foi vencedora do desafio da ImageNet 2015 de classificação e apresentou uma taxa de erro de 3,57%.

A ResNet50 [15] possui um conceito exótico para arquitetura de uma rede convolucional. Sua arquitetura é composta por blocos, onde cada bloco possui camadas (**hidden layers**) que se combinam com sua própria entrada, de tal forma que a redução da complexidade é feita pela desativação do neurônio com a aplicação do *ReLU* no final do bloco.

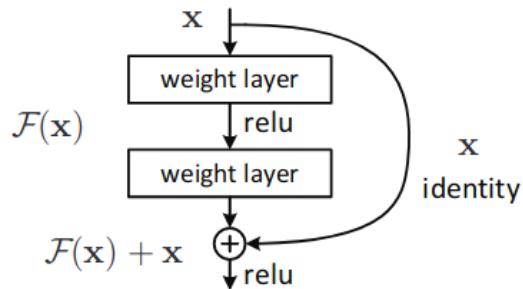


Figura 2.17: Bloco residual

Vale notar, que cada bloco pode ser constituído com qualquer quantidade de camadas internas.

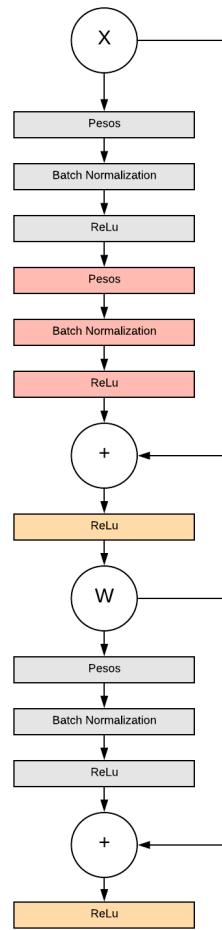


Figura 2.18: Bloco residual com *batch normalization* (BN) e quantidades diferentes de camadas por bloco

## 2.3 Tecnologias

As tecnologias elencadas para o desenvolvimento do projeto levou em consideração a linguagem de programação, bibliotecas auxiliares e ambiente onde seriam executados, visando a maior flexibilidade possível e capacidade de processamento. As seções a seguir mencionam brevemente as tecnologias aplicadas neste trabalho.

### 2.3.1 Python

Python é uma linguagem multi propósito muito utilizada na área de ciência de dados e, extremamente compatível com o ML Engine, Tensorflow e Keras. Este

texto não tem à intenção de ser um tutorial, já que existem muitos tutoriais e cursos bons disponíveis no mercado. Aqui, fica limitado apenas à informação de que todos os códigos desenvolvidos neste projeto utilizaram esta linguagem.

### 2.3.2 Google Cloud Platform

O GCP (Google Cloud Platform) é uma infraestrutura global de computação em nuvem com alto poder de processamento e fácil escalabilidade. O GCP foi escolhido para este estudo pois possui serviços específicos para *Machine Learning*, permitindo aos autores concentrar o desenvolvimento nas tarefas específicas ao problema endereçado.



Figura 2.19: Google Cloud Platform

Dentro do GCP, foram utilizados os serviços de **Compute Engine** para hospedar Jupyter Notebooks, o **Cloud Storage** como solução de armazenamento, e o **ML Engine** para treinar os modelos.

#### 2.3.2.1 Compute Engine

O Compute Engine é um módulo do GCP que possui grande flexibilidade nas opções de criação de Máquinas Virtuais de forma fácil e rápida. Permitindo criação desde micro instâncias de 1vCPU compartilhado com 600MB de memória, até instâncias com 160vCPUs e 3,75TB de memória, possui diversos recursos que podem ser consultados em <https://cloud.google.com/compute/>.



Figura 2.20: Compute Engine Logo

### 2.3.2.2 Cloud Storage

O Cloud Storage é um serviço de armazenamento distribuído globalmente de dados, capaz de armazenar exabytes de dados, com disponibilidade de 99,999999999%, e latência de início de transferência em milissegundos.



Figura 2.21: Cloud Storage Logo

Aliado à largura de banda da rede do Google Cloud Platform, o Cloud Storage permite que a transferência de arquivos muito grandes seja realizada em altíssima velocidade chegando a ser mais rápido que utilizar discos magnéticos.

### 2.3.2.3 ML Engine

ML Engine é um serviço de treinamento e disponibilização de modelos de *Machine Learning*. Projetado com alta integração ao Tensorflow, permite o treinamento distribuído em múltiplos nós de forma automática com apenas poucas configurações.

## 2.3.3 TensorFlow

Tensorflow é uma biblioteca de *opensource*, desenvolvida pela Google, para trabalhar *Machine Learning*, mais especificamente *Deep Learning*.



Figura 2.22: ML Engine Logo



Figura 2.23: Tensorflow Logo

### 2.3.3.1 Keras

Por a biblioteca do TensorFlow ser considerada complicada e necessitar de conhecimentos avançados em Álgebra Linear, a biblioteca Keras foi desenvolvida. Esta visa facilitar a implementação de modelos utilizando o TensorFlow como *backend*, mas permitindo também utilizar outros *backends* como Theano e CNTK para aumentar sua flexibilidade.

### 2.3.4 Jupyter Notebook

O Jupyter Notebook é uma aplicação *web* capaz de criar documentos interativos com diversas linguagens de programação, e, neste trabalho, foi utilizado no pré-processamento de dados e cálculos de métricas de performance dos modelos.



Figura 2.24: Keras Logo

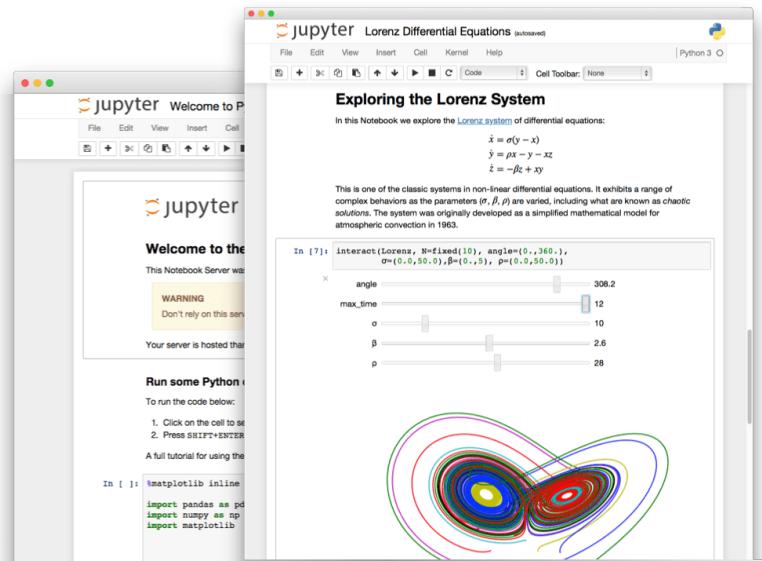


Figura 2.25: Exemplo de Jupyter Notebook

# Capítulo 3

## Desenvolvimento e Experimentos

Todo projeto em *machine learning* pode se tornar bastante complexo, mas ainda assim, com algumas variações, é comum representá-lo em um processo de seis grandes etapas, como mostra a Figura 3.1. Começando pela definição do problema a ser resolvido, aquisição e preparação de dados, é seguido pela análise de modelos e processos, pelo reporte dos resultados, e, finalmente, a utilização do modelo em situações reais (em produção) [35]. Nota-se que este fluxo não é limitado a uma única direção, mas comumente iterativo, onde passos anteriores podem ser reavaliados com novas informações adquiridas posteriormente. Por exemplo, a preparação e exploração dos dados adquiridos pode revelar informações cruciais ao problema em questão, levando a um ajuste na definição inicial do mesmo.

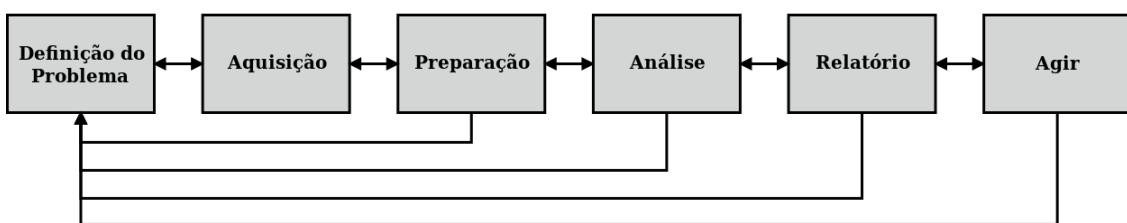


Figura 3.1: Processo genérico iterativo de um projeto em aprendizagem de máquina, com seis etapas. Fonte própria.

As seções a seguir buscam apresentar o desenvolvimento e experimentos realizados neste trabalho seguindo as etapas aqui definidas.

### 3.1 Descrição do Problema

A primeira etapa de qualquer projeto de *machine learning* deve ser a busca por um entendimento maior do contexto e do problema a ser resolvido. Este é um passo crucial para o negócio, e serve como guia para todo o futuro desenvolvimento.

Para o presente trabalho, o principal objetivo é explorar a aplicação de Redes Neurais Convolucionais em um problema clássico de classificação de imagens. Mais especificamente, foi definido um objetivo de determinação automática de espécies de flores presentes em fotos reais, como apresentado na Figura 3.2.



**Daffodil**  
(Narciso)



**Lily Valley**  
(Lírio-do-Vale)

Figura 3.2: Exemplos de fotografias de flores e seus respectivos nomes em inglês e português.

É interessante perceber que no mundo hoje já são classificadas centenas de milhares de espécies de flores diferentes, o que aumenta exponencialmente o escopo do problema definido. Para viabilizar esta execução no contexto de um Trabalho de Conclusão de Curso, um número significativamente reduzido de espécies de flores será incluído no problema de classificação. É buscado, assim, a seleção de flores populares, as quais comumente aparecem em fotografias publicadas em redes sociais, por exemplo.

Apesar de mais simplificado, com este objetivo, um modelo bem sucedido de clas-

sificação pode ser utilizado em algumas aplicações práticas bastante interessantes, tal como: mineração de dados de fotografias publicadas para avaliação da influência do aquecimento global nas épocas de florescimento de algumas flores. Não é o objetivo deste trabalho responder a este problema, mas sim desenvolver o que pode ser usado no futuro como uma ferramenta para respondê-lo.

## 3.2 Aquisição de Dados

Uma descrição clara do problema a ser resolvido auxilia na identificação dos dados necessários ao processo. A aquisição destes dados irá sustentar todas as demais etapas, e possui grande influência sobre a qualidade dos resultados finais a serem obtidos. Esta etapa é responsável por:

- Identificação de conjuntos de dados disponíveis (*datasets*);
- Busca, coleta e armazenamento de dados;
- Disponibilização dos dados armazenados para consulta.

Muitas vezes os dados não estão disponíveis em um só lugar e precisam ser coletados e agrupados em um único local em um processo chamado de ETL (do inglês *Extract Transform and Load*, ou Extração, Transformação e Carga). Para a classificação de tipos de flores, se faz necessária uma base de dados relativamente extensa de fotografias reais de diferentes espécies de flores. O presente trabalho utilizou um *dataset* público, o qual foi obtido por outros autores em um processo de três etapas, resumido a seguir [6]:

1. Acesso à API de uma rede social de fotografias chamada Flickr;
2. Busca pela API baseada em palavras-chave de nomes de flores pré-definidos;
3. *Download* das fotografias encontradas em uma estrutura organizada por tipo de flores;

Os dados prontos obtidos por este trabalho, então, incluem 5111 imagens distribuídas entre 17 diferentes tipos de flores. A Figura 3.3 lista estes tipos e apresenta alguns exemplos de flores obtidas para cada categoria.



Figura 3.3: Exemplos de imagens obtidas para cada uma das 17 categorias de flores.

### 3.3 Preparação dos dados

A etapa de preparação dos dados é comumente dividida em duas fases: exploração e pré-processamento.

#### 3.3.1 Exploração

No processo de exploração entende-se a natureza dos dados através de uma análise preliminar, onde é possível identificar correlações, tendências, distribuição e, inclusive gerar visualizações gráficas. São muitas opções de análise, mas nem sempre é necessário, ou possível, aplicar todas estas técnicas em um *dataset*.

Especificamente para o problema tratado neste trabalho, os dados são imagens de flores, as quais são representadas como um *array* de valores numéricos, os *pixels*. Sendo coloridas, estas imagens podem ser representadas por um *array* de 3 dimensões ( $x, y, c$ ):  $x$  pixels na horizontal,  $y$  na vertical, e  $c = 3$  camadas de cores no sistema RGB. Cada pixel individual possui um valor numérico variando de 0 a 255, que representa a intensidade da cor da sua respectiva camada RGB (vermelho, verde ou azul), e a combinação destes valores compõe a cor final da imagem (Figura 3.4).

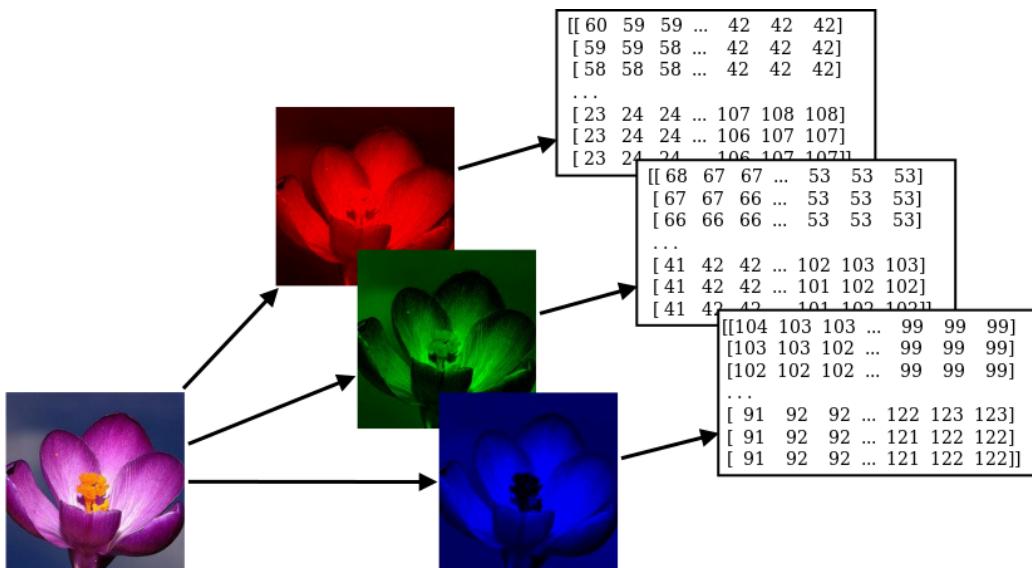


Figura 3.4: Representação de uma imagem colorida no sistema RGB, onde cada pixel corresponde a uma combinação de valores numéricos de cada canal RGB. Fonte própria.

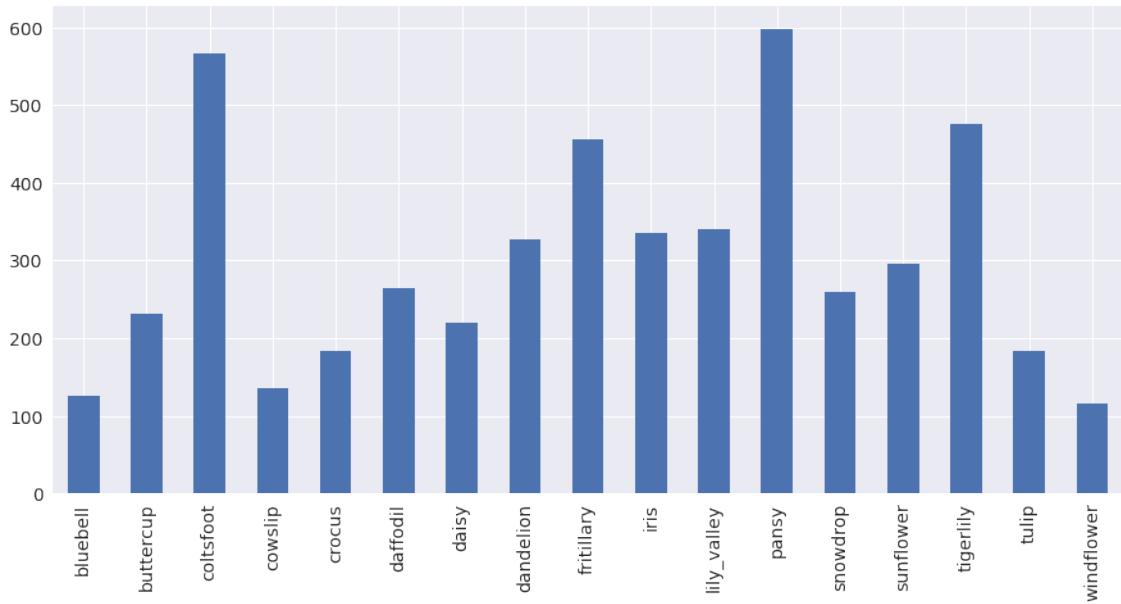


Figura 3.5: Distribuição de imagens para cada categoria de flor utilizada neste estudo.

Para o problema em análise, os valores numéricos de cada pixel das imagens não representam dados interessantes a exploração para fins de correlação, tendências, e afins. Considerando um problema de classificação, a principal característica dos dados é dada pela distribuição das imagens em cada uma categorias. A Figura 3.5 apresenta esta distribuição, mostrando a quantidade de imagens presentes para cada tipo de flor.

Pela Figura 3.5 é possível perceber que o dataset é consideravelmente desbalanceado, onde a categoria que possui maior quantidade de imagens, *pansy*, é representada por 598 arquivos, enquanto que a categoria com menos, *windflower*, possui 166. Esta é uma diferença de mais de 415% entre os extremos, sendo importante de se reconhecer para que as métricas de desempenho dos modelos de *machine learning* a serem desenvolvidos sejam avaliadas corretamente. A figura e tabela a seguir apresentam as principais características desta distribuição de dados em cada categoria, onde são notáveis os valores da média e desvio padrão: 301 e 148.0, respectivamente.

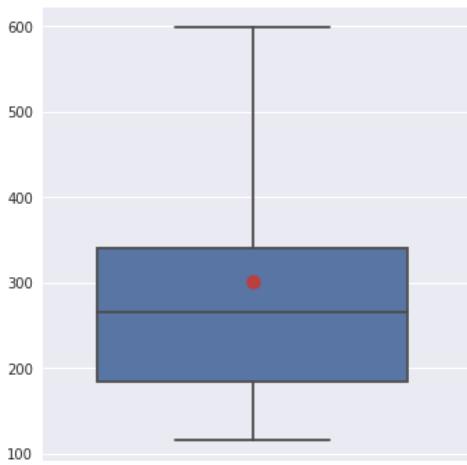


Figura 3.6: Gráfico *boxplot* da distribuição de imagens.

<b>máximo</b>	598.0
<b>Q3 (75%)</b>	340.0
<b>média</b>	300.6
<b>desvio padrão</b>	148.0
<b>mediana</b>	265.0
<b>Q1 (25%)</b>	184.0
<b>mínimo</b>	116.0

Tabela 3.1: Propriedades da distribuição de imagens.

### 3.3.2 Pré-Processamento

Na etapa de pré-processamento é realizada principalmente a limpeza e a transformação dos dados, de forma a elaborar um *dataset* de qualidade, pronto para uso em um modelo de *machine learning*.

A **limpeza de dados** é responsável por resolver erros, inconsistências e outros problemas que possam surgir no conjunto de dados, tais como valores faltantes, dados duplicados, dados inconsistentes, ruídos, *outliers*, entre outros. Como mencionado anteriormente, o *dataset* utilizado no presente trabalho foi desenvolvido por outros autores, cujo trabalho incluiu a verificação da qualidade das imagens obtidas. Desta forma, não ficou evidenciada a necessidade de se realizar nenhuma limpeza nos dados.

De qualquer forma, uma avaliação qualitativa mais simplificada por amostragem foi realizada com o objetivo de familiarização com o *dataset*. Nesta, é interessante a identificação de imagens com ruídos diversos, por vezes significativos, e que podem prejudicar a performance de um algoritmo de classificação. As imagens na Figura 3.7 exemplificam alguns destes ruídos: insetos pousados nas flores, objetos no fundo, paisagens diferentes e efeitos artificiais para estilização de fotos, como bordas e filtros. No entanto, estas imagens identificadas não foram removidas do conjunto



Figura 3.7: Exemplos de imagens do *dataset* com alto ruído como insetos, objetos, pessoas, paisagens, bordas, etc.

de dados, como uma forma de avaliar a eficácia de algoritmos *deep learning* nestas tarefas difíceis.

Em seguida, a **seleção e transformação de dados** é responsável por identificar, adicionar, combinar e remover *features* (características) dos dados. Esta etapa não tem como objetivo a correção de problemas, como na limpeza, mas sim, uma otimização do formato e conteúdo de dados de modo a facilitar, ou mesmo viabilizar, o processo de aprendizagem e aumentar a performance dos modelos.

Algumas técnicas bastante utilizadas são *encoders* e técnicas de *feature importance*. No entanto, para o problema de classificação de imagem, foram aplicadas as seguintes transformações:

- Conversão de todas as imagens para o mesmo formato **jpeg**: As imagens obtidas através da API do Flickr aparecem em formatos diversos, como **png**, **tiff** e **bmp**. Esta conversão uniformiza o dataset, e previne o aparecimento de inconsistências nos dados em etapas futuras de importação e leitura.
- Redimensionamento das imagens para um tamanho padrão de 256x256 pixels: Como as imagens serão lidas em uma Rede Neural como arrays numéricos, onde cada pixel corresponde a um neurônio na camada de entrada, é necessá-

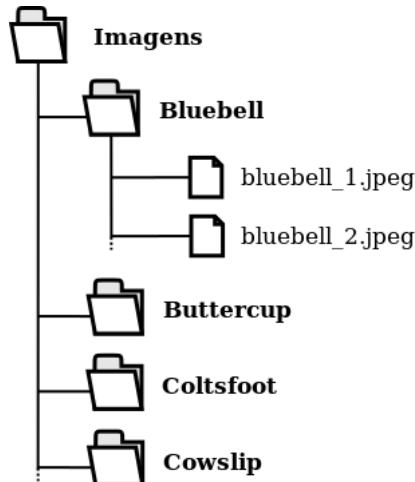


Figura 3.8: Esquema da estrutura de diretórios para armazenamento das imagens.

rio homogeneizar o tamanho de todas para serem lidas corretamente por uma mesma ANN. Para não alterar a razão de aspecto das imagens e possivelmente comprometer a classificação, as figuras retangulares foram reduzidas uniformemente até que o menor lado possua 256 pixels, e o maior lado é recortado pelos dois lados para alcançar a mesma dimensão.

Em seguida, as imagens são organizadas em uma estrutura de arquivos onde é criado um diretório para cada categoria de flor, e as fotografias são armazenadas em sua respectiva classe, como esquematiza a Figura 3.8. Neste formato, o algoritmo de leitura das imagens no modelo de *machine learning* pode facilmente identificar os *labels* a serem aplicados para cada arquivo.

Finalmente, nesta estrutura de arquivos, o *dataset* é então carregado para a nuvem, em um *storage* do Google Cloud Platform, de forma a concentrar todas as etapas futuras deste projeto em um só ambiente compartilhado que facilite a condução de todos os experimentos na etapa de análise. Para esta finalidade, também foi realizada uma última etapa de transformação dos dados: a conversão do *dataset* em arquivos binários no formato **tfrecord**.

**Tfrecord** é um formato de arquivo binário nativo do *framework* TensorFlow, e possui diversas vantagens em seu uso. Além de ocuparem menos espaço em disco e garantirem leitura e importação mais rápidas, arquivos neste formato são direta-

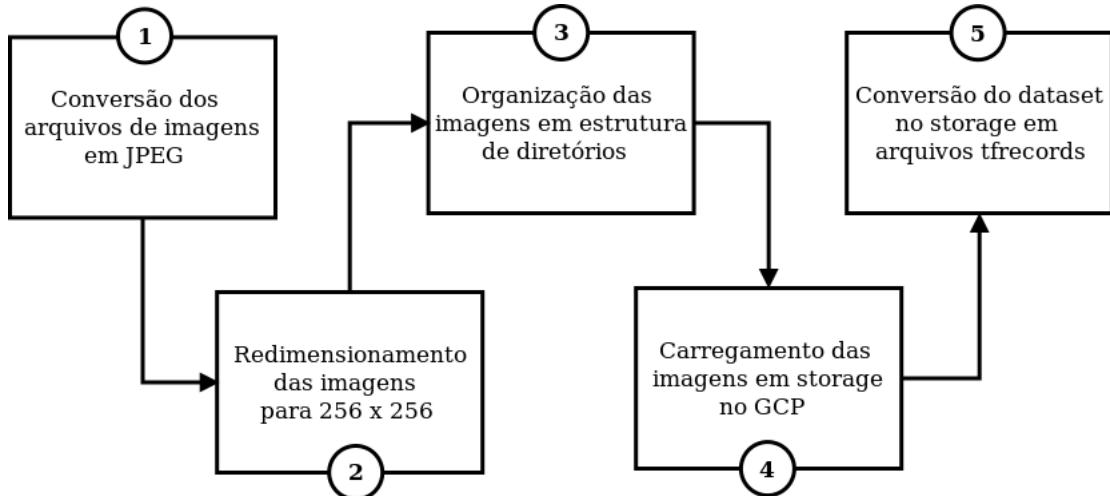


Figura 3.9: Etapas de pré-processamento realizadas nos dados.

mente integrados a diversas funções do TensorFlow, que permitem, por exemplo, uma separação automática do *dataset* em conjuntos de treino, teste e validação, e fácil mistura aleatória dos arquivos para evitar criação de viés no modelo, além de existirem funções nativas de pré-processamento e transformação (*augmentation*) de dados.

Outra grande vantagem de se utilizar o formato `tfrecord` é a aglomeração de diversos dados (neste caso, imagens) em um só arquivo, inclusive em conjunto com metadados, como os próprios *labels* das categorias as quais pertencem cada dado. Desta forma, a estrutura de diretórios apresentada na Figura 3.8 não se torna mais necessária, e o treinamento dos modelos de *machine learning* sendo realizados em *batches* lê um arquivo `tfrecord` por vez, evitando o carregamento de todo o *dataset* em memória - algo inviável para conjuntos de dados muito grandes.

Desta forma, este trabalho desenvolveu um algoritmo para conversão das imagens no *storage* do GCP em `tfrecords`, onde cada arquivo final possui 100 imagens, aleatoriamente misturadas entre categorias. Esta é a última etapa do processo de pré-processamento neste estudo, e a Figura 3.9 resume todos os passos realizados e descritos.

### 3.4 Análise de modelos

É na etapa de análise onde são selecionados os modelos de *machine learning*, é realizado o desenvolvimento e preparo dos mesmos para a condução de experimentos, e avaliação dos resultados.

No presente trabalho, foi aplicada a técnica de Transferência de Conhecimento, onde uma Rede Neural pré-treinada é utilizada no início de um outro processo de treinamento independente. A estratégia adotada foi a de se utilizar todas as camadas de *feature extraction* de uma CNN de alta performance, bem colocada em alguma versão da competição do ImageNet, com seus pesos originais do treinamento para a competição sem alteração no novo treinamento. Desta Rede, são adicionadas camadas de classificação cujos pesos, então, são efetivamente treinados com as imagens de flores deste projeto. Esta é uma abordagem que assume que as propriedades da extração de recursos treinada com imagens da base de dados do ImageNet devem ser semelhante àquelas necessárias para extração de recursos das imagens de flores, mas também garante uma grande economia de tempo de processamento no treinamento.

Já que nenhum dos parâmetros das camadas de *feature extraction* participam do treinamento da CNN, também foi aplicada uma estratégia adicional para redução de tempo de processamento, que consiste na geração de arquivos `tfrecords` das imagens após serem passadas por estas camadas. Desta forma, o resultado de todos os cálculos realizados na extração de recursos é persistido em disco, não se fazendo necessário um reprocessamento a cada vez que um novo experimento iniciar o treinamento das camadas de classificação.

Neste projeto, todo o desenvolvimento foi conduzido com a linguagem `python` em um formato que favorece a generalização. Em outras palavras, todo o código desenvolvido para o processo de treinamento e validação pode ser reutilizado em qualquer experimento do projeto sem a necessidade de modificação, apenas ajustando um *script* em shell com parâmetros que constroem e alimentam o modelo. O

principal objetivo do algoritmo é viabilizar e agilizar a implantação de um processo de treinamento de Redes Neurais Convolucionais em uma instância de máquina virtual do Google Cloud Platform, utilizando a biblioteca Keras, e acessando os dados de um *storage* dentro da mesma plataforma.

A Figura 3.10 esquematiza os principais componentes do algoritmo desenvolvido, cujo principal arquivo é o `train.py`. Neste, as principais tarefas são iniciadas e as dependências para os demais arquivos definidas. O algoritmo é iniciado por um comando descrito em um *script* em shell, que define todos os parâmetros necessários para iniciar o treinamento, tais como a localização dos dados no *storage* do GCP, definição das camadas de classificação e suas respectivas propriedades (função de ativação, número de neurônios, etc), quantidade de épocas de treinamento, função de perda, entre outros. Estes parâmetros são então iniciados dentro da estrutura do algoritmo através de uma classe definida no arquivo `parameter.py`.

Em seguida, as imagens são divididas em grupos de treino e validação, e importadas dos arquivos `tfrecords` pelo código em `dataset.py`. Nota-se que os `tfrecords` neste caso já contemplam o pré-processamento das camadas de extração de recursos, onde os cálculos são realizados previamente pelo `preprocessing.ipynb` e persistido em *storage*. Com as imagens importadas, as camadas de classificação da CNN são construídas de acordo com o especificado no comando inicial (em `train-on-cloud.sh`). Antes do modelo ser compilado em um modelo do Keras pronto para treinamento, o algoritmo otimizador, a função de perda e a métrica de performance são definidos.

Finalmente, o treinamento da Rede Neural é iniciado e, ao final, o modelo treinado é persistido em um arquivo `hdf5`. Para avaliação de resultados, a performance do experimento é medida com os dados de validação através de acurácia (calculada em tempo real durante o treinamento pelo método do Keras), e duas outras métricas descritas em `evaluate.ipynb`: F1-score, e matriz de confusão.

Com este algoritmo generalizado, diversos processos de treinamento foram ini-

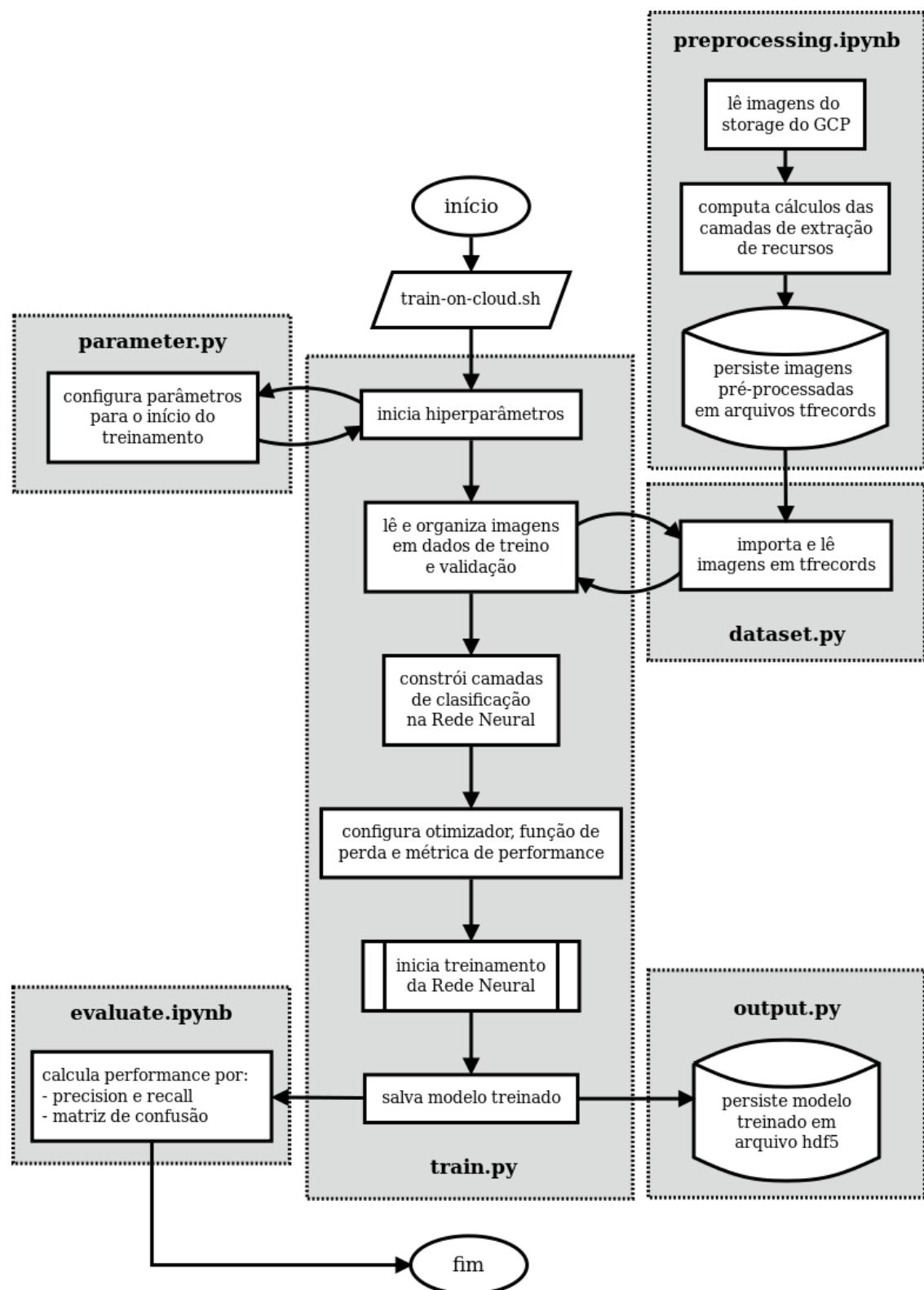


Figura 3.10: Esquema simplificado do algoritmo desenvolvido para os experimentos deste estudo. Em cinza são delimitadas as tarefas de cada arquivo de código python.

ciados como experimentos neste estudo, variando diferentes parâmetros como uma forma de avaliar a influências dos mesmos na performance final do modelo. Todos os experimentos foram executados em instâncias de máquinas virtuais com configurações idênticas no GCP, como descrito na Tabela 3.2. Por fim, todos os algoritmos desenvolvidos neste trabalho estão disponibilizado em repositório público no GitHub [5].

<b>Tipo de máquina no GCP</b>	n1-standard-4
<b>Plataforma de processamento</b>	Intel Sandy Bridge
<b>Núcleos de processamento</b>	4 vCPUs
<b>Memória</b>	15 GB

Tabela 3.2: Configuração da instância de máquina virtual utilizada em experimentos.

### 3.4.1 Experimentos

Uma avaliação profunda da influência dos parâmetros no modelo de Rede Neural Convolucional envolve a variação de dezenas de propriedades, desde a quantidade e tipo das camadas de classificação, até o algoritmo otimizador dos pesos. Técnicas como *Grid Search* ou *Randomized Search* podem ser utilizadas para o ajuste destes parâmetros, como formas de varreduras por diferentes valores e configurações de modelo. No entanto, ainda que o algoritmo desenvolvido neste projeto tenha flexibilizado a implementação de diferentes experimentos, o treinamento destes continua sendo um processo demorado, que exige alto poder computacional. Estas técnicas de ajuste de parâmetro, mesmo que otimizadas, levariam a um grande número de experimentos.

Levando em consideração o tempo de processamento necessário, e o custo associado ao uso de máquinas virtuais na plataforma do Google, não foi possível desenvolver um projeto com uma avaliação completa no contexto de um Trabalho de Conclusão de Curso. Para viabilizar este estudo, os experimentos foram determinados em etapas, e com base nos melhores resultados obtidos na etapa anterior.

Esta abordagem permite isolar a influência de um parâmetro dentro uma certa configuração, no entanto, não garante que o modelo final possui a melhor configuração possível, já que não há um teste de todos os parâmetros contra todos os demais.

A Tabela 3.3 lista os experimentos realizados, a ordem em que foram conduzidos e o que foi variado em cada um. Nota-se que a configuração-base do modelo da CNN pode variar de um experimento para outro, onde o formato com a melhor performance de um certo experimento é utilizado como a base no seguinte. As seções a seguir detalham as configurações das Redes Neurais de cada experimento.

#	Parâmetro variado	Comparações
1	Fração de neurônios desligados no <i>Dropout</i> entre camadas de classificação	0%, 25%, 50%, 75%
2	Número de neurônios da camada oculta de classificação	256, 512, 1024
3	Número de camadas ocultas de classificação	1, 2
4	Função de ativação da camada de classificação de saída	Softmax, Sigmóide, Tanh
5	Função de ativação da camada oculta de classificação	ReLU, Sigmóide, Tanh
6	Função de custo	Cross Entropy, Hinge
7	Algoritmo de otimização de pesos das camadas de classificação	RMSprop, Adam, SGD
8	Modelo de camadas e pesos da extração de recursos para <i>Transfer Learning</i>	VGG16, VGG19, InceptionV3, ResNet50, NASNet

Tabela 3.3: Sequência de experimentos realizados em etapa. A configuração com melhor performance de um experimento é utilizada no experimento seguinte.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 3.4: Parâmetros da configuração inicial da Rede Neural Convolucional aplicada neste estudo.

### 3.4.1.1 Configuração Inicial

Para iniciar os experimentos, uma primeira configuração de CNN foi estabelecida seguindo alguns padrões básicos encontrados na literatura. Para as camadas de *feature extraction*, foi utilizado o VGG16, a versão de 16 camadas do modelo desenvolvido pelo grupo Oxford Visual Geometry Group, e que obteve o segundo lugar tarefa de classificação na competição ILSVRC de 2014 [19, 30]. Os pesos calculados para esta competição foram disponibilizados pelo grupo e utilizados neste projeto.

Nas camadas de classificação, foi adicionada uma camada oculta com 256 neurônios e função de ativação ReLu, seguida por um *dropout* de 50% e finalmente a camada de classificação final com a função Softmax. O algoritmo otimizador selecionado foi o RMSprop e função de custo de entropia cruzada (*categorial cross entropy*, na biblioteca Keras). Para a configuração inicial e para todos os experimentos, a quantidade de épocas de treinamento foi mantida em 100. A Tabela 3.4 resume a configuração inicial da Rede Neural Convolucional.

### 3.4.1.2 Variando *Dropout*

O primeiro parâmetro a ser variado na sequência de experimentos é a fração de *dropout* entre as camadas de classificação, ou seja, a porcentagem de neurônios a serem desativados aleatoriamente durante o treinamento. Um pouco da teoria por trás do *dropout* é discutida na seção 2.1.3.5, e, para este estudo, foi interessante avaliar quatro configurações:

- Ausência de *dropout* (0%);
- 25% de *dropout*;
- 50% de *dropout*;
- 75 % de *dropout*;

onde os demais parâmetros da Rede Neural correspondem à configuração inicial na Tabela 3.4.

### 3.4.1.3 Variando Quantidade de Neurônios

Nesta etapa dos experimentos, apenas uma das configurações da comparação entre frações de *dropout* é selecionada e fixada, variando somente a quantidade de neurônios na camada oculta de classificação.

Como mostrado na Tabela 3.4, este experimento considera apenas uma camada oculta de classificação, onde o tamanho do vetor de entrada desta camada é definido pela extração de recursos do modelo pré-treinado VGG16, e não pode ser modificado. No entanto, a quantidade de neurônios definida para esta camada modifica o tamanho do vetor de saída, os quais vão alimentar as camadas subsequentes.

Não existe uma regra para se definir um número ideal de neurônios em uma ANN, ou mesmo para prever o comportamento da performance com a variação no tamanho de uma camada. A melhor alternativa para se definir a configuração ideal

é através de experimentação, e, para este estudo, três tamanhos diferentes foram definidos para os experimentos:

- 256 neurônios;
- 512 neurônios;
- 1024 neurônios.

#### 3.4.1.4 Variando Quantidade de Camadas Ocultas

De forma semelhante à definição da quantidade de neurônios em uma camada, não há uma regra sobre como a performance de uma Rede Neural melhora ou piora com adição de novas camadas ocultas. Pode ser constatado, no entanto, que a adição de camadas em uma ANN adiciona não-linearidade à modelagem, o que pode ser vantajoso para problemas muito complexos, altamente não lineares. Por esta possibilidade, fica interessante avaliar este parâmetro pode trazer benefícios à performance do modelo trabalho neste projeto.

Como o aumento da quantidade de camadas de uma ANN também consideravelmente o poder computacional necessário para seu treinamento, estes experimentos compararam apenas duas configurações:

- 1 camada oculta de classificação;
- 2 camadas ocultas de classificação;

onde nota-se que entre cada camada oculta, é também inserida uma camada de *dropout* (com a mesma fração).

#### 3.4.1.5 Variando Funções de Ativação da Camada de Saída

Como discutido anteriormente, funções de ativação são necessárias por duas principais razões: limitam os valores de saída de cada camada, e adicionam não-linearidade ao modelo, permitindo modelagens mais precisas para problemas com-

plexos. Em um problema de classificação com múltiplas categorias, é incrivelmente comum a aplicação da função de ativação Softmax na última camada da rede.

Esta função é uma generalização da função Sigmóide, a qual adiciona dependência dos resultados dos demais neurônios no seu cálculo, permitindo que os resultados finais representem efetivamente a probabilidade de uma certa classe, onde a soma de todos os resultados é igual a 1. Por esta razão, a Softmax foi utilizada como *default* na configuração final, mas não elimina a possibilidade de aplicação de outras funções de ativação como parte dos experimentos. Três funções foram avaliadas nesta etapa experimental:

- Função de ativação Softmax;
- Função de ativação Sigmóide;
- Função de ativação Tanh.

#### 3.4.1.6 Variando Funções de Ativação da Camada Oculta

A camada oculta de classificação também possui sua própria função de ativação, representando mais um parâmetro possível de ser variado em experimentos. A ReLU é uma função de ativação extremamente popular para camadas ocultas, tanto pela sua simplicidade, o que reduz o poder computacional necessário no treinamento e facilidade de convergência, quanto pela sua eficiência, sendo uma ótima substituta de funções mais complexas. No entanto, apesar de normalmente ser a primeira escolha na construção de uma Rede Neural, também apresenta alguns problemas, como a não limitação de valores positivos muito grandes, ou a proibição do aprendizado em casos com muitos dados de entrada negativos, o que pode levar à necessidade de se utilizar outras funções de ativação.

Nestes experimentos, a performance da CNN é avaliada variando entre três funções de ativação da camada oculta de classificação:

- Função de ativação ReLU;

- Função de ativação Sigmóide;
- Função de ativação Tanh.

#### 3.4.1.7 Variando Funções de Custo

Função de custo, em qualquer problema de otimização, se refere a um formato de cálculo da diferença entre o resultado “real” e o calculado pelo modelo sendo otimizado. É através desta equação que o algoritmo de otimização do modelo mede a evolução da performance durante o treinamento, e ajusta pesos e parâmetros para aumentá-la. Nestes experimentos, são aplicadas duas funções de custo diferentes:

- Função de custo *Hinge* (ou Função de perda de articulação);
- Função de custo *Cross-entropy* (ou Função de custo de Entropia cruzada).

#### 3.4.1.8 Variando Algoritmos de Otimização

Otimizadores são os algoritmos iterativos responsáveis por minimizar as funções de custo em um processo de treinamento, aumentando a performance do modelo a cada iteração. Como métodos numéricos, estes algoritmos podem ser os principais responsáveis não só por problemas de convergência, mas também por baixa performance, já que o processo de aprendizado pode ser direcionado e restrito a um mínimo local da função de custo, prevenindo a otimização de alcançar acurácia maiores. Para estes experimentos, três algoritmos bastante comuns de otimização foram aplicados:

- Otimizador RMSprop (*Root Mean Square Propagation*, variante do Gradiente Descendente com *momentum*, com taxa de aprendizado adaptativa);
- Otimizador Gradiente Descendente Estocástico (*Stochastic Gradient Descent*, ou SGD, variante do Gradiente Descendente onde a função de custo é computada para somente um sub-conjunto dos dados, definido aleatoriamente);

- Otimizador Adam (*Adaptive Moment Estimation*, também um método de otimização estocástico, porém com taxa de aprendizado adaptativa).

#### 3.4.1.9 Variando Modelos de *Feature Extraction*

Como discutido anteriormente, a técnica de *transfer learning* aplicada neste projeto consiste em utilizar os pesos e camadas de extração de recursos inalterados de modelos populares e disponíveis na literatura. A seção 2.2 descreve brevemente um pouco da teoria por trás dos diferentes modelos comparados nesta etapa de experimentos, e estes são listados abaixo:

- VGG16 (versão de 16 camadas do modelo desenvolvido pelo grupo Oxford Visual Geometry Group, que obteve segundo lugar na ILSVRC de 2014);
- VGG19 (versão de 19 camadas do modelo desenvolvido pelo grupo Oxford Visual Geometry Group, que obteve segundo lugar na ILSVRC de 2014);
- Inception V3 (versão 3 do modelo desenvolvido pelo Google, GoogLeNet, que obteve primeiro lugar na ILSVRC de 2014);
- ResNet50 (versão de 50 camadas do modelo desenvolvido pela Microsoft, que obteve o primeiro lugar na ILSVRC de 2015);
- NASNet (desenvolvido pelo Google, que supostamente apresentaria performance superior a todos os demais modelos que participaram das ILSVRC de 2017 e anteriores).

## 3.5 Reportar e Agir

Reportar é uma etapa fundamental de todo o projeto de Aprendizagem de Máquina, já que é a forma de expor e compartilhar os resultados obtidos para todos interessados. Ainda que resultados desfavoráveis, a compilação e exposição das

conclusões é de extrema importância para a comunidade pois auxiliará em desenvolvimentos futuros. Neste trabalho, os resultados obtidos nos experimentos são reportados no capítulo 4, utilizando as cinco principais métricas de performance a seguir:

- Acurácia: Medida da quantidade de dados classificados corretamente pelo modelo dividido pela quantidade total de dados classificados.
- Matriz de Confusão (*Confusion Matrix*): Organiza a classificação em uma matriz, onde as linhas representam os dados de uma certa categoria  $i$ , e as colunas, a categoria  $j$  prevista pelo modelo.

$$M_{ij}$$

- *Precision*: Para uma categoria  $i$ , representa a quantidade de dados classificados pelo modelo corretamente como categoria  $i$  dividido pelo número total de dados classificados como  $i$  pelo modelo.

$$P_i = \frac{M_{ii}}{\sum_j M_{ji}}$$

- *Recall*: Para uma categoria  $i$ , representa a quantidade de dados classificados pelo modelo corretamente como categoria  $i$  dividido pelo número de total de dados pertencentes à categoria  $i$ .

$$R_i = \frac{M_{ii}}{\sum_j M_{ij}}$$

- F1-score: Média harmônica das métricas de *Precision* e *Recall*);

$$F_i = 2 \cdot \frac{P_i \cdot R_i}{P_i + R_i}$$

Com resultados organizados e expostos, a etapa de “ação” é a última em um projeto bem-sucedido, e leva a extração real de valor da pesquisa conduzida. Seja ao colocar esta pesquisa em produção como um produto, ou recomeçar o processo novamente levando em consideração o aprendizado obtido em passos anteriores, agir significa utilizar os resultados obtidos. Etapas futuras deste projeto são discutidas no capítulo 5 de Conclusões.

# Capítulo 4

## Resultados

Neste capítulo são apresentados os resultados de todos os experimentos realizados no presente estudo, acompanhados de uma discussão técnica sobre cada um. As seções a seguir dividem a apresentação de resultados em cada etapa dos experimentos, como mostrado na Tabela 3.3.

Nota-se também que todos os resultados obtidos neste estudo também estão disponíveis em repositório público no [GitHub](#) [5].

### 4.1 *Dropout*

Nestes experimentos, foi variada a fração de *dropout* entre as camadas de classificação completamente conectadas da Rede Neural (Tabela 4.1). Esta propriedade é bastante importante para reduzir a possibilidade de *overfitting*, já que efetivamente significa que o treinamento considerará diversas arquiteturas diferentes ao variar quais neurônios são ativados ou desativados. Outra vantagem indireta do *dropout* é a redução do poder computacional necessário, já que a desativação de neurônios simplifica o modelo e reduz a quantidade de parâmetros a serem calculados no treinamento.

Esta característica de simplificação parece ter contribuído de forma inversa para os resultados dos experimentos conduzidos com as menores frações de *dropout*, 0

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>0, 25, 50 e 75%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 4.1: Estrutura da Rede Neural para os experimentos em *dropout*.

e 25%. Para estes casos, foram observados valores de acurácia em torno de 2,5%, extremamente baixo, e possivelmente pior que um classificador aleatório. Uma possível para isso está na complexidade do modelo e dificuldade de convergência pelo algoritmo de otimização. Com 75% ou mais de neurônios ativos na camada de classificação, é provável que o otimizador tenha se prendido a um mínimo local, prevenindo a evolução do treinamento.

Uma evidência desta possibilidade é a Matriz de Confusão destes experimentos, na Figura 4.1, que indica que o modelo treinado convergiu para uma condição de classificação uniforme, onde todas as imagens são classificadas como sendo flor *Bluebell*. Felizmente, resultados drasticamente melhores são obtidos quando o *dropout* é aumentado para 50% e 75%, como mostrado na Figura 4.2.

Nesta, observa-se que a acurácia dos dados de validação para estes experimentos foi de 54,7% e 81,6%, para *dropout* de 50 e 75%, respectivamente. Em adição aos dados de validação, também é bastante interessante observar a acurácia dos dados de treinamento para ambos experimentos, que alcançaram valores de 77,9 e 70,7%, respectivamente. Nota-se, então, que a acurácia de *dropout* 50% foi superior à de *dropout* 75% nos dados de treino, mas, na validação, essa performance caiu mais de 20% para o menor *dropout*, e subiu mais de 10% para o maior *dropout*, confirmando que este parâmetro auxilia no controle de *overfitting*.

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura 4.1: Matriz de Confusão dos experimentos de *dropout* 0 e 25%.

A Figura 4.3 apresenta a Matriz de Confusão para estes experimentos, onde é possível observar as categorias de flores com os menores e maiores erros de classificação. Para ambos valores de *dropout*, *Pansy*, *Snowdrop* e *Tigerlily* são as classes de flores que o modelo mais frequente classifica corretamente. Sobre o erro, com *dropout* 50%, nota-se que as principais classificações erradas são:

- *Buttercup*, *Cowslip* e *Crocus* classificadas como *Tulip*;
- *Coltsfoot* e *Daisy* classificadas como *Sunflower*; e
- *Iris* classificada como *Pansy*.

e com *dropout* 75%:

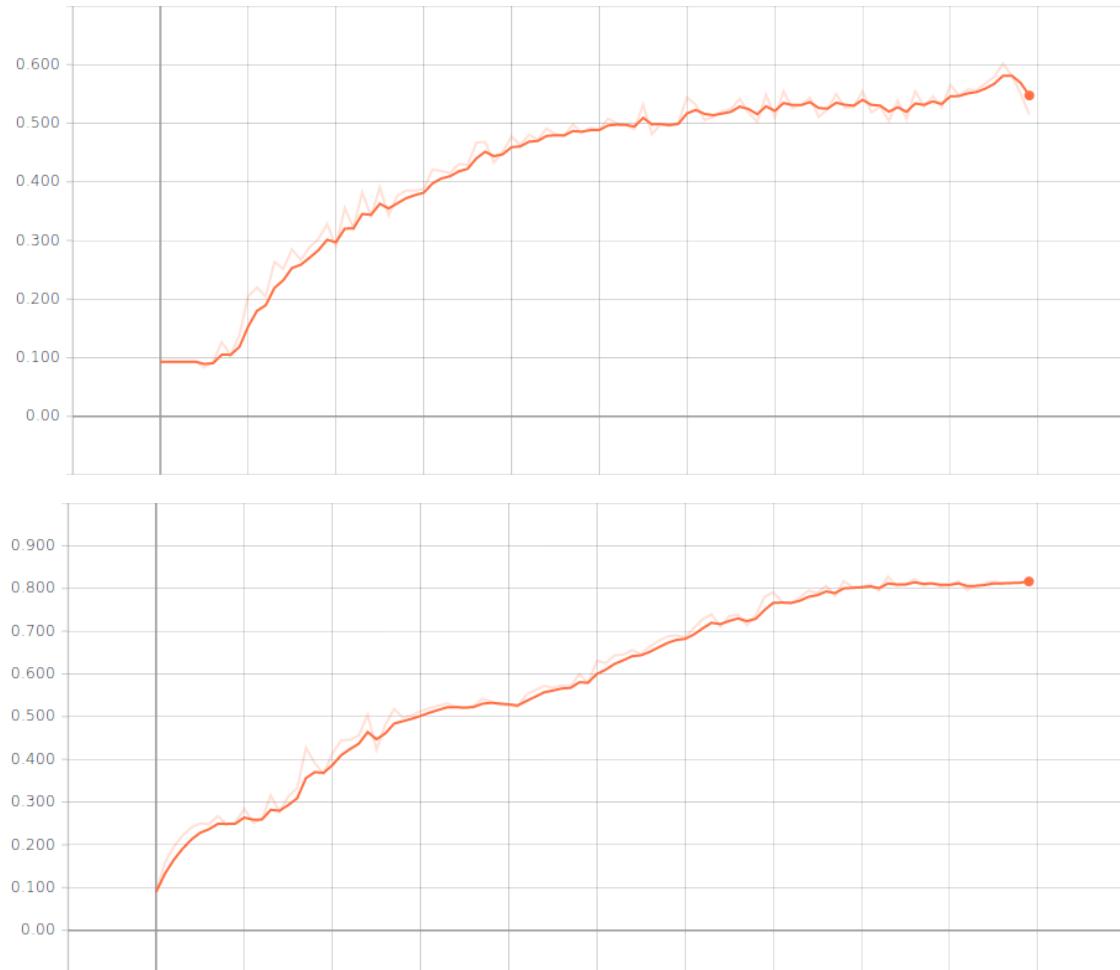
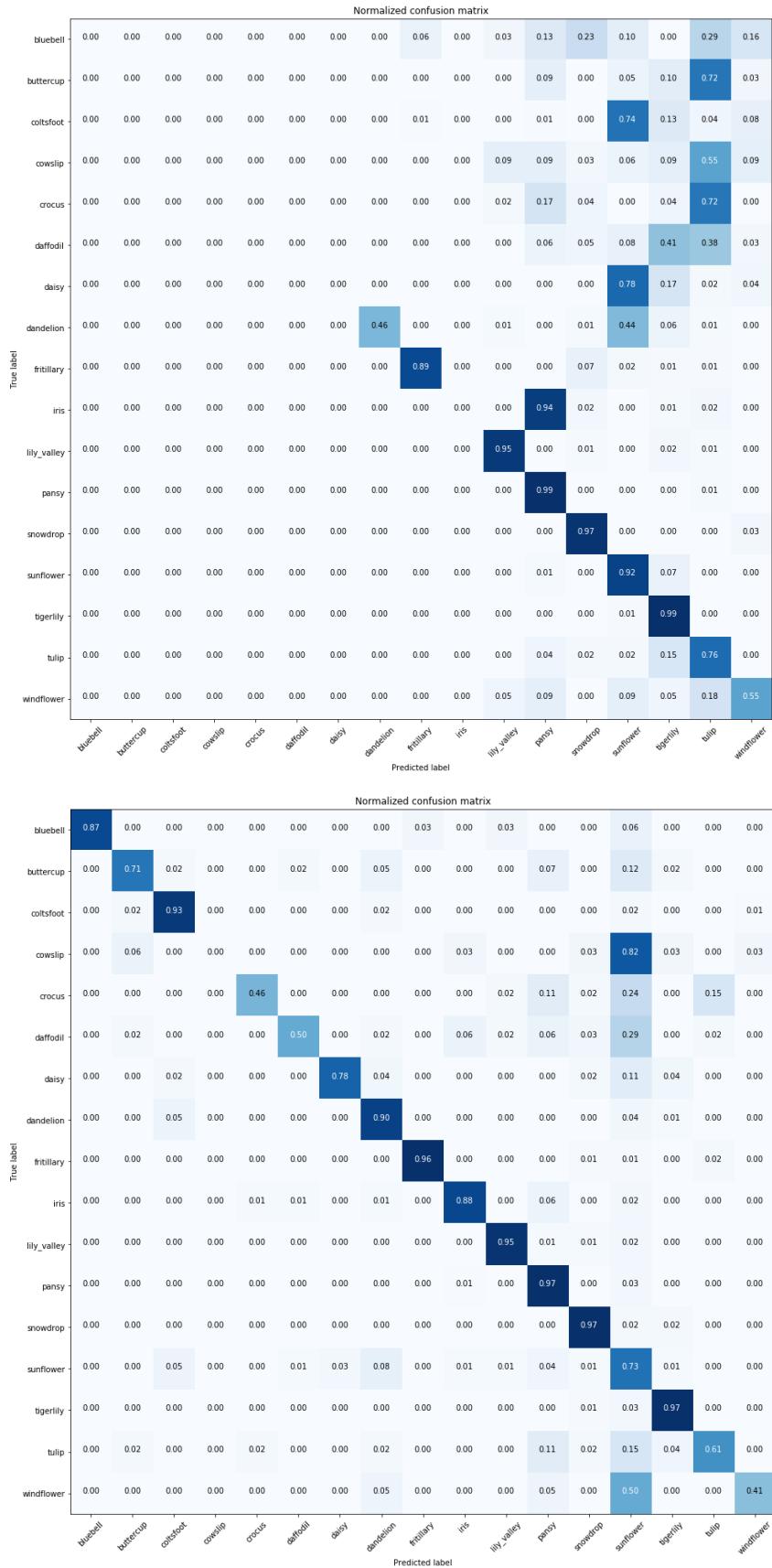


Figura 4.2: Evolução da acurácia nos dados de validação durante o processo de treinamento do experimento de *dropout* 50% (acima) e *dropout* 75%, onde o eixo *x* representa as *epochs*.

- *Cowslip* e *Windflower* classificadas como *Sunflower*.

Outra métrica de performance que mostra a superioridade do modelo com 75% de *dropout* é o F1-score, com um valor médio 0.82 para todas as categorias, quase duas vezes maior que o mesmo resultado com *dropout* 50%. A Tabela 4.2 resume os principais resultados obtidos das métricas de performance para os experimentos com variação de *dropout*, e os resultados completos destes experimentos também podem ser resgatados no Anexo A.

Figura 4.3: Matriz de Confusão dos experimentos de *dropout* 50 e 75%, em ordem.

Métrica	0%	25%	50%	75%
Acurácia de treinamento	0,025	0,025	0,779	0,707
Acurácia de validação	0,025	0,025	0,547	0,816
Média de <i>Precision</i>	0,00	0,00	0,40	0,84
Média de <i>Recall</i>	0,02	0,02	0,52	0,82
Média de F1-score	0,00	0,00	0,42	0,82

Tabela 4.2: Resultados dos experimentos variando *dropout*

## 4.2 Quantidade de Neurônios

Nesta etapa de experimentos, a quantidade de neurônios na camada oculta de classificação é variada entre 256, 512 e 1024. A estrutura da Rede Neural utilizada nestes experimentos é apresentada na Tabela 4.3.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256, 512 e 1024</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 4.3: Estrutura da Rede Neural para os experimentos em quantidade de neurônios.

De forma semelhante aos resultados encontrados nos experimentos com *dropout*, é notável como o aumento da complexidade deste modelo prejudica a convergência. Para ambos os casos onde a quantidade de neurônios foi aumentada (512 e 1024), a mesma situação de acurácia extremamente baixa é observada, em torno de 2,5%. A Matriz de Confusão, na Figura 4.4, também apresenta o mesmo resultado, onde o modelo aparentemente convergiu para uma condição de classificação uniforme para a categoria *Bluebell*.

Por fim, a Tabela 4.4 resume os principais resultados obtidos das para estes

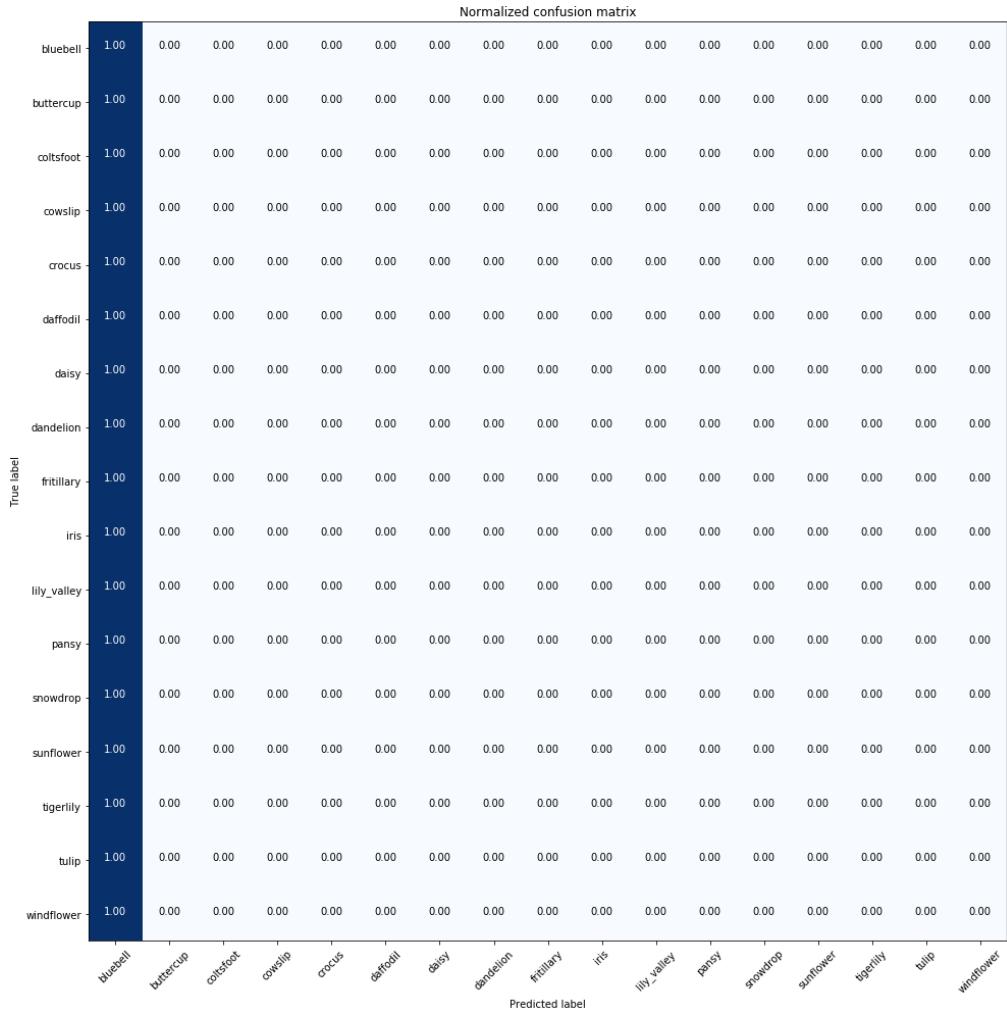


Figura 4.4: Matriz de Confusão dos experimentos de quantidade de neurônios 512 e 1024.

experimentos com variação de quantidade de neurônios, e os resultados completos são apresentados no Anexo A.

Métrica	256	512	1024
Acurácia de treinamento	0,779	0,025	0,025
Acurácia de validação	0,547	0,025	0,025
Média de <i>Precision</i>	0,40	0,00	0,00
Média de <i>Recall</i>	0,52	0,00	0,00
Média de F1-score	0,42	0,00	0,00

Tabela 4.4: Resultados dos experimentos variando em quantidade de neurônios

### 4.3 Quantidade de Camadas Ocultas

Adicionar camadas ocultas de classificação é uma boa forma de adicionar não-linearidade à Rede Neural, apesar de ser bastante difícil predizer se pode ou não beneficiar a performance do modelo. Nestes experimentos, foi utilizada a configuração na Tabela 4.5 para comparar a performance com uma e duas camadas ocultas.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1 e 2</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 4.5: Estrutura da Rede Neural para os experimentos em quantidade de camadas ocultas.

Como mostra a Figura 4.5, a acurácia de validação do experimento com duas camadas ocultas é ligeiramente maior que com apenas uma: 63,1% contra 54,7%. Não surpreendentemente, esta diferença serve como uma comprovação de que arquiteturas mais profundas de Redes Neurais normalmente levam a melhores resultados, por aumentarem a flexibilidade do modelo de se ajustar aos dados de treinamentos.

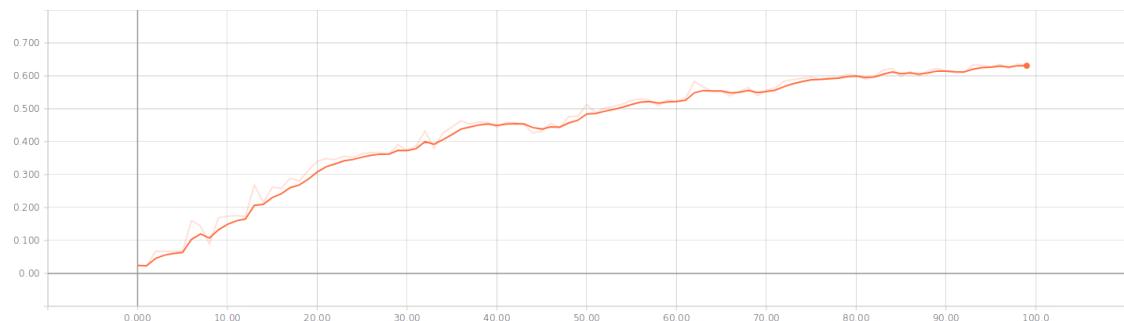


Figura 4.5: Evolução da acurácia nos dados de validação durante o processo de treinamento do experimento com duas camadas ocultas de classificação, onde o eixo *x* representa as *epochs*.

A Figura 4.6 apresenta a Matriz de Confusão do experimento com duas camadas, e é possível notar que ainda existem diversas categorias de flores com baixo índice de acerto: *Bluebell*, *Cowslip*, *Crocus*, *Daisy*, *Dandelion*, *Sunflower* e *Tulip*. A Tabela 4.6 apresenta os principais resultados, e mais detalhes são incluídos no Apêndice A.

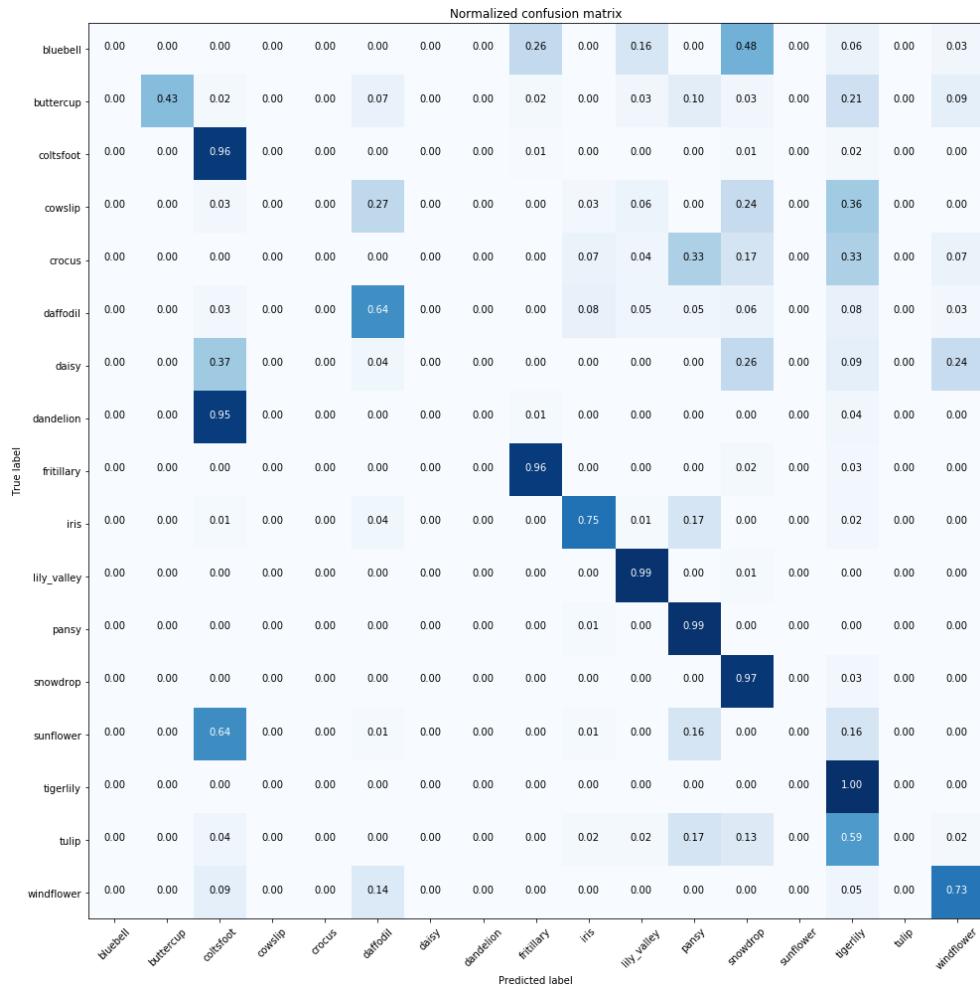


Figura 4.6: Matriz de Confusão do experimento com duas camadas ocultas de classificação.

Métrica	1 camada oculta	2 camadas ocultas
Acurácia de treinamento	0,779	0,671
Acurácia de validação	0,547	0,631
Média de <i>Precision</i>	0,40	0,49
Média de <i>Recall</i>	0,52	0,63
Média de F1-score	0,42	0,53

Tabela 4.6: Resultados dos experimentos variando camadas ocultas de classificação.

## 4.4 Funções de Ativação da Camada de Saída

Para problemas de classificação com múltiplas categorias, a literatura é bastante uniforme sobre a função de ativação Softmax ser uma das melhores para utilizar na camada de saída em uma Rede Neural. Esta etapa de experimentos busca avaliar a superioridade desta função, comparando com outras funções também populares no universo de classificação, mas não necessariamente em classificação *multi-label*. A Tabela 4.7 apresenta as configurações utilizadas nos treinamentos.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax, Tanh e Sigmóide</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 4.7: Estrutura da Rede Neural para os experimentos em funções de ativação da camada de saída.

Como esperado, pelos resultados da Tabela 4.8, é possível observar uma redução drástica na performance do modelo com a substituição da Softmax pela Sigmóide ou Tanh. A acurácia de validação cai mais de 50 pontos percentuais, para 2,5%, e as métricas de *Precision*, *Recall* e F1-score também ressaltam a precariedade do modelo. De forma semelhante ao ocorrido nos experimentos anteriores, é possível que a troca da função de ativação tenha levado o algoritmo de otimização a ficar preso em um mínimo local, proibindo a convergência, já que a Matriz de Confusão destes experimentos também apontam uma condição de classificação uniforme. Resultados detalhados destes experimentos podem ser encontrados no Apêndice A.

Métrica	Softmax	Sigmóide	Tanh
Acurácia de treinamento	0,779	0,025	0,025
Acurácia de validação	0,547	0,025	0,025
Média de <i>Precision</i>	0,40	0,00	0,00
Média de <i>Recall</i>	0,52	0,02	0,02
Média de F1-score	0,42	0,00	0,00

Tabela 4.8: Resultados dos experimentos variando função de ativação da camada de saída.

## 4.5 Funções de Ativação da Camada Oculta

A variação das funções de ativação da camada oculta é um experimento em que realmente se espera alguma melhora da performance da Rede Neural, comparada à sua configuração inicial. Nesta, foi aplicada a função de ativação ReLU, bastante popular na literatura, com diversos benefícios, mas também extremamente simples para ser usada com apenas uma camada oculta (como é o caso da configuração desta Rede Neural), e com alguns problemas clássicos bem conhecidos. Nestes experimentos, são comparadas as funções Sigmóide e Tanh na camada oculta de classificação, como mostra a Tabela 4.9.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU, Sigmóide e Tanh</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Tabela 4.9: Estrutura da Rede Neural para os experimentos em funções de ativação da camada oculta.

Ambas funções Sigmóide e Tanh são bastante parecidas, onde a principal diferença está na simetria ao eixo  $x$ . Na Sigmóide, sua saída varia entre 0 e 1, enquanto

o intervalo de saída da Tanh é simétrico, indo de -1 a 1. Em prática, a Tanh é conceitualmente superior à superior Sigmóide por permitir estes resultados negativos, e possuir um gradiente mais acentuado próximo ao centro. Curiosamente, os resultados neste modelo apresentam o inverso, onde a performance da Rede com a função de ativação Sigmóide é maior quando comparado à configuração inicial, e com a função Tanh é menor. A Figura 4.7 compara a evolução da acurácia nos treinamentos, e os valores ao final alcançam 86,1%, 54,7% e 35,4% para Sigmóide, ReLU e Tanh, respectivamente.

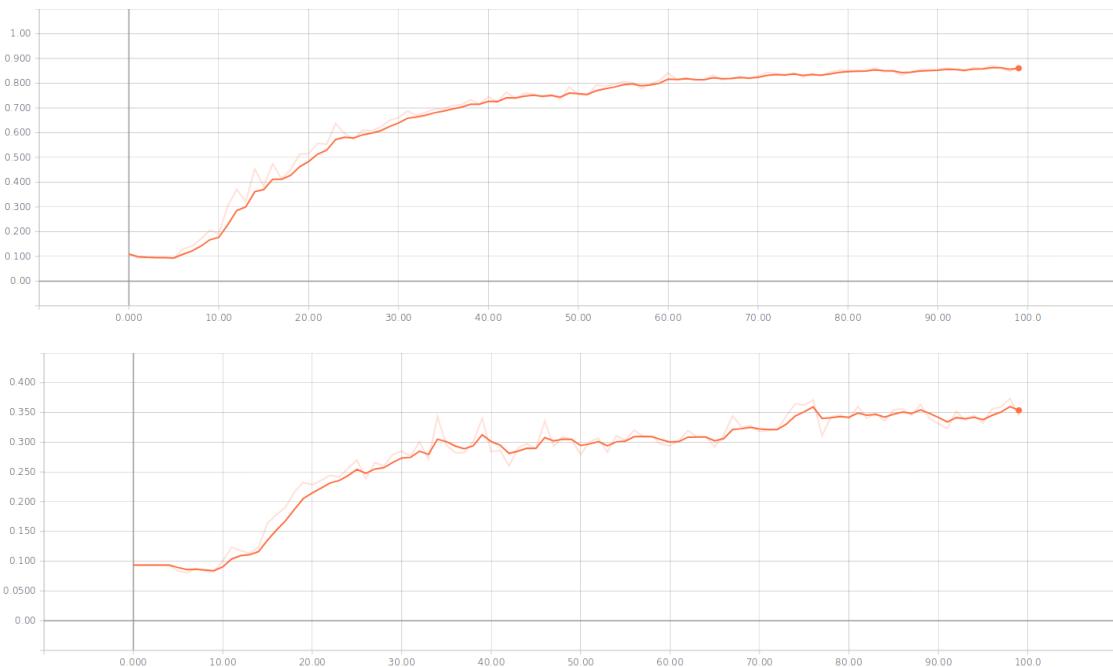


Figura 4.7: Evolução da acurácia nos dados de validação durante o processo de treinamento dos experimentos com funções ativação Sigmóide e Tanh na camada oculta, onde o eixo  $x$  representa as *epochs*.

Uma possível explicação para a superioridade da Sigmóide com relação à Tanh é a força do gradiente. Na segunda, a derivada é consideravelmente mais inclinada, e esta característica pode não ter sido favorável ao problema sendo resolvido. Outra justificativa está na adaptação a valores negativos, onde é possível que a classificação tenha se beneficiado ao tratar somente com números positivos. De qualquer forma, é bastante difícil comprovar estas ou outras justificativas, e foram apresentadas aqui apenas algumas possibilidades, baseadas na intuição do comportamento destas

funções.

Os resultados de F1-score também apresentam a mesma tendência da comparação de performance, onde foram encontrados valores médios de 86%, 42% e 26% para Sigmóide, ReLU e Tanh, respectivamente. As Matrizes de Confusão destes experimentos são apresentadas na Figura 4.8, e é possível perceber que as categorias com maiores índices de erro são semelhantes para as funções ReLU e Tanh: *Buttercup*, *Coltsfoot*, *Cowslip*, *Crocus*, *Daffodil*, *Daisy*, *Dandelion* e *Fritillary*.

No experimento da Sigmóide, apesar da alta acurácia e F1-score, este treinamento ainda obteve índices consideráveis de erro com as categorias listadas abaixo.

- *Bluebell*, bastante confundida com *Fritillary*;
- *Cowslip*, bastante confundida com *Daffodil*;
- *Crocus*, bastante confundida com *Iris*; e
- *Windflower*, bastante confundida com *Pansy*.

Por fim, os principais resultados destes treinamentos estão resumidos na Tabela 4.10, e os resultados completos apresentados no Anexo A. Devido à grande melhora na performance da Rede Neural com a função Sigmóide, optou-se por considerá-la nas configurações iniciais para os experimentos seguintes.

Métrica	ReLU	Sigmóide	Tanh
Acurácia de treinamento	0,779	0,950	0,558
Acurácia de validação	0,547	0,861	0,354
Média de <i>Precision</i>	0,40	0,87	0,26
Média de <i>Recall</i>	0,52	0,87	0,35
Média de F1-score	0,42	0,86	0,26

Tabela 4.10: Resultados dos experimentos variando função de ativação da camada oculta.

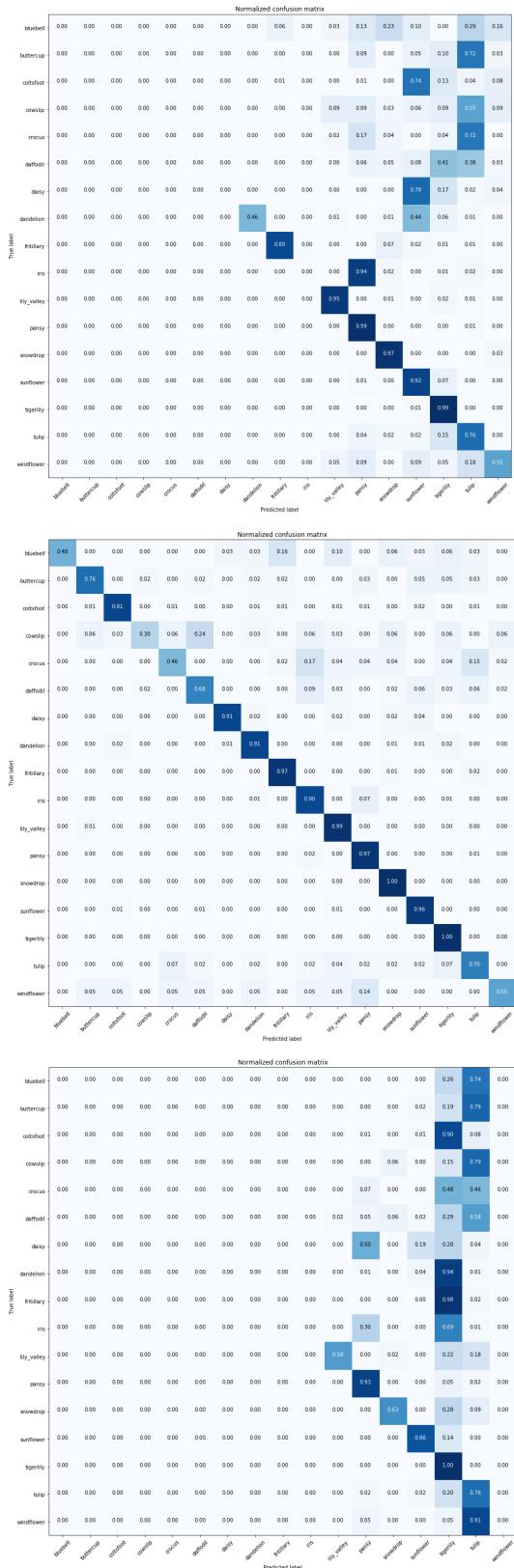


Figura 4.8: Matriz de Confusão dos experimentos de funções de ativação na camada oculta (ReLU, Sigmóide e Tanh, em ordem).

## 4.6 Funções de Custo

Neste experimento, foi comparada a função de custo da configuração inicial, *Cross-entropy*, com a função *Hinge*, como apresentado na Tabela 4.11. Modificar funções de perda significa efetivamente modificar o problema de otimização a ser resolvido no *backpropagation*, logo, os resultados desta comparação podem ser tornar bastante complicados de serem avaliados intuitivamente.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada e <i>Hinge</i></b>

Tabela 4.11: Estrutura da Rede Neural para os experimentos em funções de custo.

Pelos resultados da Tabela 4.12, nota-se um pequeno aumento no parâmetro de acurácia, e uma pequena redução nos parâmetros de *Precision* e *Recall*, mas resultados iguais para o F1-score. Esses valores mostram um equilíbrio entre ambas funções de custo, em que o modelo é capaz de convergir para uma condição de performance bem parecida. Para a seleção de qual função a ser aplicada nas próximas etapas de experimentos, a função Hinge apresentou uma duração de treinamento mais rápida, apresentando certa vantagem sob a Entropia Cruzada neste aspecto. Detalhes dos resultados destes experimentos também são apresentados no Apêndice A.

Métrica	<i>Cross-entropy</i>	<i>Hinge</i>
Acurácia de treinamento	0,950	0,952
Acurácia de validação	0,861	0,867
Média de <i>Precision</i>	0,87	0,87
Média de <i>Recall</i>	0,87	0,86
Média de F1-score	0,86	0,86

Tabela 4.12: Resultados dos experimentos variando função de custo.

## 4.7 Algoritmos de Otimização

De forma semelhante aos experimentos variando funções de custo, a modificação de otimizadores resulta na alteração do problema de otimização dos pesos, e é uma comparação com pouca intuição. Nestes experimentos, são utilizados os algoritmos de RMSprop, Adam e SGD, com a configuração da Tabela 4.13.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop, Adam e SGD</b>
Função de custo	<b>Hinge</b>

Tabela 4.13: Estrutura da Rede Neural para os experimentos em algoritmos de otimização.

Dentre os resultados obtidos, aquele que mais chama a atenção é a baixa performance do modelo otimizado com Adam, onde a acurácia de validação ficou inferior a 4%. Para este resultado, a Matriz de Confusão na Figura 4.9 demonstra que o modelo convergiu para uma condição de classificação uniforme, onde todas as imagens foram categorizada como *Tulip*. Esse resultado também é refletido nos valores de *Precision*, *Recall* e F1-score, como é mostrado na Tabela 4.14

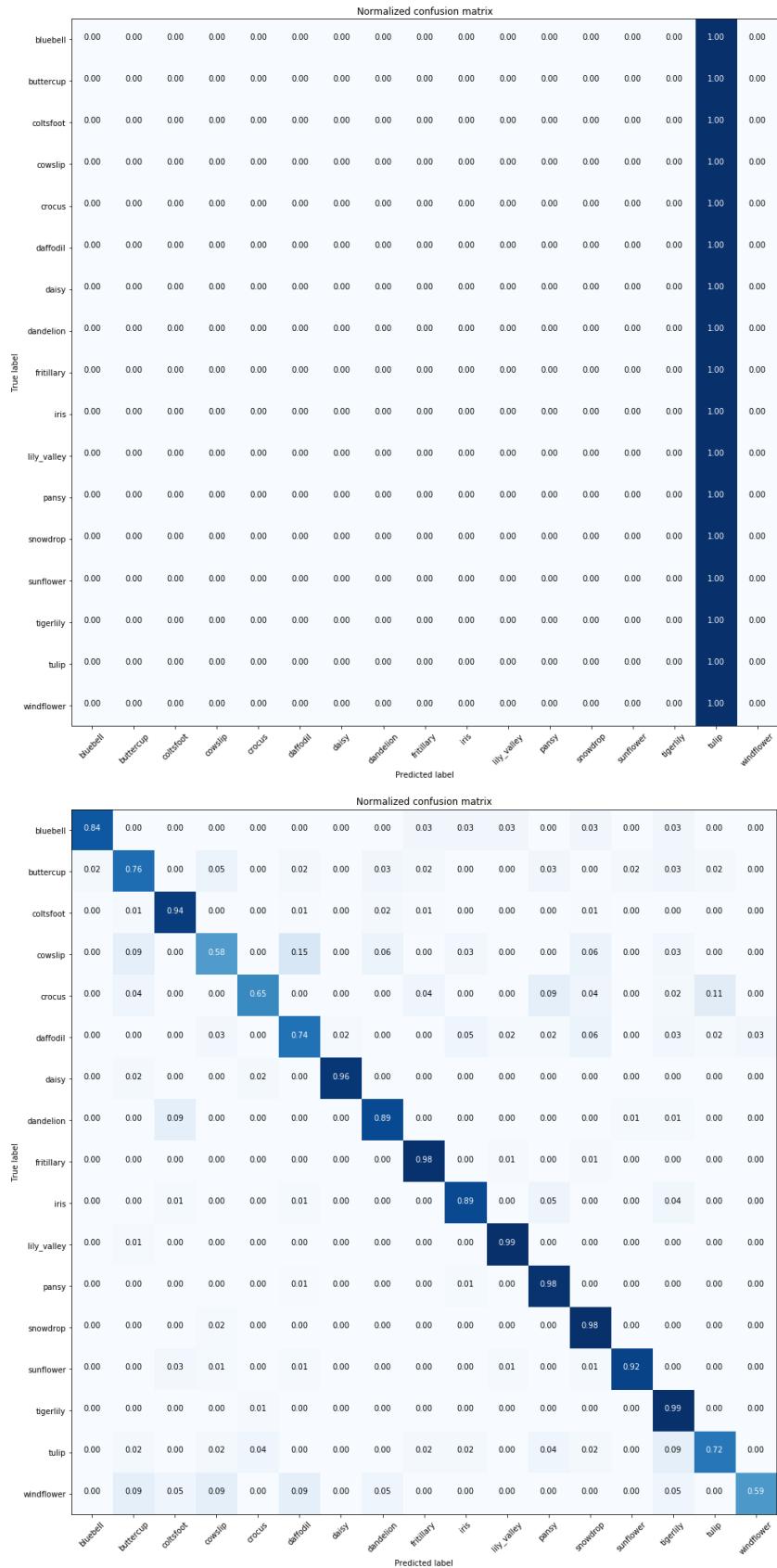


Figura 4.9: Matriz de Confusão do experimento com otimizador Adam (acima) e SGD (abaixo).

O algoritmo de otimização de Gradiente Descendente Estocástico, por outro lado, apresentou um aumento na performance de classificação, com os melhores resultados de acurácia, *Precision*, *Recall* e F1-score dos experimentos realizados até esta etapa: 89,1%, 90%, 90% e 89%, respectivamente. A Matriz de Confusão deste experimento, apresentada na Figura 4.9, mostra que as categorias de flores com maior índice de acerto são *Coltsfoot*, *Daisy*, *Fritillary*, *Lily Valley*, *Pansy*, *Snowdrop*, *Sunflower* e *Tigerlily*. Os maiores índices de erro são das classes *Cowslip*, *Crocus* e *Windflower*.

A Tabela 4.14 mostra os principais resultados desta etapa de experimentos, e mais detalhes podem ser encontrado no Apêndice A.

Métrica	RMSprop	Adam	SGD
Acurácia de treinamento	0,952	0,036	0,996
Acurácia de validação	0,867	0,036	0,891
Média de <i>Precision</i>	0,87	0,00	0,90
Média de <i>Recall</i>	0,86	0,04	0,90
Média de F1-score	0,86	0,00	0,89

Tabela 4.14: Resultados dos experimentos variando otimizadores.

## 4.8 Modelos de *Feature Extraction*

Estes experimentos buscam utilizar modelos diferentes de Redes Neurais, bem colocadas em competições do ImageNet, para a estratégia de Transferência de Conhecimento que aplica as camadas de extração de recurso com pesos pré-treinados. A configuração-base da Rede Neural para estes experimentos corresponde àquela com melhor performance na última etapa de treinamentos (Tabela 4.15), e os modelos das camadas de feature extraction é o único parâmetro modificado entre os treinamentos. Intuitivamente, alguns modelos já se comprovaram superiores a outros durante as competições do ImageNet, no entanto, estes experimentos não comparam os modelos com suas melhores configurações possíveis. Desta forma, não é esperado que os resultados desta etapa sigam a mesma tendência das performances nos ILSVRC's.

Modelo e pesos das camadas de extração de recursos	<b>VGG16, VGG19, InceptionV3, ResNet50, NAS-Net</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>SGD</b>
Função de custo	<b>Hinge</b>

Tabela 4.15: Estrutura da Rede Neural para os experimentos em modelos das camadas de extração de recursos.

Primeiramente, os resultados de ambas versões de 16 e 19 camadas do modelo do Oxford Visual Geometry Group (VGG16 e VGG19) apresentaram performances muito semelhantes, com acurárias de 89,1% e 88,3%, e F1-scores de 89% e 88%. A arquitetura destes dois modelos segue a mesma lógica de construção de camadas, onde o VGG19 apresenta 3 camadas adicionais de convolução.

Como mencionado anteriormente, quanto mais profunda uma CNN na sua parte de extração de recursos, mais características e objetos maiores e mais complexos são reconhecidos, como na Figura 2.1. Como foram aplicados os pesos pré-treinados destes modelos, é possível que o modelo VGG19 tenha se adaptado melhor às figuras da ILSVRC, e não necessariamente tão bem às imagens de flores, o que justificaria seu desempenho levemente inferior quando comparado ao modelo VGG16. De qualquer forma, os resultados ainda foram bastante parecidos, e também podem ser observados nas Matrizes de Confusão da Figura 4.10.

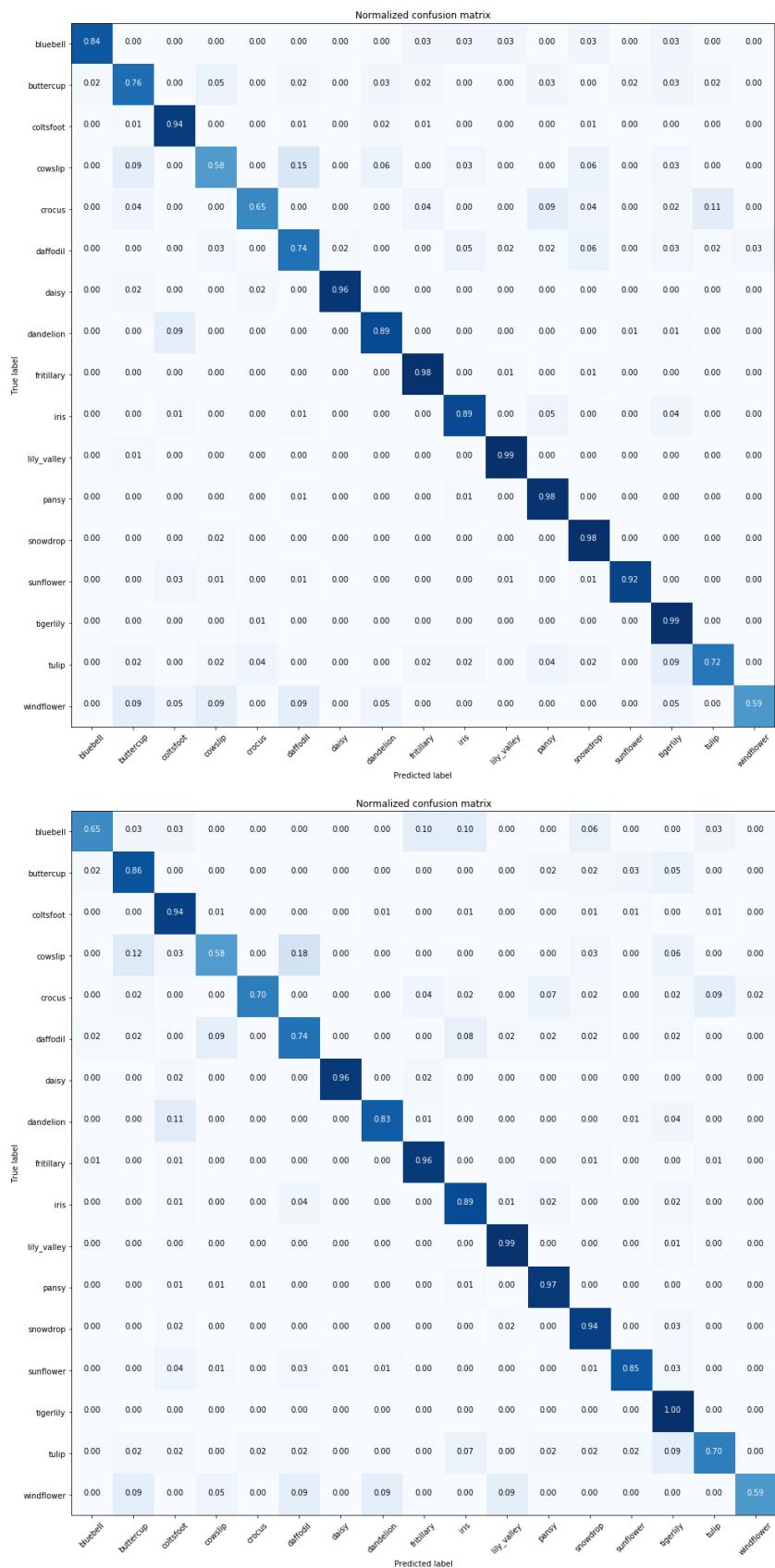


Figura 4.10: Matriz de Confusão dos experimentos com modelos VGG16 (acima) e VGG19 (abaixo).

Em seguida, é avaliado o modelo do InceptionV3, evolução do tradicional GoogLeNet, e que, a princípio, possui performance superior ao VGG. No entanto, para a configuração-base dos parâmetros da Tabela 4.15, o InceptionV3 apresentou performance bem inferior, com acurácia de 38,5% e F1-score de 19%. É importante enfatizar que este resultado não contradiz o encontrado no ILSVRC, e indica apenas que a configuração base favorece a arquitetura do VGG. A Matriz de Confusão, na Figura 4.11 mostra uma preferência de classificação para o tipo de flor *Tigerlily*.

		Normalized confusion matrix																
		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
bluebell	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.10	0.13	0.19	0.10	0.00	0.00	0.39	0.00	0.00	
buttercup	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.05	0.12	0.17	0.00	0.02	0.62	0.00	0.00	
coltsfoot	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.13	0.09	0.02	0.00	0.03	0.00	0.00	0.66	0.00	0.00	
cowslip	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.12	0.09	0.00	0.00	0.00	0.70	0.00	0.00	
crocus	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.09	0.02	0.20	0.00	0.00	0.67	0.00	0.00	
daffodil	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.02	0.06	0.02	0.26	0.00	0.00	0.62	0.00	0.00	
daisy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.02	0.04	0.06	0.17	0.04	0.00	0.63	0.00	0.00	
dandelion	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.02	0.01	0.00	0.10	0.00	0.00	0.38	0.00	0.00	
fritillary	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.20	0.00	0.04	0.07	0.01	0.00	0.65	0.00	0.00	
iris	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.08	0.06	0.13	0.01	0.01	0.69	0.00	0.00	
lily_valley	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.04	0.54	0.08	0.00	0.00	0.00	0.33	0.00	0.00	
pansy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.03	0.00	0.46	0.00	0.01	0.48	0.00	0.00	
snowdrop	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.05	0.02	0.31	0.15	0.08	0.00	0.37	0.00	0.00	
sunflower	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.04	0.86	0.00	0.00	
tigerlily	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.03	0.00	0.00	0.95	0.00	0.00	
tulip	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.09	0.22	0.02	0.00	0.65	0.00	0.00	
windflower	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.27	0.14	0.00	0.00	0.00	0.55	0.00	0.00	

Figura 4.11: Matriz de Confusão do experimento com modelo InceptionV3.

Uma justificativa para este baixo desempenho pode ser uma insuficiência de *epochs* de treinamento. A Figura 4.12 apresenta a função de custo durante o processo de treinamento deste experimento, e dado que o gráfico apresenta uma tendência de redução ainda no final das 100 épocas, é possível inferir que uma maior quantidade de *epochs* poderia ser benéfico ao desempenho final. Outras justificativas plausíveis

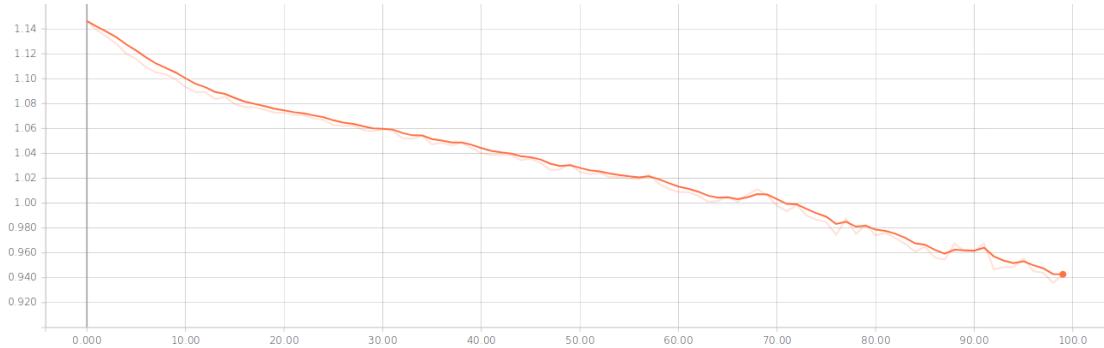


Figura 4.12: Evolução do custo calculado no treinamento do experimento com modelo InceptionV3.

também são: uma boa adaptação das camadas de *feature extraction* às figuras da ImageNet que não se repetem para as fotos de flores deste projeto; e uma baixa performance devido aos hiperparâmetros selecionados na configuração-base. No entanto, como este estudo se limitou a avaliar os experimentos de uma forma mais simples, não será aqui comprovado a melhor estratégia para se aplicar o modelo InceptionV3.

O quarto modelo avaliado foi o ResNet50, arquitetura campeã na ILSVRC de 2015, e desenvolvida pela Microsoft. ResNet apresenta uma arquitetura bastante exótica, e os resultados deste modelo, com a configuração-base da Tabela 4.15, foram os melhores obtidos neste trabalho: acurácia de validação de 95% e F1-score também de 95%. A Figura 4.13 apresenta a Matriz de Confusão deste experimento, onde é possível observar índices de acerto superiores a 95% para mais da metade das categorias. O tipo de flor com maior erro de classificação, e a única com índice inferior a 80%, foi a *Crocus*, onde 78% das imagens foram classificadas corretamente, mas 11% foram classificadas incorretamente como *Tulip*, e 7% como *Pansy*.

Assim, apesar de ter sido o treinamento mais demorado dentre os modelos comparados, a ResNet50 (com a configuração da Tabela 4.15) apresentou a melhor performance, neste estudo.

Por fim, também foi avaliado o desempenho do modelo NASNet, declarado pela Google como uma arquitetura de performance superior a todos os demais modelos

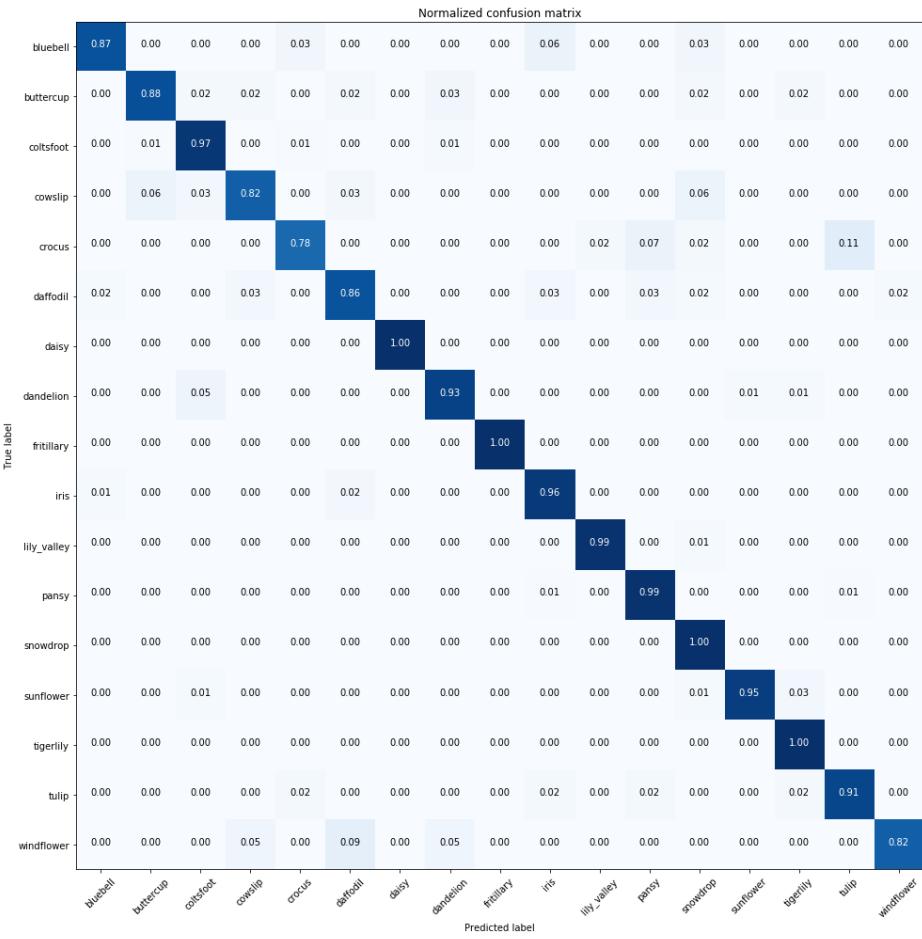


Figura 4.13: Matriz de Confusão do experimento com modelo ResNet50.

considerados neste estudo. No entanto, a configuração-base utilizada com este modelo não foi favorável no treinamento, e este experimento apresentou acurácia de somente de 19,5% e F1-score de 15%. Durante o treinamento, a mesma situação encontrada no experimento com InceptionV3 também foi observada aqui, onde a tendência da função de custo próximo a última *epoch* de treinamento indica que um treinamento mais longo poderia ser benéfico (Figura 4.14). Ainda assim, no entanto, a grande diferença entre a acurácia dos dados de treinamento (66,2%) e dos dados de validação (19,5%) pode sugerir uma situação de *overfitting*, a qual deve ser corrigida por outras formas, como aumento do *dropout*, por exemplo.

A Matriz de Confusão do experimento do NASNet é apresentada na Figura 4.15, onde também percebe-se que o experimento resultou em um classificador com preferência para o tipo de flor *Tigerlily*.



Figura 4.14: Evolução do custo calculado no treinamento do experimento com modelo NASNet.

Normalized confusion matrix																	
True label	bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.10	0.00	0.00	0.00	0.74	0.00	0.00	
	0.00	0.12	0.00	0.00	0.00	0.02	0.09	0.00	0.00	0.09	0.05	0.03	0.07	0.52	0.00	0.02	
	0.00	0.00	0.02	0.00	0.00	0.00	0.06	0.01	0.00	0.01	0.01	0.02	0.11	0.76	0.00	0.00	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.06	0.98	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.04	0.04	0.15	0.72	0.00	0.00	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.05	0.05	0.14	0.73	0.00	0.03	
	0.00	0.04	0.00	0.00	0.00	0.00	0.19	0.02	0.00	0.04	0.02	0.02	0.07	0.11	0.48	0.00	0.02
	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.32	0.01	0.00	0.04	0.02	0.04	0.01	0.52	0.00	0.02
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.01	0.00	0.37	0.58	0.00	0.00
	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.02	0.02	0.01	0.07	0.08	0.14	0.60	0.00	0.00
	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.04	0.04	0.14	0.52	0.00	0.04	
	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.03	0.00	0.00	0.09	0.01	0.39	0.46	0.00	0.00
	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.00	0.03	0.02	0.06	0.00	0.34	0.11	0.40	0.00	0.02
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.55	0.40	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	0.08	0.89	0.00	0.00
	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.02	0.04	0.09	0.00	0.28	0.46	0.00	0.04
	0.00	0.00	0.05	0.00	0.00	0.00	0.05	0.00	0.00	0.09	0.05	0.09	0.05	0.05	0.64	0.00	0.00

Figura 4.15: Matriz de Confusão do experimento com modelo NASNet.

Finalmente, um resumo dos resultados desta etapa de experimentos é listado na Tabela 4.16, mas os resultados detalhados também estão apresentados no Anexo A.

Métrica	VGG16	VGG19	InceptionV3	ResNet50	NASNet
Acurácia (treino)	0,996	0,993	0,385	0,9997	0,662
Acurácia (val.)	0,891	0,883	0,242	0,950	0,195
Média de <i>Precision</i>	0,90	0,89	0,34	0,95	0,29
Média de <i>Recall</i>	0,90	0,88	0,25	0,95	0,20
Média de F1-score	0,89	0,88	0,19	0,95	0,15

Tabela 4.16: Resultados dos experimentos variando modelos das camadas de extração de recursos.

# Capítulo 5

## Conclusões

O presente trabalho teve como objetivo explorar o universo de Redes Neurais Artificiais e *deep learning* através da aplicação de Redes Neurais Convolucionais para o problema de classificação de imagens, mais especificamente, fotografias de flores e seus tipos. Desde 2012, Redes Neurais Convolucionais começaram a se destacar para solução de problemas deste tipo, e hoje em dia já se consolidaram como a arquitetura preferida dos modelos de alta performance neste contexto. Assim, a exploração deste tópico é um objetivo bastante alinhado com os interesses da área científica, e contribuindo não somente para o aprendizado dos autores, mas também para a divulgação dos conhecimentos.

O estudo se iniciou com uma busca e compilação da literatura relevante no assunto no Capítulo 2, onde foi possível construir um histórico do desenvolvimento de Redes Neurais, remontando à década de 50, com a criação do modelo Perceptron, passando por duas épocas de expansão da pesquisa, na década de 80 e no início do milênio, até a época atual, de desenvolvimento contínuo e amadurecimento de novas técnicas e modelos. Também foram elencados e descritos fundamentos técnicos importantes na área, como Funções de Ativação e *Backpropagation*, além de um detalhamento mais profundo da arquitetura e elementos de Redes Neurais Convolucionais.

A exploração prática do problema foi iniciada com a aquisição de dados, onde optou-se por um *dataset* coletado por outros autores através da API de uma rede social de fotografias chamada Flickr. 5111 imagens, distribuídas de forma desbalanceada em 17 categorias de flores, foram então pré-tratados através da conversão para um único formato *jpeg*, e redimensionadas para um tamanho padrão de 256x256 *pixels*, antes de serem organizadas e carregadas para um *storage* no Google Cloud Platform (GCP). Para a construção e treinamento de modelos, utilizou-se uma técnica de Transferência de Conhecimento, onde as camadas de extração de recursos de outras Redes Neurais Convolucionais de alta performance pré-treinadas são utilizadas. Para isso, todas as imagens são através das camadas dos modelos, e o resultado da última camada é persistido em um arquivo *tfrecord* no GCP.

Foi desenvolvido um algoritmo generalizado para a implementação flexível de treinamentos no GCP, requerendo apenas um comando simples com parâmetros de entrada que detalham o formato do experimento desejado. Neste estudo, foram avaliadas 4 frações de *dropout* entre camadas de classificação, 3 tamanhos de camada oculta, 2 números de camadas ocultas, 3 funções de ativação da camada de saída, 3 funções de ativação da camada oculta, 2 funções de custo, 3 algoritmos de otimização e 5 modelos pré-treinados para *feature extraction*. Seguindo uma estratégia sequencial de treinamento, foram 18 experimentos realizados para avaliação da influência de cada um destes parâmetros na performance do modelo. Todos os algoritmos desenvolvidos e resultados obtidos neste trabalho estão disponibilizado em repositório público no GitHub [5].

Dos resultados obtidos, as principais conclusões são listadas a seguir, levando em consideração que estas se aplicam somente aos seus respectivos experimentos e configurações:

- Valores de *dropout* inferiores a 50% apresentaram baixa performance, possivelmente por divergência do algoritmo de otimização com cálculo de muitos parâmetros;

- Quantidade de neurônios superior a 256 na camada oculta de classificação apresentaram baixa performance, possivelmente por divergência do algoritmo de otimização com cálculo de muitos parâmetros;
- Experimento com duas camadas ocultas de classificação apresentou melhor performance compara a apenas uma camada oculta;
- A alteração da função de ativação da camada de saída de Softmax para Sigmóide ou Tanh resulta em uma redução drástica na performance do modelo;
- Quanto à função de ativação na camada oculta de classificação, a Sigmóide apresentou melhores resultados se comparada a ReLU, que, por sua vez, também teve obteve performance superior à função Tanh;
- Variando os otimizadores, o algoritmo de Gradiente Descendente apresentou melhor performance em geral quando comparado ao RMSprop, e o algoritmo Adam levou a resultados drasticamente inferiores;
- Na comparação de diferentes modelos de *feature extraction*, o modelo ResNet50 se mostrou superior a todos os demais, e inclusive se caracterizou como o experimentou com a maior acurácia e F1-score deste estudo.

## 5.1 Sugestão de Próximos Passos

Durante a execução do estudo, foram identificados diversos pontos que poderiam se beneficiar com maiores desenvolvimentos, possivelmente em estudos futuros. Estes pontos podem ser resumidos em quatro grandes tópicos:

1. Re-treinamento de parte das camadas de *feature extraction*: Em alguns casos, é possível que o congelamento das camadas de extração de recursos de um modelo pré-treinado com as imagens do ImageNet possa ter sido prejudicial à performance dos experimentos. Uma sugestão é a inclusão dos pesos

de algumas destas camadas no processo de treinamento, preferencialmente as camadas do final da etapa de *feature extraction*, que tratam de características de maior escala. Esta aumenta significativamente a quantidade de parâmetros nos cálculos para o treinamento, aumentando a complexidade do modelo e o poder/tempo computacional necessário.

2. *Data Augmentation*: Tratando-se de um dataset pequeno, com apenas 5111 imagens, uma forma de aumentar a performance do modelo de classificação é realizar pequenas variações aleatórias nos dados e utilizá-los como novas instâncias de treinamento. Por exemplo, rotacionar, transladar, cortar e aplicar filtros de cores nas fotografias são técnicas que podem aumentar a variabilidade dos dados à medida que aumenta o tamanho do *dataset*.
3. *Fine-tuning* de outros parâmetros: Outros parâmetros importantes no treinamento de uma Rede Neural não foram variados neste estudo, tal como número de *epochs* de treinamento e a taxa de aprendizado dos algoritmos de otimização. Estes hiperparâmetros podem causar mudanças significativas na performance de um modelo, e é bastante interessante considerá-los em um projeto de *Machine Learning*.
4. *Fine-tuning* mais completo: Os experimentos neste estudo foram realizados em sequência, de forma simplificada, onde apenas um parâmetro era variado por vez, sem considerar a influência de demais parâmetros no desempenho do modelo. Em algumas situações, isso foi prejudicial ao estudo, como nos casos em que os resultados possivelmente ficaram presos em mínimos locais durante a otimização, e a real influência do parâmetro variado não pôde ser cuidadosamente avaliada. É sugerida a aplicação de técnicas mais detalhadas, como um *Grid Search*, para avaliar de forma mais completa as vantagens e desvantagens de cada parâmetro.

# Apêndice A

## Resultados

Neste Apêndice, são expostos todos os resultados nos experimentos realizados.

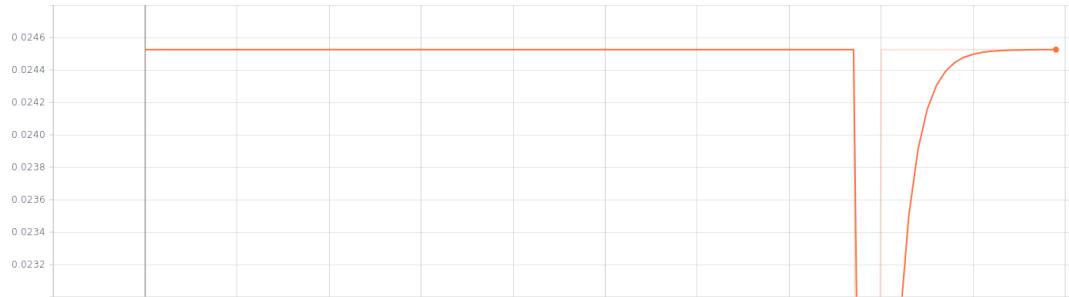
Os formatos de apresentação são:

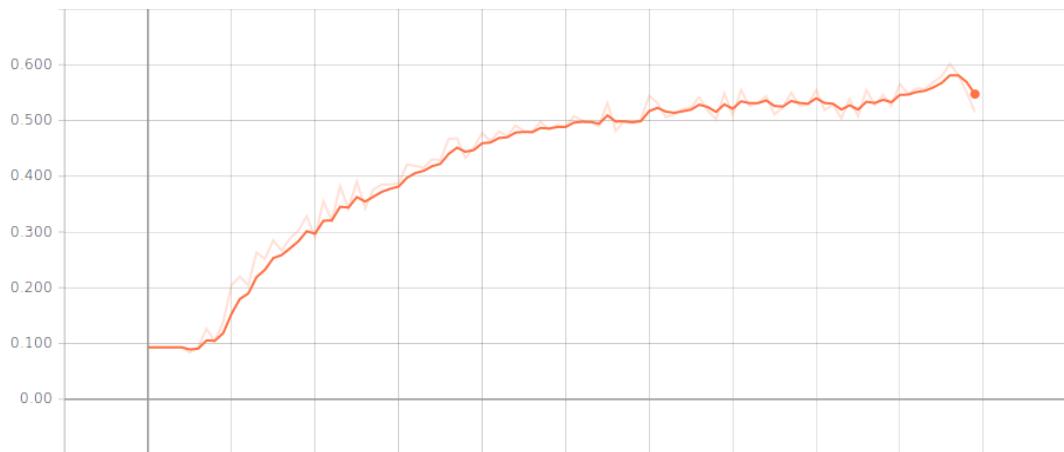
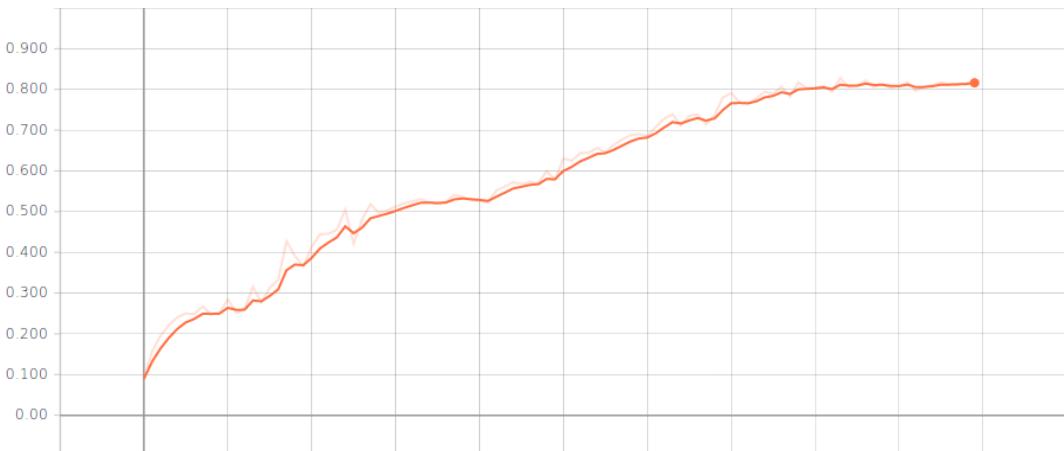
- **Acurácia:** Gráficos de evolução da acurácia calculada para os dados de validação durante o processo de treinamento (acurácia *vs epochs*);
- **Matriz de Confusão:** Imagens das matrizes de confusão;
- **Precision:** Tabela com valores de *precision* para cada categoria de flor, em cada experimento;
- **Recall:** Tabela com valores de *recall* para cada categoria de flor, em cada experimento;
- **F1-score:** Tabela com valores de F1-score para cada categoria de flor, em cada experimento;

### A.1 *Dropout*

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>0, 25, 50 e 75%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

Figura A.1: Acurácia do *Dropout* 0%: 2,5%Figura A.2: Acurácia do *Dropout* 25%: 2,5%

Figura A.3: Acurácia do *Dropout* 50%: 54,7%Figura A.4: Acurácia do *Dropout* 75%: 81,6%

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.5: Matriz de Confusão do Dropout 0%

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.6: Matriz de Confusão do Dropout 25%

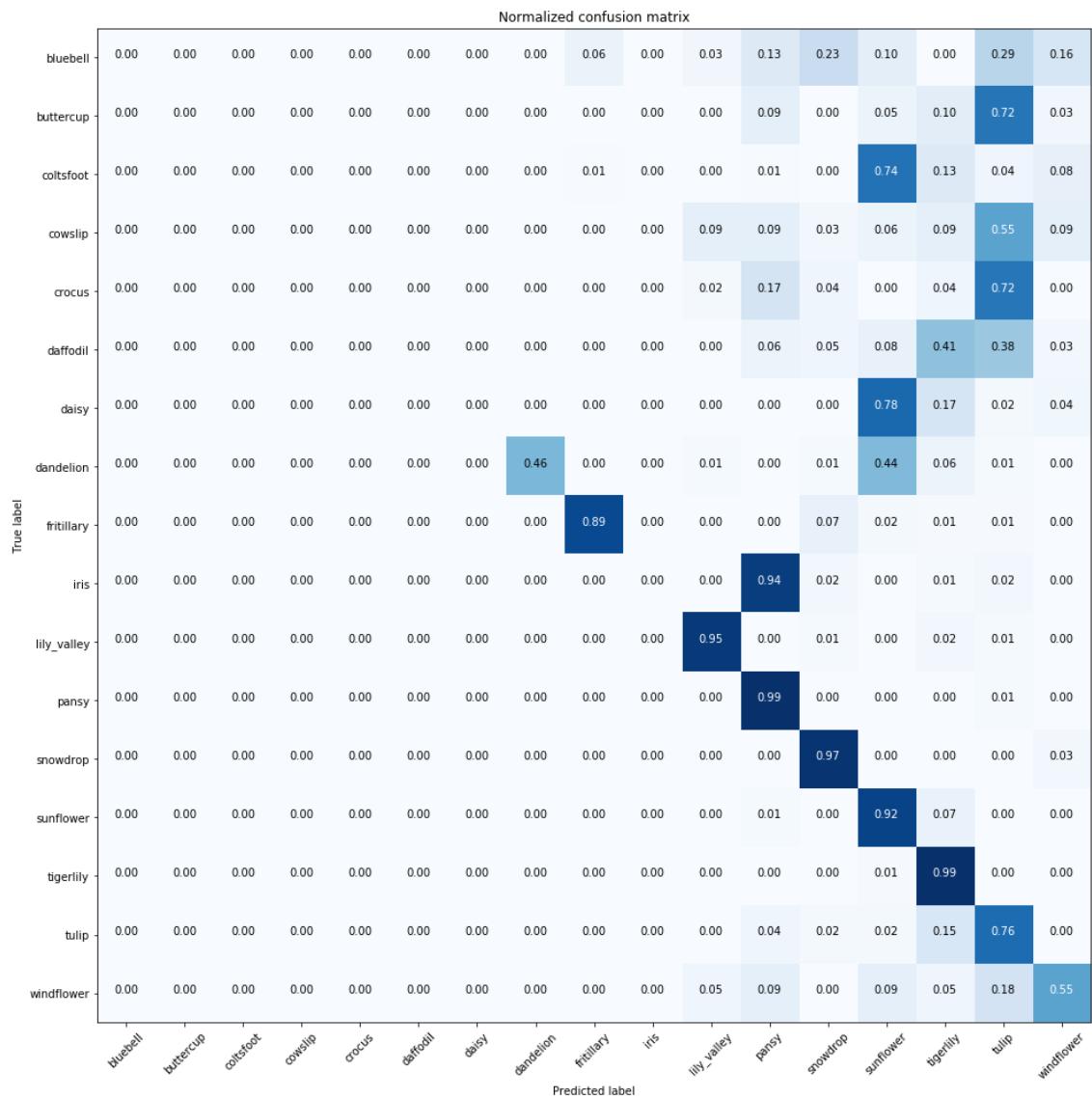


Figura A.7: Matriz de Confusão do Dropout 50%

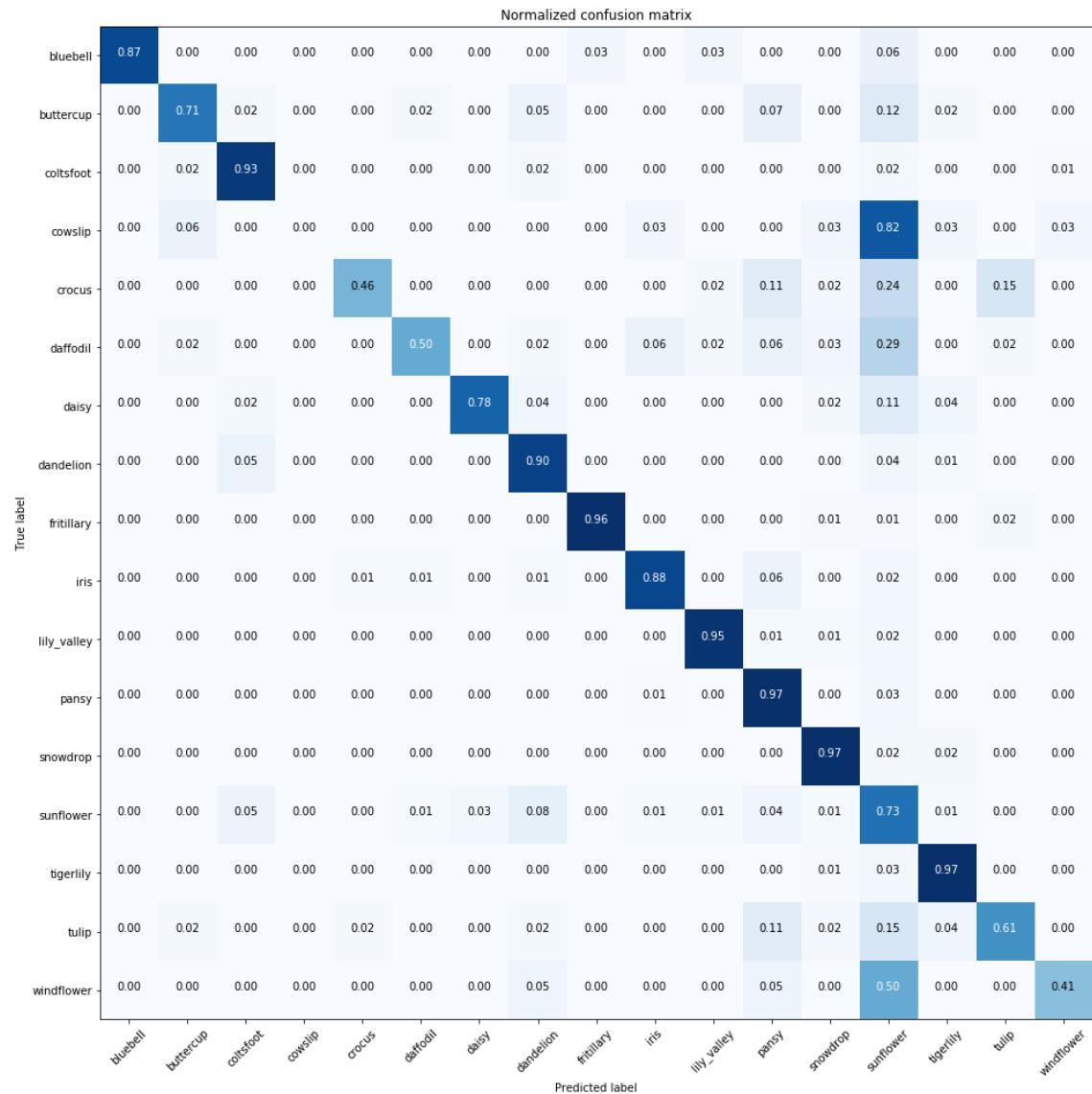


Figura A.8: Matriz de Confusão do Dropout 75%

Categoría	<i>Dropout 0%</i>	<i>Dropout 25%</i>	<i>Dropout 50%</i>	<i>Dropout 75%</i>
<i>Bluebell</i>	0.02	0.02	0.00	1.00
<i>Buttercup</i>	0.00	0.00	0.00	0.85
<i>Coltsfoot</i>	0.00	0.00	0.00	0.93
<i>Cowslip</i>	0.00	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00	0.91
<i>Daffodil</i>	0.00	0.00	0.00	0.92
<i>Daisy</i>	0.00	0.00	0.00	0.95
<i>Dandelion</i>	0.00	0.00	1.00	0.80
<i>Fritillary</i>	0.00	0.00	0.97	0.99
<i>Iris</i>	0.00	0.00	0.00	0.91
<i>Lily Valley</i>	0.00	0.00	0.92	0.95
<i>Pansy</i>	0.00	0.00	0.58	0.84
<i>Snowdrop</i>	0.00	0.00	0.71	0.86
<i>Sunflower</i>	0.00	0.00	0.25	0.33
<i>Tigerlily</i>	0.00	0.00	0.57	0.93
<i>Tulip</i>	0.00	0.00	0.19	0.74
<i>Windflower</i>	0.00	0.00	0.31	0.82
Média	0.00	0.00	0.40	0.84

Tabela A.1: Resultados de *Precision*

Categoría	<i>Dropout 0%</i>	<i>Dropout 25%</i>	<i>Dropout 50%</i>	<i>Dropout 75%</i>
<i>Bluebell</i>	1.00	1.00	0.00	0.87
<i>Buttercup</i>	0.00	0.00	0.00	0.71
<i>Coltsfoot</i>	0.00	0.00	0.00	0.93
<i>Cowslip</i>	0.00	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00	0.46
<i>Daffodil</i>	0.00	0.00	0.00	0.50
<i>Daisy</i>	0.00	0.00	0.00	0.78
<i>Dandelion</i>	0.00	0.00	0.46	0.90
<i>Fritillary</i>	0.00	0.00	0.89	0.96
<i>Iris</i>	0.00	0.00	0.00	0.88
<i>Lily Valley</i>	0.00	0.00	0.95	0.95
<i>Pansy</i>	0.00	0.00	0.99	0.97
<i>Snowdrop</i>	0.00	0.00	0.97	0.97
<i>Sunflower</i>	0.00	0.00	0.92	0.73
<i>Tigerlily</i>	0.00	0.00	0.99	0.97
<i>Tulip</i>	0.00	0.00	0.76	0.61
<i>Windflower</i>	0.00	0.00	0.55	0.41
Média	0.02	0.02	0.52	0.82

Tabela A.2: Resultados de *Recall*

Categoria	<i>Dropout 0%</i>	<i>Dropout 25%</i>	<i>Dropout 50%</i>	<i>Dropout 75%</i>
<i>Bluebell</i>	0.05	0.05	0.00	0.93
<i>Buttercup</i>	0.00	0.00	0.00	0.77
<i>Coltsfoot</i>	0.00	0.00	0.00	0.93
<i>Cowslip</i>	0.00	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00	0.61
<i>Daffodil</i>	0.00	0.00	0.00	0.65
<i>Daisy</i>	0.00	0.00	0.00	0.86
<i>Dandelion</i>	0.00	0.00	0.63	0.85
<i>Fritillary</i>	0.00	0.00	0.93	0.98
<i>Iris</i>	0.00	0.00	0.00	0.90
<i>Lily Valley</i>	0.00	0.00	0.94	0.95
<i>Pansy</i>	0.00	0.00	0.73	0.90
<i>Snowdrop</i>	0.00	0.00	0.82	0.91
<i>Sunflower</i>	0.00	0.00	0.39	0.45
<i>Tigerlily</i>	0.00	0.00	0.73	0.95
<i>Tulip</i>	0.00	0.00	0.31	0.67
<i>Windflower</i>	0.00	0.00	0.39	0.55
Média	0.00	0.00	0.42	0.82

Tabela A.3: Resultados de *F1-score*

## A.2 Quantidade de Neurônios

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256, 512 e 1024</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

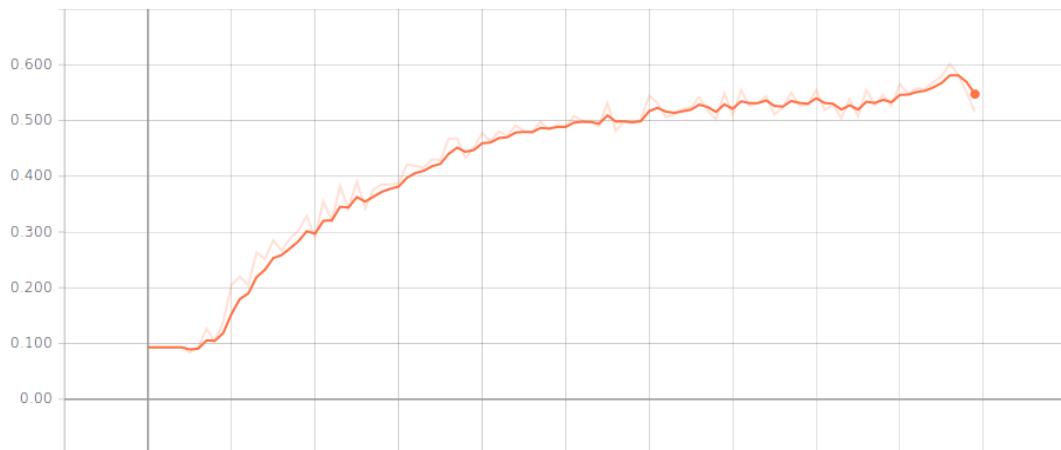


Figura A.9: Acurácia com 256 neurônios: 54,7%.

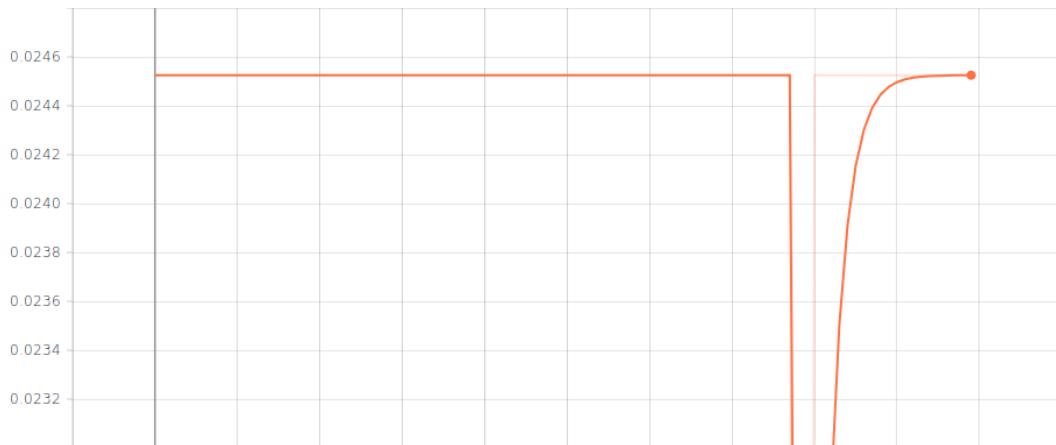


Figura A.10: Acurácia com 512 neurônios: 2,5%.

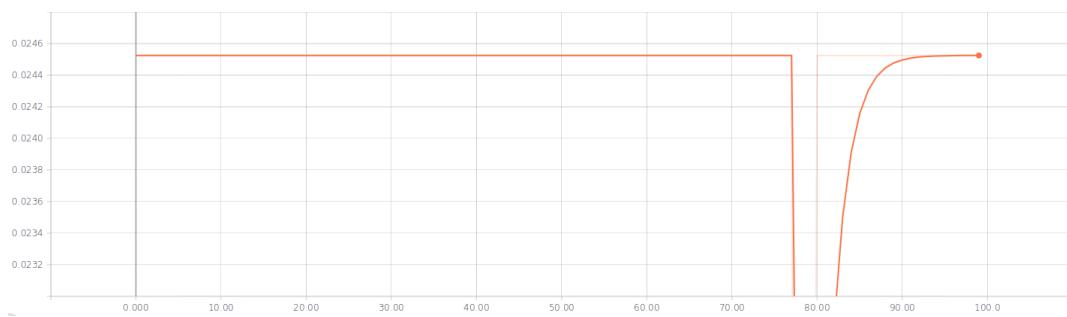


Figura A.11: Acurácia com 1024 neurônios: 2,5%.

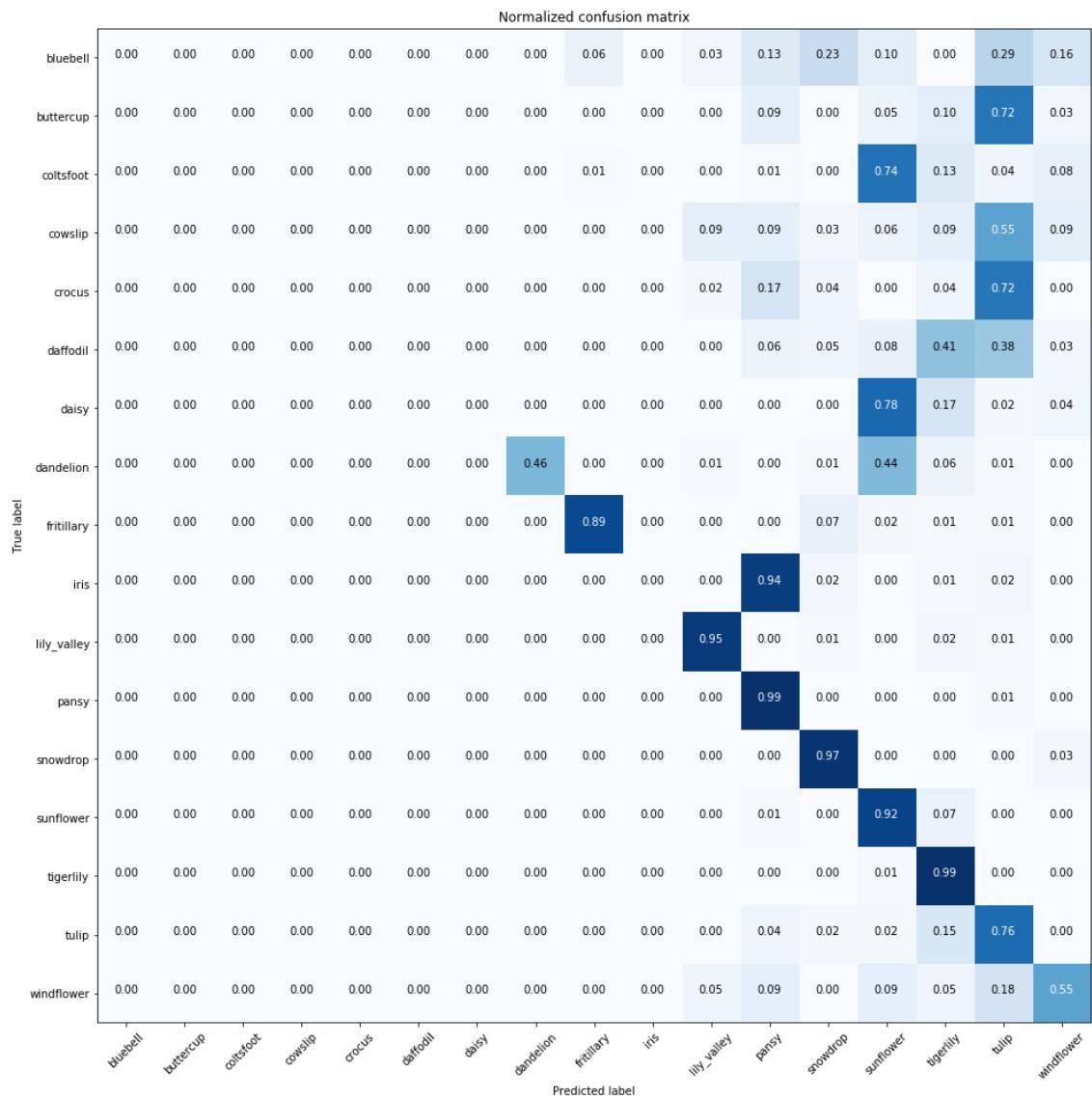


Figura A.12: Matriz de Confusão do experimento com 256 neurônios.

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	Iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.13: Matriz de Confusão do experimento com 512 neurônios.

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.14: Matriz de Confusão do experimento com 1024 neurônios.

Categoria	256 neurônios	512 neurônios	1024 neurônios
<i>Bluebell</i>	0.00	0.02	0.02
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	1.00	0.00	0.00
<i>Fritillary</i>	0.97	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.92	0.00	0.00
<i>Pansy</i>	0.58	0.00	0.00
<i>Snowdrop</i>	0.71	0.00	0.00
<i>Sunflower</i>	0.25	0.00	0.00
<i>Tigerlily</i>	0.57	0.00	0.00
<i>Tulip</i>	0.19	0.00	0.00
<i>Windflower</i>	0.31	0.00	0.00
Média	0.40	0.00	0.00

Tabela A.4: Resultados de *Precision*

Categoria	256 neurônios	512 neurônios	1024 neurônios
<i>Bluebell</i>	0.00	1.00	1.00
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	0.46	0.00	0.00
<i>Fritillary</i>	0.89	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.95	0.00	0.00
<i>Pansy</i>	0.99	0.00	0.00
<i>Snowdrop</i>	0.97	0.00	0.00
<i>Sunflower</i>	0.92	0.00	0.00
<i>Tigerlily</i>	0.99	0.00	0.00
<i>Tulip</i>	0.76	0.00	0.00
<i>Windflower</i>	0.55	0.00	0.00
Média	0.52	0.02	0.02

Tabela A.5: Resultados de *Recall*

Categoria	256 neurônios	512 neurônios	1024 neurônios
<i>Bluebell</i>	0.00	0.05	0.05
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	0.63	0.00	0.00
<i>Fritillary</i>	0.93	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.94	0.00	0.00
<i>Pansy</i>	0.73	0.00	0.00
<i>Snowdrop</i>	0.82	0.00	0.00
<i>Sunflower</i>	0.39	0.00	0.00
<i>Tigerlily</i>	0.73	0.00	0.00
<i>Tulip</i>	0.31	0.00	0.00
<i>Windflower</i>	0.39	0.00	0.00
Média	0.42	0.00	0.00

Tabela A.6: Resultados de *F1-score*

### A.3 Quantidade de Camadas Ocultas

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1 e 2</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

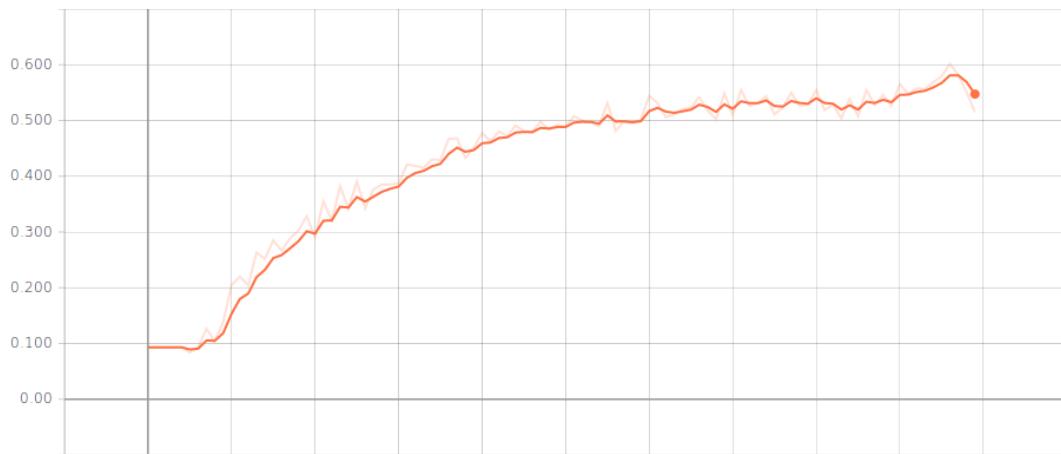


Figura A.15: Acurácia com 1 *hidden layer*: 54,7%.

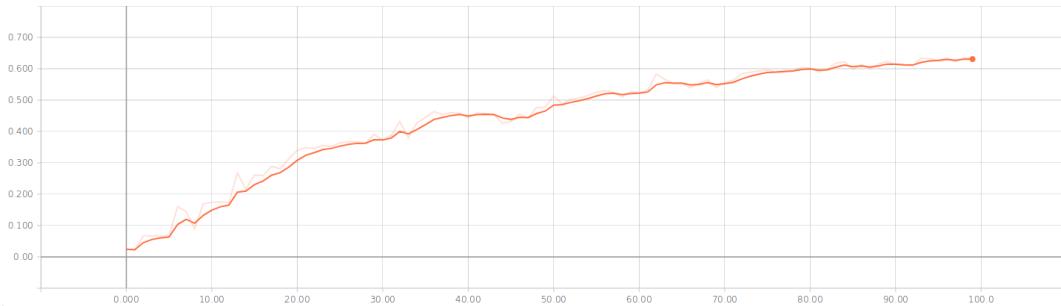
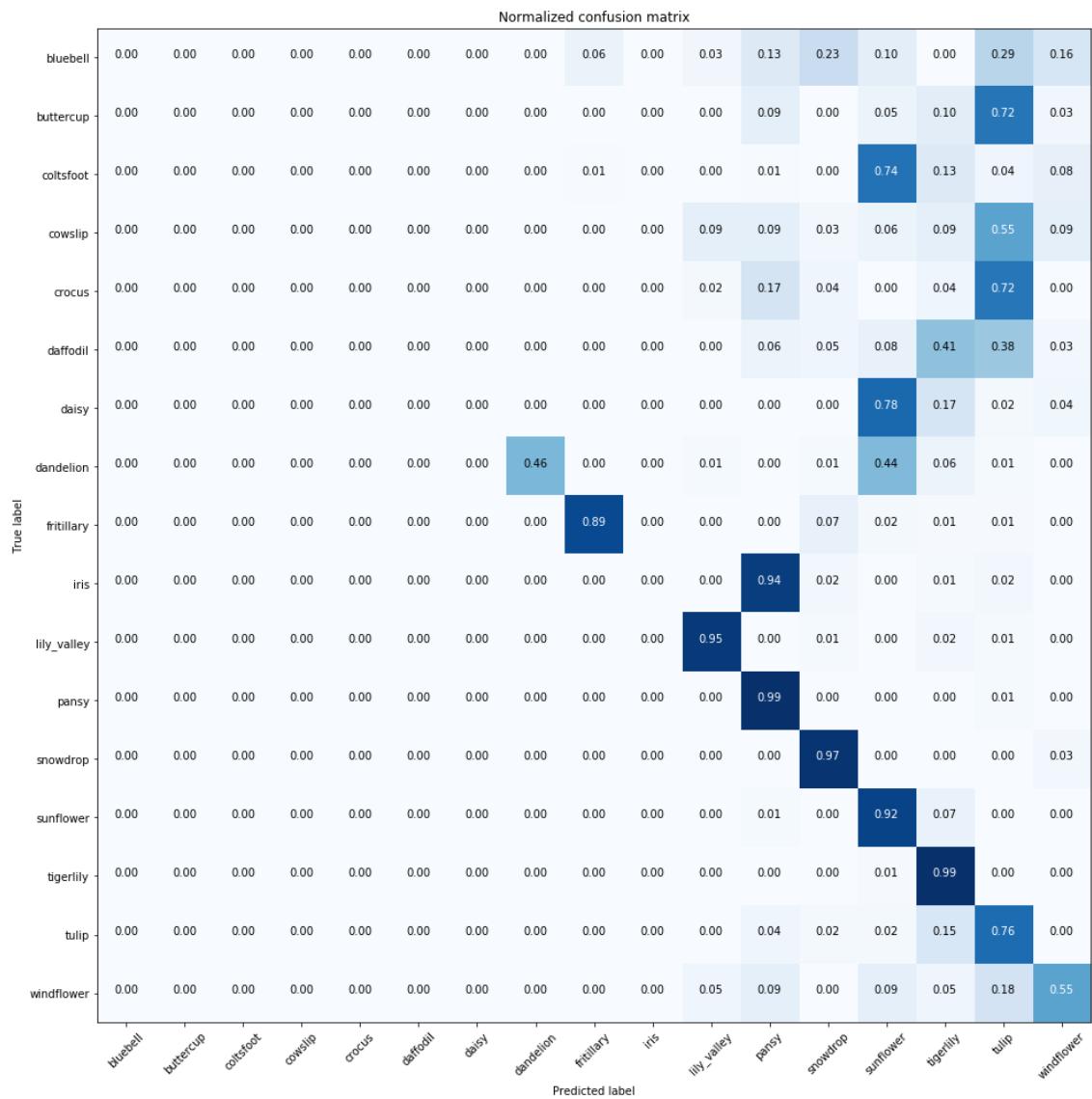
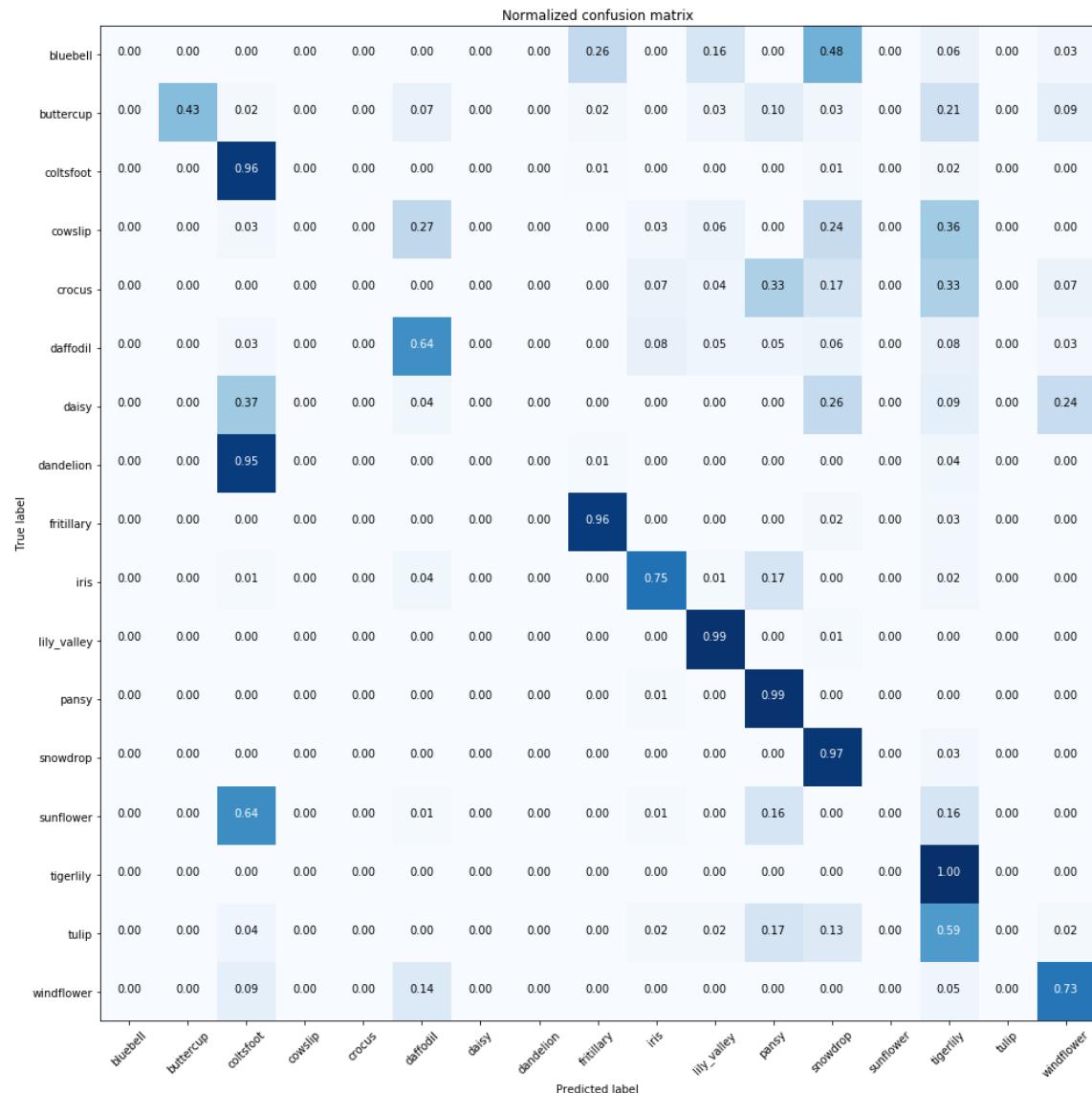


Figura A.16: Acurácia com 2 *hidden layers*: 63,1%.

Figura A.17: Matriz de Confusão do experimento com 1 *hidden layer*.

Figura A.18: Matriz de Confusão do experimento com 2 *hidden layers*.

Categoría	1 hidden layer	2 hidden layers
<i>Bluebell</i>	0.00	0.00
<i>Buttercup</i>	0.00	1.00
<i>Coltsfoot</i>	0.00	0.47
<i>Cowslip</i>	0.00	0.00
<i>Crocus</i>	0.00	0.00
<i>Daffodil</i>	0.00	0.66
<i>Daisy</i>	0.00	0.00
<i>Dandelion</i>	1.00	0.00
<i>Fritillary</i>	0.97	0.91
<i>Iris</i>	0.00	0.83
<i>Lily Valley</i>	0.92	0.84
<i>Pansy</i>	0.58	0.72
<i>Snowdrop</i>	0.71	0.50
<i>Sunflower</i>	0.25	0.00
<i>Tigerlily</i>	0.57	0.53
<i>Tulip</i>	0.19	0.00
<i>Windflower</i>	0.31	0.39
Média	0.40	0.49

Tabela A.7: Resultados de *Precision*

Categoría	1 hidden layer	2 hidden layers
<i>Bluebell</i>	0.00	0.00
<i>Buttercup</i>	0.00	0.43
<i>Coltsfoot</i>	0.00	0.96
<i>Cowslip</i>	0.00	0.00
<i>Crocus</i>	0.00	0.00
<i>Daffodil</i>	0.00	0.64
<i>Daisy</i>	0.00	0.00
<i>Dandelion</i>	0.46	0.00
<i>Fritillary</i>	0.89	0.96
<i>Iris</i>	0.00	0.75
<i>Lily Valley</i>	0.95	0.99
<i>Pansy</i>	0.99	0.99
<i>Snowdrop</i>	0.97	0.97
<i>Sunflower</i>	0.92	0.00
<i>Tigerlily</i>	0.99	1.00
<i>Tulip</i>	0.76	0.00
<i>Windflower</i>	0.55	0.73
Média	0.52	0.63

Tabela A.8: Resultados de *Recall*

Categoria	1 <i>hidden layer</i>	2 <i>hidden layers</i>
<i>Bluebell</i>	0.00	0.00
<i>Buttercup</i>	0.00	0.60
<i>Coltsfoot</i>	0.00	0.63
<i>Cowslip</i>	0.00	0.00
<i>Crocus</i>	0.00	0.00
<i>Daffodil</i>	0.00	0.65
<i>Daisy</i>	0.00	0.00
<i>Dandelion</i>	0.63	0.00
<i>Fritillary</i>	0.93	0.93
<i>Iris</i>	0.00	0.78
<i>Lily Valley</i>	0.94	0.91
<i>Pansy</i>	0.73	0.83
<i>Snowdrop</i>	0.82	0.66
<i>Sunflower</i>	0.39	0.00
<i>Tigerlily</i>	0.73	0.69
<i>Tulip</i>	0.31	0.00
<i>Windflower</i>	0.39	0.51
Média	0.42	0.53

Tabela A.9: Resultados de *F1-score*

## A.4 Funções de Ativação da Camada de Saída

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax, Tanh, Sigmóide</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

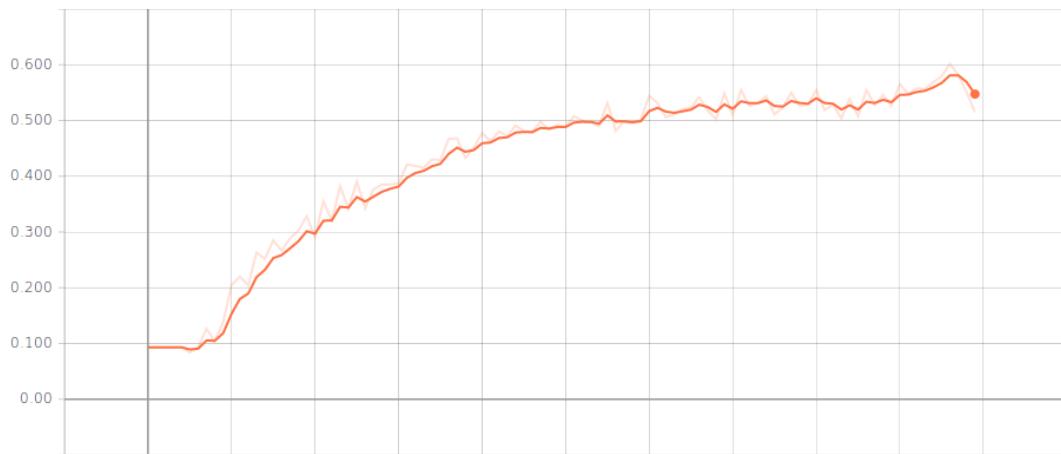


Figura A.19: Acurácia com Softmax: 54,7%.

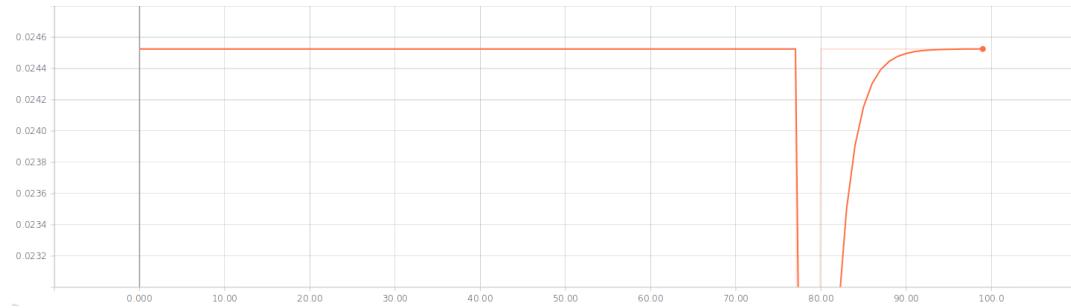


Figura A.20: Acurácia com Sigmóide: 2,5%.

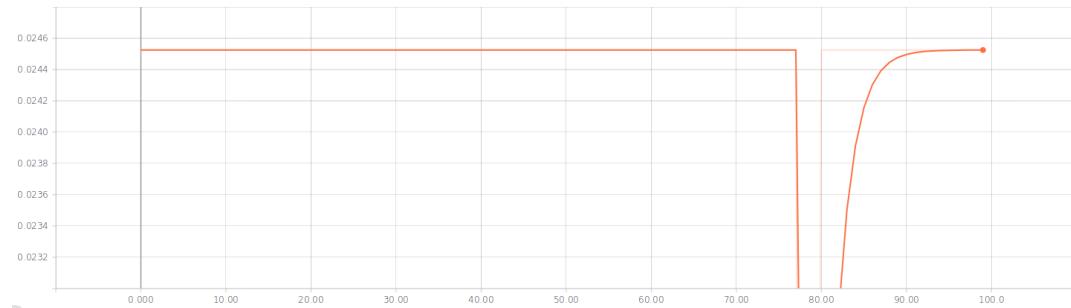


Figura A.21: Acurácia com Tanh: 2,5%.

Normalized confusion matrix																	
True label	bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.03	0.13	0.23	0.10	0.00	0.29	0.16	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.05	0.10	0.72	0.03	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.74	0.13	0.04	0.08	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.09	0.03	0.06	0.09	0.55	0.09	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.17	0.04	0.00	0.04	0.72	0.00	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.05	0.08	0.41	0.38	0.03	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.78	0.17	0.02	0.04	
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.46	0.00	0.00	0.01	0.00	0.01	0.44	0.06	0.01	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.89	0.00	0.00	0.00	0.07	0.02	0.01	0.01	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.02	0.00	0.01	0.02	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00	0.01	0.00	0.02	0.01	0.01	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.01	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.00	0.00	0.00	0.00	0.03
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.92	0.07	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.99	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.02	0.02	0.15	0.76	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.09	0.00	0.09	0.05	0.18	0.55	

Figura A.22: Matriz de Confusão do experimento com Softmax.

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	Iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.23: Matriz de Confusão do experimento com Sigmóide.

		Normalized confusion matrix																
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	bluebell	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	buttercup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	coltsfoot	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	cowslip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	crocus	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daffodil	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	daisy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	dandelion	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	fritillary	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	iris	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	lily_valley	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	pansy	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	snowdrop	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	sunflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tigerlily	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	tulip	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	windflower	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Figura A.24: Matriz de Confusão do experimento com Tanh.

Categoría	Softmax	Sigmóide	Tanh
<i>Bluebell</i>	0.00	0.02	0.02
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	1.00	0.00	0.00
<i>Fritillary</i>	0.97	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.92	0.00	0.00
<i>Pansy</i>	0.58	0.00	0.00
<i>Snowdrop</i>	0.71	0.00	0.00
<i>Sunflower</i>	0.25	0.00	0.00
<i>Tigerlily</i>	0.57	0.00	0.00
<i>Tulip</i>	0.19	0.00	0.00
<i>Windflower</i>	0.31	0.00	0.00
Média	0.40	0.00	0.00

Tabela A.10: Resultados de *Precision*

Categoría	Softmax	Sigmóide	Tanh
<i>Bluebell</i>	0.00	1.00	1.00
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	0.46	0.00	0.00
<i>Fritillary</i>	0.89	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.95	0.00	0.00
<i>Pansy</i>	0.99	0.00	0.00
<i>Snowdrop</i>	0.97	0.00	0.00
<i>Sunflower</i>	0.92	0.00	0.00
<i>Tigerlily</i>	0.99	0.00	0.00
<i>Tulip</i>	0.76	0.00	0.00
<i>Windflower</i>	0.55	0.00	0.00
Média	0.52	0.02	0.02

Tabela A.11: Resultados de *Recall*

Categoria	Softmax	Sigmóide	Tanh
<i>Bluebell</i>	0.00	0.05	0.05
<i>Buttercup</i>	0.00	0.00	0.00
<i>Coltsfoot</i>	0.00	0.00	0.00
<i>Cowslip</i>	0.00	0.00	0.00
<i>Crocus</i>	0.00	0.00	0.00
<i>Daffodil</i>	0.00	0.00	0.00
<i>Daisy</i>	0.00	0.00	0.00
<i>Dandelion</i>	0.63	0.00	0.00
<i>Fritillary</i>	0.93	0.00	0.00
<i>Iris</i>	0.00	0.00	0.00
<i>Lily Valley</i>	0.94	0.00	0.00
<i>Pansy</i>	0.73	0.00	0.00
<i>Snowdrop</i>	0.82	0.00	0.00
<i>Sunflower</i>	0.39	0.00	0.00
<i>Tigerlily</i>	0.73	0.00	0.00
<i>Tulip</i>	0.31	0.00	0.00
<i>Windflower</i>	0.39	0.00	0.00
Média	0.42	0.00	0.00

Tabela A.12: Resultados de *F1-score*

## A.5 Funções de Ativação da Camada Oculta

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>ReLU, Sigmóide, Tanh</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada</b>

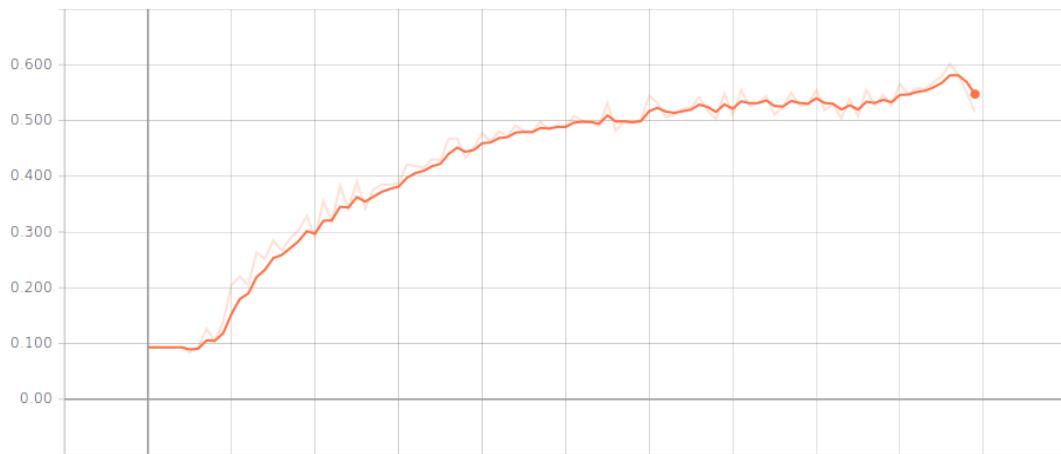


Figura A.25: Acurácia com ReLU: 54,7%.

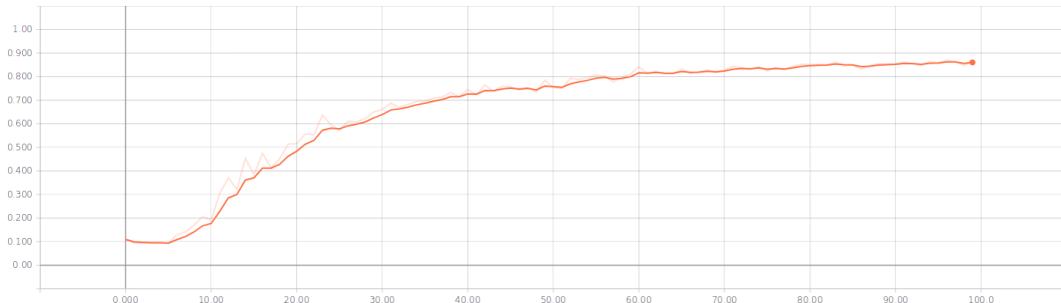


Figura A.26: Acurácia com Sigmóide: 86,1%.

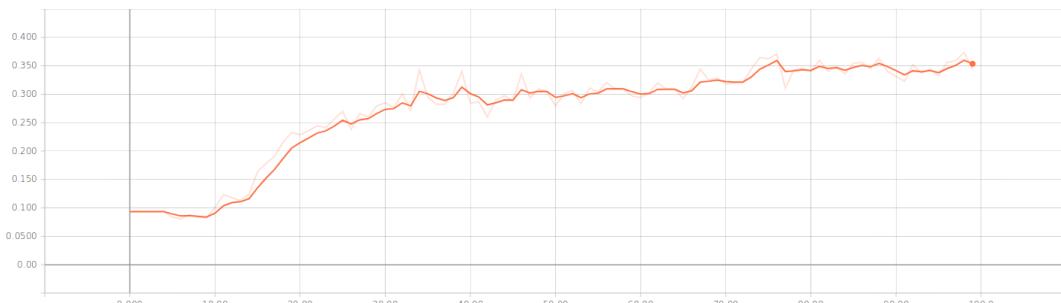


Figura A.27: Acurácia com Tanh: 35,4%.

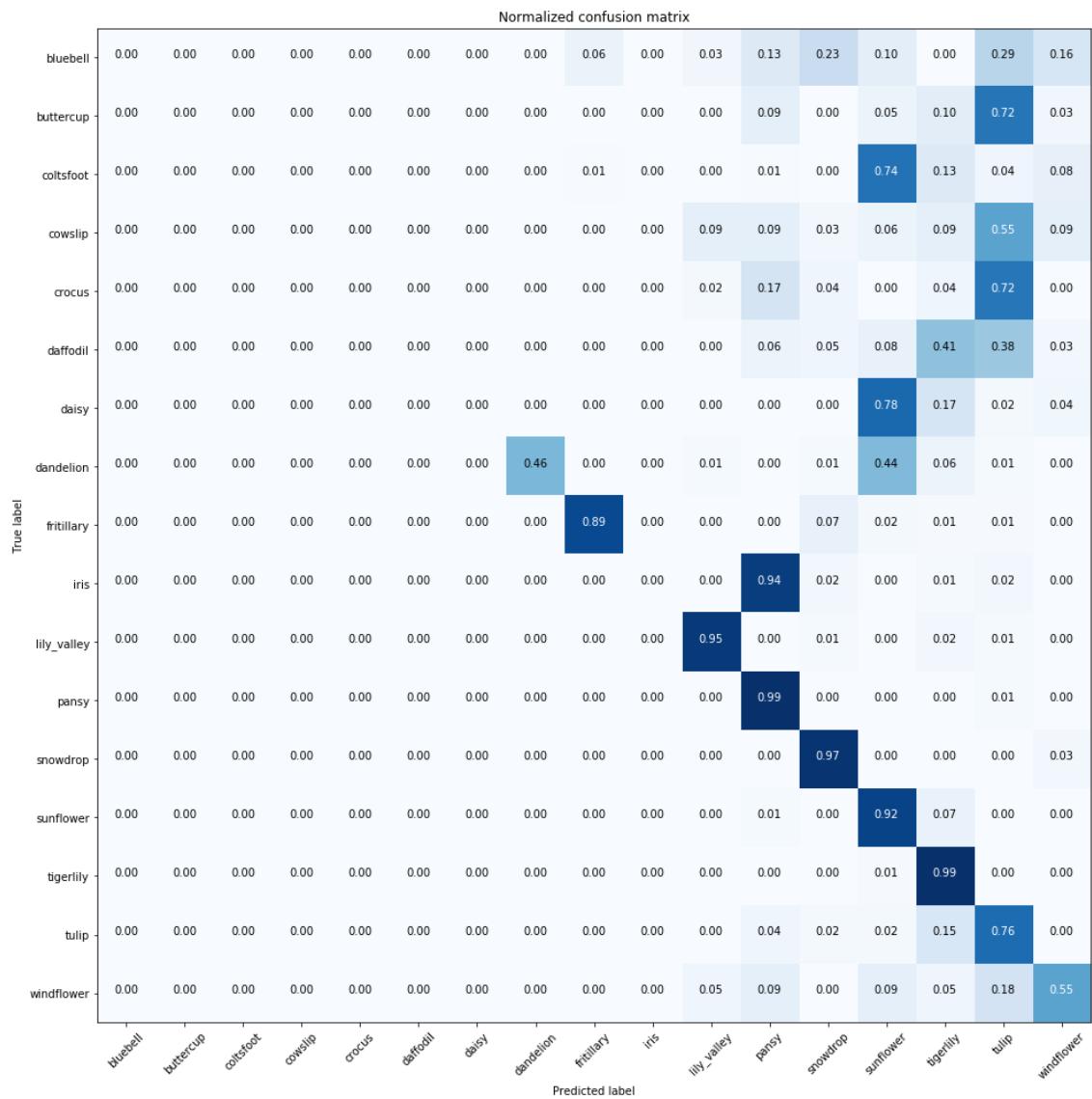


Figura A.28: Matriz de Confusão do experimento com ReLU.

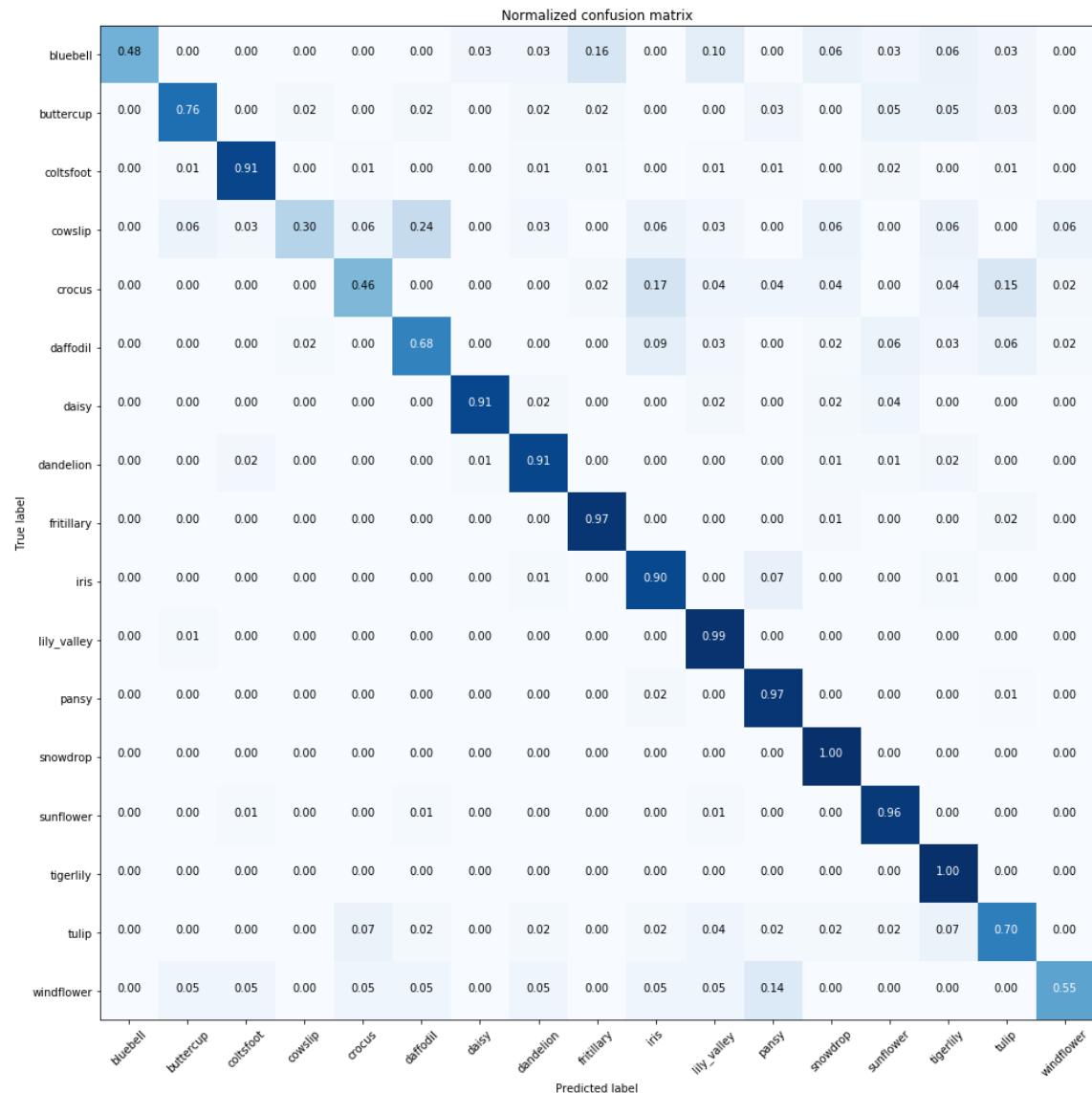


Figura A.29: Matriz de Confusão do experimento com Sigmóide.

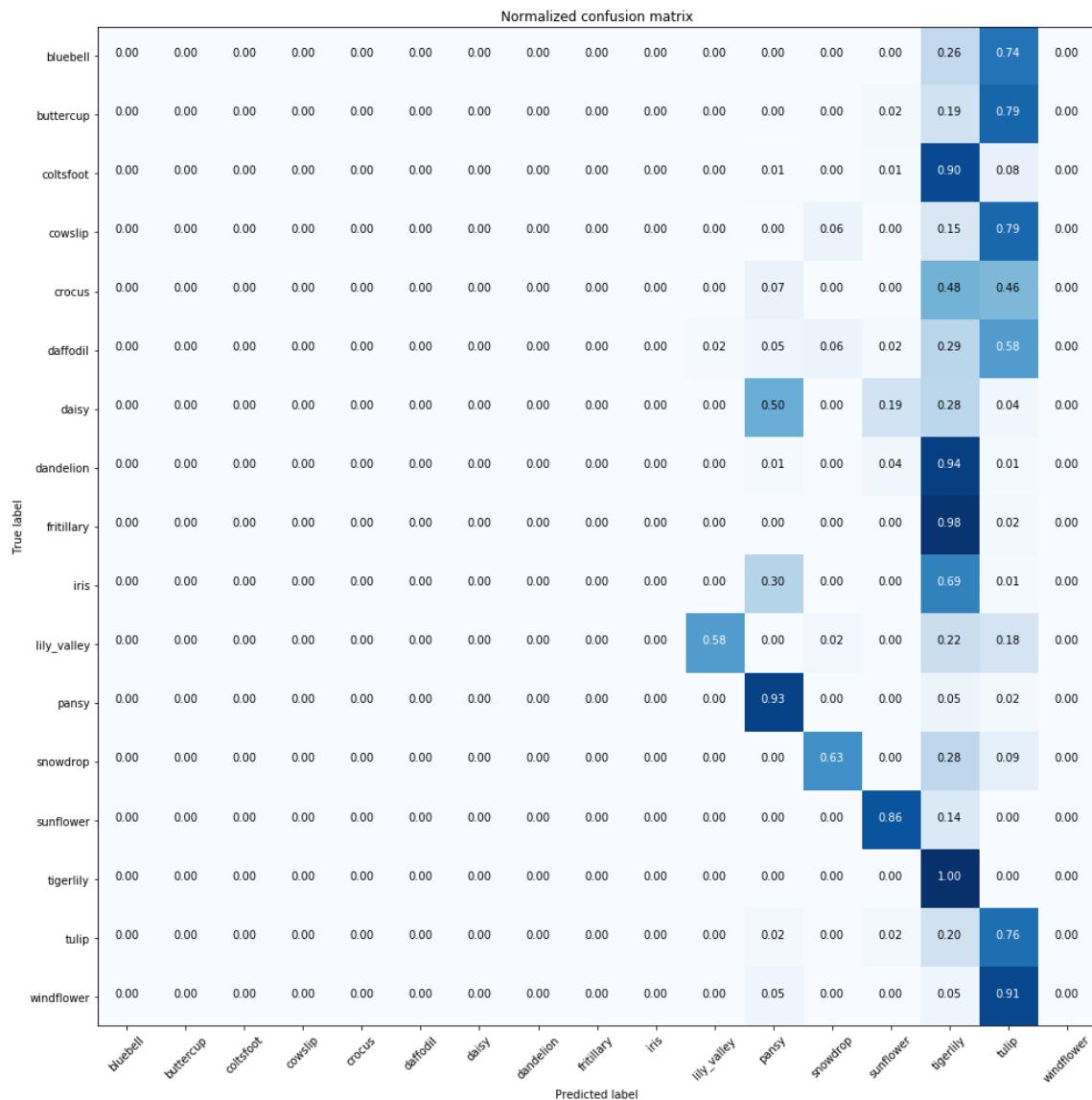


Figura A.30: Matriz de Confusão do experimento com Tanh.

Categoría	ReLU	Sigmóide	Tanh
<i>Bluebell</i>	0.00	1.00	0.00
<i>Buttercup</i>	0.00	0.90	0.00
<i>Coltsfoot</i>	0.00	0.96	0.00
<i>Cowslip</i>	0.00	0.83	0.00
<i>Crocus</i>	0.00	0.75	0.00
<i>Daffodil</i>	0.00	0.79	0.00
<i>Daisy</i>	0.00	0.96	0.00
<i>Dandelion</i>	1.00	0.89	0.00
<i>Fritillary</i>	0.97	0.92	0.00
<i>Iris</i>	0.00	0.78	0.00
<i>Lily Valley</i>	0.92	0.86	0.98
<i>Pansy</i>	0.58	0.91	0.69
<i>Snowdrop</i>	0.71	0.86	0.84
<i>Sunflower</i>	0.25	0.82	0.78
<i>Tigerlily</i>	0.57	0.87	0.19
<i>Tulip</i>	0.19	0.64	0.14
<i>Windflower</i>	0.31	0.75	0.00
Média	0.40	0.87	0.26

Tabela A.13: Resultados de *Precision*

Categoría	ReLU	Sigmóide	Tanh
<i>Bluebell</i>	0.00	0.48	0.00
<i>Buttercup</i>	0.00	0.76	0.00
<i>Coltsfoot</i>	0.00	0.91	0.00
<i>Cowslip</i>	0.00	0.30	0.00
<i>Crocus</i>	0.00	0.46	0.00
<i>Daffodil</i>	0.00	0.68	0.00
<i>Daisy</i>	0.00	0.91	0.00
<i>Dandelion</i>	0.46	0.91	0.00
<i>Fritillary</i>	0.89	0.97	0.00
<i>Iris</i>	0.00	0.90	0.00
<i>Lily Valley</i>	0.95	0.99	0.58
<i>Pansy</i>	0.99	0.97	0.93
<i>Snowdrop</i>	0.97	1.00	0.63
<i>Sunflower</i>	0.92	0.96	0.86
<i>Tigerlily</i>	0.99	1.00	1.00
<i>Tulip</i>	0.76	0.70	0.76
<i>Windflower</i>	0.55	0.55	0.00
Média	0.52	0.87	0.35

Tabela A.14: Resultados de *Recall*

Categoría	ReLU	Sigmóide	Tanh
<i>Bluebell</i>	0.00	0.65	0.00
<i>Buttercup</i>	0.00	0.82	0.00
<i>Coltsfoot</i>	0.00	0.94	0.00
<i>Cowslip</i>	0.00	0.44	0.00
<i>Crocus</i>	0.00	0.57	0.00
<i>Daffodil</i>	0.00	0.73	0.00
<i>Daisy</i>	0.00	0.93	0.00
<i>Dandelion</i>	0.63	0.90	0.00
<i>Fritillary</i>	0.93	0.95	0.00
<i>Iris</i>	0.00	0.84	0.00
<i>Lily Valley</i>	0.94	0.92	0.73
<i>Pansy</i>	0.73	0.94	0.79
<i>Snowdrop</i>	0.82	0.92	0.72
<i>Sunflower</i>	0.39	0.89	0.82
<i>Tigerlily</i>	0.73	0.93	0.31
<i>Tulip</i>	0.31	0.67	0.24
<i>Windflower</i>	0.39	0.63	0.00
Média	0.42	0.86	0.26

Tabela A.15: Resultados de *F1-score*

## A.6 Funções de Custo

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop</b>
Função de custo	<b>Entropia cruzada, Hinge</b>

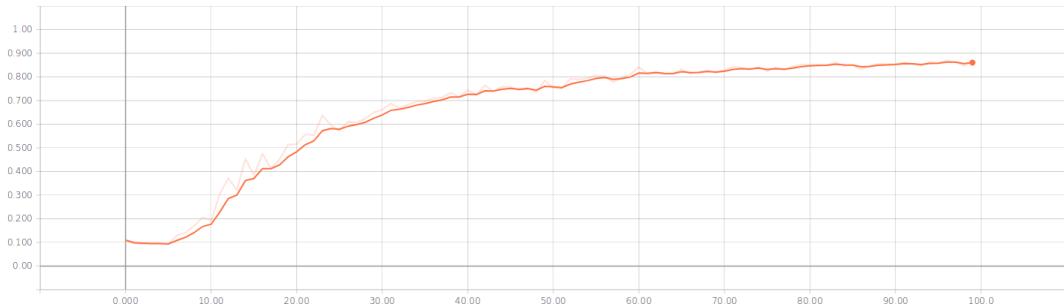


Figura A.31: Acurácia com Entropia cruzada: 86,1%.

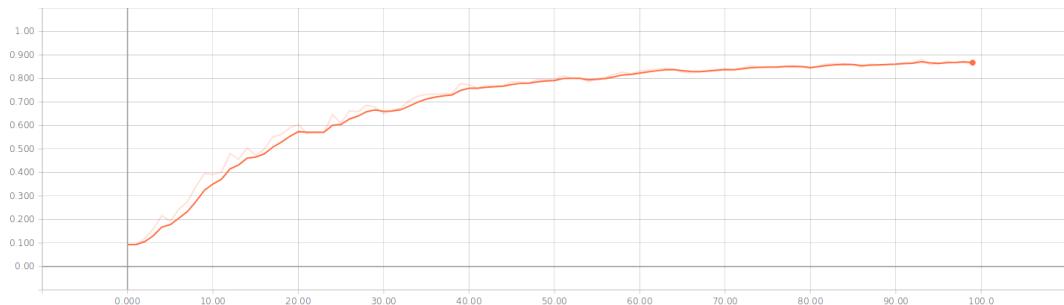


Figura A.32: Acurácia com Hinge: 86,7%.

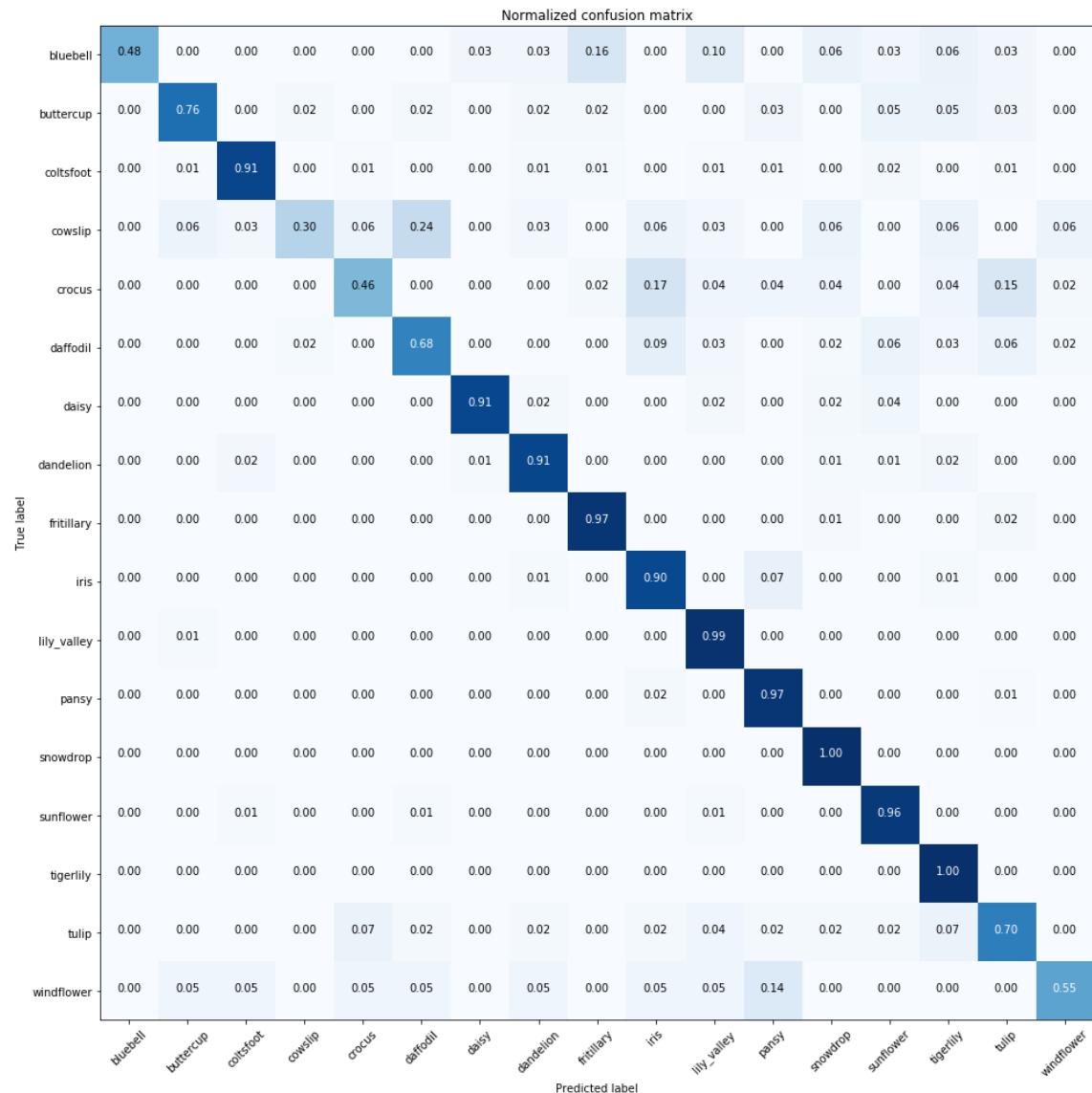


Figura A.33: Matriz de Confusão do experimento com Entropia Cruzada.

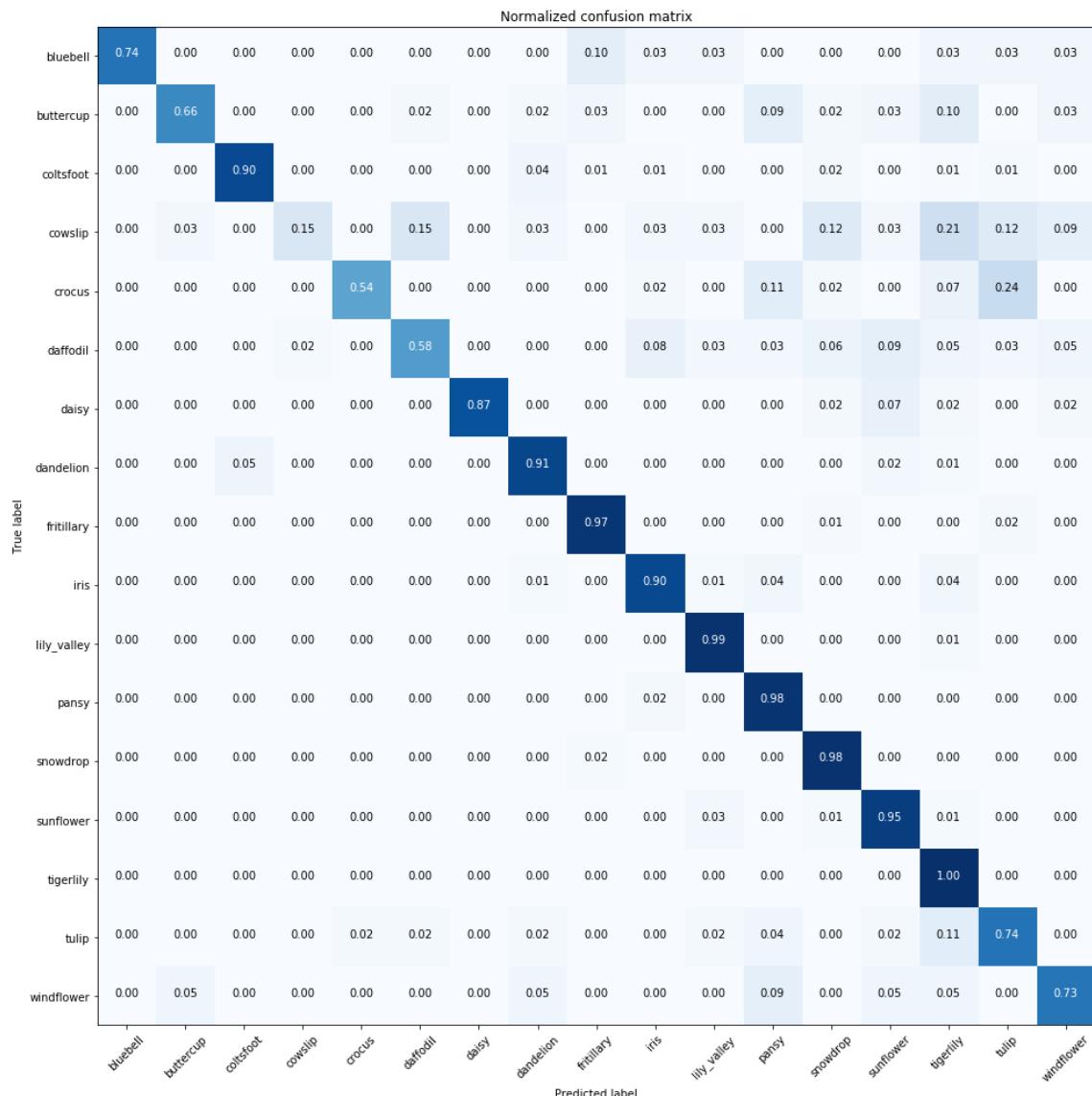


Figura A.34: Matriz de Confusão do experimento com Hinge.

Categoría	Entropia Cruzada	Hinge
<i>Bluebell</i>	1.00	1.00
<i>Buttercup</i>	0.90	0.95
<i>Coltsfoot</i>	0.96	0.97
<i>Cowslip</i>	0.83	0.83
<i>Crocus</i>	0.75	0.96
<i>Daffodil</i>	0.79	0.84
<i>Daisy</i>	0.96	1.00
<i>Dandelion</i>	0.89	0.87
<i>Fritillary</i>	0.92	0.94
<i>Iris</i>	0.78	0.86
<i>Lily Valley</i>	0.86	0.91
<i>Pansy</i>	0.91	0.88
<i>Snowdrop</i>	0.86	0.80
<i>Sunflower</i>	0.82	0.80
<i>Tigerlily</i>	0.87	0.78
<i>Tulip</i>	0.64	0.61
<i>Windflower</i>	0.75	0.62
Média	0.87	0.87

Tabela A.16: Resultados de *Precision*

Categoría	Entropia Cruzada	Hinge
<i>Bluebell</i>	0.48	0.74
<i>Buttercup</i>	0.76	0.66
<i>Coltsfoot</i>	0.91	0.90
<i>Cowslip</i>	0.30	0.15
<i>Crocus</i>	0.46	0.54
<i>Daffodil</i>	0.68	0.58
<i>Daisy</i>	0.91	0.87
<i>Dandelion</i>	0.91	0.91
<i>Fritillary</i>	0.97	0.97
<i>Iris</i>	0.90	0.90
<i>Lily Valley</i>	0.99	0.99
<i>Pansy</i>	0.97	0.98
<i>Snowdrop</i>	1.00	0.98
<i>Sunflower</i>	0.96	0.95
<i>Tigerlily</i>	1.00	1.00
<i>Tulip</i>	0.70	0.74
<i>Windflower</i>	0.55	0.73
Média	0.87	0.86

Tabela A.17: Resultados de *Recall*

Categoria	Entropia Cruzada	Hinge
<i>Bluebell</i>	0.65	0.85
<i>Buttercup</i>	0.82	0.78
<i>Coltsfoot</i>	0.94	0.93
<i>Cowslip</i>	0.44	0.26
<i>Crocus</i>	0.57	0.69
<i>Daffodil</i>	0.73	0.68
<i>Daisy</i>	0.93	0.93
<i>Dandelion</i>	0.90	0.89
<i>Fritillary</i>	0.95	0.96
<i>Iris</i>	0.84	0.88
<i>Lily Valley</i>	0.92	0.95
<i>Pansy</i>	0.94	0.93
<i>Snowdrop</i>	0.92	0.88
<i>Sunflower</i>	0.89	0.87
<i>Tigerlily</i>	0.93	0.87
<i>Tulip</i>	0.67	0.67
<i>Windflower</i>	0.63	0.67
Média	0.86	0.86

Tabela A.18: Resultados de *F1-score*

## A.7 Algoritmos de Otimização

Estes experimentos possuem a configuração de Rede Neural apresentada na tabela abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>RMSprop, Adam, SGD</b>
Função de custo	<b>Hinge</b>

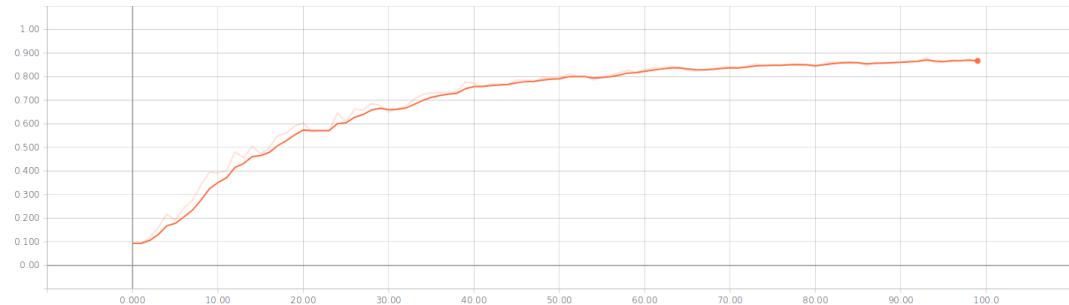


Figura A.35: Acurácia com RMSprop: 86,7%.

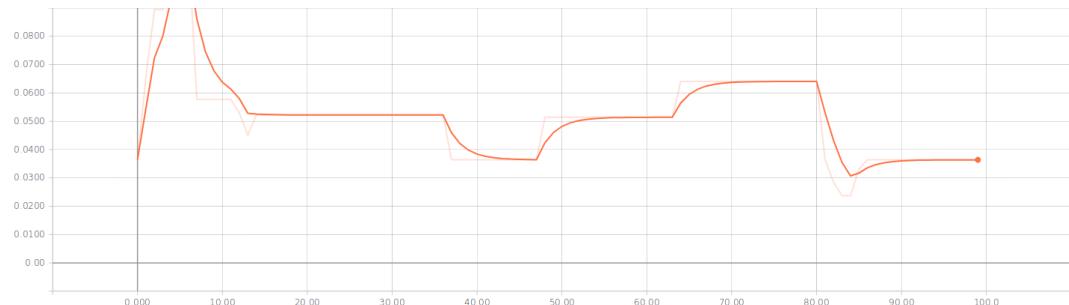


Figura A.36: Acurácia com Adam: 3,6%.

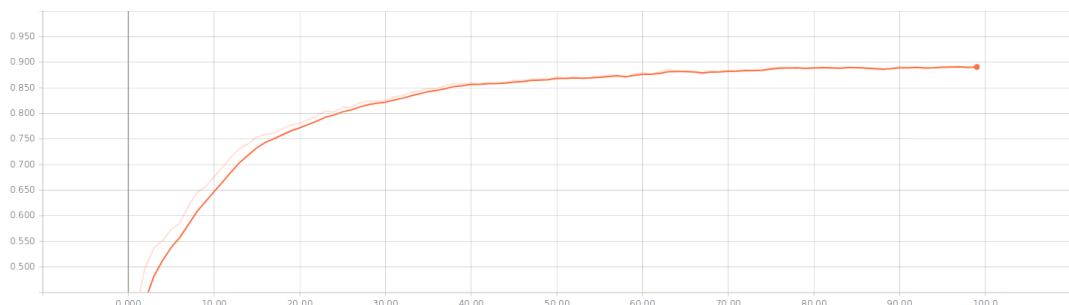


Figura A.37: Acurácia com SGD: 89,1%.

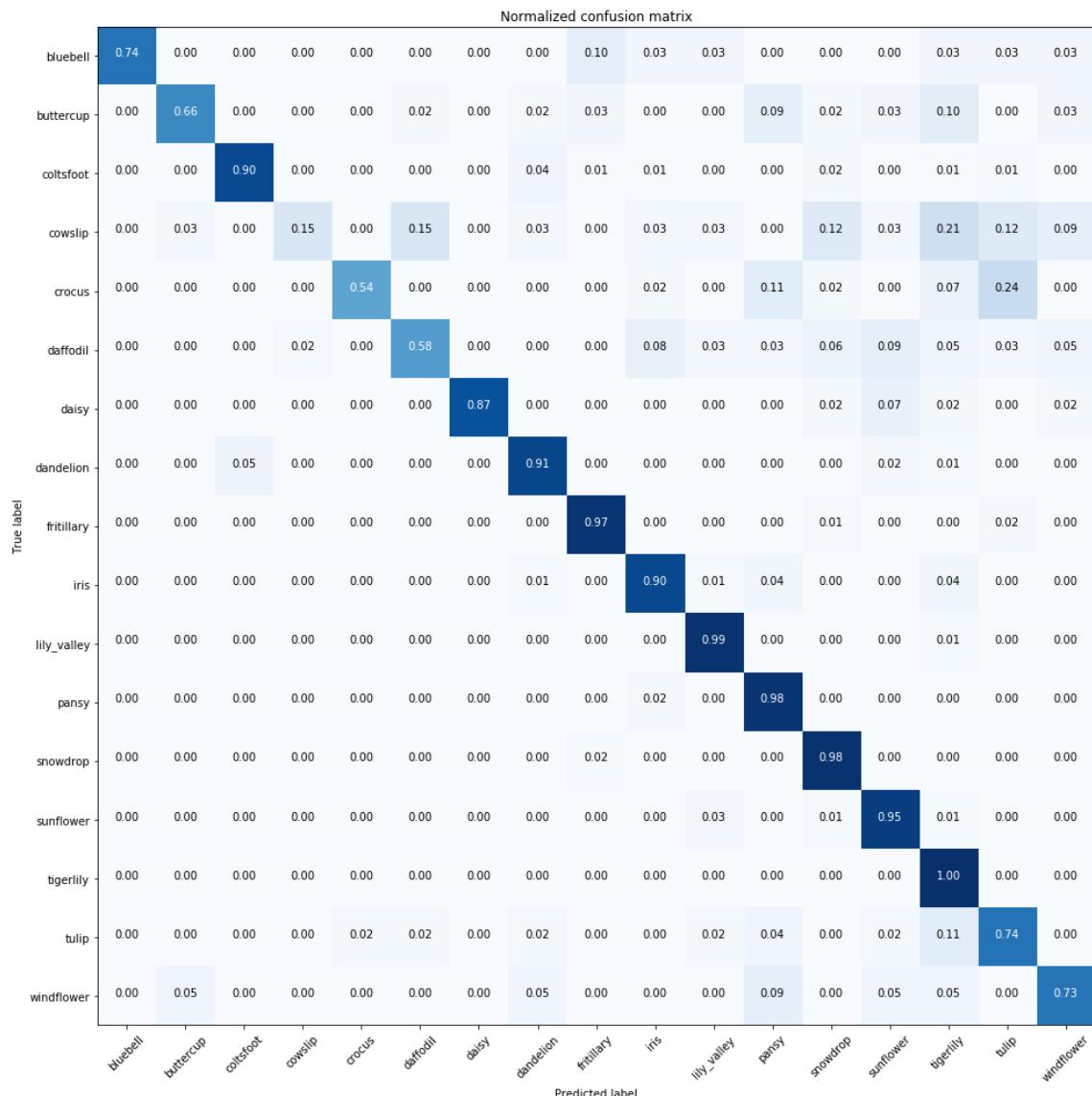


Figura A.38: Matriz de Confusão do experimento com RMSprop.

		Normalized confusion matrix																		
		bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	Iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower		
True label	bluebell	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	buttercup	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	coltsfoot	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	cowslip	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	crocus	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	daffodil	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	daisy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	dandelion	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	fritillary	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	Iris	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	lily_valley	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	pansy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	snowdrop	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	sunflower	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	tigerlily	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	tulip	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	
	windflower	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	

Figura A.39: Matriz de Confusão do experimento com Adam.

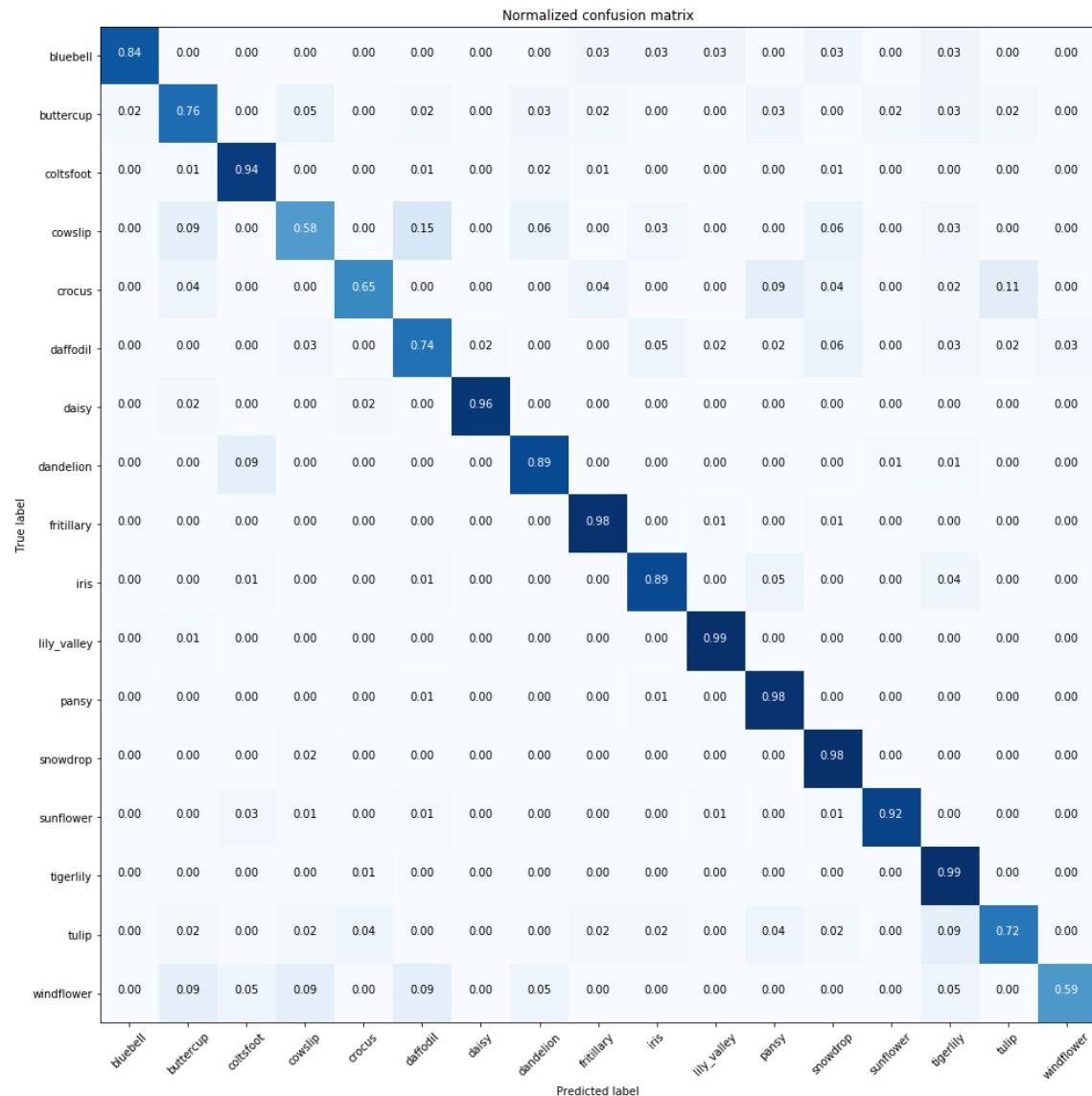


Figura A.40: Matriz de Confusão do experimento com SGD.

Categoría	RMSprop	Adam	SGD
<i>Bluebell</i>	1.00	0.00	0.96
<i>Buttercup</i>	0.95	0.00	0.79
<i>Coltsfoot</i>	0.97	0.00	0.92
<i>Cowslip</i>	0.83	0.00	0.66
<i>Crocus</i>	0.96	0.00	0.88
<i>Daffodil</i>	0.84	0.00	0.80
<i>Daisy</i>	1.00	0.00	0.98
<i>Dandelion</i>	0.87	0.00	0.90
<i>Fritillary</i>	0.94	0.00	0.95
<i>Iris</i>	0.86	0.00	0.90
<i>Lily Valley</i>	0.91	0.00	0.95
<i>Pansy</i>	0.88	0.00	0.92
<i>Snowdrop</i>	0.80	0.00	0.83
<i>Sunflower</i>	0.80	0.00	0.97
<i>Tigerlily</i>	0.78	0.00	0.88
<i>Tulip</i>	0.61	0.04	0.82
<i>Windflower</i>	0.62	0.00	0.87
Média	0.87	0.00	0.90

Tabela A.19: Resultados de *Precision*

Categoría	RMSprop	Adam	SGD
<i>Bluebell</i>	0.74	0.00	0.84
<i>Buttercup</i>	0.66	0.00	0.76
<i>Coltsfoot</i>	0.90	0.00	0.94
<i>Cowslip</i>	0.15	0.00	0.58
<i>Crocus</i>	0.54	0.00	0.65
<i>Daffodil</i>	0.58	0.00	0.74
<i>Daisy</i>	0.87	0.00	0.96
<i>Dandelion</i>	0.91	0.00	0.89
<i>Fritillary</i>	0.97	0.00	0.98
<i>Iris</i>	0.90	0.00	0.89
<i>Lily Valley</i>	0.99	0.00	0.99
<i>Pansy</i>	0.98	0.00	0.98
<i>Snowdrop</i>	0.98	0.00	0.98
<i>Sunflower</i>	0.95	0.00	0.92
<i>Tigerlily</i>	1.00	0.00	0.99
<i>Tulip</i>	0.74	1.00	0.72
<i>Windflower</i>	0.73	0.00	0.59
Média	0.86	0.04	0.90

Tabela A.20: Resultados de *Recall*

Categoria	RMSprop	Adam	SGD
<i>Bluebell</i>	0.85	0.00	0.90
<i>Buttercup</i>	0.78	0.00	0.77
<i>Coltsfoot</i>	0.93	0.00	0.93
<i>Cowslip</i>	0.26	0.00	0.61
<i>Crocus</i>	0.69	0.00	0.75
<i>Daffodil</i>	0.68	0.00	0.77
<i>Daisy</i>	0.93	0.00	0.97
<i>Dandelion</i>	0.89	0.00	0.89
<i>Fritillary</i>	0.96	0.00	0.97
<i>Iris</i>	0.88	0.00	0.90
<i>Lily Valley</i>	0.95	0.00	0.97
<i>Pansy</i>	0.93	0.00	0.95
<i>Snowdrop</i>	0.88	0.00	0.90
<i>Sunflower</i>	0.87	0.00	0.94
<i>Tigerlily</i>	0.87	0.00	0.93
<i>Tulip</i>	0.67	0.07	0.77
<i>Windflower</i>	0.67	0.00	0.70
Média	0.86	0.00	0.89

Tabela A.21: Resultados de *F1-score*

## A.8 Modelos de *Feature Extraction*

Estes experimentos possuem a configuração de Rede Neural apresentada abaixo.

Modelo e pesos das camadas de extração de recursos	<b>VGG16, VGG19, InceptionV3, ResNet50, NAS-Net</b>
Qtde. de camadas ocultas de classificação	<b>1</b>
Qtde. de neurônios nas camadas ocultas de classificação	<b>256</b>
Função de ativação das camadas ocultas de classificação	<b>Sigmóide</b>
Fração de <i>dropout</i> entre camadas de classificação	<b>50%</b>
Função de ativação na camada de classificação de saída	<b>Softmax</b>
Algoritmo de otimização dos pesos	<b>SGD</b>
Função de custo	<b>Hinge</b>

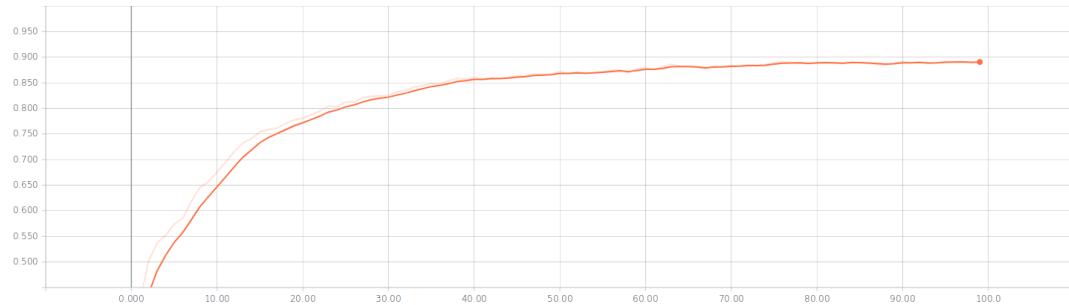


Figura A.41: Acurácia com VGG16: 89,1%.

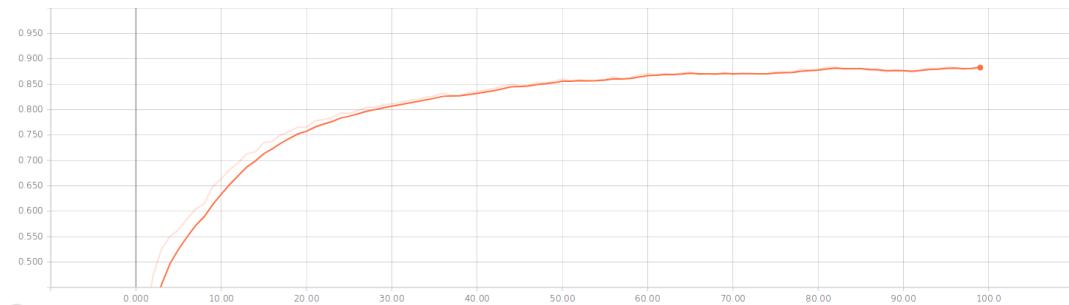


Figura A.42: Acurácia com VGG19: 88,3%.

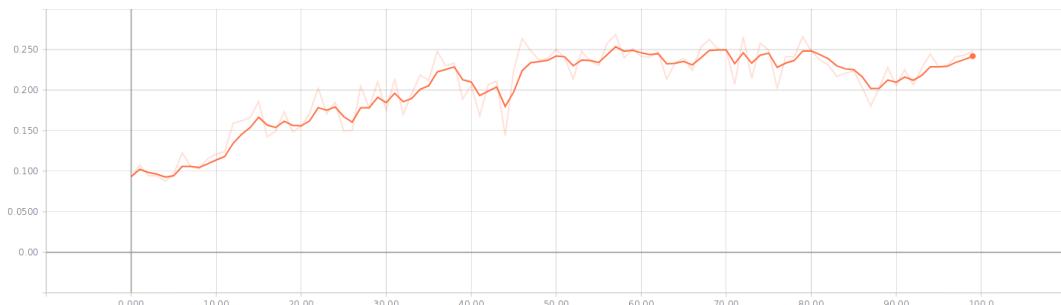


Figura A.43: Acurácia com InceptionV3: 24,2%.

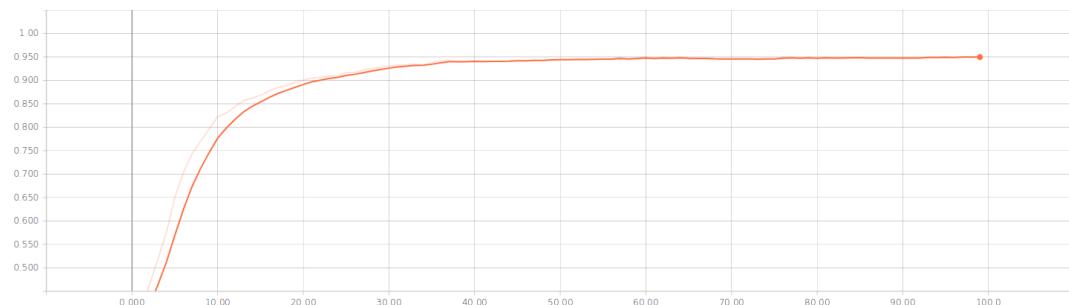


Figura A.44: Acurácia com ResNet50: 95,0%.

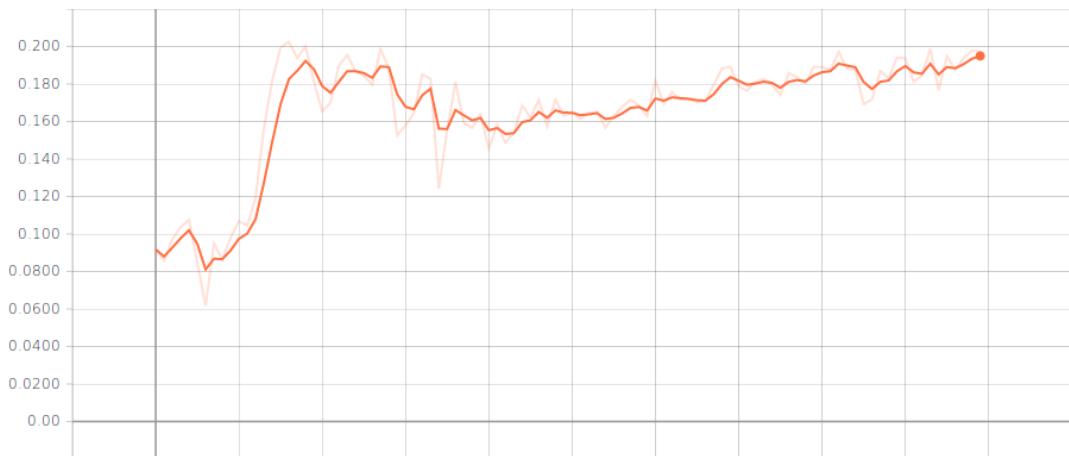


Figura A.45: Acurácia com NASNet: 19,5%.

Normalized confusion matrix																	
	bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
True label	0.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.03	0.00	0.03	0.00	0.03	0.00	0.00
bluebell	0.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.03	0.00	0.03	0.00	0.03	0.00	0.00
buttercup	0.02	0.76	0.00	0.05	0.00	0.02	0.00	0.03	0.02	0.00	0.00	0.03	0.00	0.02	0.03	0.02	0.00
coltsfoot	0.00	0.01	0.94	0.00	0.00	0.01	0.00	0.02	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
cowslip	0.00	0.09	0.00	0.58	0.00	0.15	0.00	0.06	0.00	0.03	0.00	0.00	0.06	0.00	0.03	0.00	0.00
crocus	0.00	0.04	0.00	0.00	0.65	0.00	0.00	0.00	0.04	0.00	0.00	0.09	0.04	0.00	0.02	0.11	0.00
daffodil	0.00	0.00	0.00	0.03	0.00	0.74	0.02	0.00	0.00	0.05	0.02	0.02	0.06	0.00	0.03	0.02	0.03
daisy	0.00	0.02	0.00	0.00	0.02	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dandelion	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.89	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00
fritillary	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00
iris	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.89	0.00	0.05	0.00	0.00	0.04	0.00	0.00
lily_valley	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pansy	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00
snowdrop	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00
sunflower	0.00	0.00	0.03	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.01	0.92	0.00	0.00	0.00	0.00
tigerlily	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00
tulip	0.00	0.02	0.00	0.02	0.04	0.00	0.00	0.00	0.02	0.02	0.00	0.04	0.02	0.00	0.09	0.72	0.00
windflower	0.00	0.09	0.05	0.09	0.00	0.09	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.59	0.00

Figura A.46: Matriz de Confusão do experimento com VGG16.

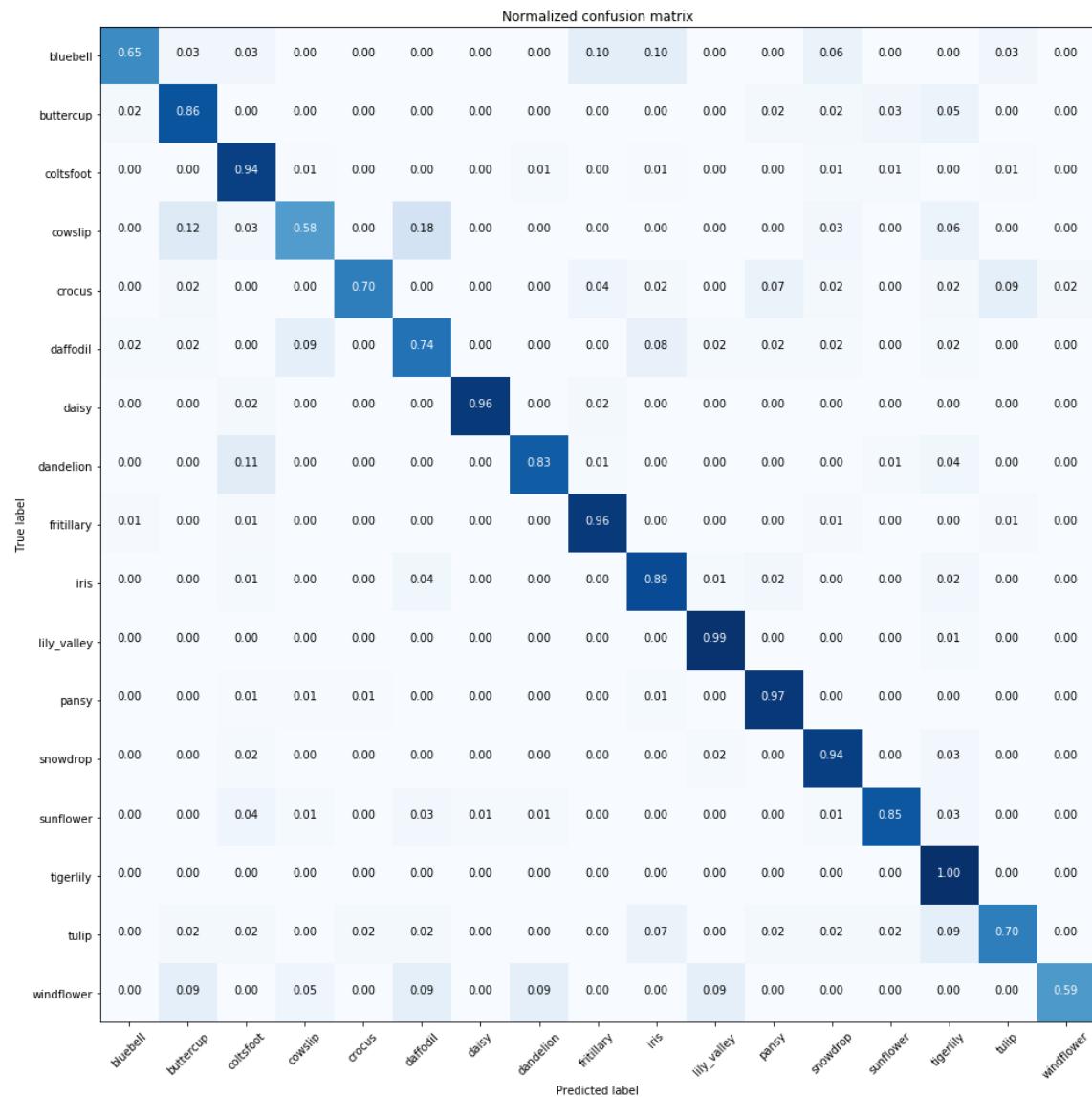


Figura A.47: Matriz de Confusão do experimento com VGG19.

Normalized confusion matrix																	
True label	bluebell	buttercup	coltsfoot	cowslip	crocus	daffodil	daisy	dandelion	fritillary	iris	lily_valley	pansy	snowdrop	sunflower	tigerlily	tulip	windflower
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.10	0.13	0.19	0.10	0.00	0.00	0.39	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.05	0.12	0.17	0.00	0.02	0.62	0.00	0.00
	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.13	0.09	0.02	0.00	0.03	0.00	0.00	0.66	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.12	0.09	0.00	0.00	0.00	0.70	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.09	0.02	0.20	0.00	0.00	0.67	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.02	0.06	0.02	0.26	0.00	0.00	0.62	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.02	0.04	0.06	0.17	0.04	0.00	0.63	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.02	0.01	0.00	0.10	0.00	0.00	0.38	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.20	0.00	0.04	0.07	0.01	0.00	0.65	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.08	0.06	0.13	0.01	0.01	0.69	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.04	0.54	0.08	0.00	0.00	0.33	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.03	0.00	0.46	0.00	0.01	0.48	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.05	0.02	0.31	0.15	0.08	0.00	0.37	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.04	0.00	0.86	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.03	0.00	0.00	0.95	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.09	0.22	0.02	0.00	0.65	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.27	0.14	0.00	0.00	0.00	0.55	0.00	0.00

Figura A.48: Matriz de Confusão do experimento com InceptionV3.

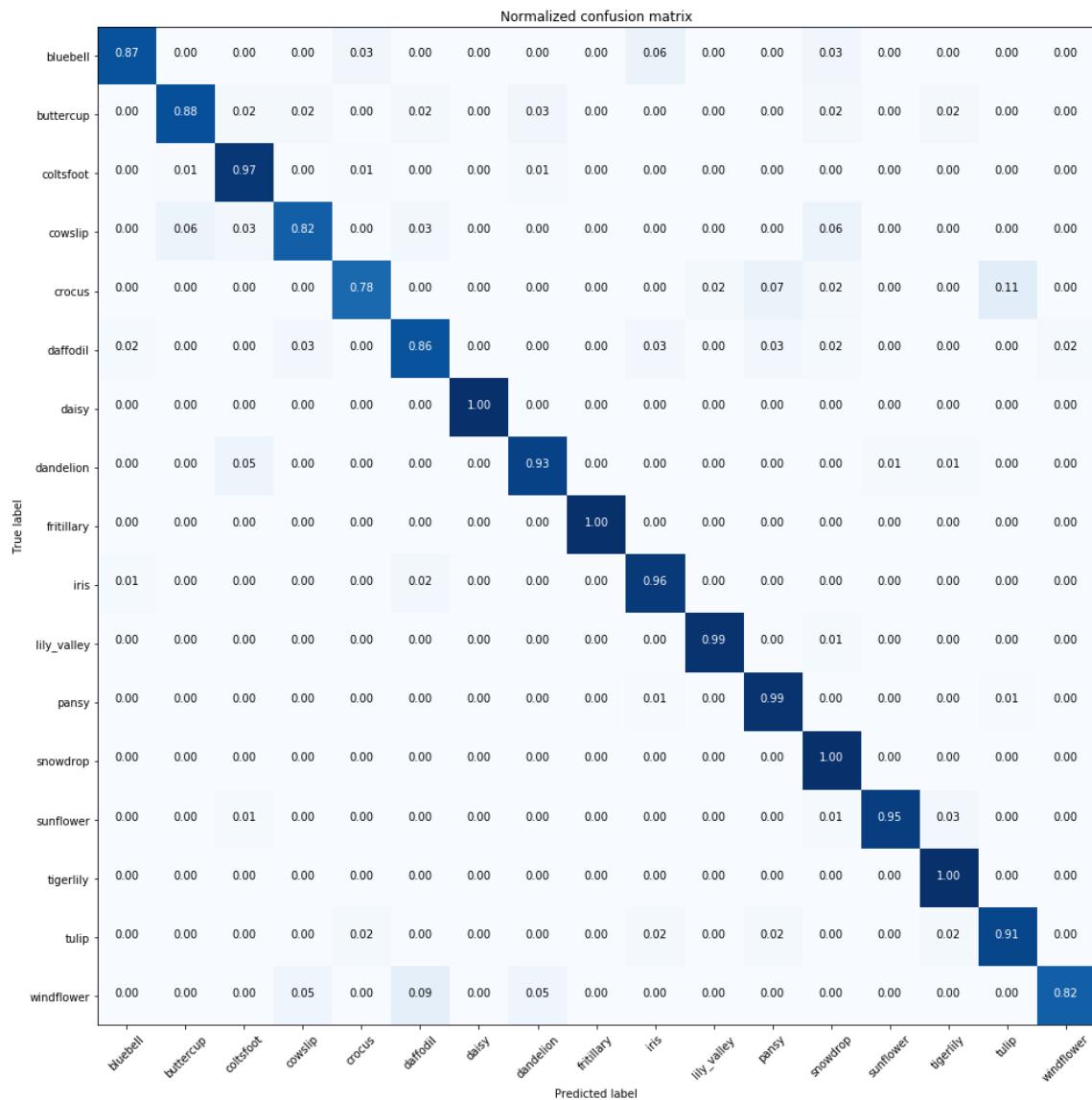


Figura A.49: Matriz de Confusão do experimento com ResNet50.

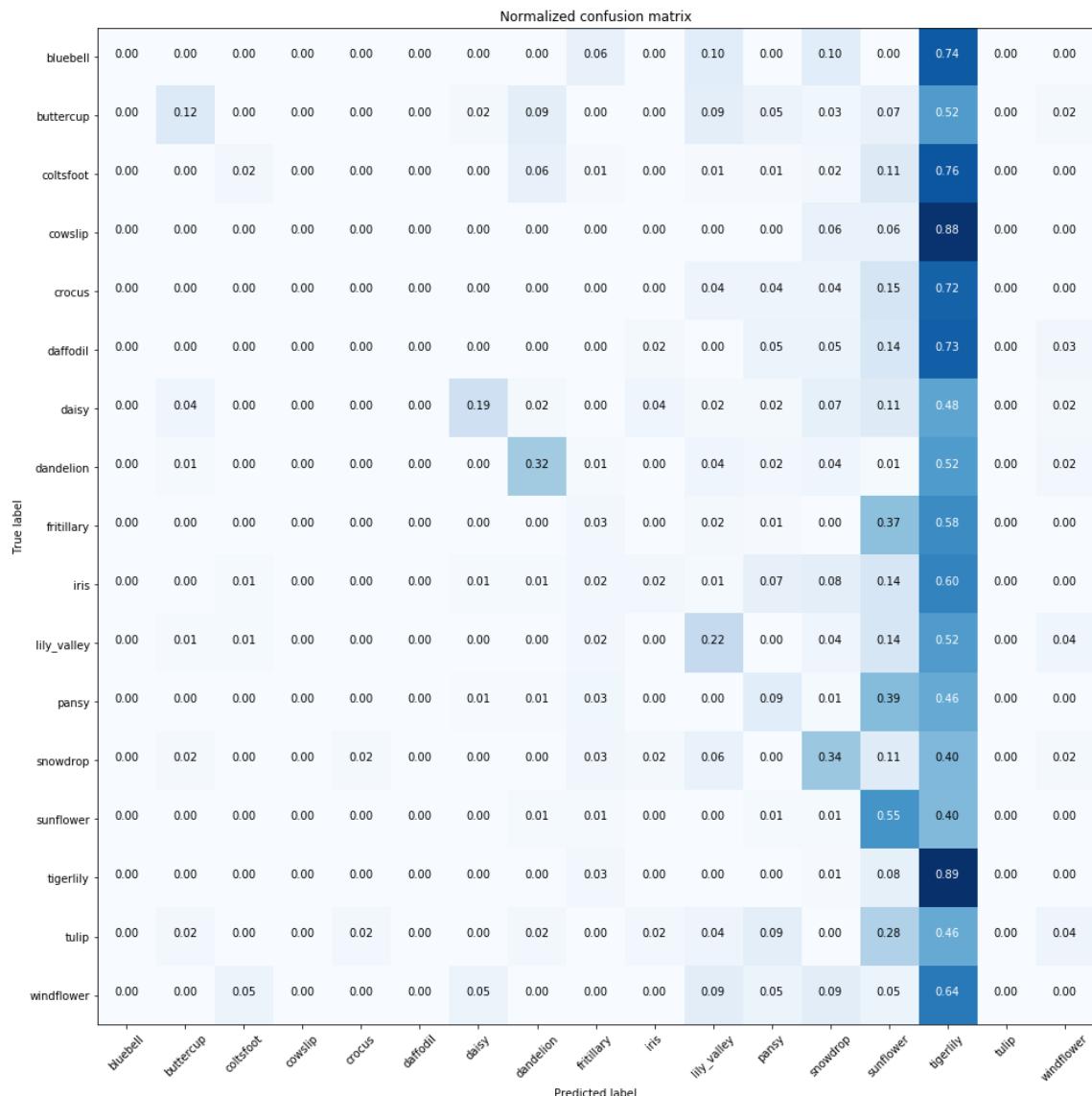


Figura A.50: Matriz de Confusão do experimento com NASNet.

Categoría	VGG16	VGG19	InceptionV3	ResNet50	NASNet
<i>Bluebell</i>	0.96	0.87	0.00	0.93	0.00
<i>Buttercup</i>	0.79	0.83	0.00	0.94	0.54
<i>Coltsfoot</i>	0.92	0.87	1.00	0.95	0.50
<i>Cowslip</i>	0.66	0.66	0.00	0.87	0.00
<i>Crocus</i>	0.88	0.94	0.00	0.92	0.00
<i>Daffodil</i>	0.80	0.78	0.00	0.90	0.00
<i>Daisy</i>	0.98	0.98	0.00	1.00	0.71
<i>Dandelion</i>	0.90	0.93	0.48	0.94	0.58
<i>Fritillary</i>	0.95	0.94	0.20	1.00	0.14
<i>Iris</i>	0.90	0.83	0.08	0.93	0.29
<i>Lily Valley</i>	0.95	0.94	0.54	0.99	0.42
<i>Pansy</i>	0.92	0.95	0.46	0.96	0.36
<i>Snowdrop</i>	0.83	0.85	0.08	0.89	0.37
<i>Sunflower</i>	0.97	0.91	0.04	0.99	0.17
<i>Tigerlily</i>	0.88	0.85	0.95	0.96	0.14
<i>Tulip</i>	0.82	0.82	0.00	0.88	0.00
<i>Windflower</i>	0.87	0.93	0.00	0.95	0.00
Média	0.90	0.89	0.25	0.95	0.29

Tabela A.22: Resultados de *Precision*

Categoría	VGG16	VGG19	InceptionV3	ResNet50	NASNet
<i>Bluebell</i>	0.84	0.65	0.00	0.87	0.00
<i>Buttercup</i>	0.76	0.86	0.00	0.88	0.12
<i>Coltsfoot</i>	0.94	0.94	0.07	0.97	0.02
<i>Cowslip</i>	0.58	0.58	0.00	0.82	0.00
<i>Crocus</i>	0.65	0.70	0.00	0.78	0.00
<i>Daffodil</i>	0.74	0.74	0.00	0.86	0.00
<i>Daisy</i>	0.96	0.96	0.00	1.00	0.19
<i>Dandelion</i>	0.89	0.83	0.48	0.93	0.32
<i>Fritillary</i>	0.98	0.96	0.20	1.00	0.03
<i>Iris</i>	0.89	0.89	0.08	0.96	0.02
<i>Lily Valley</i>	0.99	0.99	0.54	0.99	0.22
<i>Pansy</i>	0.98	0.97	0.46	0.99	0.09
<i>Snowdrop</i>	0.98	0.94	0.08	1.00	0.34
<i>Sunflower</i>	0.92	0.85	0.04	0.95	0.55
<i>Tigerlily</i>	0.99	1.00	0.95	1.00	0.89
<i>Tulip</i>	0.72	0.70	0.00	0.91	0.00
<i>Windflower</i>	0.59	0.59	0.00	0.82	0.00
Média	0.90	0.88	0.25	0.95	0.20

Tabela A.23: Resultados de *Recall*

Categoría	VGG16	VGG19	InceptionV3	ResNet50	NASNet
<i>Bluebell</i>	0.90	0.74	0.00	0.90	0.00
<i>Buttercup</i>	0.77	0.85	0.00	0.91	0.20
<i>Coltsfoot</i>	0.93	0.90	0.13	0.96	0.04
<i>Cowslip</i>	0.61	0.61	0.00	0.84	0.00
<i>Crocus</i>	0.75	0.80	0.00	0.85	0.00
<i>Daffodil</i>	0.77	0.76	0.00	0.88	0.00
<i>Daisy</i>	0.97	0.97	0.00	1.00	0.29
<i>Dandelion</i>	0.89	0.88	0.49	0.93	0.41
<i>Fritillary</i>	0.97	0.95	0.28	1.00	0.04
<i>Iris</i>	0.90	0.86	0.11	0.95	0.04
<i>Lily Valley</i>	0.97	0.97	0.48	0.99	0.29
<i>Pansy</i>	0.95	0.96	0.41	0.97	0.15
<i>Snowdrop</i>	0.90	0.89	0.13	0.94	0.35
<i>Sunflower</i>	0.94	0.88	0.08	0.97	0.26
<i>Tigerlily</i>	0.93	0.92	0.25	0.98	0.24
<i>Tulip</i>	0.77	0.75	0.00	0.89	0.00
<i>Windflower</i>	0.70	0.72	0.00	0.88	0.00
Média	0.89	0.88	0.19	0.95	

Tabela A.24: Resultados de *F1-score*

# Bibliografia

- [1] Análise em google trends do termo de pesquisa “*deep learning*”. Acessado em <https://trends.google.com.br> em Novembro de 2018.
- [2] Imagenet. Acessado em <http://image-net.org/> em Novembro de 2018.
- [3] Princeton university, about wordnet. Acessado em <https://wordnet.princeton.edu/> em Novembro de 2018.
- [4] ACADEMY, D. S. Deep learning book. Acessado em <https://deeplearningbook.com.br/> em Novembro de 2018.
- [5] BATISTA, A., AND MACIEL, T. Big data image classifier. <https://github.com/allanbatista/bigdata-image-classifier.git>. Repositório no GitHub com códigos desenvolvidos e resultados obtidos.
- [6] BERG, T., HAYS, J., AND MACIEL, W. Acessado em [https://github.com/wmaciel/flowerSpotter/tree/master/Flickr\\_code](https://github.com/wmaciel/flowerSpotter/tree/master/Flickr_code) em Setembro de 2018.
- [7] BERNARD WIDROW, M. E. H. Adaptive switching circuits. *Proc IRE WES-CON Conf* (1960), 96–104.
- [8] DEMUTH, H. B., BEALE, M. H., AND INC, M. *Neural Network Toolbox User’s Guide*. Mathworks, Incorporated, 1992.
- [9] DENG, L., AND YU, D. *Deep Learning: Methods and Applications*. NOW Publishers, May 2014.

- [10] GOODFELLOW, I., Bengio, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [11] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.
- [12] GRAUPE, D. *Principles of Artificial Neural Networks*, 2 ed. World Scientific Publishing, 2007.
- [13] GUPTA, V. Image classification using convolutional neural networks in keras. Acessado em <https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/> em Novembro de 2018.
- [14] HAN, J., PEI, J., AND KAMBER, M. *Data mining: concepts and techniques*. 2011.
- [15] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).
- [16] HE, K., ZHANG, X., REN, S., AND SUN, J. Identity mappings in deep residual networks. *CoRR abs/1603.05027* (2016).
- [17] HINTON, G., OSINDERO, S., AND TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 7 (2006), 1527–1554.
- [18] HOPFIELD, J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America* 78, 8 (1982), 2554–2558.

- [19] IMAGENET. Imagenet large scale visual recognition challenge. Acessado em <http://image-net.org/challenges/LSVRC> em Novembro de 2018.
- [20] KINSEY, M. *Machine Learning for Beginners*. Publicado independentemente, 2018.
- [21] MAHAPATRA, S. Why deep learning over traditional machine learning? Acessado em <https://towardsdatascience.com/> em Novembro de 2018.
- [22] McCULLOCH, W. S., AND PITTS, W. A logical calculus of the ideas imminent in nervous activity. *Bulletin Mathematical Biophysics* 5 (1943), 115–133.
- [23] NIELSEN, M. *Neural Networks and Deep Learning*. Determination Press, 2015.
- [24] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* (1958), 65–386.
- [25] RUMELHART, D., HINTON, G., AND WILLIAMS, R. Learning representations by back-propagating errors. *Nature* 323 (1986), 533–536.
- [26] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
- [27] RUSSEL, S., AND NOVIG, P. *Inteligência Artificial*. Elsevier, 2013.
- [28] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.
- [29] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).
- [30] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

- [31] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.
- [32] SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S. E., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. Going deeper with convolutions. *CoRR abs/1409.4842* (2014).
- [33] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the inception architecture for computer vision. *CoRR abs/1512.00567* (2015).
- [34] TORREY, L., AND SHAVLIK, J. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends*, E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez, Eds. Information Science Reference, 2009, pp. 242–264.
- [35] UNIVERSITY OF CALIFORNIA SAN DIEGO. Machine learning with big data. <https://www.coursera.org/learn/big-data-machine-learning/>. Aula online, cursada em 2018.
- [36] VEEN, F. V. The neural network zoo. Asimov Institute, 2016. Acessado em <http://www.asimovinstitute.org/neural-network-zoo/> em Novembro de 2018.
- [37] YADAV, N., KUMAR, M., AND YADA, A. *An Introduction to Neural Network Methods for Differential Equations*. Springer Netherlands, 2015, ch. History of Neural Networks.
- [38] ZOPH, B., VASUDEVAN, V., SHLENS, J., AND LE, Q. V. Learning transferable architectures for scalable image recognition. *CoRR abs/1707.07012* (2017).