Classificação de Tipos de Pokémons via Deep Learning

ANDREZA A. SANTOS, DANIELA PALUMBO, GUILHERME FURLAN, LUCAS CUNHA E THAMIRIS COELHO *

*Ciência da Computação - Graduação (UNICAMP)
E-mail: a164213@dac.unicamp.br, d166301@dac.unicamp.br,
g160160@dac.unicamp.br, l172655@dac.unicamp.br,
t187506@dac.unicamp.br

***Resumo* – Este trabalho propõe uma solução para o problema de classificação de tipos de Pokémons. Para realizar a classificação uma base de dados com 9870 imagens foi gerada a partir de imagens do jogo de cartas do Pokémon. Um dos problemas enfrentados neste trabalho além da obtenção e organização dos dados foi o desbalanceamento de Pokémons por tipo. Para realizar a classificação foram testadas as três redes com melhores resultados na classificação da *imagenet*, são elas: Inception-ResNet V2, NASNet Large e Xception. A fim de obter melhores resultados foram aplicadas técnicas de aumentação de dados e *ensemble*, combinando as predições de modelos distintos. O resultado mais promissor foi obtido a partir do *ensemble* da Xception com a Inception-ResNet V2 que foram treinadas com 80% dos dados disponíveis aplicando diversas técnicas de aumentação de dados aleatoriamente. O resultado obtido no conjunto de teste foi de 44,3% de acurácia.**

***Palavras-chave* – Classificação de Pokémons, Deep Learning, Redes Neurais**

I. INTRODUÇÃO

Este trabalho foi realizado utilizando os conceitos de aprendizado de máquina vistos na disciplina de Inteligência Artificial. O problema foi elaborado pelos membros da equipe e consiste em classificar os tipos de Pokémons a partir de imagens de cartas do jogo utilizando redes neurais profundas.

II. TRABALHO PROPOSTO

Para participar de uma batalha Pokémon é sempre importante que o jogador reconheça o tipo do Pokémon de seu adversário de modo a escolher o melhor Pokémon para enfrentá-lo. Com o objetivo de ajudá-lo nessa tarefa, este trabalho apresenta uma solução baseada em classificação de imagens através de deep learning.

III. MATERIAIS E MÉTODOS

A. Base de Dados

A base de dados para o problema é composta por imagens de Pokémons nos formatos PNG (Portable Network Graphics) e JPEG (Joint Photographic Experts Group), obtidas através do Kaggle [1] e da API desenvolvida pelo Pokémon TCG Developers [2].

No total são 9870 imagens no conjunto de treinamento, elas são coloridas de tamanho 300×300 pixels e apresentam tanto fundos simples quanto complexos. A Figura 2 mostra exemplos dessa base de dados. A base de dados é dividida em 9 tipos de Pokémons e apresenta Pokémons de todas as gerações. A quantidade de imagens para cada tipo de Pokémon é bastante variada e o gráfico da Figura 1 mostra a distribuição de imagens coletadas para cada tipo de Pokémon no conjunto de treinamento.

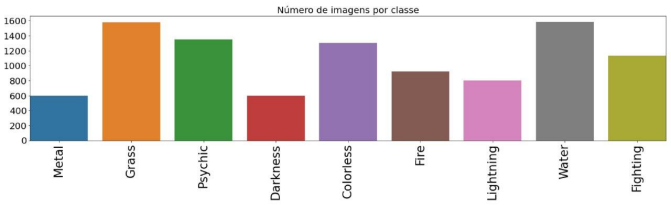

Figura 1: Distribuição de imagens por classe no conjunto de treinamento

Para o conjunto de teste, temos um total de 330 imagens, sendo que essas imagens possuem a característica de apresentar o Pokémon com o fundo completamente preto ou branco.

Pela dificuldade de encontrar uma base de dados de Pokémons pronta com imagens de qualidade razoável e completa, o trabalho de montar a base final para utilizar nesse trabalho passou por muitos desafios.

A primeira tentativa foi utilizar a base presente no Kaggle [1], porém essa base tem o problema de ser limitada em quantidade de imagens, necessitando que mais imagens fossem coletadas. Afim de aumentar a quantidade de imagens, foi implementado um algoritmo usando Scrapy [3] para coletar imagens de Pokémons do site do Pokémon Database [4], porém as imagens coletadas eram muito pequenas e de pouca qualidade. Como alternativa, foi encontrada a API desenvolvida pelo Pokémon TCG Developers [2], de onde foram baixadas todas as imagens das cartas de Pokémons através de um script.

Algumas imagens de cartas possuem mais de um tipo, sendo um principal e um secundário. Para facilitar a separação dos Pokémons por tipos, foi desconsiderados os tipos secundários. Nas cartas também existem exemplos do mesmo Pokémon com tipos diferentes e, para resolver isso, foi realizada uma votação entre os tipos das cartas de cada Pokémon e atribuído a todas as cartas daquele Pokémon o tipo mais frequente.

As imagens das cartas passaram por um pós-processamento

Figura 2: Exemplos de imagens de cada classe presentes na base de dados

que realiza o corte da carta na região da imagem, eliminando os textos presentes nas cartas, e o redimensionamento para 300×300 pixels. As imagens do conjunto de treino e de teste foram separadas em pastas por classe.

B. Implementação

Para realizar esse trabalho, foi escolhida o *framework* Tensorflow versão 2.2 [5]. Os principais motivos dessa escolha foram: (1) A facilidade de utilização em conjunto com o *framework* Keras [6]; (2) é um *framework* otimizado para rodar em GPU; (3) possui muita documentação e tutoriais acessíveis.

Data Generator e Aumentação de Dados

Como a base é composta por muitas imagens de tamanho 300×300 pixels seria impraticável carregar a base de dados inteiramente na memória, por esse motivo foi necessário o uso da classe *Data Generator* do *Tensorflow*. Essa classe gera *batches* de imagens a partir de um diretório, ou seja, as imagens são carregadas *batch* a *batch* no momento do treinamento de forma aleatória.

Além de realizar a importação dos dados por *batch* a classe também disponibiliza uma série de técnicas de aumentação de dados que é realizada em tempo real (enquanto treina o modelo). Aplicar aumentação de dados no conjunto de treinamento é interessante, porque aumenta a variabilidade do conjunto de dados. Por esse motivo aplicamos as seguintes técnicas de aumentação ao nosso conjunto de dados: rotação, zoom, espelhamento horizontal, cisalhamento, deslocamento vertical e horizontal.

Para o treinamento das redes o conjunto de treino foi repartido na proporção 80% para treino e 20% para validação. As técnicas acima citadas foram aplicadas apenas a esses conjuntos. O conjunto de teste não sofreu aumentação.

Early Stop

O *early stop* é um *callback* disponível no Tensorflow para treinamento de redes neurais, esse *callback* monitora uma métrica, podendo ser acurácia ou erro, o objetivo é que a rede finalize o treinamento caso não esteja mais melhorando seus resultados após N épocas, ou seja, a rede neural parou de aprender.

Neste trabalho o *early stop* foi utilizado monitorando o valor do erro no conjunto de validação, de modo que o treinamento fosse parado após 15 épocas sem diminuição do valor do erro. O erro na validação foi utilizado para monitoramento a fim de evitar *overfitting*.

C. Arquiteturas

As arquiteturas escolhidas foram selecionadas dentre as arquiteturas existentes utilizadas para classificação no *framework* Keras [6]. O critério de escolha foi selecionar as três arquiteturas com melhor acurácia na base de dados ImageNet [7].

Inception-ResNet V2 [8]

Inception-ResNet V2 é a versão residual da arquitetura Inception [9]. Ser Residual significa que essa arquitetura utiliza conexões residuais, ou seja, funcionam transmitindo a saída de uma camada tanto para a camada imediatamente após quanto também o propaga para alguma camada a frente no fluxo da rede. Nas versões residuais das redes Inception, são utilizados blocos de Inception mais baratos que o Inception original. Cada bloco de Inception é seguido por uma camada de expansão de filtro (1×1 convolução sem ativação), que é usada para aumentar a dimensionalidade do banco de filtros antes da adição de modo a ficar com a profundidade da entrada. Isso é necessário para compensar a redução de dimensionalidade induzida pelo bloco de iniciação.

Outra diferença técnica entre as variantes residuais e não residuais da Inception é que, no caso da Inception-ResNet, é usada a normalização de *batch* apenas no topo das camadas tradicionais, mas não no topo das somatórias. Isso é feito para limitar o consumo de memória, uma vez que foi identificado que camadas de ativação grandes consomem uma quantidade desproporcional de memória.

NASNet Large [10]

Inspirada pelo *Neural Architecture Search (NAS) framework [11]*, que utiliza um método de busca por aprendizado por reforço para otimizar as configurações de arquitetura, a NASNet é uma arquitetura criada aplicando o NAS em cima da base de dados CIFAR-10 [12]. Para tornar possível a transferência da arquitetura aprendia para a ImageNet [7],

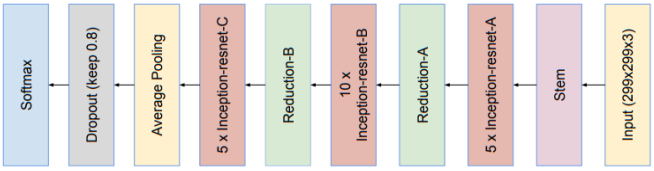

Figura 3: Arquitetura da Inception-ResNet V2[8]

foi projetado um espaço de busca chamado "Espaço de busca da NASNet" para que a complexidade da arquitetura fosse independente da profundidade da rede e do tamanho das imagens de entrada. Todas as redes neurais dentro desse espaço de busca são compostas por camadas convolucionais com estrutura idêntica, porém com pesos diferentes. Assim a busca pela melhor arquitetura passa a se tornar busca pela melhor estrutura dessas camadas de convolução. A arquitetura base da NASNet Large é mostrada na Figura 4.

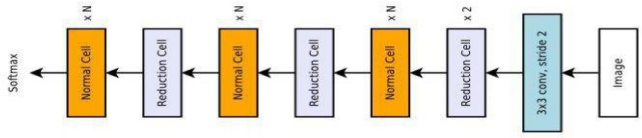

Figura 4: Arquitetura da NASNet Large[10]

Xception [13]

Criada por François Chollet, a Xception [13] possui sua arquitetura inspirada nos módulos Inception, introduzidos por Szegedy [9], onde esses módulos foram substituídos por convoluções separáveis em profundidade (convoluções separáveis).

Para a elaboração dessa arquitetura, seu autor deixou a hipótese utilizada pela Inception [9] mais forte, assumindo que o mapeamento de correlações entre canais e correlações espaciais nos mapas de características de redes neurais convolucionais pode ser completamente desacoplado.

A arquitetura da Xception, mostrada na Figura 5, possui 36 camadas convolucionais, formando a base de extração de recursos da rede. As 36 camadas convolucionais são estruturadas em 14 módulos, todos com conexões residuais lineares em torno deles, exceto o primeiro e o último módulo. Podemos resumir essa arquitetura como uma pilha linear de camadas de convolução separáveis em profundidade com conexões residuais.

D. Ensemble

A técnica de ensemble consiste em classificar um dado fazendo uma votação entre as predições de um conjunto de classificadores. Um conjunto de classificadores é, frequentemente, mais preciso que um único classificador, gerando melhores resultados [14].

O método de ensemble utilizado foi *bagging* que consiste em treinar diversos modelos paralelamente e após os treinamentos realizar uma combinação dos resultados individuais a fim de obter um resultado melhor.

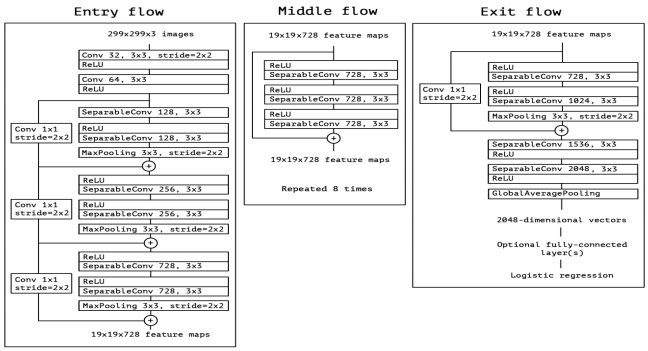

Figura 5: Arquitetura da Xception [13]

Neste trabalho as redes descritas anteriormente foram treinadas e para a predição final foi realizada uma votação ponderada entre os dois melhores modelos, dando 10% a mais de peso para o que atingiu melhores resultados no conjunto de validação. A predição final é dada por uma votação entre a predição dos dois melhores modelos.

IV. Resultados e Discussão

A. Experimentos Iniciais

Devido a quantidade limitada de amostras para realizar o treinamento, foi considerado realizar *transfer learning* a partir dos pesos pré treinados na ImageNet [7]. Foi realizado dois treinamentos: um partindo dos pesos na ImageNet e outro treinando todo o modelo. Como os resultados obtidos com *transfer learning* foram piores aos obtidos treinando o modelo por completo, neste relatório discutiremos apenas os resultados sem *transfer learning* afim de tentar respeitar o limite de páginas.

A seguir realizamos dois treinamento da rede Xception: um utilizando aumentação de dados e outro sem aumentação afim de verificar se a utilização dessa técnica é benéfica ou não. A Figura 6 mostra os resultados obtidos sem aplicar aumentação e a Figura 7 mostra os resultados obtidos aplicando aumentação. Podemos ver que no caso em que ela é utilizada, os resultados no conjunto de validação foram melhores, portanto foi decidido que a aumentação seria utilizada nos demais treinamentos das redes.

Outra decisão que precisou ser tomada foi na questão do modo de balancear os dados. Como apontado pelo gráfico 1, temos algumas classes que chegam a estar $3\times$ menor que outras e isso pode dificultar o treinamento dos modelos.

Primeiramente tentamos buscar mais imagens para complementar as classes menores, porém sem sucesso. Duas estratégias foram, então, testadas: (1) balancear as imagens cortando exemplos das classes com muita imagem de modo a aproximar a quantidade de exemplos por classe; (2) utilizar pesos proporcionais às quantidades para cada classe durante o treinamento dos modelos. A estratégia (1) apresentou resultados inferiores ao da estratégia (2). Acreditamos que o motivo desse resultado foi o fato de que cortar dados é algo bem prejudicial quando tentamos treinar modelos que possuem

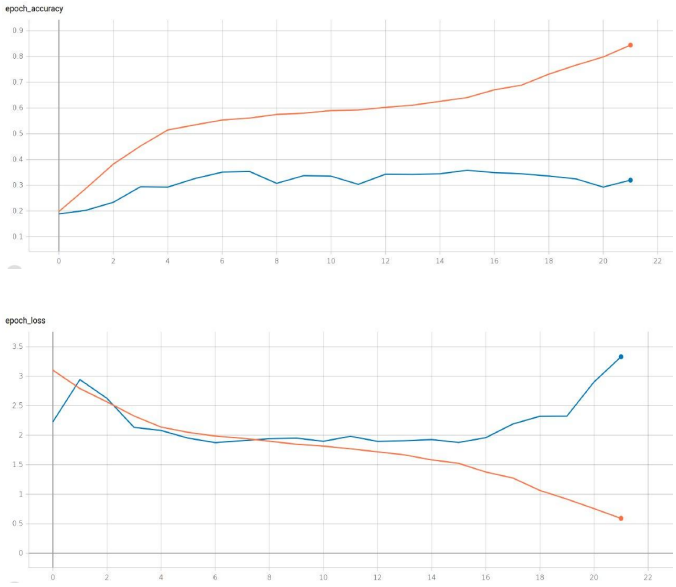

Figura 6: Gráfico da acurácia por época e da *loss* por época no treinamento da Xception [13] com tamanho de *batch* igual a 8 e **sem** aumentação de dados. Em laranja temos os valores para o conjunto de treino e em azul os valores para o conjunto de validação.

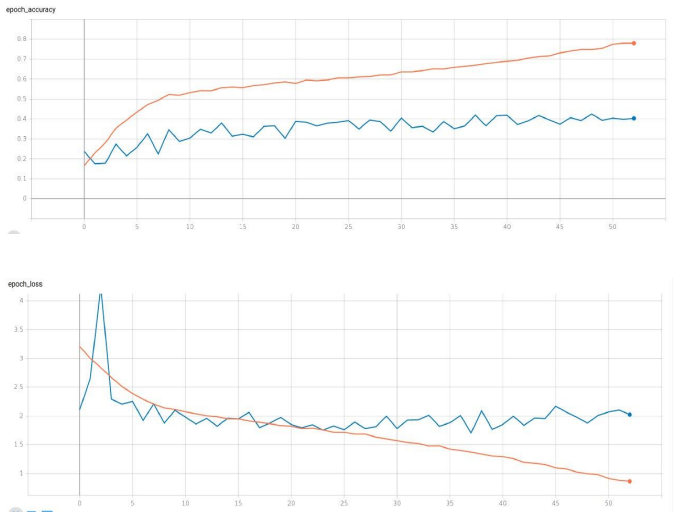

Figura 7: Gráfico da acurácia por época e da *loss* por época no treinamento da Xception [13] com tamanho de *batch* igual a 8 e **com** aumentação de dados. Em laranja temos os valores para o conjunto de treino e em azul os valores para o conjunto de validação.

milhões de parâmetros, uma vez que nossa quantidade de dados original já é muito limitada. Por esse motivo, eliminar imagens não resulta em resultados melhores por mais que as classes fiquem balanceadas. Dessa forma, a estratégia (2) foi utilizada nos demais treinamentos dos modelos.

B. Experimentos Exploratórios

Os experimentos a seguir foram executados nas GPUs presentes no laboratório *REasoning for COmplex Data (RECOD)*, devido a necessidade de mais memória para realizar experimentos com diferentes tamanhos de *batch*.

Os primeiros experimentos executados foram com *batch* igual a 8. As Figuras 8 e 9 mostram os gráficos do treinamento para a Inception-ResNet v2 e NASNet Large, respectivamente.

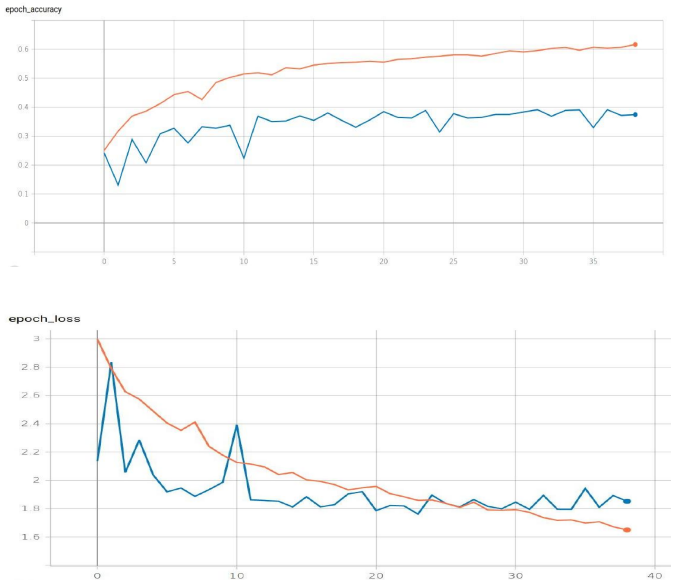

Figura 8: Gráfico da acurácia por época e da *loss* por época no treinamento da Inception-ResNet v2 [8] com tamanho de *batch* igual a 8. Em laranja temos os valores para o conjunto de treino e em azul os valores para o conjunto de validação.

Podemos ver que para todos os gráficos mostrados nas Figuras 7 e 8, após um certo ponto a rede começa a dar indícios de *overfitting*, quando os valores de *loss* e acurácia continuam melhorando no conjunto de treino, mas pioram no conjunto de validação. Devido a configuração do *early stop* para interromper o treinamento após 15 épocas consecutivas sem melhora no valor de *loss* no conjunto de validação, foi necessário fazer a seleção dos pesos da melhor época. Isso foi feito utilizando o menor valor de *loss* no conjunto de validação.

No caso da NASNet Large [10] mostrado na Figura 9, podemos ver que a rede não conseguiu obter um resultado utilizável na tarefa proposta nesse trabalho. Podemos supor que isso ocorreu devido a natureza da rede e da nossa base de dados: uma vez que essa rede foi criada a partir da busca otimizada de sua configuração de arquitetura em cima da base de dados CIFAR-10 [12] e submetida a um *transfer learning* para obter bons resultados na ImageNet [7], acreditamos que realizar o seu treinamento completo em nossa base da dados não funcionou, pois temos uma base de dados muito limitada em quantidade de amostras e muito diferente das bases de

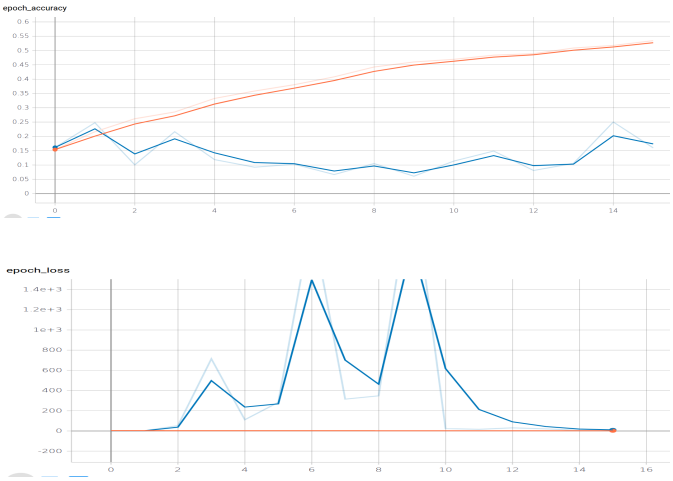

Figura 9: Gráfico da acurácia por época e da *loss* por época no treinamento da NASNet Large [10] com tamanho de *batch* igual a 8. Em laranja temos os valores para o conjunto de treino e em azul os valores para o conjunto de validação.

dados em que ela foi especialmente configurada. Vale lembrar que realizar o *transfer learning* a partir do pesos obtidos na ImageNet não obteve bons resultados para nenhuma rede testada neste trabalho. A Tabela I mostra os valores de *loss* e acurácia no conjunto de validação para o treinamento das três arquiteturas utilizando *batch* igual a 8.

Arquitetura	***Loss***	**Acurácia**
NASNet Large	2,045	0,1599
Inception-ResNet V2	1,762	0,3888
Xception	1,706	0,4193

Tabela I: Valores de *loss* e acurácia das três arquiteturas. Esses valores dizem respeito a época selecionada a partir do critério anteriormente apresentado.

Foram realizados experimentos com as três arquiteturas variando o tamanho de *batch* para 16 e 32, porém esses dois valores de *batch* resultaram em piora quando comparados aos resultados obtidos com *batch* igual a 8.

A partir da seleção dos pesos da melhor época de acordo com o critério apresentado anteriormente, realizamos a avaliação dos modelos da Inception-ResNet V2 e da Xception utilizando nosso conjunto de teste. O modelo da NASNet Large não foi avaliado após analisarmos pelos seus gráficos na Figura 9 que ele não conseguiu de fato realizar um aprendizado em cima da base de dados.

Além disso, como os gráficos [7, 8] do treinamento dessas duas redes ficaram muito semelhantes assim como seus melhores valores de *loss* no conjunto de validação mostrados na Tabela I, foi decidido verificar como seria o resultado da combinação das saídas dessas duas redes, utilizando *ensemble*. Os resultados estão mostrados na Tabela II.

A Figura 10 mostra exemplos de predições bem sucedidas utilizando *ensemble* entre os modelos. Podemos ver que, utilizando essa técnica, conseguimos corrigir algumas predições nos casos em que uma rede acerta a predição e a outra erra, como nos exemplos mostrados nas duas primeiras linhas da figura.

Arquitetura	**Acurácia**
Inception-ResNet V2	0,4000
Xception	0,4273
Ensemble	0,4424

Tabela II: Valores acurácia no conjunto de teste para as duas arquiteturas testadas e o *ensemble* entre elas.

Além disso, para realizar a votação dos modelos foi decidido dar 10% mais peso para a Xception, pois ela apresentou resultados superiores a Inception-ResNet V2 tanto no conjunto de validação, quanto no conjunto de teste. Podemos ver na terceira linha da Figura 10 que essa abordagem também ajudou na obtenção de melhores resultados, já que a Inception-ResNet V2 apontou o resultado com maior porcentagem de certeza, porém errou a classe, por causa do peso maior a predição final veio da Xception, mesmo com uma porcentagem de certeza um pouco menor, desse modo a predição final foi correta.

Figura 10: Exemplos de predições que foram bem sucedidas devido a utilização do *ensemble*. Na primeira coluna temos as predições da Inception-ResNet, na segunda coluna as predições para a Xception e na última coluna as predições utilizando o *ensemble* das duas redes.

A Figura 11 mostra mais resultados utilizando apenas as saídas preditas através do *ensemble*. Podemos ver que temos muitas predições corretas, mas vale apontar que muitas das predições erradas fazem sentido, como nos casos mostrados nessa figura.

Na primeira fileira, que temos o Rhydon, predito como sendo de água, provavelmente devido a sua cor azulada. Na segunda fileira temos o Beedrill, predito erroneamente como sendo do tipo metal, provavelmente por ter seus ferrões similares com características de Pokémons desse tipo. E na terceira fileira temos o Gengar predito como escuridão, provavelmente pelas cores mais escuras do Pokémon e o formato que se assemelha a outros Pokémons dessa classe.

Vale perceber também que muitos dos exemplos que a rede erra a predição a porcentagem de certeza é menor que 50%, mostrando que esses exemplos provavelmente poderiam se encaixar em mais de uma classe. É importante ressaltar que muitos Pokémons possuem mais de um tipo, sendo um principal e outro secundário e para este trabalho foram retirados os tipos secundários a fim de ter apenas um tipo para cada Pokémon.

Figura 11: Exemplos de predições obtidas através do *ensemble* da Xception e da Inception-ResNet V2. Quadrados vermelhos indicão predições erradas e verdes indicão predições corretas.

V. Divisão de Tarefas

Para a realização desse trabalho o grupo se reuniu via Google Meet e além de tomar todas as decisões juntos todo o código foi escrito e definido em conjunto. Portando as tarefas não foram divididas entre os membros, mas feitas em conjunto. Os experimentos foram executados no computador pessoal do Lucas e nas GPUs presentes no RECOD.

VI. Conclusões

O problema de classificação de tipos de Pokémons foi estudado. Uma base de dados foi criada a partir de imagens obtidas na internet [2], [4]. Criar a base de dados foi um desafio pela dificuldade em encontrar amostras para compôr a base, o que limita a quantidade de amostras por classe. Além disso a diversidade entre os tipos de Pokémon é grande, de modo que alguns Pokémons no jogo possuem mais de um tipo, sendo um primário e outro secundário. Para facilitar o problema, foi utilizado apenas o tipo primário para classificar o Pokémon. A quantidade de amostras por tipo de Pokémon também varia muito, gerando uma base de dados desbalanceada e necessitando da aplicação de técnicas de balanceamento.

Uma solução utilizando *deep learning* para o problema proposto foi testada. Três arquiteturas diferentes foram selecionadas e avaliadas, foram elas: Xception, Inception-ResNet V2 e NASNet Large. As redes Xception e Inception-ResNet V2 apresentaram resultados promissores atingindo respectivamente 41,9% e 38,9% de acurácia no conjunto de validação com a utilização de técnicas de aumentação de dados. Já a NASNet Large não apresentou bons resultados, chegando a 15,9% de acurácia no conjunto de validação, atribuímos esse resultado a limitações da arquitetura e da base de dados.

Como as duas melhores arquiteturas apresentadas atingiram resultados muito próximo, a técnica de *ensemble* foi aplicada afim de avaliar o desempenho da combinação desses modelos. Pra isso foi realizada uma votação ponderada, dando 10% a mais de peso para a Xception, já que essa atingiu resultados superiores no conjunto de validação.

É possível concluir que, para a solução final apresentada, as técnicas de aumentação de dados, balanceamento dos dados e *ensemble* dos melhores modelos das redes Xception e Inception-ResNet V2 foram fundamentais para o alcance dos resultados apresentados, chegando a uma acurácia no conjunto de teste de 44,2%.

Podemos apontar alguns fatores como possíveis limitantes dessa solução: a grande diversidade dentro de um mesmo tipo de Pokémon faz com que seja mais difícil que os modelos consigam encontrar padrões para cada classe além de que a quantidade pequena de amostras faz com que o modelo comece a sofrer *overfitting* muito rapidamente; a limitação dos Pokémons aos seus tipos primários também pode trazer dificuldades no reconhecimento de padrões de uma classe, já que na verdade o Pokémon pode não pertencer exclusivamente aquela classe.

Referências

[1] Kaggle, "One-Shot-Pokemon Images," https://www.kaggle.com/aaronyin/oneshotpokemon, 2018, [Online; accessed 25-July-2020]. 1

[2] "Pokemon TCG Developers," https://pokemontcg.io/, [Online; accessed 25-July-2020]. 1, 6

[3] D. Kouzis-Loukas, *Learning Scrapy*. Packt Publishing Ltd, 2016. 1

[4] "Pokemon Database," https://pokemondb.net/sprites, [Online; accessed 25-July-2020]. 1, 6

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/ 2

[6] "Keras," https://keras.io/getting_started/, [Online; accessed 05-August-2020]. 2

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009. 2, 3, 4

[8] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016. 2, 3, 4

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. 2, 3

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," 2017. 2, 3, 4, 5

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016. 2

[12] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html 2, 4

[13] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.195 3, 4

[14] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15. 3