Classificação multirrótulo de Pokémon através de mecanismos de *transfer learning*

George Gigilas Junior, Rodrigo Duarte de Meneses

{g216741@dac.unicamp.br, r197962@dac.unicamp.br}

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Campinas, SP, Brasil

Resumo - Neste trabalho, vamos trabalhar com a classificação multirrótulo de Pokémon. Dado um conjunto de imagens rotuladas, o modelo deverá classificar cada uma como sendo pertencentes ou não a cada um dos 18 tipos existentes no universo de Pokémon. Utilizamos, também, técnicas de *Data Augmentation* e *Transfer Learning*.

1. Introdução

Os problemas de classificação multirrótulo constituem uma ampla área de estudo dentro das pesquisas associadas a aprendizado de máquina. Em particular, para um grande número de classes com características semelhantes, a classificação de um grupo de amostras exige o projeto de redes neurais bastante complexas. Por conseguinte, o estudo desses problemas é imensamente proveitoso para o melhor entendimento das diferentes abordagens estratégicas que podem ser utilizadas no campo de aprendizado profundo.

Neste trabalho, buscamos atacar o problema de classificação de Pokémon (plural, pokémon), espécies ficcionais de monstros colecionáveis. Cada um possui *design* e características específicas, sendo divididos em um conjunto de 18 *tipos* – como fogo, água, elétrico, entre outros. Os pokémon também contam com o mecanismo de *evolução,* em que um dado pokémon passa por uma série de mudanças de status, envolvendo alterações em sua aparência e/ou acesso a uma maior variedade de ataques. Ademais, há ainda a presença de pokémon *shiny,* consistindo em uma variação mais rara em que sua coloração pode ser completamente diferente de sua versão original. Por fim, existem vários pokémon pertencentes a múltiplos tipos – comumente, uma classe principal e uma classe secundária, de forma que o pokémon pode apresentar características associadas a ambas as classes em níveis diferentes. Os pokémon são catalogados na Pokédex, com números de identificação únicos, atualmente totalizando 905 espécies.

Devido à natureza artística, o problema de classificação de pokémon em seus respectivos tipos (i.e., classes) é considerado bastante complexo (até mesmo para humanos). Os melhores resultados que encontramos atingem cerca de 46% [3] (de medida macro f1, veremos mais detalhes ao longo do relatório). Diante disso, esperamos conseguir chegar perto desse valor e entender as limitações do problema.

Para abordar esse problema, optamos por desenvolver esquemas de redes neurais convolucionais (CNN), os mais utilizados para os problemas de classificação multirrótulo e processamento de imagens, devido à sua capacidade de reconhecimento de padrões. Exploraremos, ao longo deste trabalho, alguns modelos de CNN e analisaremos comparativamente os resultados obtidos, esperando promover uma discussão produtiva acerca dos assuntos abordados. Também descreveremos as particularidades de problemas multirrótulo. Adicionalmente, estudaremos a utilização de *transfer learning*, a fim de avaliar o desempenho dos modelos utilizando redes neurais pré-treinadas, buscando entender como o problema em questão pode ser abordado aproveitando os resultados de modelos bem consolidados da literatura. Para implementarmos os modelos, utilizamos o *framework Tensorflow* e o *Keras*.

2. Análise do balanceamento das classes

Uma análise muito relevante para discussão dos modelos que apresentaremos diz respeito a como as amostras do conjunto de dados estão separadas em classes. Assim, fizemos um balanço interpretando a quantidade de amostras por classe para o *dataset* utilizado. Conforme discutido, as classes correspondem ao tipos de pokémon, cada uma com um determinado número de amostras. A quantidade de instâncias para cada classe é apresentada no histograma da Figura 1.

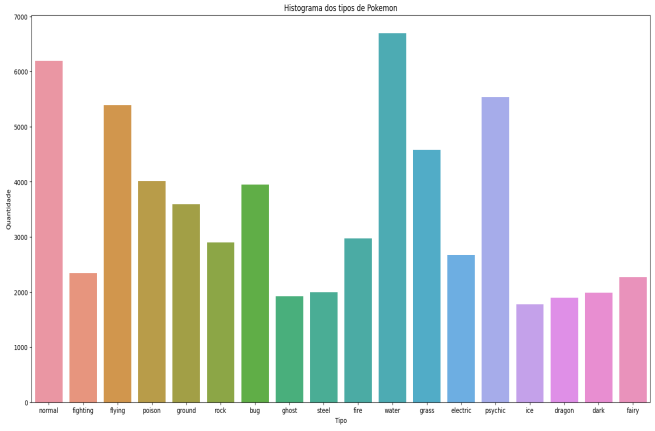

Figura 1. Histograma de tipos de Pokémon no nosso dataset.

Como podemos ver na Figura 1, o *dataset* utilizado é desbalanceado, com um número de instâncias significativamente maior para certas classes. Esse desbalanceamento deve ser levado em conta para a implementação dos modelos, já que a presença de classes majoritárias e minoritárias pode levar o modelo a um processo de aprendizagem tendencioso e a uma convergência mais lenta. Uma solução para esse problema consiste em eliminar amostras da classe majoritária a fim de estabelecer um equilíbrio entre as classes; porém, essa alternativa leva à perda de parte dos dados a serem utilizados na análise. Como será discutido posteriormente, o desbalanceamento das classes foi levado em consideração para determinar as métricas de avaliação do modelo.

3. *Data augmentation*

Uma técnica importante utilizada em alguns dos nossos modelos é a de *data augmentation*. Esse método consiste na ampliação artificial do conjunto de dados, comumente utilizado como forma de mitigar os efeitos associados ao *overfitting*. Isso pode ser feito através da adição de amostras ligeiramente modificadas do próprio conjunto de treinamento à base de dados utilizada no treinamento do modelo. Tais modificações podem ser aplicadas na iluminação, orientação e tamanho dos objetos nas imagens utilizadas. As instâncias alteradas auxiliam o modelo a generalizar melhor, tornando-o mais complacente às variações que podem vir a ser observadas durante a etapa de teste do modelo.

Entretanto, alguns detalhes particulares do nosso problema exigem que adotemos uma seleção mais cuidadosa quanto ao conjunto de mudanças aplicadas. Caso optássemos por realizar as alterações de forma imprudente, poderíamos agir de forma contraproducente aos nossos objetivos, dificultando a classificação das amostras. Isso se deve ao fato de que certos aspectos das imagens (como saturação, iluminação e demais atributos associados às cores das amostras) são sensíveis para a correta classificação dos padrões.

Por isso, limitamos nossas escolhas a um conjunto limitado de mudanças a serem aplicadas nas imagens, restringindo-se a transformações envolvendo *flip* horizontal e/ou vertical, rotações, translações e *zoom*. Avaliamos essa estratégia como uma forma de expandir consideravelmente o conjunto de dados, sem adicionar dificuldades para a classificação correta dos padrões. Isso foi feito a partir de métodos do Keras, que configuram camadas de pré-processamento a serem implementadas na arquitetura da rede durante a etapa de treinamento e realizam as transformações de forma aleatória para uma dada *seed*. Optamos por manter os mesmos parâmetros dados na documentação do TensorFlow [5] para cada uma dessas funções, já que os testes preliminares que realizamos indicaram um bom desempenho para esses conjuntos de parâmetros.

4. *Transfer learning*

Outra abordagem utilizada em nosso projeto foi a aplicação de *transfer learning*, que consiste na transferência do conhecimento utilizado para solução de um determinado problema para uma posterior aplicação em um outro problema relacionado. Essas técnicas têm se tornado cada vez mais comuns em problemas de *machine learning*, por permitirem o aproveitamento de redes neurais pré-treinadas. Assim, diminui-se consideravelmente o tempo gasto nas etapas de treinamento e no projeto da arquitetura das redes.

Entretanto, para o contexto deste projeto, é necessário avaliar quais modelos pré-treinados são mais adequados. Como os Pokémon possuem uma variedade muito significativa de *designs* (semelhantes a animais, objetos, plantas, etc.), pesquisamos por redes treinadas para classificação em classes bastante distintas e encontramos um modelo que pareceu compatível – a MobileNetV2. Essa é uma rede neural convolucional profunda (53 camadas), desenvolvida pela Google [4]. A MobileNetV2 foi projetada para aplicações de visão computacional e desenvolvida buscando uma boa performance em aparelhos celulares. Utilizamos o modelo pré-treinado com os pesos utilizados na ImageNet, que é capaz de classificar imagens em cerca de 1000 classes de objetos diferentes (como animais, flores, objetos e padrões diversos). Como podemos observar na Figura 2, a arquitetura do esquema de *transfer learning* consiste em dois blocos principais: 1. a rede neural de base (MobileNetV2) e 2. o bloco de classificação, em que são especificadas as classes para as quais as amostras serão atribuídas.

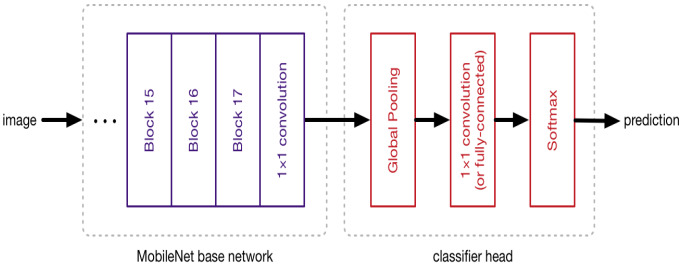

Figura 2. Arquitetura do esquema de *transfer learning*.

Em nosso projeto, optamos por manter o primeiro bloco em sua forma integral e alteramos o bloco de classificação apenas substituindo a camada *Softmax* por uma camada densa e uma camada de saída com 18 neurônios, com função de ativação sigmóide.

Também avaliamos o desempenho de outros modelos como a VGG16 e ResNet50, mas obtivemos resultados similares para todos os casos. Como a MobileNetV2 apresentou melhores resultados de forma geral, optamos por mantê-la para os resultados finais deste projeto.

5. Classificação Multirrótulo

O problema da classificação multirrótulo consiste em ter dados que podem pertencer a mais de uma classe, conforme dito anteriormente. Por isso, a função *softmax* não funciona tão bem, por expressar a probabilidade de o dado pertencer a cada classe e destacar uma classe sobre as demais. Aqui, o problema consiste em determinar se o dado pertence a cada uma das classes e, por isso, utilizamos a função sigmóide, como se tivéssemos 18 classificadores diferentes (um por classe). Tal função retorna um número próximo de 1, se o dado pertencer a determinada classe, e próximo de 0, caso contrário. Vale ressaltar que os rótulos estão representados como um vetor *multi-hot encoding*.

Além dessa peculiaridade, também buscamos uma função de *loss* e uma métrica de avaliação do modelo adequadas. Inicialmente, utilizamos a acurácia binária, que produziu resultados próximos de 90%. Esse valor era muito alto, então investigamos e percebemos que o modelo estava atribuindo números próximos de 0 para todas as classes. Com isso, descobrimos que esta métrica leva em conta cada valor do vetor *multi-hot encoding*, que possuía 16 ou 17 valores zerados.

Sendo assim, buscamos outras alternativas e descobrimos que a função *macro f1 score* era adequada para avaliação do modelo [1]. Tal métrica combina os valores de precisão e *recall*, conforme visto em aula. A única diferença é que ela consiste na média do valor da medida f1 para cada classe, atribuindo um peso igual para cada uma delas. Essa abordagem é vantajosa porque, como vimos no início deste relatório, existe um desbalanceamento de classe considerável. Quanto à função de *loss*, descobrimos a *soft f1 loss*, uma versão compatível com a medida macro f1 e que foi adaptada para ser diferenciável [2]. Isso é importante para a *backpropagation*, trazendo melhor desempenho para o treinamento do modelo. Com isso, podemos avaliar como o modelo evolui com relação à medida macro f1.

6. Tratamento dos dados

O primeiro grande desafio deste projeto foi a obtenção e modelagem dos dados, para que pudéssemos utilizá-los junto ao modelo. Para isso, encontramos um repositório que contém diversas informações sobre Pokémon: desde imagens dos modelos dos jogos, até tipos, atributos, itens, personagens, entre outros [6]. Para rotular as imagens, fizemos um *script* que extrai o número identificador de cada Pokémon do nome de cada imagem. Assim, tendo as imagens nomeadas no padrão do PokeAPI, conseguimos relacionar a imagem, o número de cada Pokémon e o(s) tipo(s) de cada Pokémon (a partir de seu número). Conseguimos reunir cerca de 43 mil imagens com seus respectivos tipos, a partir de várias fontes oficiais de Pokémon.

Além de rotular as imagens, transformamos o rótulo em um esquema *multi-hot encoding*, que foi feito nesse mesmo *script*. Ademais, separamos os dados em treinamento, validação e teste, sendo 60% dos dados de treinamento, 20% de validação e 20% de teste (escolhidos arbitrariamente, de acordo com resultados dos EFCs). Todos os dados foram embaralhados, pois a ordem numérica dos Pokémon apresenta as evoluções de forma consecutiva e, como elas tendem a preservar seus tipos, isso pode trazer um viés para os dados. Por fim, utilizamos métodos do *Keras* para padronizar o tamanho das imagens, com cada *pixel* normalizado entre 0 e 1, para melhorar o desempenho do modelo.

7. Implementação das redes

Para implementarmos as redes, fizemos quatro modelos diferentes, todos com *transfer learning*: um modelo simples, com as imagens em 128x128, e sem normalização dos *pixels* entre 0 e 1; um modelo parecido com o primeiro, mas com *Data Augmentation* e normalizando os *pixels*; um modelo adaptado da principal referência encontrada [3], com as imagens em 224x224; e uma variação deste modelo adaptado com *Data Augmentation*. Ao longo do relatório, vamos nos referir a esses modelos como primeiro modelo, segundo modelo, terceiro modelo e quarto modelo, de acordo com a ordem aqui apresentada.

Em todos os casos, o consumo de memória RAM é muito grande para carregar todas as imagens em memória. Por isso, utilizamos o *Dataset* do *Keras*, que produz um *lazy iterator* para carregar os dados por *batches*, conforme necessário. Tal abordagem funcionou muito bem, possibilitando manter o conjunto integral dos dados. Há uma pequena diferença na forma como os dois primeiros modelos importam as imagens, com relação aos outros dois, mas a ideia é a mesma e, a princípio, não deve haver diferenças.

A principal diferença entre os nossos modelos originais e os adaptados está no pré-processamento dos dados. Em nossos modelos, apenas transformamos os dados em 128x128, aplicamos *Data Augmentation* (ou

não, dependendo do modelo), e normalizamos os *pixels*. Já no modelo da referência, decodificamos os arquivos PNG e transformamos as imagens em tensores (antes de fazer parte do *Dataset*), além das imagens ficarem em 224x224, de acordo com o padrão da *MobileNet V2*. Com essa pequena mudança, houve significativas diferenças nos resultados, discutidos na próxima seção.

Quanto às redes em si, todos os modelos tinham a mesma base: utilizamos uma "camada" de extração de características, correspondente ao "*feature vector*" da rede *MobileNet V2*; uma camada densa com 1024 neurônios; e uma camada de saída, com 18 neurônios com função sigmóide. A camada de extração de características é congelada para mantermos os pesos utilizados na classificação dos dados da *ImageNet*.

Para todos os modelos, utilizamos *learning rate* de 0,00001, valor baixo e mais adequado para *transfer learning*. O otimizador escolhido foi o *Adam*, pois tivemos bons resultados com ele no EFC3, e rodamos os modelos por 60 épocas, de acordo com o modelo de referência. Assim, conseguimos comparar os resultados sob as mesmas condições. A função de *loss* e a métrica para avaliar o modelo já foram discutidas anteriormente. Os códigos utilizados, juntamente com as imagens deste relatório em um tamanho maior, se encontram no seguinte repositório. Cada código levou entre 9 e 13 horas para rodar, destacando a contribuição do *Transfer Learning* para treinar menos parâmetros.

8. Resultados

Como resultados, apresentamos as variações das funções de *loss* e da medida macro f1, que podem ser vistos nas Figuras 3, 4, 5 e 6. Com esses gráficos, podemos observar aspectos importantes. Primeiramente, vamos analisar o efeito do *Data Augmentation*, presente no segundo e no quarto modelo. Comparando as versões dos modelos com e sem esse artifício, notamos algo importante: analisando as escalas, ambas as versões com o artifício fizeram com que as métricas de validação e treinamento se aproximassem. Isso faz bastante sentido e é muito positivo, pois a técnica tem como principal objetivo a redução dos efeitos de *overfitting*, que ocorre justamente quando as métricas de validação ficam distantes das de treinamento.

Outro efeito notado é que, para os primeiros modelos, a introdução de *Data Augmentation* melhorou os resultados, já para os outros dois, os números caíram. Isso, possivelmente, se deve à normalização dos *pixels* introduzida no 2º modelo, que melhorou os números, sem nenhuma interferência do *Data Augmentation*.

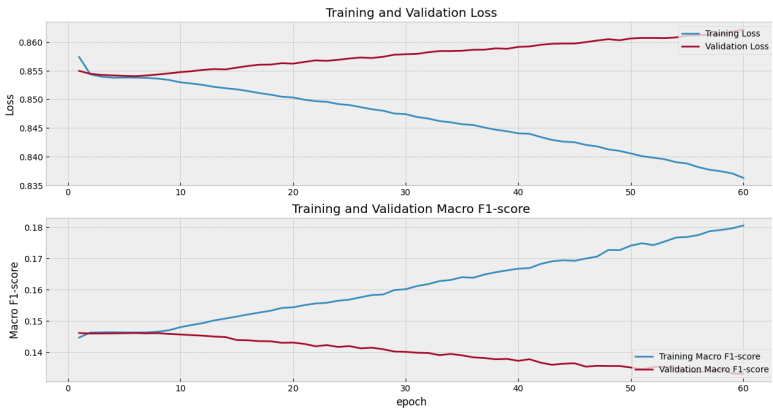

Figura 3. Função de *loss* e medida macro f1 do 1º modelo.

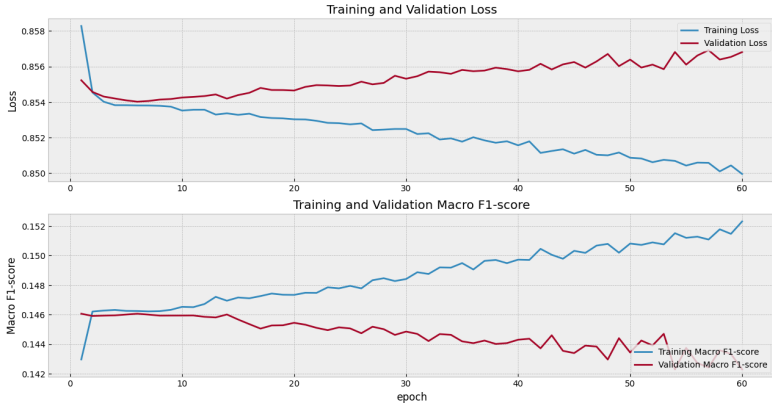

Figura 4. Função de *loss* e medida macro f1 do 2º modelo.

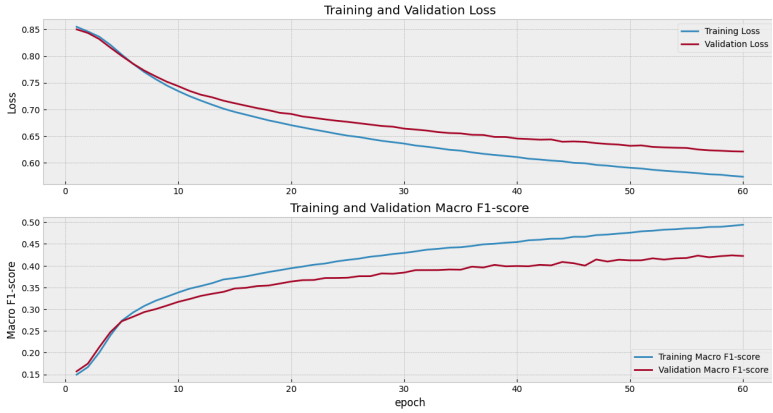

Figura 5. Função de *loss* e medida macro f1 do 3º modelo.

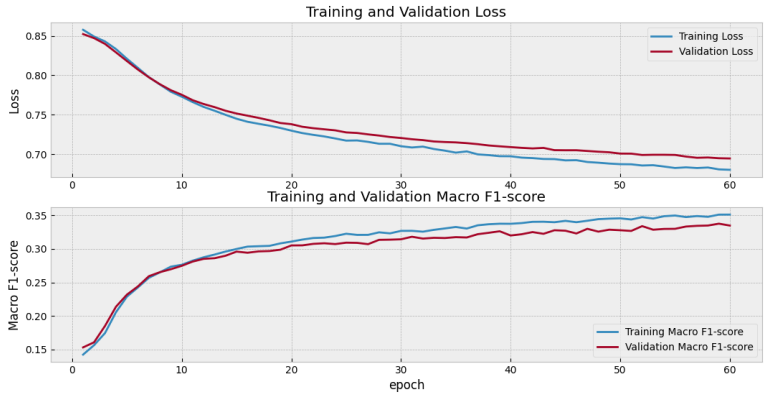

Figura 6. Função de *loss* e medida macro f1 do 4º modelo.

Outra possibilidade é que a técnica tenha adiado o sobreajuste em algumas épocas, analisando a escala,

nos primeiros modelos. Já nos últimos, não houve sobreajuste no treinamento, podendo aumentar a medida macro f1 se treinarmos com mais épocas, corroborando o que foi observado nos primeiros modelos.

Comparando, agora, a diferença nos modelos em si (primeiro com terceiro e segundo com quarto), vamos identificar o impacto da forma como os dados foram tratados. Os primeiros modelos não transformam as imagens de forma adequada e, provavelmente por isso, não foram capazes de extrair das imagens suas características principais. Notamos que a diferença é bastante grande, apesar de o modelo ser bem parecido, fortalecendo tal possibilidade. Os resultados da medida macro f1, junto aos dados de teste foram, do primeiro até o quarto modelo: 0,1330; 0,1451; 0,4271; 0,3329. Esses valores reforçam o que foi dito, tanto para a diferença nos modelos quanto para *Data Augmentation*.

Utilizando o melhor modelo (por questões de espaço no relatório), vamos mostrar 5 classificações para ilustrar como a classificação está sendo feita:

Tipo verdadeiro: water e rock
[0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0]
Tipo previsto: water
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Tipo verdadeiro: ice
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
Tipo previsto: steel, water e ice
[0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0]

Tipo verdadeiro: psychic
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
Tipo previsto: normal
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Tipo verdadeiro: normal e flying
[1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Tipo previsto: normal e flying
[1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Tipo verdadeiro: water
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
Tipo previsto: nenhum
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Ficamos bem satisfeitos com os resultados das classificações. Dentre os 5 Pokémon (incluindo versão de costas e versão *shiny*), 3 tiveram ao menos 1 tipo classificado corretamente, sendo que um teve seus 2 tipos classificados corretamente. O terceiro *Pokémon* não possui nenhuma característica que remete ao tipo psíquico (como a cor rosa), o que explica o resultado. Já para o último, o modelo não deu probabilidade maior que 50% de ser de algum tipo, o que pode acontecer.

9. Conclusões e Perspectivas Futuras

Neste relatório, conseguimos implementar um modelo que faz a classificação multirrótulo dos 18 tipos de *Pokémon*, a partir de suas imagens, utilizando conceitos de *transfer learning* e *data augmentation*. Foram implementadas 4 versões de modelos, todas utilizando como base a rede *MobileNet V2*. Nosso melhor modelo apresentou 0,4271 de medida macro f1, junto ao conjunto de teste. Considerando que são 18 tipos (e que um *Pokémon* pode pertencer a 1 ou 2 tipos), esse resultado foi bastante satisfatório.

No entanto, gostaríamos de obter resultados ainda melhores para classificação dos tipos com maior taxa de acerto. Para isso, futuramente, podemos utilizar *fine-tuning* e descongelar alguma(s) camada(s) do modelo de *transfer learning*. Recomenda-se deixar o número de camadas a serem descongeladas como hiperparâmetro a ser determinado. Assim, espera-se obter resultados melhores, com os pesos do modelo adaptados para o problema, mas o tempo de treinamento será maior, e deve ser levado em consideração.

Um outro aspecto importante são os efeitos do desbalanceamento das classes nos modelos utilizados. Ao implementar *Data Augmentation* no quarto modelo, a taxa de acerto diminuiu. Tal fenômeno pode ter sido causado pela utilização da técnica sobre um conjunto de dados desbalanceado – em nossa implementação, as imagens a serem modificadas eram escolhidas aleatoriamente do conjunto de dados, podendo amplificar os efeitos do desbalanceamento no modelo. Para trabalhos futuros é possível avaliar o desempenho de modelos cujos conjunto de dados sejam balanceados, o que pode ser feito, por exemplo, utilizando *data augmentation* de forma seletiva nas classes minoritárias.

Referências

[1] MEMARI, Issa. “Precision, Recall, Accuracy, and F1 Score for Multi-Label Classification”. *Synthesio Engineering*, 12 de Janeiro de 2021.
[2] MAIZA, Ashref. “The Unknown Benefits of using a Soft-F1 Loss in Classification Systems”. *Towards Data Science*, 4 de Dezembro de 2019.
[3] SHAHIR, Jamshaid. “Multi-Label Classification of Pokemon Types with TensorFlow”. *Towards Data Science*, 4 de Fevereiro de 2022.
[4] SANDLER, Mark; HOWARD, Andrew. “MobileNetV2: The Next Generation of On-Device Computer Vision Networks”. *Google Research*, 3 de Abril de 2018.
[5] *Tensorflow Documentation*. Acesso em 12/12/2022.
[6] *PokeAPI* 2.3.0. Acesso em 12/12/2022.