Detecção de Alimentos Através da Rede EfficientNet

Brenda Alves Moura
Departamento de Ciência Da Computação
IFC-Videira
Videira
brendamoura857@gmail.com

***Resumo*—Nesse presente artigo será demostrado a forma de detecção de objetos por meio da rede EfficientNet, utilizando um modelo criado em 2022 para auxiliar a detecção de objetos e treinamentos de rede neurais na área.**

Keywords— EfficientNet, modelo, rede neural.

I. INTRODUÇÃO

Com o surgimento da computação em nuvem e o aumento significativo da capacidade computacional de big data com a ajuda do processamento de hardware da unidade de processamento gráfico (GPU), o reconhecimento de imagens atingiu um nível de desempenho que pode justificar mais adequadamente sua sobrecarga econômica e permitir seu uso mais amplo. Gigantes de TI como Google e Amazon viram isso como uma oportunidade para expandir seu catálogo de produtos e oferecer recursos de visão computacional como um serviço.

Entretanto, certas circunstâncias podem fazer com que os serviços de reconhecimento de imagem produzam rótulos errôneos, potencialmente embaraçosos, como foi visto com o desastre do Google Fotos de 2015, onde dois indivíduos negros foram rotulados como gorilas [1]. Conforme mostrado nos resultados do trabalho de conclusão de curso "Uso de detecção de objetos em imagens para o ensino de vocabulário estrangeiro", que utilizou o serviço de reconhecimento de imagens do Google Vision, o serviço obteve 69,5% de acertos, não sendo capaz de detectar classes específicas [2]. O serviço conseguia detectar, por exemplo, que em uma imagem existia alimento, frutas, vegetais, porém não era capaz de apontar diretamente quais tipos de alimentos, frutas e vegetais haviam na imagem, conforme mostra na Figura 1.

Figura 1. Reconhecimento de alimentos do aplicativo Bazhenov. Fonte: A autora

Diante dos expostos, o que se pretende abordar neste trabalho é o desenvolvimento de um modelo utilizando rede neural capaz de detectar alimentos para complementar os resultados obtidos pelo Google Vision no aplicativo mobile Bazhenov, construído como trabalho de conclusão de curso em 2021.

II. REVISÃO LITERÁRIA

A. Machine Learning

Machine Learning é uma área da Inteligência Artificial que fornece ao computador a habilidade de aprender uma determinada tarefa sem ser explicitamente programada, ela consegue identificar padrões através de um conjunto de dados robusto [3]. Para [4], Machine Learning é um sistema que pode modificar seu comportamento autonomamente tendo como base a sua própria experiência, com pouca interferência humana.

B. Visão computacional

Nesse projeto, a visão computacional foi utilizado sem o Deep Learning. Nessa situação, o aprendizado de máquina, para cada imagem do Dataset usado, é realizado um processo chamado extração de características (feature extraction), em que a imagem de entrada é quantificada de acordo com um algoritmo específico (feature extractor), que retorna um vetor que tem o objetivo de quantificar o conteúdo da imagem, que é usado como entrada para o modelo de Machine Learning com a quantificação da imagem (o processo de extração de recursos), que, por sua vez, é usado como entrada para o modelo de Machine Learning [5].

III. METODOLOGIA

A. Linguagem e bibliotecas

A linguagem utilizada foi a Python, e o editor foi o Google Colab. As principais bibliotecas utilizadas foram o objeto da *pathlib* e *os.path* para trabalhar com os dados. scikit-learn para o treinamento do modelo de classificação. A visualização dos dados foi feita através do Matplotlib. Essa biblioteca é útil para a visualização do resultado do processamento da imagem carregada.

B. Padronização das imagens

É importante possuir as imagens padronizadas, por isso todas as faces identificadas foram recortadas, redimensionadas e salvas em uma lista. Para o recorte, é utilizado o slice e para o redimensionamento é utilizado o resize, que padroniza as imagens com o valor da altura e da largura desejados, nesse caso, 254x254, como é exemplificado pela figura.

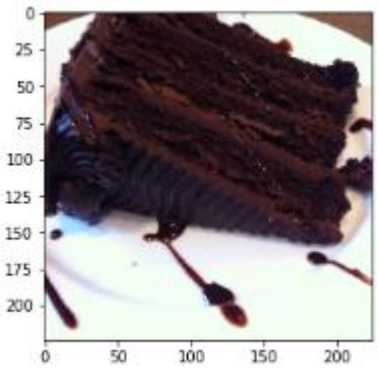

Figura 2. Redimensionamento da imagem. Fonte: A autora

C. Testagem

Para testar o modelo, foram processadas imagens de alimentos com base no *dataset* de imagens de alimentos encontrado no site *kaggle*. O *dataset* possui 101 classes, sendo que cada classe possui 1000 imagens. Apesar de cada classe conter 1000 imagens, sendo uma quantidade elevada de dados, algumas classes não possuem uma grande diversidade de imagens. A falta de diversidade de imagens pode afetar a performance do aprendizado e reconhecimento. Para contornar este problema foi utilizado o recurso do keras *ImageDataGenerator* para gerar imagens a partir das imagens já existentes. Entretanto apenas 10% de cada classe de imagens foi utilizada para não sobrecarregar a GPU do Colab.

Nesta etapa, os algoritmos são executados diversas vezes para que se perceba uma consistência no tempo de execução, visto que o hardware da máquina pode causar oscilações no resultado. Então é feita verificação da precisão dos algoritmos, onde foram utilizadas métricas populares na classificação de algoritmos de ML, como *accuracy*, *precision*, *recall* e *f1-score*.

D. Aplicação utilizando o modelo treinado

O algoritmo fica responsável por realizar a classificação entre a variedade de classes, utilizando a rede pré-treinada EfficientNet. A partir do treinamento do modelo foi possível criar um script que realiza a verificação de uma imagem para dizer quais alimentos contém nela, como é ilustrado pela Figura 3.

```
[ ]  Categories = sorted(os.listdir('/content/images'))
     Categories

[ ]  # Prediciting
     Categories[np.argmax(model.predict([img_preprocessed]))]

     'chocolate_cake'
```

Figura 3. Reconhecimento do alimento como chocolate_cake na imagem enviada para o modelo. Fonte: A autora

IV. Desenvolvimento

Para implementar o algoritmo, primeiramente foram importadas as bibliotecas *numpy*, *pandas*, Matplotlib e Path.

Após a importação, uma célula do notebook foi utilizada para armazenar o caminho para a pasta de imagens, local onde contém os arquivos que serão processados e analisados. A pasta é importada pelo elemento *Path*, que indica o caminho no qual a pasta se encontra e, em seguida, armazena o caminho em uma variável

Então é feita uma lista do *dataframe* contendo uma coluna com todos os caminhos de arquivo e, em seguida, outra coluna com todos os rótulos associados com o caminho do arquivo. O algoritmo percorre todos os diretórios utilizando a função *glob* que consegue direcionar arquivos dentro de uma pasta. Todas as imagens que contém dentro das pastas são de extensão *.jpg*.

Após armazenar os rótulos, no caso, cada caminho de arquivo que existe é identificado e, usando a função *glob*, é possível armazenar os caminhos. A *função os.path.split* é responsável por dividir a *string* ou o caminho do objeto para que se possa ter o prefixo do caminho, assim, resultando apenas no nome do arquivo em vez do diretório. Desta forma, a função retorna o nome de cada classe.

O armazenamento dos rótulos é aplicado a todos os caminhos de arquivo, e para cumprir este objetivo foi usada a função *map*. A função lambda recebe uma variável *x* que vai ser um determinado caminho e então vai aplicar a função para todos, e essa função é mapeada para a variável *filepaths*. Depois, o resultado é transformado em uma lista e então a variável *labels* armazena os rótulos retirados com essa função.

Como resultado do processo acima, obtém-se duas listas, uma com os caminhos dos arquivos e a outra com os rótulos, ambos em ordem alfabética. Logo, os dados são transformados em panda series e concatenados em um *dataframe*. E então utiliza-se o *panda.concat* para concatenar as *labels* e os caminhos no eixo 1, que significa lado a lado.

Logo após, é feita a divisão entre o conjunto de teste e o conjunto de treino. A divisão foi feita enviando 70% do dataset para o conjunto de treino e 30% para o conjunto de teste. A função *train_test_split* foi chamada para pré-pocessar o modelo construído e em seguida analisar a performance do modelo com a matriz de confusão e a classificação do *sklearn*.

Depois, os *generators* foram criados, este recurso tem finalidade de gerar mais imagens a partir das existentes. Para esta etapa, foi utilizado o *Efficientnet*. Nos geradores, o primeiro parâmetro passa os dados de treino separados. O segundo parâmetro é a *x_column*, que se trata da coluna que tem os caminhos do arquivo para as imagens e as *strings*. No terceiro parâmetro denominado *y_column*, é passado o caminho para os rótulos. No quarto parâmetro se passa o tamanho da imagem que, para melhor classificação, foi redimensionado para 224 de altura por 224 de largura. A cor usada é a *rgb*, e o modelo de classe utilizado trata-se do *categorical* porque ele é um classificador multiclasses. A semente foi fixada em 42 para poder reproduzir os resultados.

Depois, são ajustados os dados de validação que tem basicamente as mesmas configurações. A validação pega 20% dos dados de treinamento para poder validar.

Por último ajusta-se os dados de teste, que tem as configurações parecidas com as configurações dos dados anteriores, retirando apenas os parâmetros *subnet* e o *shuffle*, pois o conjunto de teste não precisa de validação nem embaralhar os dados. Na Figura 4 mostra-se o a quantidade de imagens que está sendo usada em cada conjunto de treino, validação e teste.

```
Found 30300 validated image filenames belonging to 101 classes.
Found 7575 validated image filenames belonging to 101 classes.
Found 12625 validated image filenames belonging to 101 classes.
```

Figura 4. Quantidade de imagens para treino, validação e teste. Fonte: A autora

Foi utilizado o modelo pré-treinado que vem do *keras.applications*, com os *inputs* da imagem sendo de resolução 224x224, e as imagens dispondo de 3 canais (*rgb*). O parâmetro *Include_top* foi colocado como falso, que significa que não é desejado manter a camada de classificação final do modelo original treinado. O propósito de usar um modelo pré-treinado como esse é chamado de transfer-learning. Esse modelo foi construído para ser muito bom em extrair recursos úteis das imagens, isso é conhecido como *feature extractor* porque toda a rede convolucional, toda a parte da rede que não é o topo (que é destinada à classificação) serve para extrair recursos das imagens, então estamos extraindo recursos bidimensionais das imagens, são apenas pedaços de informação que são úteis. Consequentemente, é utilizado todo esse treinamento que foi feito na *imagenet* e somente o topo é retirado, que vou usar para classificar de acordo com os meus propósitos.

O *EficientNet* foi treinado originalmente no *dataset* da *imagenet*, então os pesos utilizados foram *imagenet*, para os mesmos pesos serem mantidos.

Em seguida, utilizou-se o comando *GlobalAveragePooling2D()* para garantir que a saída do modelo será unidimensional, visto que será um vetor. O otimizador escolhido para utilizar neste caso foi o Adam, a função *loss* foi definida para *categorical_crossentropy* por oferecer melhor abordagem da nomenclatura das classes.

Depois, algumas funções foram chamadas para o algoritmo interromper a execução caso ocorra a estagnação da melhoria sem que tenha treinado todas as épocas, que totaliza em 10. Portanto, se por 3 épocas consecutivas não ocorrer melhora, o algoritmo para de executar e volta para os pesos da melhor época. Não foi possível utilizar mais épocas devido às limitações do plano gratuito do Colab.

V. RESULTADOS

A Figura 5 exibe os resultados de precisão da acurácia, treinamento e validação do modelo treinado, enquanto a Figura 6 exibe os resultados de precisão relacionados ao treinamento e validação da perda.

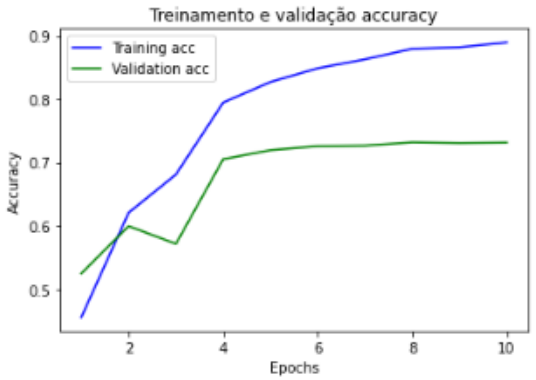

Figura 5. Precisão da acurácia do treinamento e validação. Fonte: A autora

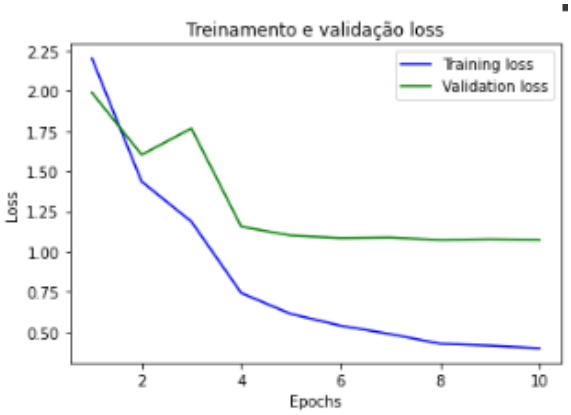

Figura 6. P recisão da perda do treinamento e validação. Fonte: A autora

O tamanho do conjunto de dados e a complexidade do modelo justificam essa tendência de desempenho observada. Com modelos com maior complexidade como o modelo utilizado modelo, há uma maior probabilidade de obter uma melhor métrica de desempenho. No entanto, ao mesmo tempo, eles são mais propensos a sobreajustar o conjunto de dados.

No relatório de classificação, são utilizadas métricas como precisão (fração de instâncias recuperadas que foram previstas corretamente), *recall* (fração de instâncias de uma classe que foram previstas como sendo daquela classe), e o *f1-score* (a média harmônica da precisão e *recall*). Os resultados podem ser vistos na Tabela 1.

	precision	**recall**	**score**	**support**
accuracy			0.73	12625
macro avg	0.74	0.73	0.73	12625
weighted avg	0.74	0.73	0.73	12625

Tabela 1.Resultados relacionados ao treinamento do modelo. Fonte: A autora

Algumas descobertas gerais são que o modelo teve um aceitável desempenho na classe majoritária (Acurácia = 73%). A precisão média do modelo foi de 74%. O modelo, observando a escala *f1-score*, teve pior desempenho nas classes *foie_gras* (42%) e *pork_chop* (46%), enquanto entregou melhor resultado nas classes *seaweed_salad* (91%) e *edame* (96%). Os resultados completos podem ser observados na Tabela 2.

Classes	Precision	recall	f1-score	support
apple_pie	0.53	0.43	0.47	134
baby_back_ribs	0.72	0.77	0.74	118
baklava	0.75	0.88	0.81	126
beef_carpaccio	0.73	0.83	0.78	123
beef_tartare	0.87	0.61	0.72	132
beet_salad	0.64	0.66	0.65	126
beignets	0.77	0.89	0.83	113
bibimbap	0.94	0.83	0.88	138
bread_pudding	0.59	0.56	0.57	134
breakfast_burrito	0.70	0.53	0.60	125
bruschetta	0.65	0.62	0.63	133
caesar_salad	0.84	0.83	0.83	127
cannoli	0.91	0.77	0.84	123
caprese_salad	0.79	0.73	0.76	124
carrot_cake	0.90	0.64	0.75	138
ceviche	0.52	0.64	0.58	128
cheese_plate	0.73	0.74	0.74	109
cheesecake	0.61	0.55	0.58	121
chicken_curry	0.68	0.60	0.64	123
chicken_quesadilla	0.87	0.69	0.77	125
chicken_wings	0.86	0.78	0.82	127
chocolate_cake	0.51	0.81	0.63	134
chocolate_mousse	0.46	0.58	0.52	132
churros	0.68	0.80	0.73	128
clam_chowder	0.70	0.86	0.77	129
club_sandwich	0.88	0.83	0.85	127
crab_cakes	0.74	0.56	0.63	135
creme_brulee	0.93	0.81	0.87	131
croque_madame	0.79	0.83	0.81	131
cup_cakes	0.81	0.81	0.81	122
deviled_eggs	0.95	0.86	0.90	107
donuts	0.63	0.78	0.70	124
dumplings	0.82	0.81	0.82	134
edamame	0.93	0.99	0.96	127
eggs_benedict	0.81	0.83	0.82	120
escargots	0.68	0.81	0.74	115
falafel	0.85	0.58	0.69	132
filet_mignon	0.53	0.66	0.59	125
fish_and_chips	0.85	0.78	0.81	118
foie_gras	0.34	0.58	0.42	118
french_fries	0.79	0.90	0.84	118
french_onion_soup	0.68	0.84	0.75	100
french_toast	0.67	0.62	0.64	123
fried_calamari	0.87	0.80	0.83	129
fried_rice	0.88	0.75	0.81	125
frozen_yogurt	0.82	0.82	0.82	138
garlic_bread	0.78	0.71	0.74	135
gnocchi	0.56	0.62	0.59	127
greek_salad	0.76	0.79	0.78	131
grilled_cheese_sandwich	0.65	0.62	0.63	124
grilled_salmon	0.72	0.65	0.68	135
guacamole	0.88	0.82	0.85	124
gyoza	0.69	0.77	0.73	113
hamburger	0.76	0.65	0.70	114
hot_and_sour_soup	0.90	0.90	0.90	126
hot_dog	0.53	0.71	0.61	105
huevos_rancheros	0.83	0.38	0.53	130
hummus	0.73	0.58	0.65	128
ice_cream	0.70	0.57	0.63	127
lasagna	0.75	0.61	0.67	123
lobster_bisque	0.72	0.84	0.78	118
lobster_roll_sandwich	0.80	0.87	0.83	116
macaroni_and_cheese	0.72	0.73	0.73	119
macarons	0.79	0.94	0.86	116
miso_soup	0.84	0.93	0.88	120
mussels	0.83	0.90	0.87	114
nachos	0.73	0.64	0.68	140
omelette	0.73	0.61	0.66	135
onion_rings	0.83	0.80	0.82	119
oysters	0.86	0.86	0.86	132
pad_thai	0.74	0.94	0.83	131
paella	0.77	0.77	0.77	118
pancakes	0.80	0.74	0.77	133
panna_cotta	0.59	0.66	0.62	120
peking_duck	0.56	0.75	0.64	121
pho	0.75	0.92	0.83	132
pizza	0.77	0.79	0.78	116
pork_chop	0.46	0.45	0.46	132
poutine	0.94	0.86	0.90	124
prime_rib	0.71	0.76	0.73	128
pulled_pork_sandwich	0.68	0.73	0.70	105
ramen	0.75	0.72	0.74	123
ravioli	0.58	0.46	0.51	126
red_velvet_cake	0.94	0.75	0.83	122
risotto	0.62	0.64	0.63	115
samosa	0.73	0.73	0.73	127

sashimi	0.71	0.88	0.78	112
scallops	0.48	0.63	0.54	111
seaweed_salad	0.88	0.95	0.91	134
shrimp_and_grits	0.71	0.69	0.70	124
spaghetti_bolognese	0.93	0.87	0.90	128
spaghetti_carbonara	0.86	0.91	0.89	119
spring_rolls	0.83	0.74	0.79	141
steak	0.45	0.38	0.41	120
strawberry_ shortcake	0.69	0.74	0.71	134
sushi	0.88	0.64	0.74	129
tacos	0.72	0.63	0.67	134
takoyaki	0.83	0.86	0.84	142
tiramisu	0.74	0.65	0.69	122
tuna_tartare	0.63	0.60	0.61	141
waffles	0.81	0.76	0.79	121

Tabela 2. Resultados por classe. Fonte: A autora

CONCLUSÃO

Neste artigo, foi demonstrado que o modelo EfficientNet, projetado principalmente com a detecção de objetos, localização de objetos e segmentação de instâncias de imagens naturais em mente, pode ser usado para produzir resultados de alta qualidade para auxiliar em modelos preditivos customizados para o próprio propósito.

Por fim a EfficientNetB0 tem muito futuro e algumas melhorias que podem ser feitas para a melhor detecção como um treinamento com quantidade maior de épocas ou maior quantidade de imagens. Um ponto para determinar isso seria com mais treinamento para a rede neural entender melhor as áreas de contato e conseguir distinguir os falsos positivos.

REFERENCES

[1] Hern, Alex, *"Google's solution to accidental algorithmic racism: ban gorillas",* 2018; acesso em 10 julho, 2022, https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people.

[2] 2019-06-12.
MOURA, Brenda. *Uso de detecção de objetos em imagens para o ensino de vocabulário estrangeiro***.** Ciência da Computação, Instituto Federal Catarinense, Santa Catarina, 2021.

[3] W. L. HOMEM, "Apostila de machine learning," UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO PROGRAMA DE EDUCAÇÃO TU- TORIAL ENGENHARIA MECÂNICA, 2020.

[4] E. ALECRIM, "Machine learning: o que é e por que é tão importante," 2018. Online; acesso em 11 Julho, 2022, https://tecnoblog.net/247820/machine-learning-ia-o-que-e/.

[5] M. TIBAU, "Visão computacional na era do deep learning," 2021. Online; acesso em 11 Julho, 2022, https://www.updateordie.com/2021/05/14/visao-computacional-na-era-do-deep-learning/.