COINTER: Aplicativo para Reconhecimento e Segmentação de Imagens de Moedas

Rickson G. Monteiro*, Ana Luísa G. Pires†, Gabriella C. B. Costa‡ , Luan S. Oliveira§, Tatiana B. de Azevedo.¶
Centro Federal de Educação Tecnológica
Leopoldina, Minas Gerais
Email:{*ricksonencaut, †aninhaluisa09}@gmail.com, {‡gabriella, §luan, ¶tatianaazevedo}@cefetmg.br

***Resumo*—Este artigo apresenta uma solução para facilitar o turista brasileiro na identificação e cálculo do valor de moedas em dólar, já que estas não apresentam seus respectivos valores através de algarismos arábicos retratados nelas. Para este fim, foi desenvolvido um aplicativo multiplataforma capaz de fotografar essas moeda e, através da rede neural MobileNetV2, fazer a análise das mesmas determinando seus valores, retornando ao usuário o valor total das moedas que foram fotografadas. Os testes iniciais feitos utilizando esse aplicativo demonstram que a rede neural utilizada, quando aplicada em situações reais, apresenta uma taxa de acurácia de 97,65%.**

***Abstract*—This article introduces a solution to facilitate the Brazilian tourist in the identification and calculation of the value of foreign currencies that don't present their respective values through Arabic numerals portrayed on them. For this purpose, a multiplatform application capable of photographing coins was developed and, through the MobileNetV2 neural network, analyze them by determining their values, returning to the user the total value of the coins that were photographed. The initial tests using this application demonstrate that the neural network used, when applied in real situations, presents an assertiveness rate of 97.65%.**

***Palavras-chaves*—MobileNetV2, Segmentação, Reconhecimento, Fine Tuning, Aplicativo**

I. Introdução

Estamos em uma era marcada pela grande interação virtual e física entre pessoas das mais variadas nacionalidades do mundo. Atualmente, somente no Brasil, há mais de 220 milhões de smartphones ativos. Nunca se comunicou tanto e nunca se viajou tanto. Entretanto, uma viagem a um país estrangeiro, seja a trabalho, estudo ou lazer, envolve inúmeros desafios sociais e culturais. Além das dificuldades de adaptação a uma nova alimentação, diferenças culturais e ao idioma. Há também dificuldade em compreender a moeda corrente do país. Isso porque diversos países utilizam notas e moedas que apresentam apenas imagens e símbolos, sem apresentar impresso o número ou valor a que se refere ou apresentá-lo apenas na língua do próprio país. Diante disso, este projeto idealiza o COINTER, que tem por objetivo a utilização de técnicas de reconhecimento de imagens para o desenvolvimento de um aplicativo que visa fotografar moedas em dólar, fazer o reconhecimento de cada uma delas e informar o valor total das moedas fotografadas. Através do COINTER será possível, por meio da câmera do smartphone, identificar moedas, calcular e informar ao usuário o valor total fotografado. Com esse aplicativo, visamos facilitar a interação do turista com os indivíduos de outra nação no que tange à utilização de moedas estrangeiras. Cabe ressaltar que a estrutura desse projeto foi realizada em três etapas: o front-end, a rede neural e a integração de ambas as partes.

II. Trabalhos Relacionados

No trabalho de Shubham Mittal e Shiva Mittal [1], os autores apresentaram uma estrutura de identificação de denominações de notas de Rúpia, moeda corrente indiana, a partir de suas imagens coloridas. É enfatizado nesse trabalho que o conjunto de dados apresenta imagens de quatro denominações de notas em diferentes pontos de vistas e condições de iluminação. Por possuírem um DataSet com poucas imagens, estas foram submetidas a rotações aleatórias, deslocamentos horizontais e verticais e escalonamento para cima ou para baixo, o que aumentou consideravelmente esse conjunto de dados. Nesse artigo, também é mencionado que esse modelo pode auxiliar os deficientes visuais na identificação das denominações escritas nas notas. O modelo retratado usa o conceito de aprendizagem por transferência através da maior versão da MobileNet, é treinado para o reconhecimento da moeda indiana da Rúpia com denominação de dez, cinquenta, cem e quinhentos. A experimentação foi realizada com uma máquina equipada com CPU dual-core Intel Core i5, a solução foi implementada em python com auxílio do TensorFlow. O modelo treinado exibe uma precisão de 96,6% e o trabalho futuro visa portar o método para smartphones para desenvolver um aplicativo robusto para reconhecimento de notas.

Em [2], Samuel S. De Freitas elabora uma proposta semelhante a aqui apresentada. Em seu trabalho ele busca desenvolver um algoritimo eficiente para contagem de moedas com o intuito de ajudar os deficientes visuais e comerciantes. Foi apresentado dois algoritimos para classificação, o Modelo A, utilizando pontos de interesse, no qual foi baseado no algoritimo SIFT e o Modelo B, onde é utilizado os raios das moedas para classificação, ambos detalhados em [2]. Apesar do processo de segmentação ser semelhante ao utilizado no COINTER, seu processo de classificação difere, sendo que neste projeto foi utilizada a rede neural convolucional MobileNetV2 para classificação das moedas, na qual apresentou uma taxa de assertividade melhor que os algoritmos citados acima.

III. Front-End

O *"front-end"* é a interface gráfica do aplicativo que interage diretamente com o usuário através de seu dispositivo e, nesse projeto, essa interface foi estruturada com um *framework* baseado em JavaScript capaz de gerar código nativo para iOS e Android denominado React Native [3]. Além disso, para facilitar a criação dessa interface, foi usado o Expo, uma ferramenta utilizada no desenvolvimento mobile com React Native que permite o fácil acesso às API'S nativas do dispositivo, oferecendo grande parte de recursos de forma nativa e integrada [4]. A escolha desse *framework* foi baseada na possibilidade desta linguagem gerar versões nativas multiplataforma (iOS e Android) com o mesmo código-fonte [5], utilizando apenas JavaScript. No entanto, todo o código desenvolvido com o React Native é convertido para linguagem nativa do sistema operacional, o que torna o aplicativo muito mais fluído [6]. Como já mencionado, o intuito do aplicativo é fazer o reconhecimento das moedas fotografadas e retornar o valor que o usuário possui. Para que haja tal funcionalidade, foi necessário fazer a integração do *front-end* e a rede neural. Sendo assim, houve a necessidade de se criar uma API PHP que através de uma requisição HTTP no código JavaScript recebe a foto da moeda tirada pelo usuário, salva no servidor e após esse processo, envia a fotografia para uma outra API feita com a linguagem Python que irá realizar a segmentação e o reconhecimento das moedas e retornar o valor que o usuário possui. A versão do aplicativo para Android pode ser obtido clicando aqui.

IV. DataSet

Para primeira versão do aplicativo, foi estipulado apenas o reconhecimento de moedas de dólar americano, dessa forma foi montado um conjunto de dados com 575 imagens distribuídas em 5 classes, 115 imagens de cada moeda, onde 400 foram destinadas para o treinamento, 90 para validação e 85 para teste. A fim de simplificar o problema, a primeira versão do aplicativo só reconhece moedas na posição coroa. Dessa forma, pôde-se reduzir o número de classes do problema em 5. As duas primeiras classes são referentes à moeda de *"one cent"*, a primeira refere-se a versão da moeda com o escudo da união; já a segunda tem referência ao Memorial Lincoln. Apesar dessas duas classes fazerem referência a um mesmo valor (*"one cent"*), apresentam características visuais diferentes na posição coroa o que influencia diretamente no treinamento da rede neural. Afim de um melhor resultado na classificação foi realizada essa divisão. A terceira classe representa o *"nickel"* ou *"five cents"*, a quarta e quinta classes são as moedas de *"one dime"* e *"quarter dollar"*; respectivamente. Visando uma maior generalização do modelo, foi aplicada uma técnica conhecida como *In-place/on-the-fly data augmentation*, disponibilizado pela classe "ImageDataGenerator" que realiza transformações no conjunto de treinamento como rotações, zoom, alteração de brilho entre outras [7]. As transformações utilizadas neste projeto podem ser vistas no algoritimo 1.

```
data_train = ImageDataGenerator(rescale = 1./255,
                                rotation_range = 7,
                                horizontal_flip =
                                    True,
                                shear_range = 0.2,
                                height_shift_range =
                                    0.05,
                                zoom_range = 0.3)
```

Algoritimo 1. Transformações aplicadas no conjunto de treino

V. Segmentação

O processo de segmentação das moedas é realizado totalmente com o auxílio da biblioteca de visão computacional OpenCV, esta que possui a função "HoughCircles()" utilizada para detecção dos círculos das moedas para posterior extração, no qual se baseia no *"Hough Gradiente Method"* [8], explicado em [9]. O método utilizado para segmentação das moedas retorna uma lista contendo a posição do centro do círculo encontrado (x, y) e o raio do mesmo. Dessa forma, é possível realizar a extração da imagem da moeda e após um redimensionamento enviá-la a rede neural para classificação. A função utilizada para segmentação pode ser visualizada no algoritimo 2.

```
def image_segmentation(image):
    imagens_cortadas=[]
    gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur=cv2.GaussianBlur(gray,(9,9),cv2.
        BORDER_DEFAULT)
    all_circs=cv2.HoughCircles(blur, cv2.
        HOUGH_GRADIENT, 0.9, 300, param1=50,
                                   param2= 25,
                                minRadius=270,
                                maxRadius=350)
    all_circs=np.uint16(np.around(all_circs))
    for corte in (all_circs[0]):
        crop= image[corte[1]-corte[2]:corte[1]
        +corte[2], corte[0]-corte[2]:corte[0]
        +corte[2]]
        imagens_cortadas.append(crop)
    return imagens_cortadas
```

Algoritimo 2. Funcão de utilizada na API para segmentação das moedas treino

A função elaborada para segmentação das moedas recebe a imagem contendo todas as moedas como parâmetro, realiza uma conversão de cor do espaço BGR (*"Blue"* (Azul), *"green"* (Verde) e *"Red"* (Vermelho)) para o *"gray"* (cinza). Desse modo, há uma redução nos canais de cores da imagem de 3 para 1. Posteriomente, é aplicado um filtro passa baixa na imagem já convertida, com o intuito de reduzir os possíveis ruídos presentes e evitar o reconhecimento de círculos falsos. Logo após, a imagem é enviada ao método "HoughCircles()", já mencionado aqui e com as coordenadas obtidas é feita uma manipulação na imagem colorida, afim de extrair as moedas encontradas salvando-as em uma lista vazia que é retonada para outra função, onde é realizada o reconhecimento das mesmas.

VI. Rede Neural

A arquitetura escolhida para o reconhecimento das imagens foi a MobileNetV2, esta, voltada para aplicações mobile buscou assimilar precisão e desempenho. A MobileNetV2 utiliza

camadas de convolução separáveis em profundidade no lugar das camadas convolucionais comuns. Nesta, o trabalho da camada de convolução é dividido em subtarefas que juntas formam um "bloco" recebendo esse nome, sendo possível assim reduzir o custo computacional em 8 a 9 vezes, sem grandes alterações na precisão [10]. Na MobileNetV2, existem três camadas de convolução em um bloco que recebe o nome de *"Bottleneck Residual block"*. A primeira, recebe o nome de camada de expansão, esta é uma camada de convolução 1x1, que tem como objetivo expandir o número de canais antes da entrada na segunda camada [11], que é uma camada de convolução em profundidade 3x3 que realiza a filtragem dos canais de entrada, por último uma camada de convolução 1x1 que recebe o nome de camada de projeção ou camada de gargalo (*"Bottleneck layer"*), reduzindo a dimensão dados. Após cada camada há um processo de *"batch normalization"* e a função de ativação utilizada é a *"ReLu6"*, com exceção da saída da camada de projeção, pois percebeu-se que ocorria a perda de informações úteis com a aplicação de uma não linearidade em dados de baixa dimensão [11]. Dessa forma temos uma arquitetura formada por uma camada de convolução inicial com 32 filtros de kernel 3x3 seguidos de 19 blocos *"Bottleneck Residual"* [10] descritos no parágrafo acima.

A. Treinamento

O processo de treinamento foi realizado com o uso a técnica de *"fine tuning"* devida a baixa quantidade de dados presentes. Esta técnica é baseada no conceito de *"transfer learning"*, no qual são transferidas informações da arquitetura treinada sob um conjunto maior para um conjunto mais específico [12]. Neste caso, foi realizado o congelamento de parte da rede treinada no conjunto de dados do imagenet [13] e realizado o treinamento do restante do modelo com o conjunto já detalhado aqui por 250 épocas. Foi utilizado o *"Adam"*[14] como otimizador com uma taxa de aprendizado igual a 0,0001. A função de perda utilizada foi a *"Categorical crossentropy"*[15] como e a função de ativação *"Softmax"* na camada de saída. Nas figuras 1, 2 e 3 pode-se ver os valores da função de perda, da acurácia do modelo e a matriz de confusão de resultados do conjunto de testes respectivamente.

Dessa forma, após a aplicação da técnica citada junto ao processo de treinamento, foi alcançado uma acurácia de 97,65% no conjunto de testes. Também foram realizados testes com outras arquiteturas aplicando os mesmos procedimentos, a VGGNet19 e a InceptionResNetV2 que apesar de terem apresentado bons resultados, 97,65% e 96,47% de acurácia respectivamente, requiriam bastante espaço de amazenamento, 212mb no caso da VGGNet19 e 469mb para a InceptionResNetV2 contra 23mb da MobileNetV2, o que torna inviável o uso delas para a aplicação.

VII. API

Após o treinamento da rede e desenvolvimento do procedimento para segmentação foi desenvolvido uma API em

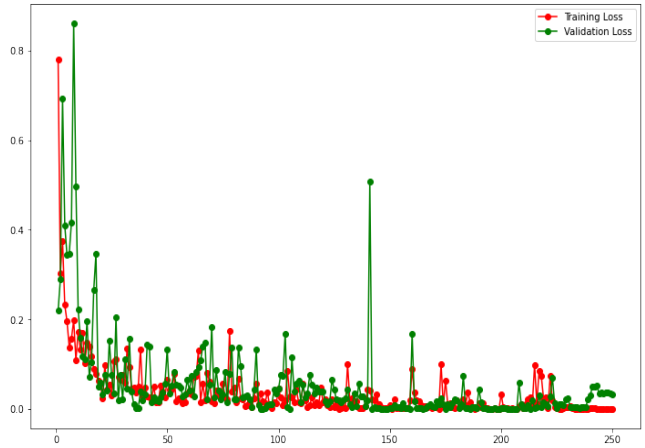

Figura. 1. Valores da função de perda no conjunto de validação e treinamento

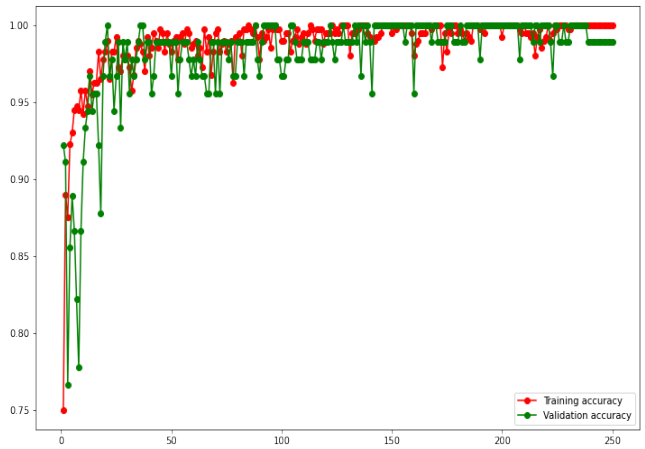

Figura. 2. Valores da acurácia do modelo no conjunto de validação e treinamento

python com auxílio do *framework* flask que permite o desenvolvimento de aplicações web de forma simples. A API desenvolvida comporta basicamente o modelo treinado e o algoritimo de segmentação que após finalizada foi hospedada no servidor Heroku que disponibiliza gratuitamente 500mb para uso, por esse fator uma arquitetura mais leve foi escolhida. Seu funcionamento ocorre através de uma requisição HTTP realizada pela API PHP, no qual é passado como parâmetro o nome da imagem que foi salva anteriormente em outro servidor também pela API em PHP, dessa forma a API em python consegue resgatar a imagem enviada pelo usuário, realizando os processos de segmentação, reconhecimento e posteriormente retornando ao usuário o valor total na imagem.

VIII. Conclusão

O COINTER, busca facilitar o dia a dia do indivíduo que não esteja familiarizado com a moeda corrente local, baseando-se em dois princípios: segmentação e reconhecimento de imagem. Este foi organizado em uma interface simples de forma que o usuário não encontre dificuldade em utilizá-la. O uso de uma rede neural convolucional voltada para os

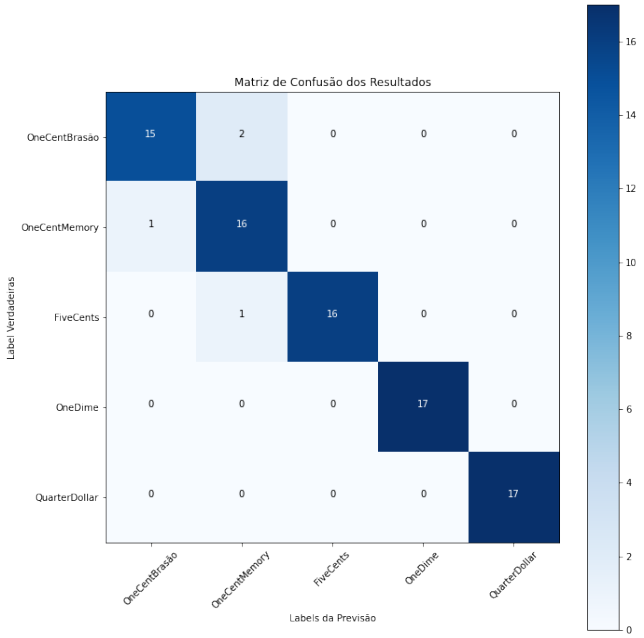

Figura. 3. Matriz de confusão dos resultados

dispositivos móveis permitiu ótimos resultados, uma acurácia de 97,65%, sem exigir grande poder de processamento. A ideia de hospedar o *"back-end"* do projeto em um servidor externo ajudou a tornar o aplicativo mais fluído. Entretando, ainda existem alguns problemas a serem resolvidos nos projetos futuros. Há uma grande necessidade de um processo mais inteligente para a extração de moedas das imagens, a transformada de *"hough"* baseada na orientaçao do gradiente disponibilizada no OpenCV é bastante robusta em ambientes controlados. No entanto, ainda depende de parâmetros fixos o que leva a possíveis erros de segmentação em imagens com iluminação ou ambiente muito distinto do ajustado durante o desenvolvimento. Além disso, a necessidade de resgatar a imagem enviada pelo usuário de outro servidor ocasiona uma lentidão na resposta do usuário devido a necessidade de download. A primeira versão do aplicativo tem suporte para moedas de dólar americano, no entanto é de grande interesse da equipe adicionar funcionamento para outras as moedas, como a Libra esterlina (£), o Iene (¥) e o Dólar canadense ($). A equipe de desenvolvimento do aplicativo também visa adicionar uma funcionalidade para usuários com possíveis deficiências (Visual, por exemplo). Adicionando uma mensagem por voz será possível esse indivíduo identificar o valor monetário consigo sem ajuda externa.

REFERÊNCIAS

[1] MITTAL, Shubham; MITTAL, Shiva. Indian Banknote Recognition using Convolutional Neural Network. In: 2018 3rd International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU). IEEE, 2018. p. 1-6.

[2] Freitas, Samuel Sanches de. "Determinação do valor total de moedas em imagens digitais."(2014).

[3] PEDRASSANI, Carlos Eduardo. Uma solução em Node.js e React Native para busca e oferta de emprego. 2018. Disponível em: http://t2ti.com/erp3/pdf/TCC_CARLOS_EDUARDO_PEDRASSANI.pdf Acesso em: 26 jul. 2020.

[4] FERNANDES, Diego. Expo: o que é, para que serve e quando utilizar?. Rocketseat, 2018. Disponível em: https://blog.rocketseat.com.br/expo-react-native/. Acesso em: 26 jul. 2020.

[5] Por que React Native é a linguagem adequada para criação de apps?. Computerworld, 2019. Disponível em: https://computerworld.com.br/2019/02/12/por-que-react-native-e-a-linguagem-adequada-para-criacao-de-apps/. Acesso em: 26 jul. 2020.

[6] ANDRADE, Ana Paula de. O que é React Native?. TreinaWeb, 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-o-react-native/. Acesso em: 26 jul. 2020.

[7] ROSEBROCK, Adrian. Keras ImageDataGenarator and Data Argumentation, 2019. Disponível em: https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/. Acesso em: 26 jul. 2020.

[8] OpenCV https://docs.opencv.org/master/d3/de5/tutorial_js_houghcircles.html

[9] PETKOVIC, T., LONCARIC, S. An Extension to Hough Transform Based on Gradient Orientation. Proceedings of the Croatian Computer Vision Workshop, Year 3. 2015.

[10] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., CHEN, L. MobileNetV2: Inverted Residual Linear Bottlenecks, 2018 In Proceedings of the IEEE conference on computer vision and pattern recognition, p. 4510-4520. 2018.

[11] HOLLEMANS, Matthijs. MobileNet Version 2, 2018. Disponível em: https://machinethink.net/blog/mobilenet-v2/. Acesso em: 26 jul. 2020.

[12] REYES, A. K., CAICEDO, J. C., CAMARGO, J. E. Fine-tuning Deep Convolutional Networks for Plant Recognition, CLEF (Working Notes) p. 467-475. 2015.

[13] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

[14] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[15] Zhang, Z., Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. In Advances in neural information processing systems (pp. 8778-8788).