Insper

Relatório de Projeto da Disciplina Machine Learning

Classificador de Imagens do Jogo Pedra, Papel, Tesoura

Marcelo Cesário Miguel

William Augusto Reis da Silva

Professor Fábio Ayres

São Paulo

Dezembro/2021

1. Introdução

"Pedra, papel e tesoura", também conhecido como Jockenpô, ou *rock paper scissors*, é um jogo muito conhecido mundialmente, utilizado como brincadeira infantil ou até em disputas, visando à decisão de alguma ideia, ou tópicos semelhantes. A ideia é que uma pessoa irá jogar contra a outra, simulando sua mão com um dos três objetos, enquanto a outra pessoa faz o mesmo movimento e, ao final, após simularem juntas, a partir do objeto selecionado por cada, determina-se a vencedora.

Tabela 1 – Explicação dos Resultados do Jogo

Vencedor	**Pessoa A**	**Pessoa B**
Papel (B)	Pedra	Papel
Tesoura (A)	Papel	Tesoura
Pedra (B)	Tesoura	Pedra
Empate	Pedra	Pedra

Ou seja, há uma situação para cada objeto vencer e, caso ambos joguem igual, dá empate.

Conhecendo a relevância do jogo e almejando trabalhar com redes neurais e classificação de imagens, o grupo decidiu construir um classificador, utilizando como base imagens obtidas no site Kaggle[1], que contém imagens de mãos representando os objetos. Com isso, serão utilizadas técnicas de Machine Learning para poder classificar as imagens sozinhas, determinando se são de pedra, papel ou tesoura. Ademais, no final será demonstrado um jogo, onde se tem duas imagens, cada uma representando um jogador, onde o resultado será definido a partir do que a rede previu, para dar um ar mais real para o modelo construído.

2. Desenvolvimento

a. Análise das Imagens e Divisão entre Treinamento e Testes

De início, foi feito o download das imagens presentes no link do kaggle, onde foram colocadas no mesmo diretório do projeto a fim de começar a utilizar. Analisou-se que havia 2188 imagens, sendo 726 de pedra, 713 de papel e 750 de tesoura. Juntou-se todas em uma lista, embaralhando para ficarem aleatoriamente distribuídas, além de se fazer a divisão entre **conjunto de treinamento** e **conjunto de testes**, algo muito importante em projetos de Machine Learning. 80% das imagens foram para o treinamento do modelo e 20% para os testes.

Mas antes de iniciar toda a construção do modelo, é relevante observar como são as imagens em si, para ver se são semelhantes, ou obter outras características.

Figura 1 – Imagens do Conjunto de Treinamento

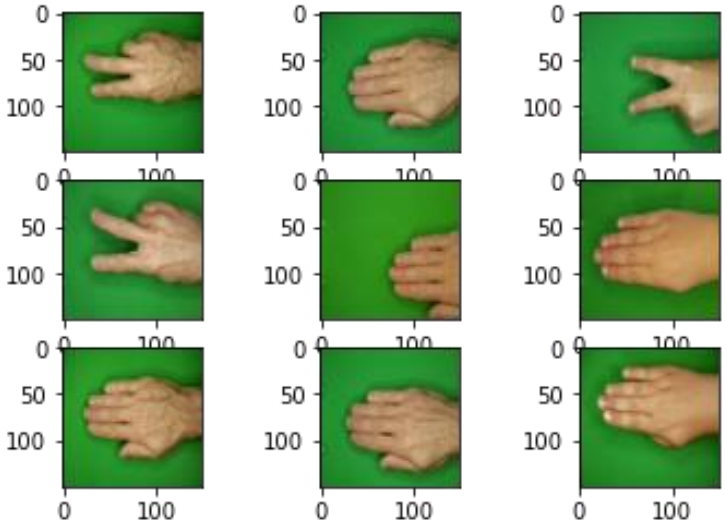
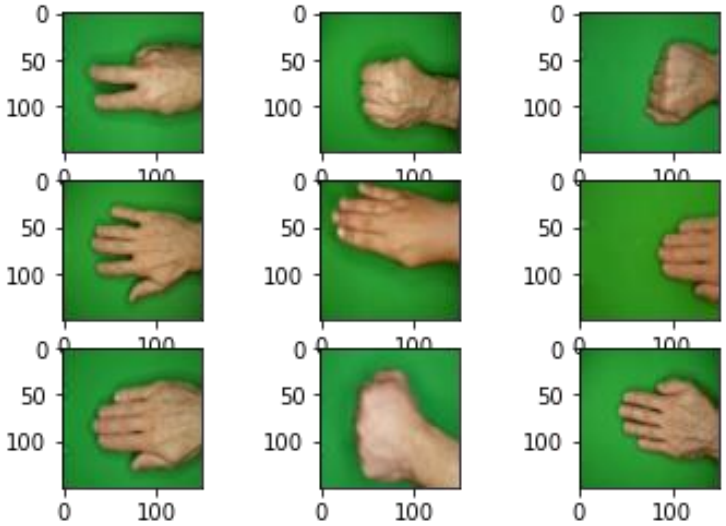


Figura 2 – Imagens do Conjunto de Teste



Como é possível notar, as imagens são padronizadas, com um fundo verde e com mãos representando os objetos em posições relativamente semelhantes, além das peles terem a mesma tonalidade. As imagens possuem o mesmo tamanho também e é importante deixar claro isso, pois pode impactar se o modelo, por algum motivo, quiser prever uma imagem com características diferentes, de forma a observar como ele iria se comportar, o que o grupo realizará no final também.

b. Construção do Modelo e Resultados

Com o conjunto de treinamento e testes dividido, foi feita uma preparação dos dados para serem compatíveis com os objetos utilizados no TensorFlow, como um tensor na variável X. Na variável Y, foi feita a categorização dos dados, de forma que, se é:

- Uma pedra: 0;
- Uma tesoura: 1;

- Um papel: 2.

Além disso, padronizamos o tamanho das imagens utilizando o ImageDataGenerator, que faz esse trabalho, para que não haja muita divergência no tamanho. Todas essas mudanças são com o intuito de preparar tudo para que o modelo possa agir. A imagem abaixo representa como fica então o X, que já foi explicitado acima, demonstrando que há 1749 imagens, cada uma com tamanho (150, 150), coloridas (o 3 representa R, G, B).

Figura 3 – Conjunto X de treinamento representando as imagens

```
X_train.shape
```

```
(1749, 150, 150, 3)
```

A partir disso, pode-se partir para a construção do modelo, que será construído a partir do TensorFlow, uma biblioteca do Python muito relevante para construção de redes neurais. Assim, cria-se a rede neural com o Sequential do TensorFlow.Keras.

Um dos pontos interessantes de se explicar é o summary do model, onde consegue se entender melhor como está funcionando essa rede neural. A imagem abaixo auxilia na discussão e explicação:

Figura 4 – Summary do Modelo

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2D  (None, 74, 74, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 64)       0
 2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 128)               2367616

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 2,391,587
Trainable params: 2,391,587
Non-trainable params: 0
_________________________________________________________________
```

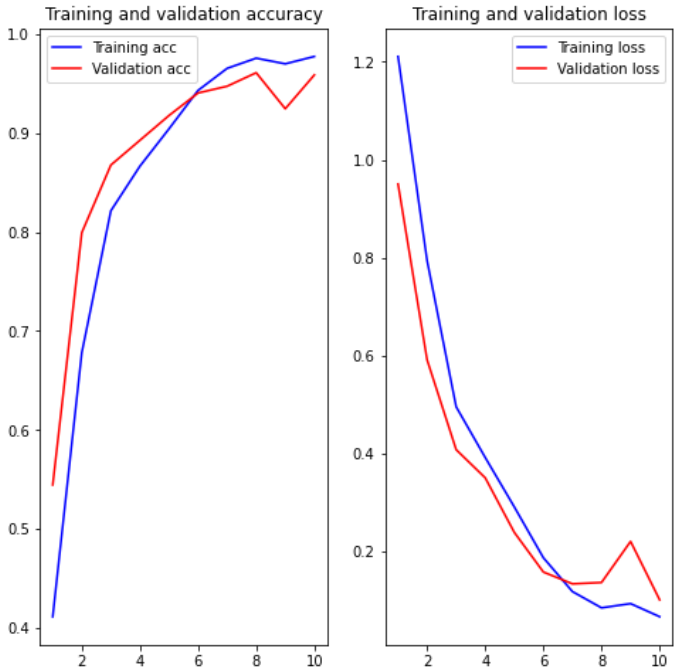
Como é possível observar, foi criada uma rede pelo Sequential do keras, utilizando redes como a conv2d, maxpooling2d, flatten e dense. Na última Dense, foi utilizado o parâmetro de 3 neurônios, pois é o número de classes que possuímos, com a função de ativação softmax. A base utilizada para a consturção dessa rede foi de um tutorial que possuía um problema semelhante, de um tutorial oficial do tensorflow[2].

Com a rede criada, foi utilizado o fit, para fazer o treinamento e analisar qual o comportamento no conjunto de testes. Foram utilizadas 10 epochs, que é uma iteração de treinamento. No caso, implementamos 10 iterações, ou seja, ela vai repassar os dados de treinamento 10 vezes.

Na primeira iteração, teve-se acurácia de 42% para o conjunto de treinamento e 54% para o conjunto de testes, o que é algo que chamou atenção, porém ainda estava na primeira iteração. A última, que é a mais relevante para o modelo em si, teve-se o desempenho incrível de 98% para o conjunto de treinamento e 95% para o conjunto de testes.

Para compreender melhor a evolução dos valores conforme cada iteração, tem-se um gráfico muito interessante de vai demonstrando como evolui o desempenho (acurácia) do conjunto de treinamento e testes:

Figura 5 – Gráficos de Acurácia e Perdas



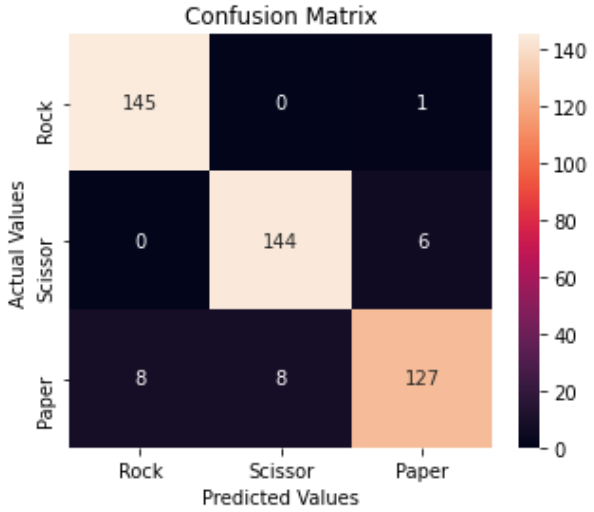
Como é possível observar no primeiro gráfico, de acurácias, tanto os dados de treinamento quanto os de testes têm seus resultados crescendo conforme o número de iterações aumentam, o que trás uma visão de que está funcionando bem a rede construída. Uma coisa interessante a se observar também é de que ambas as linhas continuam perto uma da outra, o que trás um ponto positivo. Um problema que geralmente acontece, demonstrando overfitting, é a linha do treinamento subir bastante, enquanto a de validação para em um certo número. Ou seja, para o conjunto de treinamento, o modelo funciona, porém para o de testes não, necessitando fazer ajustes, o que não é nosso caso.

O segundo gráfico demonstra as perdas, que também descrece juntas conforme o modelo tem as suas iterações. No decorrer, elas abaixam, o que é justamente o que se almeja em um modelo construído, demonstrando que o modelo construído foi relativamente positivo.

c. Predição e Resultados

Após tudo que foi proferido, fizemos a predição a partir dos dados de teste e observamos inicialmente através de uma matriz de confusão como foi o desempenho.

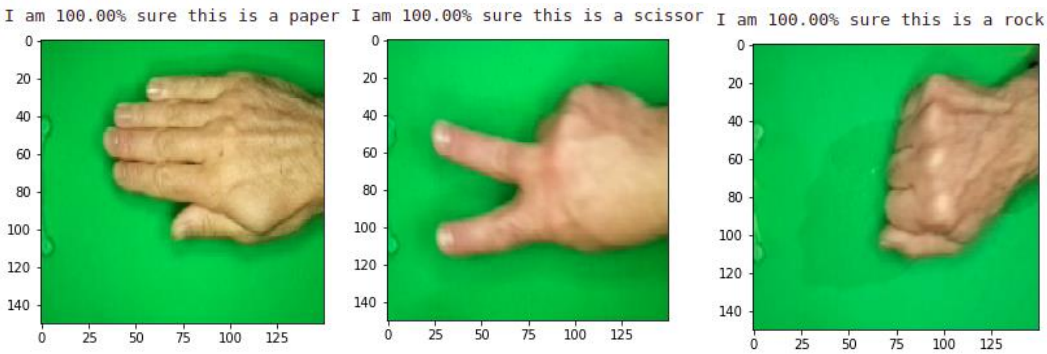
Figura 6 – Matriz de Confusão

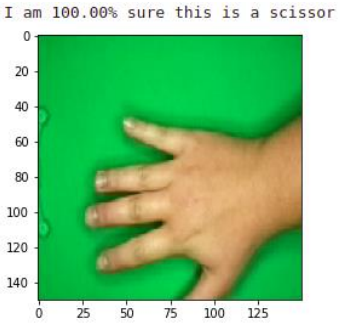


A partir da matriz de confusão, consegue-se analisar as métricas *precision* e *recall*, por exemplo, que ajuda a entender um pouco como o modelo performou para cada uma das possibilidades.

Para as três categorias, o precision foi de 95%, o que significa que: quando o modelo previu que seria pedra, por exemplo, ele acertou 95% das vezes. 5% das vezes ele previu que seria papel ou tesoura. Outra medida interessante é o *recall.* A pedra tem *recall* de 99%, o que significa que 99% das vezes em que a classe apareceu nos dados de teste, ela foi prevista corretamente.

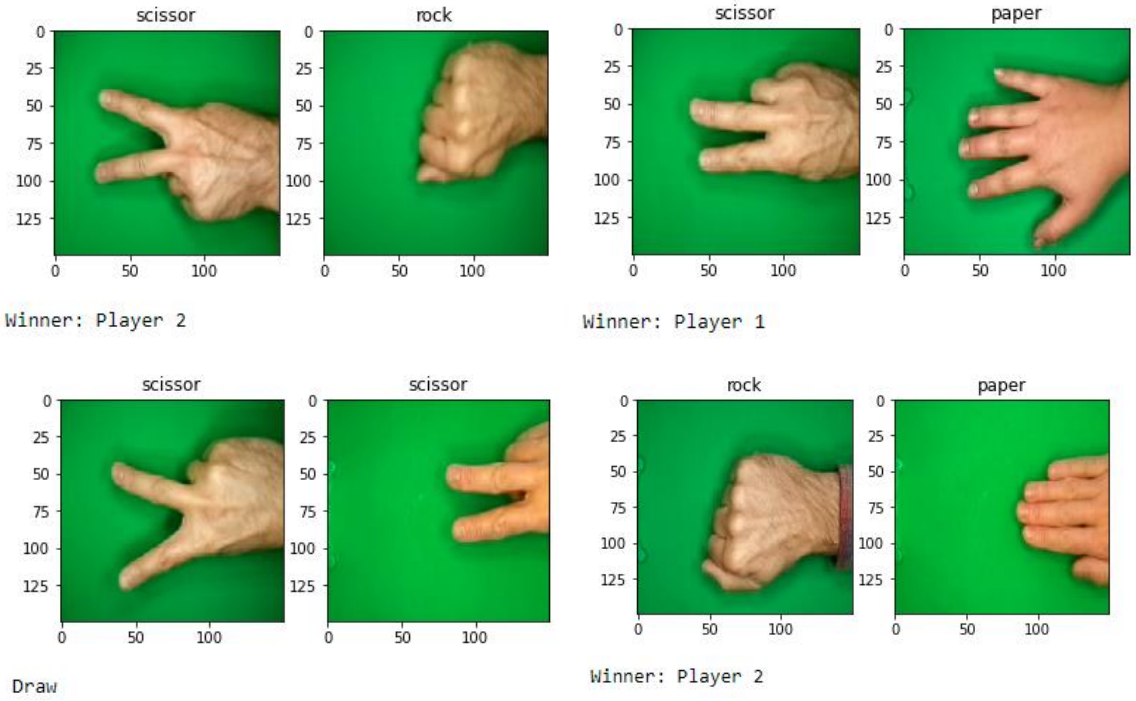
Para observar melhor o desempenho, nada melhor do que observar na prática com algumas imagens. Um ponto importante é de que nosso modelo ficou 100% convicto em todas as imagens, mesmo errando em alguns casos. As imagens abaixo são exemplos de acertos e erros do modelo:





d. Jogo de Imagens

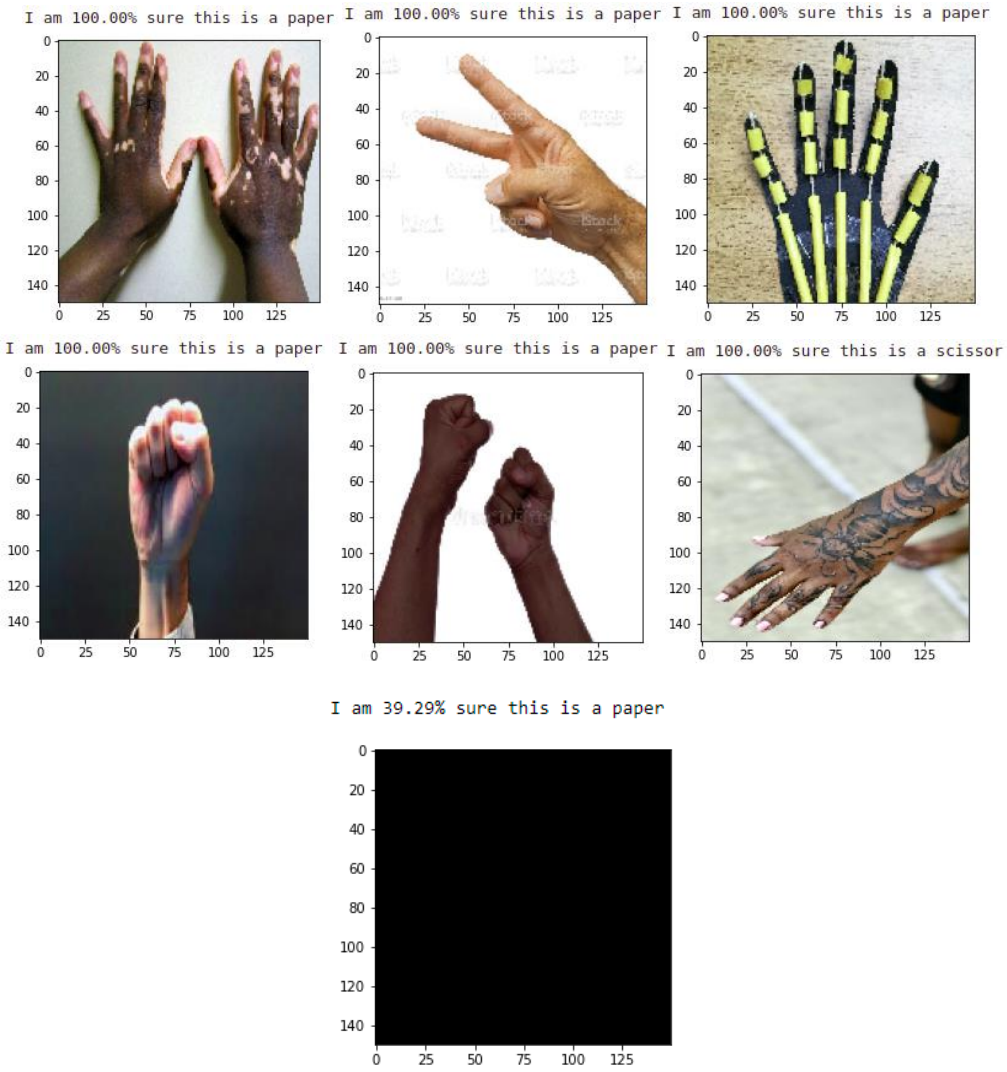
Como foi proferido inicialmente no relatório, um dos objetivos era fazer um jogo com predições. Selecionaremos duas imagens aleatórias, o modelo irá dizer o que ele acha que é e nós programamos para dizer qual das imagens é a vencedora de uma disputa. Essa é a parte mais divertida do projeto e algumas disputas estão demonstradas abaixo:



e. Imagens diferentes

O último tópico que exploramos no trabalho foi observar o desempenho do modelo em imagens diferentes, com mãos diferentes, de pele divergente, cenários alterados, além de outros tópicos, como por exemplo algo que não é nem uma mão.

As imagens abaixo demonstram os testes que fizemos, onde alguns ele acertou e outros passou longe!



3. Conclusão

Foi muito interessante realizar esse projeto, onde consegue-se observar que os resultados foram relativamente bons com a construção da rede que foi feita e os testes são divertidos e demonstraram que é possível fazer análises interessantes e observar como se comporta uma rede neural em um caso específico, que foi o que escolhemos, de “Pedra, Papel e Tesoura”. Para uma iteração futura, seria relevante mudar alguns parâmetros para ver como o modelo iria se comportar, diminuindo *epochs*, construindo outra rede, etc.

4. Bibliografia

[1] Rock-Paper-Scissors Images. Disponível em <*https://www.kaggle.com/drgfreeman/rockpaperscissors*>. Acesso em 24 nov. 2021.

[2] Classificação de Imagem. Disponível em <*https://www.tensorflow.org/tutorials/images/classification*>. Acesso em 2 dez. 2021.