Trabalho 2 - Classificador de gato/cachorro para Android

Breno Cunha Queiroz - 11218991
Lucas de Medeiros França Romero - 11219154
Lucas Yuji Matubara - 10734432

Índice

Introdução

O objetivo deste trabalho é aprender a utilizar o TensorFlow Lite para utilizar modelos em sistemas embarcados. Será desenvolvido um modelo para um aplicativo Android para conseguir classificar entre gatos e cachorros a partir da imagem da câmera do celular. Foi utilizado a técnica de transfer learning a partir do modelo MobileNetV2 para criar um modelo para a classificação, que posteriormente foi testado em no aplicativo para verificar a precisão e tempo de inferência. Iremos apresentar como o modelo inicial foi gerado e em seguida o processo iterativo de otimização.

Dataset de treinamento

O modelo foi treinado em cima de um dataset de cães e gatos público no google cloud[1]. Este dataset possui 2000 imagens de treinamento e 1000 imagens de validação.
Inicialmente foi feita uma análise da distribuição das classes no dataset e foi descoberto que tanto no dataset de treinamento, quanto no de validação, exatamente metade das imagens são de cachorros e metade de gatos, não sendo necessário um balanceamento. Na Figura 1 temos algumas fotos retiradas deste dataset.

[1] URL do dataset https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip

Figura 1 - Algumas imagens do dataset.

Como este dataset não possui um set de imagens para teste, o dataset de validação foi dividido em dataset de validação e dataset de teste.

Modelo desenvolvido

O treinamento foi feito no Google Colab utilizando Keras/TensorFlow. Utilizamos a técnica de *transfer learning* a partir do modelo MobileNetV2 para criar um classificador binário de cachorro e gato.

Input layer - A primeira camada do modelo desenvolvido é para recepção de uma imagem de 160x160.

Augmentation layer - A próxima camada aplica técnicas de aumento de dataset na imagem para evitar *overfitting* no dataset. Nesta camada é aplicado rotação e flip horizontal no vetor da imagem de input.

Preprocess layer - Após isto, existe outra camada para transformar este input na entrada esperada para o MobileNetV2.

MobileNetV2 - A saída do MobileNetV2 é um vetor de features de tamanho (5, 5, 1280). Neste momento é necessário extrair a informação destas features para definir se é um gato ou cachorro.

Average layer - Outra camada foi inserida para fazer a média entre os valores e converter o vetor de dimensão (5, 5, 1280) em um vetor de 1280 valores.

Dropout layer - Na saída do average layer utilizamos *dropout* de 20% para evitar *overfitting* do modelo.

Dense layer - A última camada de extração de features recebe os 1280 e os condensa para um único valor. A partir deste último valor obtemos se foi um gato ou cachorro.

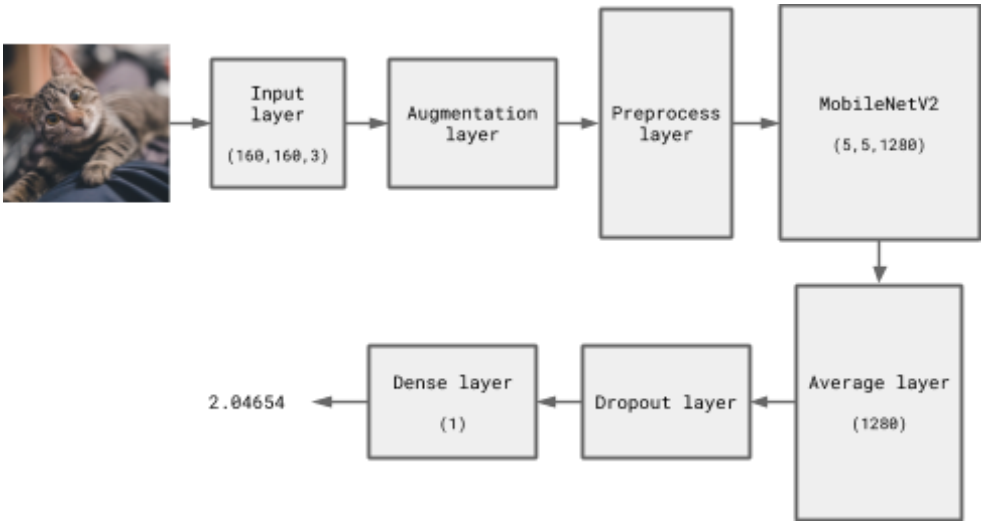

Figura 2 - Arquitetura do modelo desenvolvido.

Primeiro treinamento

Dense Layer

Inicialmente fizemos um treinamento do modelo para verificar como ela iria se comportar e definir as técnicas que utilizaremos para comparar os outros modelos. Além disso, este teste inicial também foi importante para conferir como esta rede iria se comportar em um celular Android.
O primeiro treinamento foi realizado somente nas camadas do **Dense layer**, totalizando 1.2K parâmetros para treinar. Neste treinamento as 2.5M variáveis do MobileNetV2 foram congeladas.
Antes do treinamento obtivemos uma *accuracy* de 0.4032 e *loss* de 0.9266, o que está de acordo com o esperado visto que o modelo estaria adivinhando. Após o treinamento da Dense layer obtivemos uma *accuracy* de 0.9427 e *loss* de 0.1437 no dataset de treino (nunca visto pelo modelo).
Abaixo é apresentado a curva *accuracy* e l*oss* durante este primeiro treinamento.

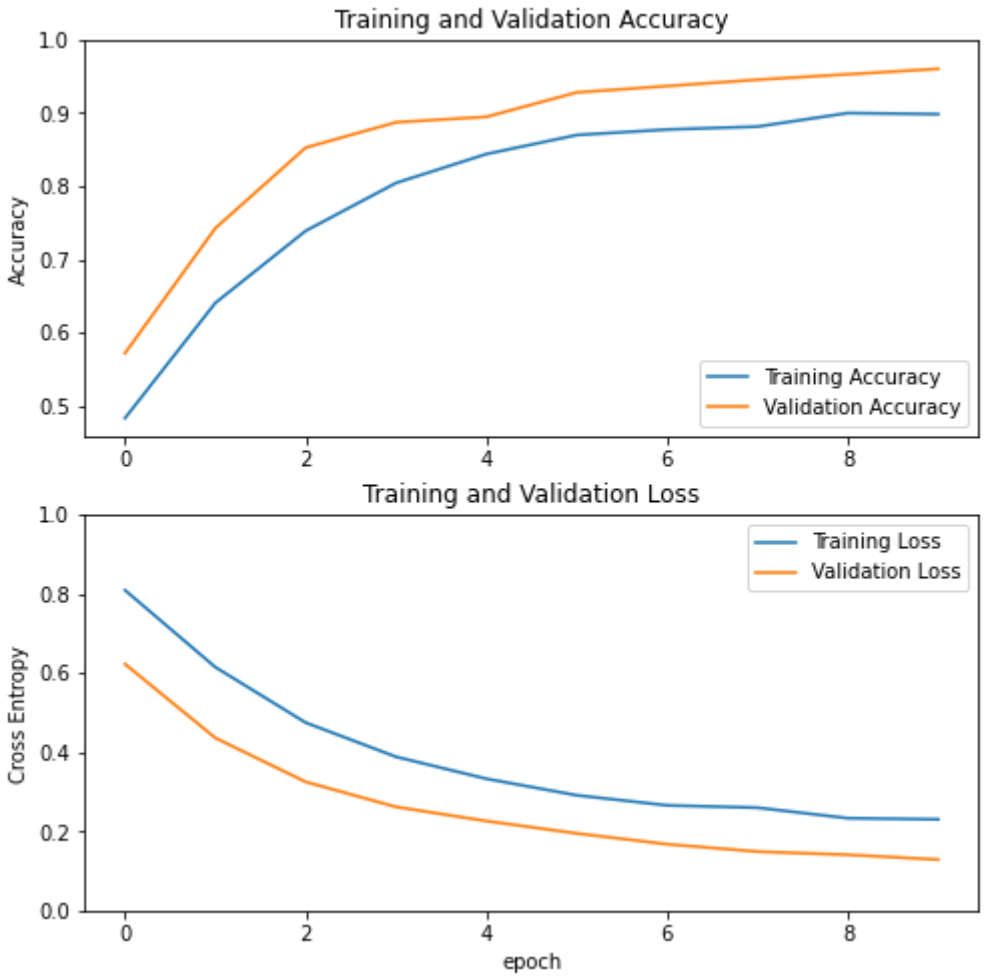

Figura 3 - Curvas após treinamento do Dense layer.

Fine-tuning com a MobileNetV2

Após este primeiro treinamento realizamos um *fine-tuning* na rede para tentar melhorar a acurácia. Nesta etapa descongelamos as últimas 50 camadas do MobileNetV2 e continuamos a treinar o modelo. Obtivemos uma *accuracy* de 0.9792 e *loss* de 0.0399 no dataset de teste. É possível perceber por estes resultados que o *fine-tuning* realmente funcionou para melhorar o modelo, visto que o mesmo está performando melhor em imagens nunca antes vistas. Na Figura 4 estão apresentadas as curvas com a etapa de *fine-tuning*.

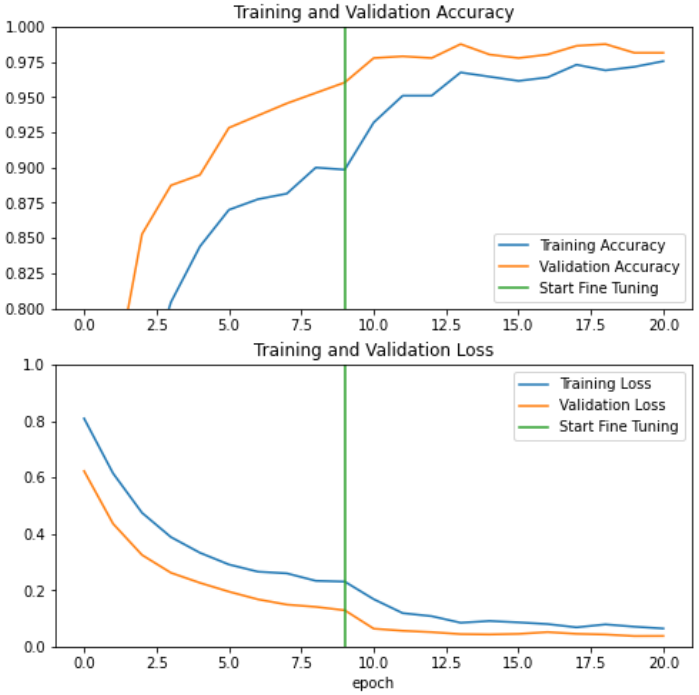

Figura 4 - Curvas após treinamento de *fine-tuning*. A linha verde separa os dois treinamentos.

Teste em embarcado

Após treinar este primeiro modelo resolvemos testar este em um celular[2] Android para verificar o tempo de classificação e se ocorreria perda de precisão. Para adaptamos um código exemplo do TFLite (TensorFlow Lite) para android, foi necessário primeiro exportar o modelo treinado no colab para `.tflite`, um formato entendido pela Lib do TFLite. Na Figura 5 é apresentado a interface do aplicativo executando com diferentes modelos. Em especial a primeira imagem mostra nossa modificação para executar o modelo desenvolvido.

[2] Foi utilizado um Samsung Galaxy S9+ nestes testes.

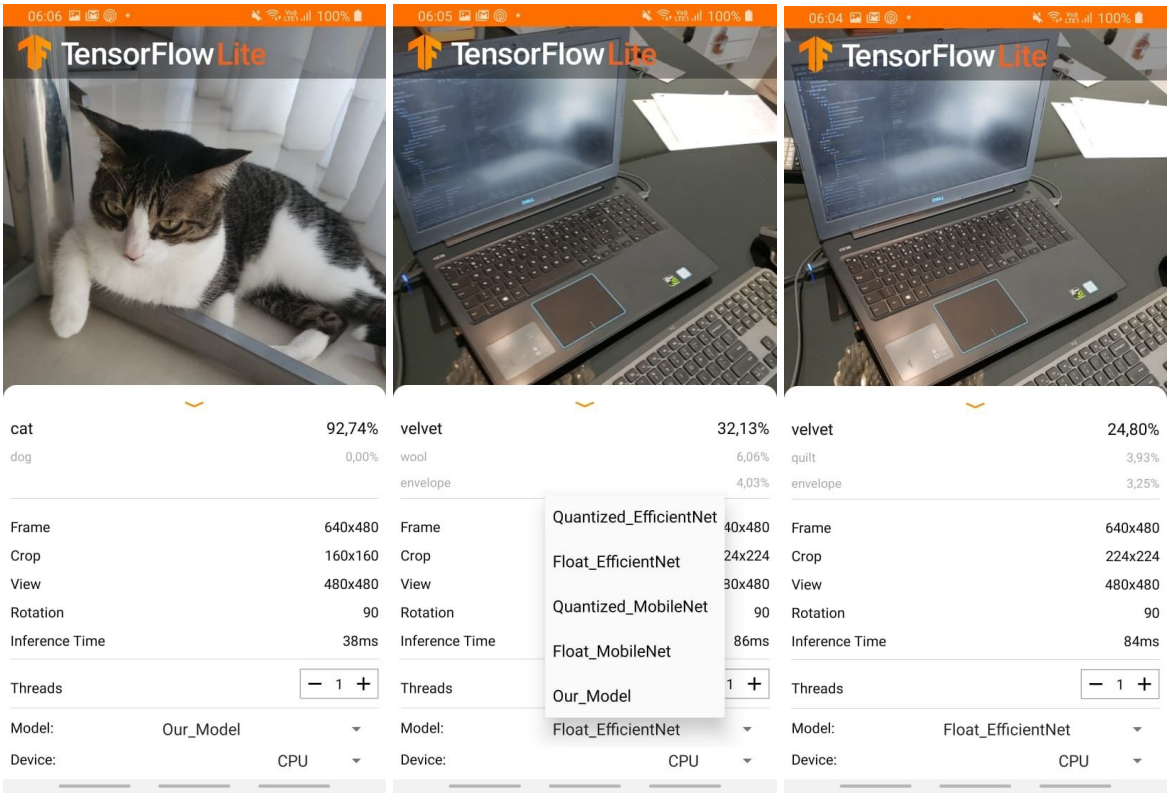

Figura 5 - Na primeira imagem é possível ver o output do nosso modelo. Na segunda o *combobox* de seleção de modelo, e na última um outro modelo executando.

Na Tabela 1 abaixo é apresentado o tempo de execução dos diferentes modelos no aplicativo.

Quantized_EfficientNet	100 ms
Float_EfficientNet	90 ms
Quantized_MobileNet	103 ms
Float_MobileNet	98 ms
Our_Model (MobileNetV2)	35 ms

Tabela 1 - Tempo para processar a imagem.

Uma dúvida que surgiu ao testar nosso modelo no celular é se ocorre alguma perda de dados dos pesos do modelo ao salvar como `.tflite`. Para conferir isto executamos a classificação utilizando o nosso modelo no colab e também o modelo carregado do `.tflite`. Obtivemos o output

-11.069564 no modelo do colab e -11.127892 no modelo do tflite, o que nos leva a deduzir que existe uma pequena perda de informação. Vale ressaltar que o resultado negativo significa que é um gato, visto que no dataset de treinamento gatos eram 0 e cachorros 1. O valor negativo vem do comportamento da função de loss utilizada no treinamento (*Binary Cross Entropy*).

Testes nos parâmetros do modelo

Neste momento treinamos a rede alterando alguns parâmetros para analisar o impacto destes sobre o resultado final. Para acelerar os testes não foi realizada a etapa de *fine-tuning*.

Retirar augmentation

Após retirar o layer de *augmentation* obtivemos resultados inesperados. Inicialmente pensávamos que o accuracy do modelo iria aumentar durante o treinamento pois as imagens seriam mais parecidas, o que aconteceu. Entretanto, como isto seria uma forma de overfitting no dataset, deveria ter um desempenho pior no dataset de teste (nunca visto), o que não aconteceu. Na realidade o desempenho foi superior. O modelo atingiu 0.9740 de *accuracy* e 0.0977 de *loss* no dataset de teste (contra 0.9427 de *accuracy* e 0.1437 de *loss* no modelo com augmentation).

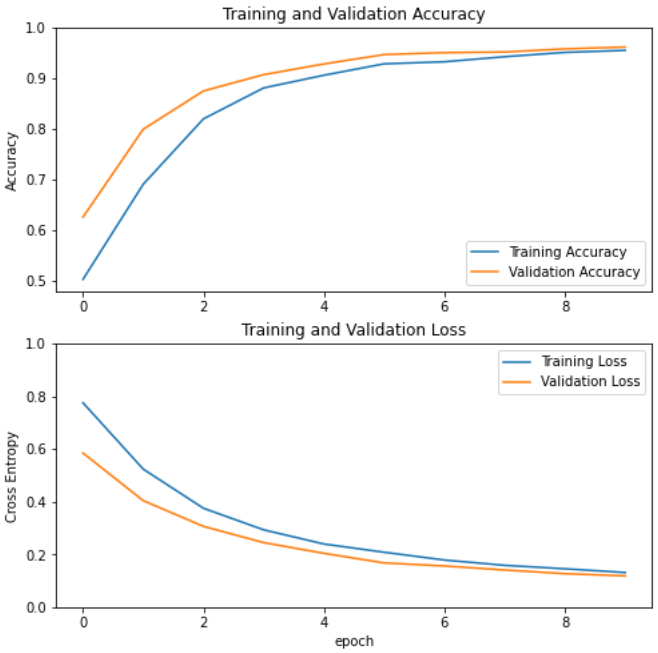

Figura 6 - Curvas de treinamento do modelo sem *augmentation*.

Aumentar *dropout*

Após aumentar o dropout de 20% para 50%, obtivemos um resultado pior que o obtido utilizando 20%. Obtivemos 0.9323 de *accuracy* e 0.1595 de *loss* no dataset de teste.

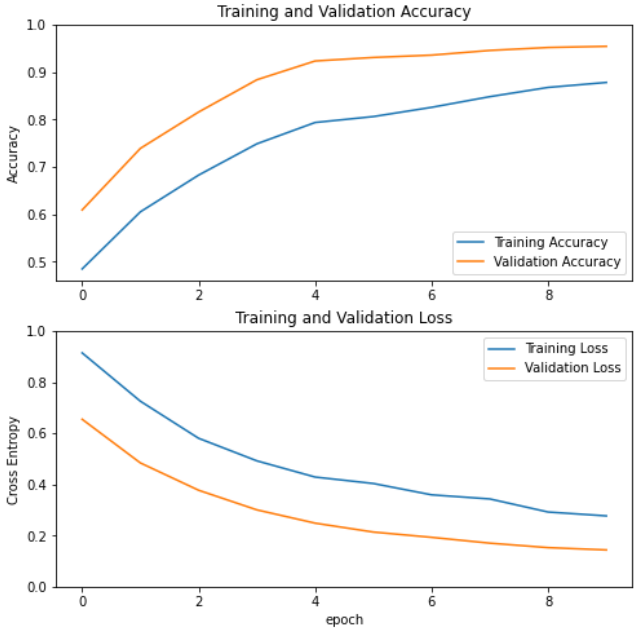

Figura 7 - Curvas de treinamento do modelo com *dropout* maior.

Modelo final

A partir destes testes decidimos aplicar o fine-tuning no modelo sem *augmentation* para utilizar como modelo definitivo no aplicativo do celular. Como aumentar o *dropout* resultou em perda no *accuracy,* mantivemos em 20% no modelo final.
Conseguimos atingir a melhor performance até o momento com este fine-tuning.
Atingindo *accuracy* de 0.9896 e *loss* de 0.0189 no dataset de teste (contra *accuracy* de 0.9792 e *loss* de 0.0399 no primeiro fine-tuning).
Vale ressaltar que é a primeira vez que as curvas de treinamento e validação se invertem neste trabalho. Isto indica que houve um overfitting do modelo no conjunto de dados de treinamento. Apesar disso, foi possível ver que este modelo está dando melhores resultados em um dataset nunca visto.

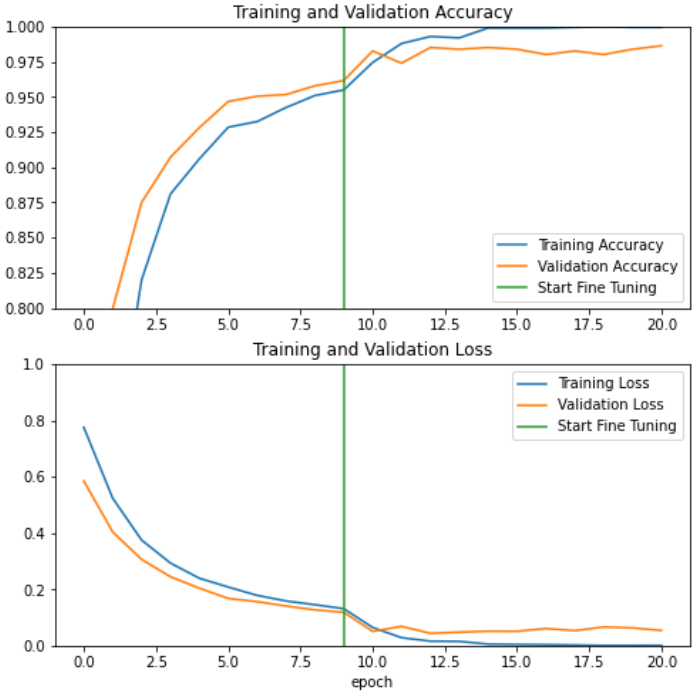

Figura 8 - Curvas de treinamento do modelo final.

Análise com animais reais

Nesta parte iremos analisar a performance de três modelos desenvolvidos neste trabalho a partir dos dados do aplicativo ao filmar um gato ou cachorro.

Os três modelos serão:

- Our_model -> Primeiro modelo
- No_Aug_Model -> Modelo sem *augmentation*
- No_Aug_Fine_Model -> Modelo sem *augmentation fine-tuned*

Para realizar estas análises gravamos um vídeo de 30 segundos com cada um dos três modelos com gato e com cachorro, sempre alterando a distância e orientação do celular, totalizando 6 vídeos [2]. As imagens a seguir foram retiradas dos videos.

Aplicativo com gato

Iremos analisar o comportamento dos modelos em três situações principais: quando a câmera está próxima, quando a câmera está distante, e quando o animal está ocluso por algum objeto.

Gato perto

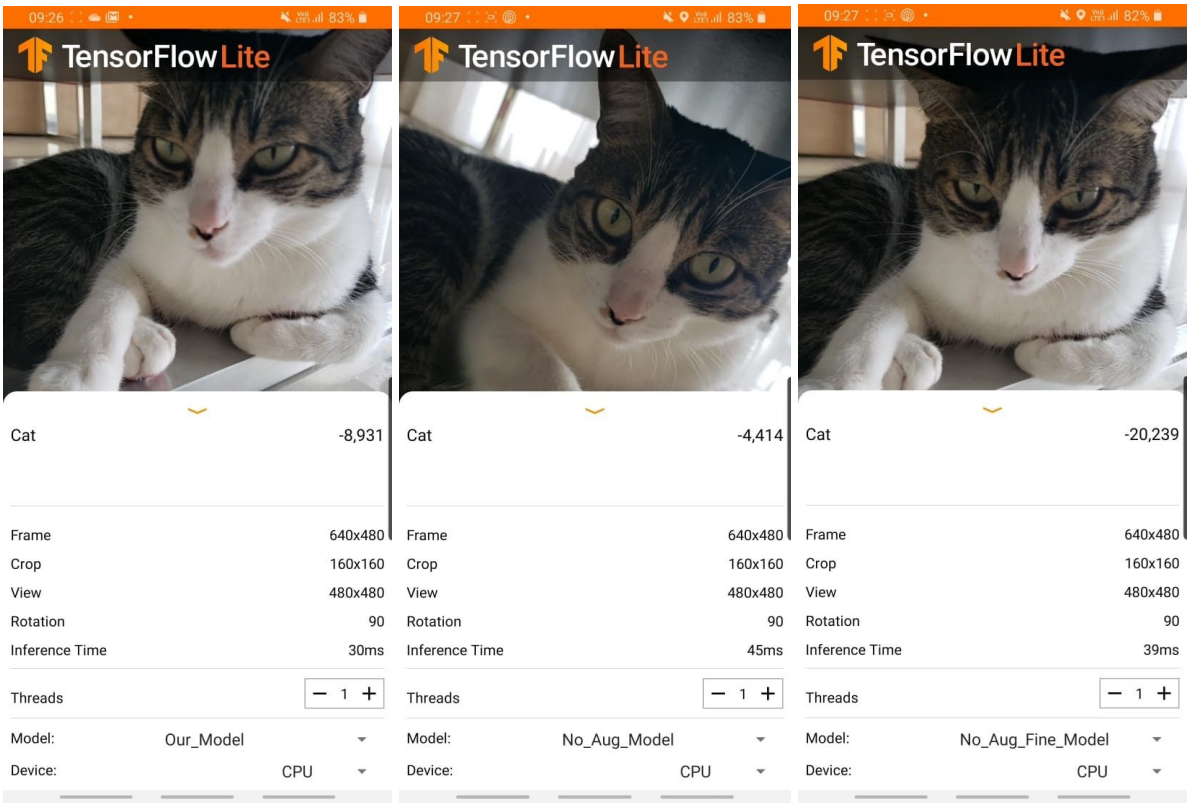

Figura 9 - Resultados de um gato próximo da câmera.

É possível perceber que o modelo fine-tuned ficou com um valor bem mais negativo que os outros. Acreditamos que isto tenha acontecido por causa do *overfitting* do modelo. O primeiro e o segundo modelos estão com valores aceitáveis neste caso.

Gato distante

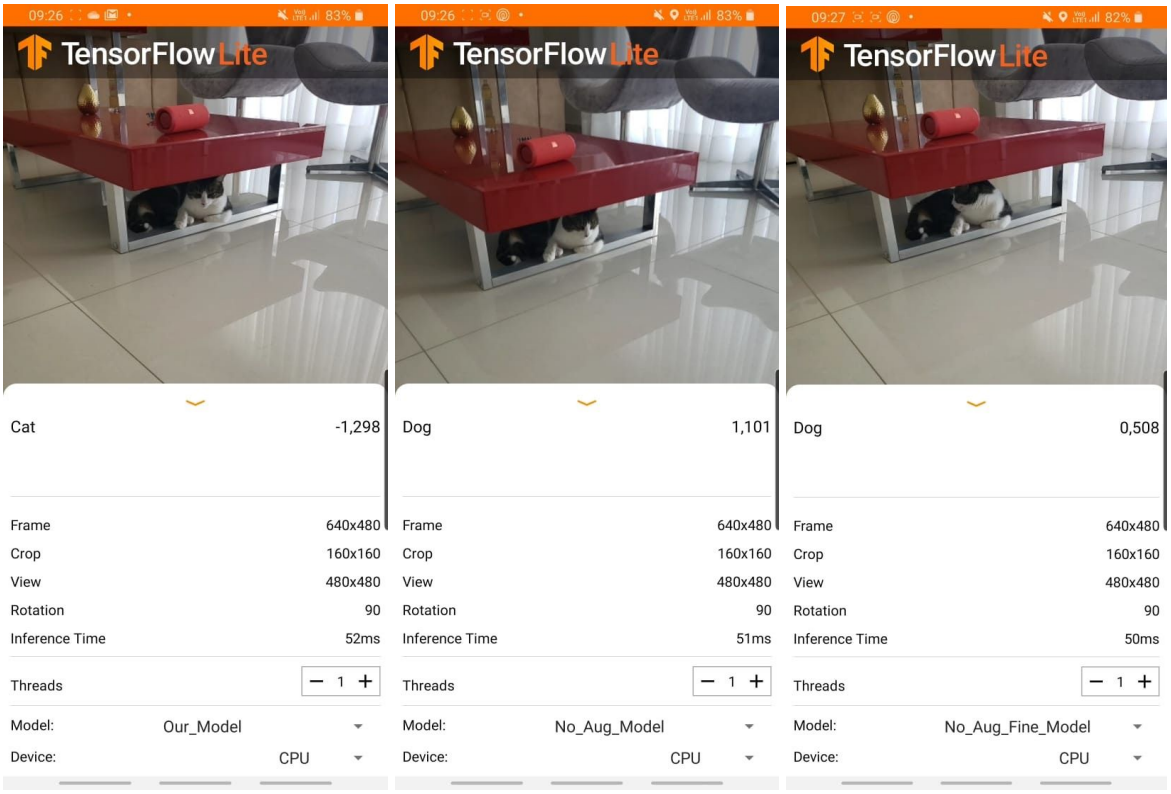

Figura 10 - Resultados de um gato longe da câmera.

No caso do gato distante, os modelos sem augmentation tiveram mais dificuldades em diferenciar entre gato e cachorro. O primeiro modelo conseguiu detectar bem mas o valor está próximo de 0.5, o que indica uma certa indecisão.

Gato ocluso

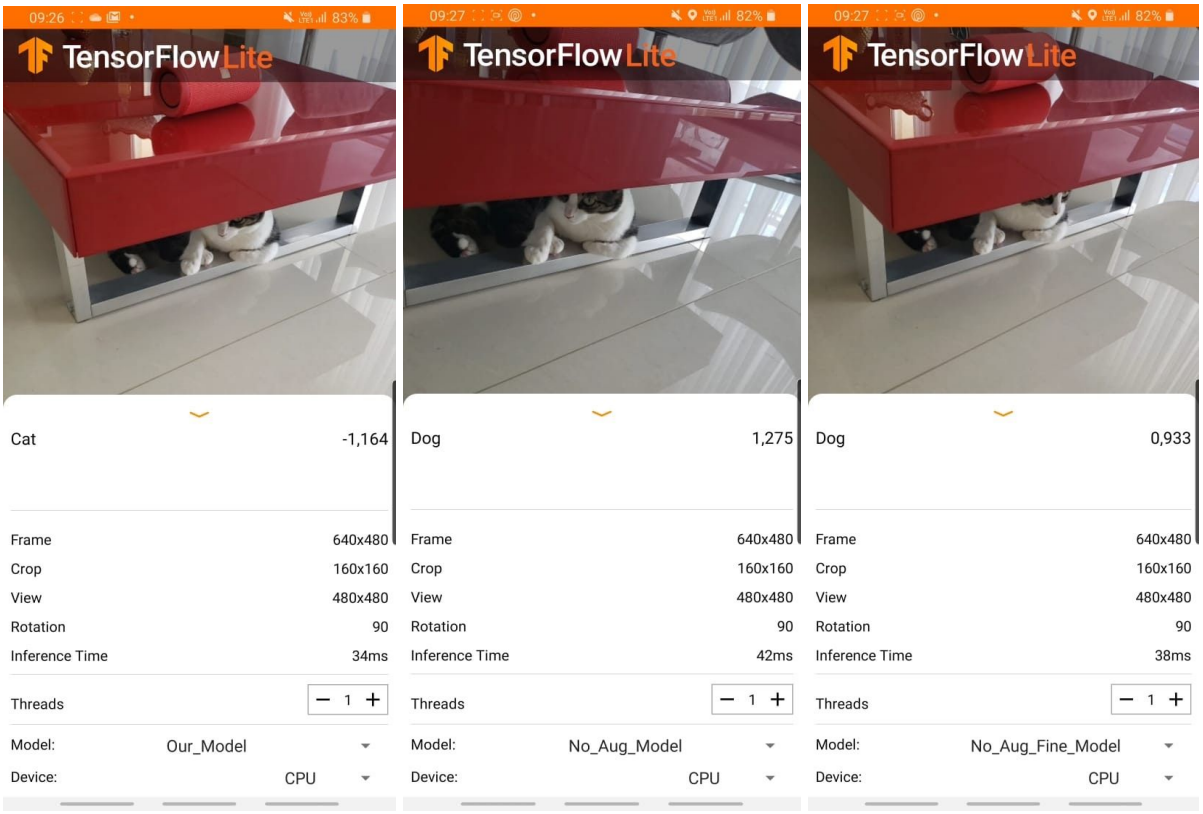

Figura 11 - Resultados de um gato ocluso por objeto.

No caso do gato ocluso por um objeto tivemos resultados semelhantes com o do gato distante. Após realizar testes com o caso ocluso perto percebemos que o resultado foi semelhante com o do gato perto, o que nos leva a concluir que a oclusão parcial não afeta muito o modelo.

Aplicativo com cachorro

Cachorro perto

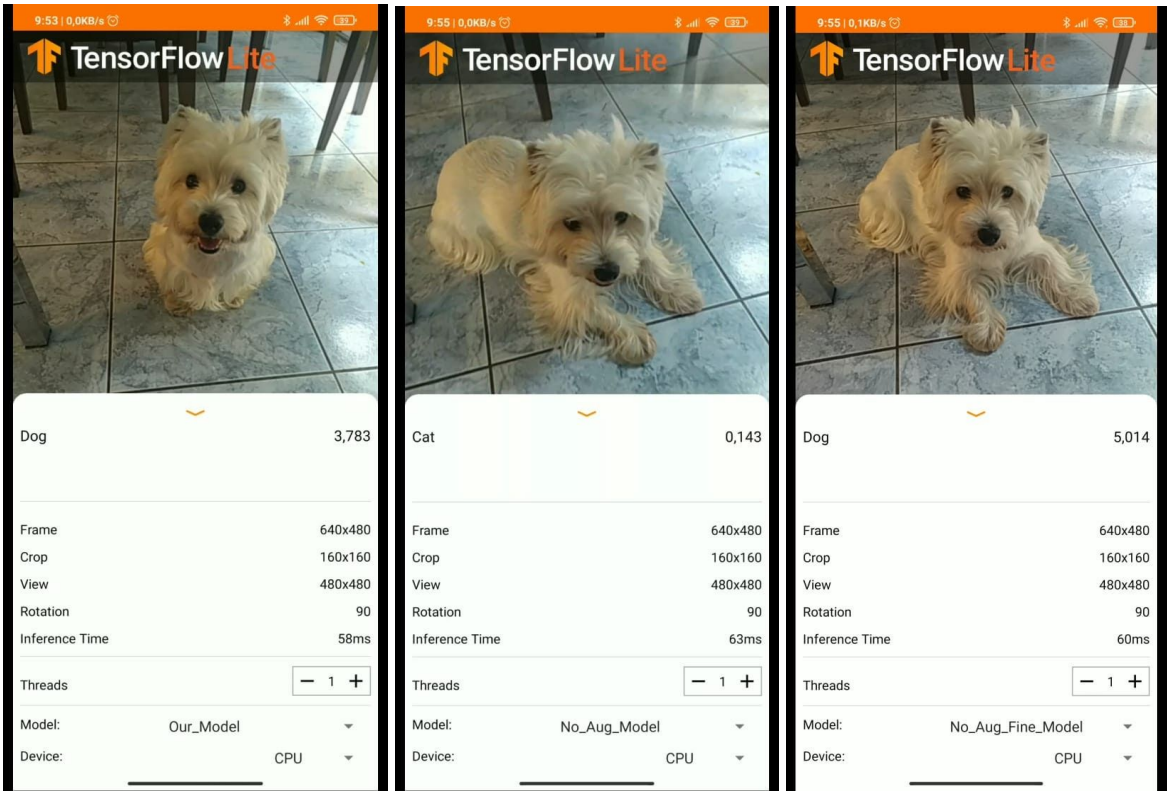

Figura 12 - Resultados de um cachorro perto.

O segundo modelo teve dificuldade em distinguir entre gato e cachorro mesmo com o cachorro próximo. Os outros dois modelos conseguiram identificar corretamente.

Cachorro distante

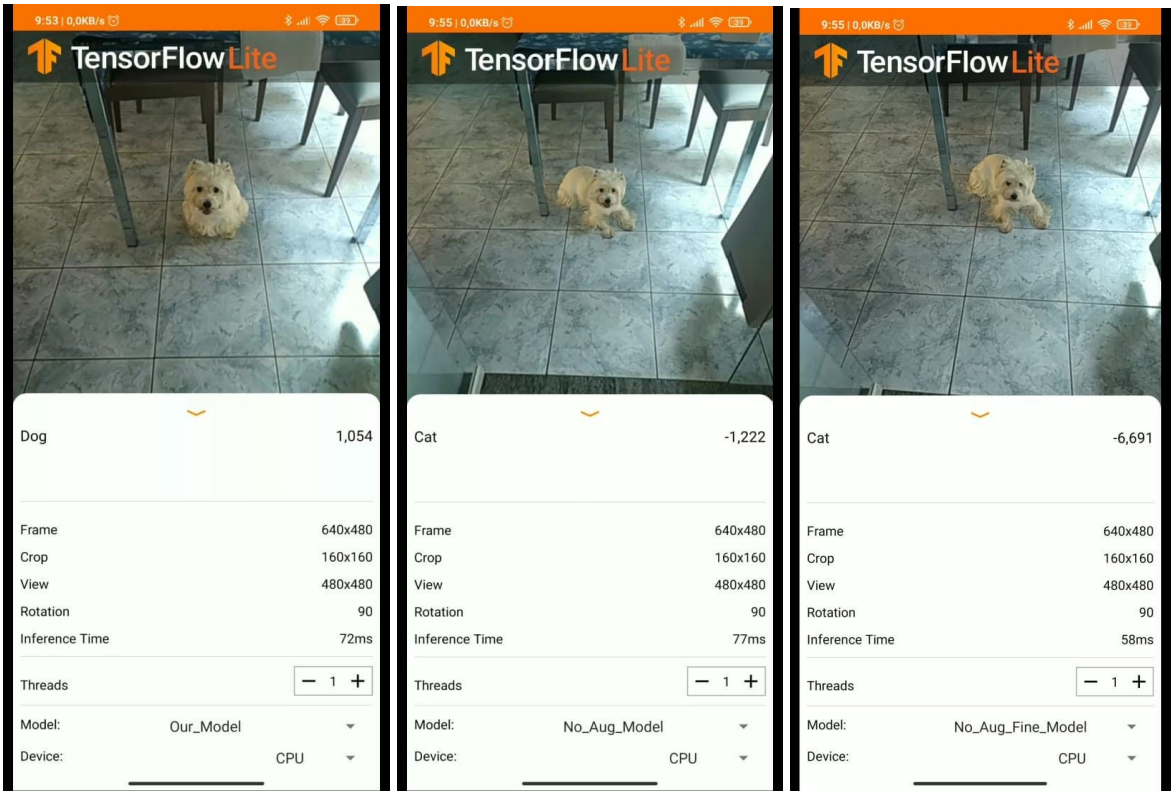

Figura 13 - Resultados de um cachorro distante.

Os últimos dois modelos tiveram bastante dificuldade em detectar se era um gato ou cachorro. O primeiro identificou mas ficou com variação entre gato e cachorro.

Cachorro ocluso

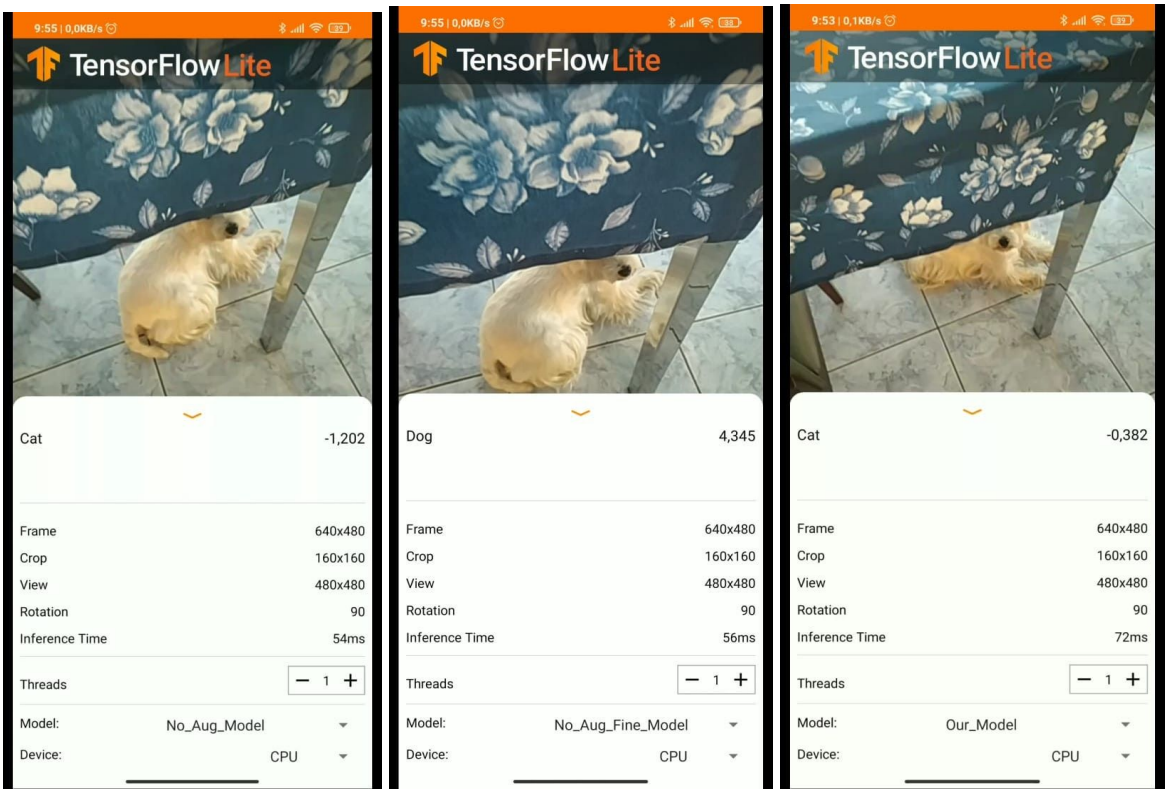

Figura 14 - Resultados de um cachorro ocluso por um objeto.

Neste caso os três modelos apresentaram bastante variação. O que obteve o melhor resultado foi o segundo, detectando gato somente quando o focinho do cachorro estava coberto.

Dificuldades encontradas

A principal dificuldade encontrada foi adaptar o aplicativo para executar o modelo treinado no colab. Durante nossos primeiros testes obtivemos resultados que não estavam condizentes com o esperado para o output do modelo. Posteriormente descobrimos que isto aconteceu pois o código exemplo aplicava transformações na imagem antes de enviar para o modelo (normalização dos valores dos da imagem e *resize-crop*). Após retirado estas transformações o modelo funcionou como esperado.

Conclusão

Foi possível utilizar a técnica de transfer learning para treinar uma rede neural para classificação de gatos e cachorros a partir do modelo MobileNetV2. Testamos o modelo com diferentes parâmetros para tentar encontrar um modelo mais eficiente e apresentamos os dados de convergência.
Após decidir por três modelos, testamos estes modelos em casos reais, detectando um gato e um cachorro. Os três modelos apresentaram variação ao tentar classificar os animais, mas conseguiram no geral na maior parte do tempo indicar o animal correto se a imagem não estiver muito desafiadora.
A partir dos resultados, o modelo que performou melhor foi o primeiro, isto provavelmente aconteceu pois os modelos sem *augmentation* se especializaram no conjunto de treinamento.
Futuramente para melhorar os modelos pretendemos re-treinar o modelo com mais imagens de gatos e cachorros para melhorar a precisão do classificador.

Referências

[1] Repositório do Github: https://github.com/Brenocq/CatDogClassification
[1] Vídeos dos resultados: https://youtu.be/HPxRi8cztHM

Apêndices

Treinamento dense layer

initial loss: 0.94
initial accuracy: 0.39
Epoch 1/10
63/63 [==============================] - 62s 940ms/step - loss: 0.8095 - accuracy: 0.4830 - val_loss: 0.6229 - val_accuracy: 0.5718
Epoch 2/10
63/63 [==============================] - 59s 933ms/step - loss: 0.6149 - accuracy: 0.6405 - val_loss: 0.4361 - val_accuracy: 0.7426
Epoch 3/10
63/63 [==============================] - 59s 932ms/step - loss: 0.4747 - accuracy: 0.7390 - val_loss: 0.3250 - val_accuracy: 0.8527
Epoch 4/10

63/63 [==============================] - 59s 937ms/step - loss: 0.3883 - accuracy: 0.8045 - val_loss: 0.2617 - val_accuracy: 0.8874
Epoch 5/10
63/63 [==============================] - 60s 955ms/step - loss: 0.3328 - accuracy: 0.8440 - val_loss: 0.2261 - val_accuracy: 0.8948
Epoch 6/10
63/63 [==============================] - 61s 969ms/step - loss: 0.2912 - accuracy: 0.8700 - val_loss: 0.1948 - val_accuracy: 0.9282
Epoch 7/10
63/63 [==============================] - 59s 938ms/step - loss: 0.2657 - accuracy: 0.8775 - val_loss: 0.1675 - val_accuracy: 0.9369
Epoch 8/10
63/63 [==============================] - 59s 941ms/step - loss: 0.2598 - accuracy: 0.8815 - val_loss: 0.1489 - val_accuracy: 0.9455
Epoch 9/10
63/63 [==============================] - 60s 945ms/step - loss: 0.2332 - accuracy: 0.9000 - val_loss: 0.1409 - val_accuracy: 0.9530
Epoch 10/10
63/63 [==============================] - 60s 941ms/step - loss: 0.2304 - accuracy: 0.8985 - val_loss: 0.1287 - val_accuracy: 0.9604

Treinamento fine-tuning

Epoch 10/20
63/63 [==============================] - 88s 1s/step - loss: 0.1992 - accuracy: 0.9103 - val_loss: 0.0634 - val_accuracy: 0.9777
Epoch 11/20
63/63 [==============================] - 81s 1s/step - loss: 0.1182 - accuracy: 0.9519 - val_loss: 0.0559 - val_accuracy: 0.9790
Epoch 12/20
63/63 [==============================] - 82s 1s/step - loss: 0.1085 - accuracy: 0.9489 - val_loss: 0.0511 - val_accuracy: 0.9777
Epoch 13/20
63/63 [==============================] - 82s 1s/step - loss: 0.0874 - accuracy: 0.9695 - val_loss: 0.0441 - val_accuracy: 0.9876
Epoch 14/20
63/63 [==============================] - 82s 1s/step - loss: 0.0894 - accuracy: 0.9635 - val_loss: 0.0429 - val_accuracy: 0.9802
Epoch 15/20
63/63 [==============================] - 82s 1s/step - loss: 0.0871 - accuracy: 0.9645 - val_loss: 0.0450 - val_accuracy: 0.9777
Epoch 16/20
63/63 [==============================] - 81s 1s/step - loss: 0.0778 - accuracy: 0.9644 - val_loss: 0.0514 - val_accuracy: 0.9802
Epoch 17/20
63/63 [==============================] - 81s 1s/step - loss: 0.0702 - accuracy: 0.9732 - val_loss: 0.0452 - val_accuracy: 0.9864
Epoch 18/20

63/63 [==============================] - 83s 1s/step - loss: 0.0766 - accuracy: 0.9681 - val_loss: 0.0427 - val_accuracy: 0.9876
Epoch 19/20
63/63 [==============================] - 82s 1s/step - loss: 0.0735 - accuracy: 0.9706 - val_loss: 0.0372 - val_accuracy: 0.9814
Epoch 20/20
63/63 [==============================] - 82s 1s/step - loss: 0.0560 - accuracy: 0.9805 - val_loss: 0.0375 - val_accuracy: 0.9814

Treinamento augmentation

initial loss: 0.91
initial accuracy: 0.34
Epoch 1/10
63/63 [==============================] - 55s 835ms/step - loss: 0.7755 - accuracy: 0.5035 - val_loss: 0.5849 - val_accuracy: 0.6262
Epoch 2/10
63/63 [==============================] - 53s 836ms/step - loss: 0.5238 - accuracy: 0.6910 - val_loss: 0.4041 - val_accuracy: 0.7995
Epoch 3/10
63/63 [==============================] - 53s 836ms/step - loss: 0.3752 - accuracy: 0.8200 - val_loss: 0.3064 - val_accuracy: 0.8750
Epoch 4/10
63/63 [==============================] - 53s 843ms/step - loss: 0.2933 - accuracy: 0.8810 - val_loss: 0.2448 - val_accuracy: 0.9072
Epoch 5/10
63/63 [==============================] - 53s 835ms/step - loss: 0.2394 - accuracy: 0.9060 - val_loss: 0.2041 - val_accuracy: 0.9282
Epoch 6/10
63/63 [==============================] - 53s 838ms/step - loss: 0.2082 - accuracy: 0.9285 - val_loss: 0.1676 - val_accuracy: 0.9468
Epoch 7/10
63/63 [==============================] - 53s 834ms/step - loss: 0.1786 - accuracy: 0.9325 - val_loss: 0.1561 - val_accuracy: 0.9505
Epoch 8/10
63/63 [==============================] - 53s 836ms/step - loss: 0.1586 - accuracy: 0.9425 - val_loss: 0.1408 - val_accuracy: 0.9517
Epoch 9/10
63/63 [==============================] - 53s 837ms/step - loss: 0.1454 - accuracy: 0.9510 - val_loss: 0.1269 - val_accuracy: 0.9579
Epoch 10/10
63/63 [==============================] - 53s 845ms/step - loss: 0.1313 - accuracy: 0.9550 - val_loss: 0.1185 - val_accuracy: 0.9616

Treinamento maior dropout

initial loss: 0.91
initial accuracy: 0.43

Epoch 1/10
63/63 [==============================] - 61s 916ms/step - loss: 0.9150 - accuracy: 0.4845 - val_loss: 0.6552 - val_accuracy: 0.6089
Epoch 2/10
63/63 [==============================] - 57s 904ms/step - loss: 0.7258 - accuracy: 0.6050 - val_loss: 0.4843 - val_accuracy: 0.7389
Epoch 3/10
63/63 [==============================] - 58s 915ms/step - loss: 0.5807 - accuracy: 0.6825 - val_loss: 0.3773 - val_accuracy: 0.8156
Epoch 4/10
63/63 [==============================] - 57s 903ms/step - loss: 0.4924 - accuracy: 0.7485 - val_loss: 0.3005 - val_accuracy: 0.8837
Epoch 5/10
63/63 [==============================] - 57s 900ms/step - loss: 0.4289 - accuracy: 0.7935 - val_loss: 0.2485 - val_accuracy: 0.9233
Epoch 6/10
63/63 [==============================] - 57s 902ms/step - loss: 0.4037 - accuracy: 0.8060 - val_loss: 0.2136 - val_accuracy: 0.9307
Epoch 7/10
63/63 [==============================] - 57s 909ms/step - loss: 0.3593 - accuracy: 0.8255 - val_loss: 0.1930 - val_accuracy: 0.9356
Epoch 8/10
63/63 [==============================] - 57s 908ms/step - loss: 0.3432 - accuracy: 0.8480 - val_loss: 0.1702 - val_accuracy: 0.9455
Epoch 9/10
63/63 [==============================] - 58s 911ms/step - loss: 0.2920 - accuracy: 0.8675 - val_loss: 0.1527 - val_accuracy: 0.9517
Epoch 10/10
63/63 [==============================] - 57s 907ms/step - loss: 0.2770 - accuracy: 0.8780 - val_loss: 0.1436 - val_accuracy: 0.9542

Treinamento *augmentation fine-tuning*

Epoch 10/20
63/63 [==============================] - 82s 1s/step - loss: 0.0790 - accuracy: 0.9697 - val_loss: 0.0506 - val_accuracy: 0.9827
Epoch 11/20
63/63 [==============================] - 78s 1s/step - loss: 0.0310 - accuracy: 0.9872 - val_loss: 0.0684 - val_accuracy: 0.9740
Epoch 12/20
63/63 [==============================] - 78s 1s/step - loss: 0.0134 - accuracy: 0.9935 - val_loss: 0.0438 - val_accuracy: 0.9851
Epoch 13/20
63/63 [==============================] - 77s 1s/step - loss: 0.0118 - accuracy: 0.9952 - val_loss: 0.0479 - val_accuracy: 0.9839
Epoch 14/20
63/63 [==============================] - 77s 1s/step - loss: 0.0048 - accuracy: 0.9996 - val_loss: 0.0514 - val_accuracy: 0.9851
Epoch 15/20

63/63 [==============================] - 77s 1s/step - loss: 0.0048 - accuracy: 0.9993 - val_loss: 0.0509 - val_accuracy: 0.9839
Epoch 16/20
63/63 [==============================] - 78s 1s/step - loss: 0.0022 - accuracy: 0.9994 - val_loss: 0.0607 - val_accuracy: 0.9802
Epoch 17/20
63/63 [==============================] - 79s 1s/step - loss: 0.0011 - accuracy: 0.9998 - val_loss: 0.0532 - val_accuracy: 0.9827
Epoch 18/20
63/63 [==============================] - 79s 1s/step - loss: 4.1906e-04 - accuracy: 1.0000 - val_loss: 0.0663 - val_accuracy: 0.9802
Epoch 19/20
63/63 [==============================] - 78s 1s/step - loss: 6.3731e-04 - accuracy: 0.9993 - val_loss: 0.0630 - val_accuracy: 0.9839
Epoch 20/20
63/63 [==============================] - 78s 1s/step - loss: 3.4948e-04 - accuracy: 0.9998 - val_loss: 0.0539 - val_accuracy: 0.9864