Atividade 4: Predizer a raça de um cachorro

Jonlenes Castro
Camila Moura
Anderson Rocha

Resumo

Por intermédio da deep learning foi possível realizar a predição das raças de cachorros analisando imagens. Este método explora as técnicas de transfer learning, meta learning e data augmentation para realizar a classificação. Com o melhor modelo, obtido a partir dos experimentos, obteve-se a accucary de aproximadamente 97%. No decorrer deste trabalho estão apresentados os modelos treinados, a composição da Convolutional Neural Network (CNN), os experimentos realizados, resultados e conclusões.

1. Introdução

O método de classificação será utilizado na identificação da raça de um cachorro, onde cada imagem do *dataset* será classificada baseada em suas características. Para tanto serão explorados diversos modelos para classificação utilizando *Deep Learning*.

Composto por 19742 imagens, o *dataset* utilizado neste trabalho, esta dividido em três conjuntos: treinamento com 8300, validação com 6022 e testes com 5420.

2. Atividades

Com o intuito de identificar a raça de um cachorro, foram realizadas e apresentadas as seguintes atividades:

- Construção de uma *Convolutional Neural Network* (CNN) básica para realizar a classificação;
- *Feature extration* com CNN;
- Transfer Learning
- Data augmentation
- Meta Learning

3. Construção de uma CNN básica

A construção e o treinamento da CNN, foram auxiliadas pela biblioteca *Keras* (*The Python Deep Learning library*), por ser esta uma API de alto nível para redes neurais [1].

A CNN criada para a classificação das raças de cachorros é composta por uma imagem de entrada com tamanho 224 x 224 pixels, 2 camadas de tipos distintos que se repetem, sendo uma de *convolution* e a outra de *max pooling*, seguida por uma camada de *global average pooling* e uma camada *fully-connected* que gera o *output*. A representação da arquitetura desta CNN pode ser visualizada na Figura 1.

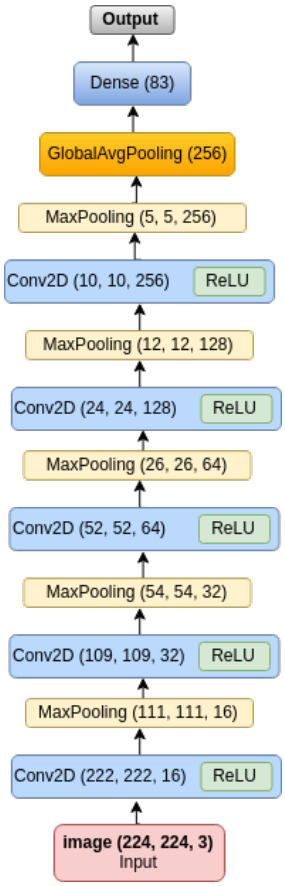

Figura 1. Arquitetura CNN

A camada de *convolution* utiliza um *kernel* de tamanho 3, *strides* com tamanho 1 para altura e largura, e função de ativação *ReLU*. Inicialmente com 16 filtros de saída (*depth*),

que serão enviados para a próxima camada acrescido com vetores de *bias*. Em seguida, o modelo utiliza a camada *max pooling* com *pool size* e *stride* ambos iguais a 2. Ressalta-se que estas camadas são repetidas 4 vezes, sendo alterado o *depth* em cada *convolution*.

Quanto a camada *fully-connected*, a função de ativação *softmax* é utilizada para produz, a partir de um volume de entrada, um vetor N dimensional, onde N é o número de classes [2].

As imagens de entrada foram redimensionadas e repassada para o modelo, que obteve uma *accuracy* de 12.26%, após o treinamento por 5 *epochs* e 10 exemplos por passe (*batch size*). Treinando o mesmo modelo por 10 *epochs* e 20 *batchs* obteve-se uma *accuracy* de 17.87%.

Ainda neste modelo, adicionou-se uma camada de normalização (*Batch Normalization*) após cada camada de *pooling*, o resultado apresentou melhoria com 27.13% de *accuracy*.

O próximo passo realizado visando melhorar o processo de classificação foi a utilização de modelos prontos e pré-treinados com outro *dataset* para a extração *features* e a realização de *transfer learning*, conforme apresentado a seguir.

4. Modelos pré-treinados

Nesse trabalho foram exploradas a utilização de modelos de *deep learning* pré-treinados para a extração de *features* e *transfer learning*.

4.1. Features extraction com CNN

O primeiro modelo utilizado foi o ResNet50 pré-treinado com o *dataset* ImageNet [3].

O ResNet50, também disponibilizado na *Keras*, foi carregado e utilizado para realizar o *features extraction*. Para isso foi necessário remover sua camada *fully-connected* e os 2048 valores que eram repassados para esta camada foram utilizados como *features*.

Após a extração de *features* para cada imagem, foi realizado o *scaling* desses dados e os mesmos foram treinados com os classificadores listados na Tabela 1, com as respectivas *accuracy* normalizada.

Tabela 1. Resultados da classificação

Classificador	Accuracy
Nearest Neighbors	61.50%
Linear SVM	76.53%
RBF SVM	69.19%
Decision Tree	48.33%
Random Forest	81.61%
Neural Network	79.75%
Naive Bayes	69.86%
Logistic Regression	78.82%

Como pode ser observado, a utilização de modelos pré-treinados pode auxiliar no processo de classificação, pois foi obtido 81.61% de *accuracy* sobre o conjunto de validação. No próximo passo será utilizada a técnica de *transfer learning* na busca por melhores resultados.

4.2. Transfer Learning

O *transfer learning* é uma técnica de aprendizado de máquina em que um modelo treinado em uma tarefa é redirecionado outra tarefa relacionada [4].

Uma das maneiras para realizar *transfer learning* é por meio do *fine-tuning*, que consiste no processo de ajustar as redes existentes, treinadas em outro *dataset*, e continuar o treinamento com *dataset* desejado [5].

Neste trabalho foram utilizados os modelos ResNet50, InceptionV3 (IncV3) e Xception (Xcp) pré-treinados com o *dataset* ImageNet. Para todos os modelos, o processo de *fine-tuning* realizado foi similar, consistindo na remoção da última camada do modelo original e na adição de novas camadas para realizar o treinamento com o *dataset* deste trabalho.

Para determinar as novas camadas a ser adicionadas ao modelo, foram realizados alguns experimentos, sendo estes apresentados em seguida.

Experimento 1 - Fine-tuning 1

Este experimento consiste em receber o *output* da última camada do modelo pré-treinado (sem o topo), adicionar uma camada de *glabal average pooling*, uma camada *fully-connected* com 512 saídas e ativação *ReLU* e, por fim, uma camada de *output* com 83 saídas e função de ativação *softmax*.

Ao final deste processo, todas as camadas do modelo pré-treinado foram congeladas e as novas camadas adicionadas foram treinadas por uma *epoch*. Em seguida, descongela-se 10% das últimas camadas e realiza-se novo treinamento por mais 75 *epochs*, obtendo-se os resultados apresentados na Tabela 2.

Tabela 2. Experimento 1

Modelos	Epochs	Accuracy val
ResNet50	75	81.45%
InceptionV3	75	83.88%
Xception	75	84.18%

Os resultados obtidos com este experimento foi superior aos encontrados utilizando a CNN para a extração de *features*, chegando a 84.18%.

Experimento 2 - Fine-tuning 2

Neste experimento utiliza-se a mesma configuração do Experimento 1, mas remove-se todas as camadas existentes entre a camada de *pooling* e a de *output*, ou seja, foi

adicionada somente uma camada após a *output* do modelo pré-treinado. O resultado deste experimento pode ser visualizados na Tabela 3

Tabela 3. Experimento 2

Modelos	Epochs	Accuracy val
ResNet50	3	79.49%
InceptionV3	3	83.81%
Xception	3	84.24%

Os resultados encontrados com este experimento foram equivalentes aos do experimento anterior. Como o modelo deste experimento é mais simples, então optou-se por continuar os testes com este.

Experimento 3 - Taxa de congelamento

Este experimento visa identificar a melhor taxa de congelamento das camadas pré-treinadas da rede, utilizando o modelo do Experimento 2. Neste intuito, foram considerados alguns percentuais de congelamento - o percentual congela as primeiras X% camadas da rede - conforme apresentado na Tabela 4.

Tabela 4. My caption

Percentual	Epochs	ResNet50	IncV3	Xcep
90%	3	79.49%	83.81%	84.24%
75%	3	78.49%	82.46%	84.02%
50%	3	79.23%	82.35%	83.41%
25%	3	79.18%	81.47%	82.93%

Novamente os resultados foram bem similares aos anteriores, no entanto, quanto maior a quantidade de camadas descongeladas, maior é a necessidade de poder computacional para realizar o treinamento, logo optou-se por descongelar uma quantidade pequena de camada. Outra observação importante, foi o aumento da *accuracy* sobre o próprio conjunto de treinamento e uma diminuição da *accuracy* sobre o conjunto de validação, o que caracteriza um possível *overfitting*, então o próximo experimento visa explorar a utilização de camadas de *dropout* juntamente com diferentes taxas de congelamento.

Experimento 4 - Dropout

Nesse experimento foram adicionadas camadas de *dropout* após camadas *dense*. Na Tabela 5 são mostrados os resultados considerando diferentes *epocs* e camadas de *dropout*. Nesta tabela 0.75 significa uma camada de *dropout* de 0.50 seguida por uma *dense* e outro *dropout* de 0.25. A coluna *Perc* representa o percentual de congelamento, conforme Experimento 3.

Tabela 5. Adicionando dropout

Perc	Epochs	Dropout	ResNet50	IncV3	Xcep
25%	3	0.25	78.8%	81.8%	81.8%
25%	3	0.50	79.5%	81.9%	82.3%
25%	3	0.75	75.2%	77.9%	81.0%
25%	20	0.75	81.3%	81.5%	84.7%
30%	10	0.75	79.9%	82.9%	84.0%

As melhorias encontradas com este experimento não justificam o aumento da complexidade do modelo, optando-se assim por não utilizar nenhuma camada de *dropout* até o momento.

Experimento 5 - Optimizers

Até o momento, a atualização dos pesos das novas camadas adicionadas foram realizadas utilizando o *optimizer rmsprop* e o treinamento após o descongelamento dos pesos era realizado com SGD. Para este experimento alguns testes com a função de *optimizer* foram realizados. Ao trocar o *optimizer* a *accuracy* sobre o conjunto de treinamento aumentou bastante (alcançando 98%) e a *accuracy* sobre a validação diminui bastante não ultrapassando 50%, o que caracteriza *overfitting*.

Para tentar resolver esse problema, foram acrescentadas algumas camadas de *dropout* e *data augumentation*, descritas na próxima seção.

5. Data augmentation

Para evitar *overffiting* e ajudar na generalização do modelo, foi utilizado *data augmentation*. Tendo como transformações utilizadas neste trabalho:

- Rotação entre $-40°$ e $40°$;
- *Shift* (altura e largura), com taxa de 20% do tamanho;
- *Zoom* com taxa de 20%;
- Flip horizontal e vertical;
- *Shear* com intensidade de 20%

Para realizar este processo também foi utilizada a biblioteca *Keras*, que modifica os dados através de várias transformações aleatórias, de tal modo que o modelo nunca utilize duas vezes a mesma imagem [6].

Todos os modelos treinados até o momento foram retreinados considerando a alteração da função de *optimizer*, adicionando camadas de *dropout* e utilizando *data augmentation*.

Com isso, foi possível resolver o problema de *overffiting*, no entanto, os resultados não foram significativamente melhores do que os obtidos em outros experimentos, como apresentado na Tabela 6.

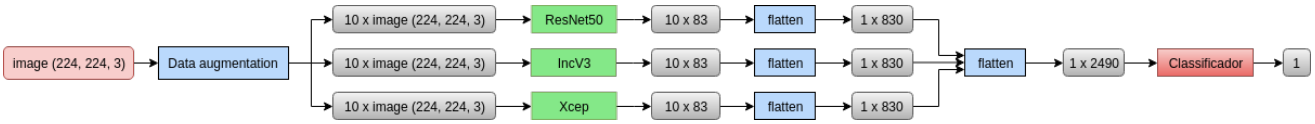

Figura 2. Meta learning com Data augmentation.

Tabela 6. Data augmentation

Modelos	Accuracy train	Accuracy val
ResNet50	71.17%	79.19%
InceptionV3	76.91%	81.24%
Xception	78.08%	82.63%

6. Resultados preliminares dataset de teste

Os três melhores modelos treinados de cada rede foram selecionados e utilizados para fazer a predição do conjunto de teste. Os resultados podem ser vistos na Tabela 7.

Tabela 7. Resultados

Modelos	Accuracy val	Accuracy test
ResNet50	81.45%	85.23%
InceptionV3	83.88%	89.47%
Xception	84.70%	89.55%

7. Meta Learning

Nesta etapa, foram utilizados os três melhores modelos da seção anterior e obtidas as distribuições de probabilidade para cada exemplo de treinamento (predição). Cada modelo gera 83 valores, então estes valores foram concatenados para os três modelos, tendo um total de 249 valores para cada exemplo.

Esses valores foram utilizados como *features* para treinar outros classificadores, obtendo os seguintes resultados para o conjunto de *test* (Tabela 8).

Tabela 8. Meta learning

Classificador	Accuracy
Linear SVM	91.88%
Random Forest	89.57%
Neural Network	90.09%
Logistic Regression	91.42%

Com a utilização de *meta learning*, obteve-se uma melhora de aproximadamente 2% nas predições sobre o conjunto de *test*, sendo este o melhor resultado obtido neste trabalho até o momento.

8. Meta Learning com Data augmentation

Também foi testado a utilização de *data augmentation* com *meta Learning*. A Figura 2 representa o experimento realizado.

Para cada imagem do *dataset* foram geradas 30 novas imagens utilizando *data augmentation*, onde cada nova imagem representa um subconjunto de transformações aleatório, conforme Seção 5.

Dez imagens foram repassadas para cada um dos três melhores modelos, definidos na Seção 6, que por sua vez, gera um vetor de saída com 83 valores para cada imagem, representando o vetor de distribuição de probabilidade.

Os vetores de probabilidades de cada imagem foram concatenados, como há 10 imagens por modelo tem-se 10 vetores resultando em 830 valores, como são três modelos totaliza-se 2490 valores. Logo, as imagens do *dataset* foram convertidas em 2490 valores, que por sua vez foram utilizadas como *features* no treinamento do classificador. O resultado desse experimento pode ser visto na Tabela 9

Tabela 9. Meta learning

Classificador	Accuracy	f1 score
Linear SVM	94.48%	94.34%
Random Forest	94.39%	94.25%
Neural Network	96.84%	96.55%
Logistic Regression	95.70%	95.45%

9. Resultados

O melhor resultado encontrado nesse trabalho consiste na *accuracy* normalizada de aproximadamente 97%, e *f1-score* de 96%, conforme seções anteriores. Um resumo do percentual por classe por ser visto na Tabela **??**, onde a coluna quantidade (qtd) representa quantas classes obtiveram aquele percentual de *accuracy*.

Tabela 10. Classes por accuracy

Qtd	**Accuracy**	Qtd	Accuracy
1	11%	2	94%
1	53%	1	95%
1	64%	4	96%
2	88%	1	97%
1	90%	15	98%
1	91%	6	99%
1	92%	43	100%
3	93%		

10. Conclusão e Trabalhos Futuros

Conforme os experimentos realizados é possível realizar a predição da raça de um cachorro com bons percentuais de

acertos. Porém, algumas classes tiveram o índice de acerto muito abaixo das demais, como por exemplo, nas classes 78 e 82 (possivelmente Yorkshire Terrier vs. Silk Terrier Australiano), com *accuracy* de 11% e 56% respectivamente.

Uma das análises realizadas no decorrer deste trabalho foi a possibilidade de o problema acontecer devido as duas raças serem bastante parecidas e de porte pequeno, fazendo com que a predição fosse realizada incorretamente para vários exemplos.

Apesar do modelo constuido apresentar boa *accuracy*, acredita-se que seja possível aprimorá-lo ainda mais, principalmente na realização do *fine-tuning* que obteve *accuracy* entre 80% e 90%, sendo esta uma das sugestões para trabalhos futuros.

Referências

[1] keras team. Keras: The python deep learning library, 2018. Disponível em: https://keras.io/. 1

[2] Adit Deshpande. A beginner's guide to understanding convolutional neural networks, 2016. Disponível em: https://adeshpande3.github.io/A-Beginner%27sGuideTo-UnderstandingConvolutionalNeuralNetworks/. 2

[3] Stanford Vision Lab. Imagenet, 2016. Disponível em: http://www.image-net.org/. 2

[4] Jason Brownlee. A gentle introduction to transfer learning for deep learning, 2017. Disponível em: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. 2

[5] Felix Yu. A comprehensive guide to fine-tuning deep learning models in keras, 2016. Disponível em: https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html. 2

[6] Francois Chollet. Building powerful image classification models using very little data, 2016. Disponível em: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. 3