Classificação de espécies de peixes via rede neural convolucional

André G. C. Pacheco
Elivelto Eberman
Programa de Pós Graduação em Informática - PPGI
Universidade Federal do Espírito Santo

***Abstract*—Classificação de dados está presente em diversos problemas reais, tais como: reconhecer padrões em imagens, diferenciar peças defeituosas em uma linha de produção, classificar tumores benignos e malignos, dentre diversas outras. Muitos desses problemas possuem padrões de dados difíceis de serem identificados, o que requer, consequentemente, técnicas mais avançadas para sua resolução. Recentemente, diversos trabalhos abordando diferentes arquiteturas de redes neurais artificiais vêm sendo aplicados para solucionar problemas de classificação. Quando a classificação do problema deve ser obtida por meio de imagens, atualmente a metodologia padrão é uso de redes neurais convolucionais. Sendo assim, neste trabalho são utilizadas redes neurais convolucionais para classificação de espécies de peixes.**

Keywords*—*Classificação, Rede Neural Convolucional, Aprendizado profundo

I. Introdução

As Redes Neurais Artificiais (RNA) se tornaram populares no final da década de 1980 com o surgimento e aperfeiçoamento do algoritmo de treinamento *backpropagation* [1]. Desde então, as RNA's vêm sendo aplicadas em diversas áreas da computação, tais como: robótica [2], tratamento de imagens [3], predição de séries temporais [4], classificação de dados [5], dentre muitas outras áreas. Ao longo do tempo, as teorias desenvolvidas com base em RNA's direcionaram ao desenvolvimento de uma nova abordagem, com uma maior capacidade de aprendizagem e generalização, para problemas nos quais a natureza dos padrões são de difícil compreensão. Essa nova abordagem é conhecida como aprendizado profundo, que apresenta-se como uma melhor alternativa para problemas que envolvam comportamento inteligente, como classificação de padrões [6].

No contexto de aprendizagem profunda, pode-se destacar as redes profundas de crença (DBNs) [7], obtidas, por exemplo, por meio de empilhamento de máquinas de Boltzmann restrita (RBMs) [8], e as redes neurais convolucionais (CNN) [9]. Ambas arquiteturas utilizam diversas camadas de neurônios ocultos com intuito de extrair características de um dado conjunto de dados. Atualmente, as duas abordagens vêm sendo utilizadas em diversos trabalhos nas mais diversas áreas.

CNNs foram primeiramente propostas em 1998 por LeCun et al. [9], na qual os autores desenvolveram uma arquitetura neural conhecida como LeNet5, utilizada para reconhecer dígitos escritos à mão. Na ocasião, tal arquitetura estabeleceu um novo estado da arte ao atingir 99.2% de acurácia na base de dados MNIST [10]. Anos mais tarde, a Krizhevsky et. al [5] estabeleceram um marco na área ao propor a AlexNet, arquitetura vencedora do desafio ImageNet [11]. Desde então, aplicação mais popular relacionada a CNNs vêm sendo reconhecer padrões em imagens, todavia, diversos trabalhos vêm aplicando CNNs em outros tipos de tarefas [12], [13].

Neste trabalho são utilizadas diferentes arquiteturas de CNNs para classificação de espécie de peixes. Tal base de dados é composta por milhares de imagens que pode conter um, mais de um ou nenhum peixe, sempre da mesma espécie. Este problema de classificação faz parte do desafio *The Nature Conservancy Fisheries Monitoring*, hospedado da plataforma *Kaggle* [14]. O restante deste trabalho esta organizado da seguinte forma: na seção 2 são apresentados os conceitos básicos relacionados a CNNs; na sequência, na seção 3 a base de dados e o desafio são apresentados; em seguida, na seção 4, os resultados experimentais são descritos; por fim, na seção 5 são apresentadas as conclusões do trabalho.

II. Redes neurais convolucionais

Redes neurais convolucionais são similares a redes neurais tradicionais: ambas são compostas por neurônios que possuem pesos e *bias* que necessitam ser treinados. Cada neurônio recebe algumas entradas, aplica o produto escalar das entradas e pesos além de uma função não-linear. Ademais, ambas possuem a última camada toda conectada e todos os artifícios utilizados para melhorar a rede neural tradicional também são aplicados nesta camada. Dessa forma, qual a vantagem de se utilizar uma CNN? Uma CNN assume que todas as entradas são imagens, o que permite codificar algumas propriedade na arquitetura. Redes neurais tradicionais não são escaláveis para imagens, uma vez que a mesma produz um número muito alto de pesos a serem treinados [15].

Uma CNN é composta por uma sequência de camadas. Além da camada de entrada, que normalmente é composta por uma imagem com largura, altura e profundidade (RGB), exitem três camadas principais: camada convolucional, camada de *pooling* e camada totalmente conectada. Além disso, após uma camada de convolução é comum uma camada de ativação (normalmente uma função ReLu). Essas camadas, quando colocadas em sequência (ou empilhadas), forma uma arquitetura de uma CNN, como ilustrada na Fig. 1. Na sequência as camadas principais serão descritas, bem como suas funções.

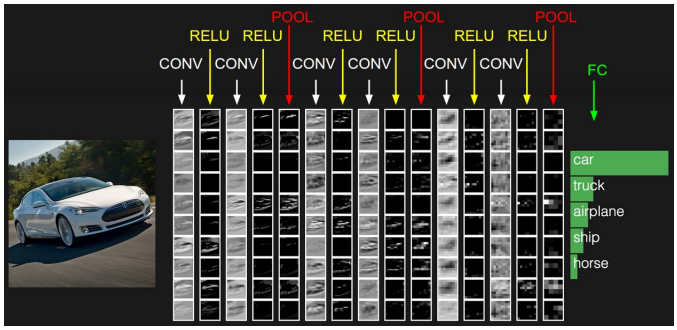


Fig. 1. Exemplo de arquitetura de uma CNN

A. Camada convolucional

A camada convolucional é a camada mais importante da rede. Nela é realizada a parte mais pesada do processamento computacional. Esta camada é composta por um conjunto de filtros (ou *kernels*) capazes de aprender de acordo com um treinamento. Os filtros são matrizes pequenas (por exemplo, $5 \times 5 \times 3$) compostas por valores reais que podem ser interpretado como pesos. Esses filtros são convoluídos com os dados de entradas para obter um mapa de características. Estes mapas indicam regiões na qual características específicas em relação ao filtro, são encontradas na entrada. Os valores reais dos filtros se alteram ao longo do treinamento (assim como os pesos de uma rede neural tradicional) fazendo com que a rede aprenda a identificar regiões significantes para extrair características do conjunto de dados.

A convolução entre um filtro e a imagem é ilustrado na Fig. 2. Neste exemplo é apresentada uma imagem $4 \times 4 \times 3$ e um filtro $2 \times 2 \times 3$. A convolução é realizada através do produto escalar de uma região da imagem do tamanho do filtro pelo filtro propriamente dito. Na sequência o filtro é deslizado para outra região e o produto escalar é realizado novamente até que toda a imagem seja percorrida. Perceba que cada canal é convoluído por uma dimensão diferente do filtro, que neste exemplo esta representado pela mesma cor do canal (RGB). Neste exemplo o filtro desliza na imagem deslocando 1 pixel para lado ou para baixo. Este deslizamento é controlado pelo parâmetro conhecido como *stride*, que neste caso é 1. Este valor pode ser alterado, todavia deve respeitar os limites da imagem. Caso o tamanho do filtro e do *stride* desejado não seja possível para a imagem, pode-se utilizar *zero-padding*, que nada mais é do que adicionar zeros na borda da imagem para tornar o deslizamento do filtro possível. Tanto o *stride* quanto *zero-padding* são parâmetros de projetos que deve ser analisados pelo projetista. Por fim, vale a pena destacar que podem existir (e normalmente existem) mais de um filtro por camada de convolução. Dessa forma, cada filtro resulta em uma saída de três dimensões, como o da Fig. 2.

Nas matrizes resultados da convolução é aplicado a função de ativação. A função mais utilizada é a ReLU (unidade de retificação linear), que é simplesmente aplicar a função $max(0, x)$ em cada elemento do resultado da convolução. No exemplo da Fig. 2, todos os elementos são maiores do que zero, sendo assim, o resultado da ReLU são os próprios valores. Por fim, como os pesos utilizados por cada filtro são os mesmos em todas as regiões, ele são conhecidos como pesos compartilhados. Essa característica reduz consideravelmente a quantidade de pesos da CNN quando comparadas a uma RNA.

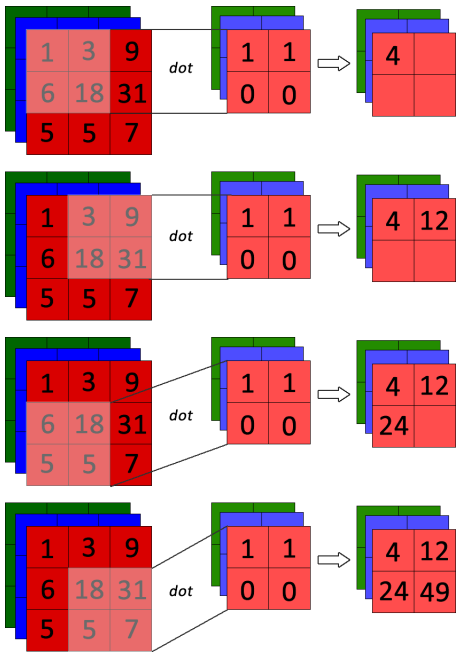


Fig. 2. Exemplo de convolução entre a imagem e um filtro

B. Camada de pooling

É muito comum após uma camada de convolução existir uma camada de *pooling*. A técnica de *pooling* é utilizada com objetivo de reduzir o tamanho espacial das matrizes resultantes da convolução. Consequentemente, essa técnica reduz a quantidade de parâmetros a serem aprendidos na rede, contribuindo para o controle de *overfitting*. As camadas de *pooling* operam de maneira independente em cada um dos canais do resultado da convolução. Além disso, é necessário determinar previamente o tamanho do filtro e *stride* para realizar o *pooling*. Na Fig. 3 é ilustrado a realização de *pooling* considerando que o resultado da convolução seja uma matriz $4 \times 4 \times 3$, o filtro do *pooling* abrange regiões 2×2 em cada canal, com *stride* igual a 2. Além disso, a função de agregação é $max(X)$, ou seja, escolhe o maior valor da região.
Como pode ser observado, a camada de *pooling* não reduz a quantidade de canais e sim a quantidade de elementos em cada canal. Além do *maxpooling*, pode ser utilizadas outras funções, como por exemplo a média dos valores. Todavia, a função *max* vem obtendo melhores resultados, uma vez que pixels vizinhos são altamente correlacionados.

C. Camada totalmente conectada

Diferentemente da camada convolucional, na qual os pesos são conectados apenas em uma região, a camada totalmente

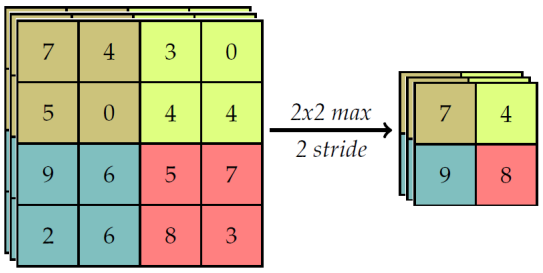


Fig. 3. Exemplo de *maxpooling* de uma imagem 4×4

conectada, como o próprio nome já sugere, é completamente conectada com a camada anterior. Tipicamente são utilizadas como última camada da CNN e funciona da mesma maneira que as redes neurais tradicionais. Portanto, os mesmos artifícios para melhorar o desempenho de uma RNA, como *dropout* [16] por exemplo, também são aplicáveis nesta camada.

Como a camada totalmente conectada vêm após uma camada convolucional ou de *pooling*, é necessário conectar cada elemento das matrizes de saída de convolução em um neurônio de entrada. A Fig. 4 ilustra a conexão de uma camada convolucional com uma camada totalmente conectada. É possível observar que os 48 mapas de características 4×4 são colocados de forma linear formando 768 entradas para camada totalmente conectada, que por sua vez, possui 500 neurônios ocultos que resultam em 10 saídas para rede. Neste ponto, é possível observar visualmente um rede neural tradicional.

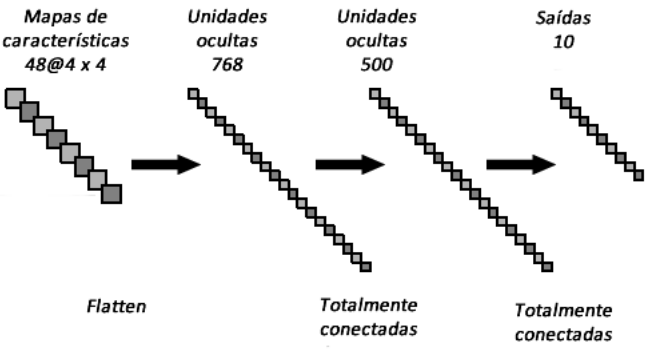


Fig. 4. Ilustração de uma camada totamente conectada

Nas unidades de saída, que no exemplo da Fig. 4 são 10, é utilizado uma função *softmax* para se obter a probabilidade de dada entrada pertencer a uma classe. Neste ponto é realizado o algoritmo de treinamento supervisionado *backpropagation*, assim como em uma rede neural tradicional. O erro obtido nesta camada é propagado para que os pesos dos filtros das camadas convolucionais sejam ajustados [9]. Dessa forma, os valores dos pesos compartilhados são aprendidos ao longo do treinamento.

D. Arquitetura de uma CNN

A arquitetura de uma CNN é relacionada com a maneira que as camadas descritas anteriormente são organizadas. De fato o projetista pode organizá-las da maneira que julgar adequado. A maior parte das arquiteturas utilizam uma camada de entrada, obviamente, seguida de um bloco de N camadas convolucionais com ativação ReLU conectada a uma camada de *pooling*. Esse bloco é repetido M vezes ao longo da rede que ao final é conectado a uma camada totalmente conectada para determinar a classificação final. Além disso, na montagem da arquitetura também são definidos parâmetros como tamanho do filtro, *stride* e *zero-padding*. Todas essas escolhas impactam diretamente na quantidade de pesos que a rede deve treinar e por consequência o quanto de computação é necessário para que esses valores sejam bem ajustados.

Existem diversas arquiteturas bem conhecidas na área de redes convolucionais. Três das mais utilizadas são descritas na sequência:

- **LeNet-5:** desenvolvida por LeCun et al. [9] foi a primeira aplicação de CNN que obteve sucesso. Foi utilizada para classificar dígitos, por conta disso, muito utilizada em tarefas como leitura de códigos postais e afins. A Fig. 5 ilustra a arquitetura de uma LeNet-5, que possui uma camada de entrada com imagens de 32×32 *pixels* e mais 7 camadas de pesos treináveis. As camadas convolucionais utilizam filtros $5 \times 5 \times 3$ e $6 \times 6 \times 3$ para computar os mapas de características e filtros $2 \times 2 \times 3$ para realização de *pooling*. Por fim, além da camada de saída que possui conexão para 10 classes, a rede possui duas camadas totalmente conectadas de 120 e 84 neurônios.

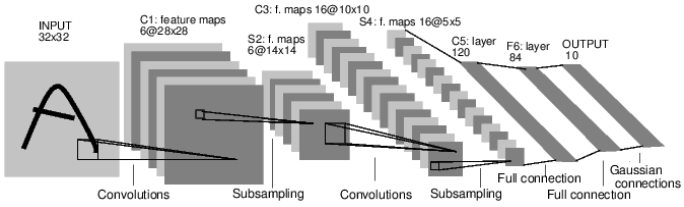


Fig. 5. Arquitetura de uma LeNet-5, CNN muito utilizada para reconhecimento de dígitos

- **AlexNet:** a arquitetura AlexNet foi o trabalhou que popularizou a CNNs para visão computacional. Proposta por Krizhevsky et al. [5], como já mencionado anteriormente, foi a metodologia vencedora do desafio ImageNet em 2012 [11]. Sua arquitetura é bastante similar à da LeNet-5, porém ela é mais profunda, com mais camadas convolucionais, e possui muito mais mapas de característica, como mostrado na Fig. 6. A AlexNet recebe como entrada imagens de 224×224 *pixels* por canal. Na primeira camada de convolução utiliza um filtro de $11 \times 11 \times 3$, na segunda $5 \times 5 \times 3$ e na terceira a diante $3 \times 3 \times 3$. Além disso, a terceira, quarta e quinta camada são conectadas sem utilização de *pooling*. Por fim, a a rede possui duas camadas totalmente conectadas com

2048 neurônios cada e uma camada de saída com 1000 neurônios, quantidade de classes existentes no problema. Vale a pena destacar que a AlexNet foi a primeira rede a utilizar *dropout* para auxiliar no treinamento da camada totalmente conectada. Além disso, o treinamento de toda a rede foi realizado utilizando duas GPUs. Como pode ser observado na Fig. 6, a rede é dividida em duas partes, uma GPU executou a parte de cima e a outra a parte de baixo da rede.

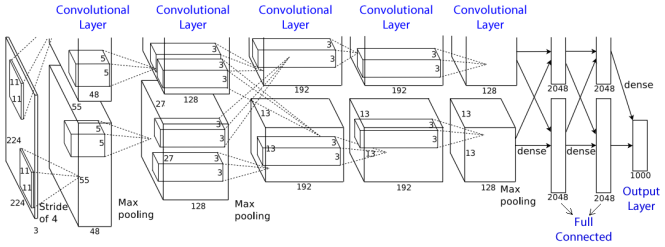


Fig. 6. Arquitetura de uma AlexNet, CNN vencedora do desafio ImageNet

- **VGGNet:** a arquitetura VGGNet foi proposta por Simonyan & Zisserman [17] e teve como principal contribuição mostrar que a profundidade da rede é um componente crítico para uma boa performance. O modelo padrão da VGGNet possui diversas camadas de convolução aplicando filtros $3 \times 3 \times 3$ e *max-pooling* com filtros $2 \times 2 \times 3$. Na Fig. 7 é ilustrado uma comparação entre uma AlexNet e uma VGGNet. É possível observar uma camada de *pooling* sempre após duas de convolução. Além disso, a rede possui três camadas totalmente conectadas além da camada de saída. Devido a sua profundidade, a VGGNet é bem cara computacionamente e necessita de muita memória para computar em cima de seus parâmetros (140M).

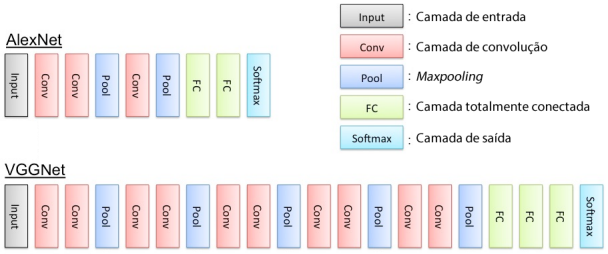


Fig. 7. Comparação entre as arquiteturas AlexNet e VGGnet

III. A BASE DE DADOS PA CLASSIFICAÇÃO DE ESPÉCIES DE PEIXE

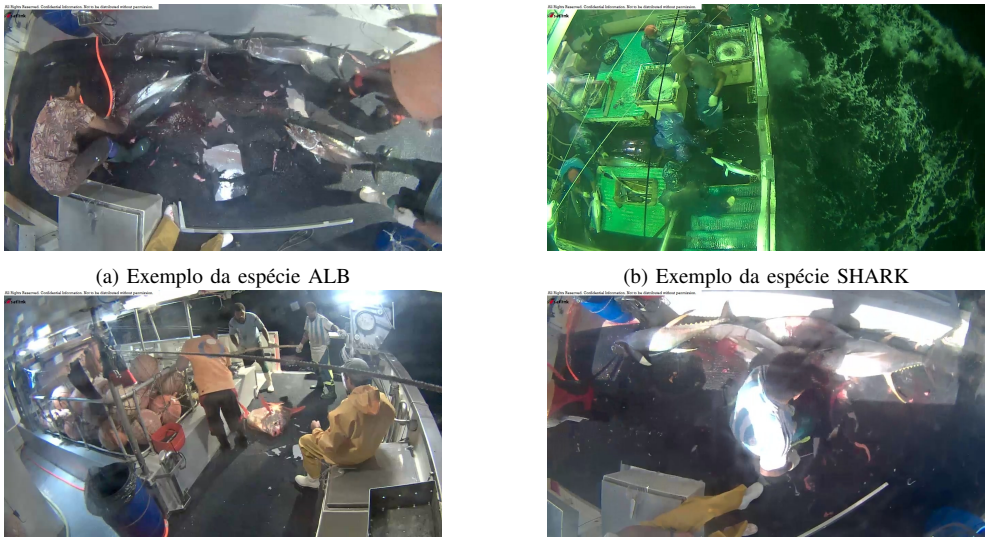
A base de dados utilizada neste trabalho foi coletada do desafio *The Nature Conservancy Fisheries Monitoring*, da plataforma *Kaggle*. *The Nature Conservancy* é uma organização internacional, sem fins lucrativos, líder na conservação da biodiversidade e do meio ambiente, cuja missão é conservar plantas, animais e comunidades naturais que representam a diversidade da vida na Terra, protegendo espaços que necessitam para sobreviver. De acordo com a organização, cerca de 60% do atum pescado no mundo é realizado de forma ilegal e não reportado. Essa prática prejudica o ecossistema marinho e os suprimentos de frutos do mar. Com isso, a organização está monitorando essas atividades por meio de imagens, que apesar de funcionar bem, a quantidade de dados produzido é muito grande para serem classificados manualmente. Sendo assim, a organização propôs o desafio para comunidade Kaggle desenvolver algoritmos capazes de detectar e classificar espécies de peixes a partir de imagens obtidas por câmeras instaladas em barcos.

Existem oito categorias a serem classificadas, as seis espécies de peixes ilustradas na Fig. 8 além da classe *outro*, indicando uma espécia de peixe que não seja a da figura, e uma outra classe *no fish*, indicando que não existe nenhum peixe na imagem.

Fig. 8. Espécies de peixes contidas no desafio em escala ilustrativa

Cada imagem da base de dados possui 1280×974 *pixels* e apenas uma classe como resposta. O algoritmo deve retornar oito saídas com a probabilidade da amostra de entrada pertencer a cada classe. Obviamente, a classe com maior probabilidade deve ser a escolhida.

Na Fig. 9 são ilustradas exemplos de imagens utilizadas no desafio. Como pode ser observado, as imagens possuem diferentes ângulos, iluminação e algumas possuem pequenos peixes, utilizados como isca, que não podem ser confundidos com as espécies desejadas. A quantidade de amostras para cada classe é descrita na Tab. I, o que mostra que a base de dados é bastante desbalanceada, tendo a classe ALB com 1719 amostras e a classe LAG com apenas 67.



(a) Exemplo da espécie ALB

(b) Exemplo da espécie SHARK

(c) Exemplo da espécie LAG

(d) Exemplo da espécie YFT

Fig. 9. Exemplos de imagens da base dados para classificação de espécies de peixes

Espécie	Nº de amostras
ALB	1719
BET	200
DOL	117
LAG	67
NoF	465
Outros	299
Shark	176
YFT	734
Total	3777

Tab. I. Número de amostras por espécie contidas na base de dados

IV. Resultados Experimentais

Para atacar o problema de classificação de espécie de peixes, neste trabalho foi utilizado uma rede neural convolucional com arquitetura que será descrita a seguir. Para implementar a rede foi utilizada a linguagem *Python* com as bibliotecas *TensorFlow* e *Keras*. A *TensorFLow* permite utilizar GPUs de maneira rápida e fácil e a *Keras* disponibiliza uma série de funções para construção de modelos neurais utilizando a *TensorFlow* para interface com GPUs. Como a *TensorFlow* é baseada em CUDA, para utilizá-la é necessário um *hardware* da NVIDIA.

Para realizar a classificação da base de dados foram testadas diferentes arquiteturas de CNN. A arquitetura que obteve melhor desempenho, e consequentemente foi escolhida para ser utilizada neste trabalho, está ilustrado na Fig. 10. Como pode ser observado, o modelo é semelhante à AlexNet, todavia a rede possui quatro camadas de convolução, realizando *maxpooling* a cada duas camadas, duas camadas totalmente conectadas e uma camada de saída *softmax*. Baseado nesta arquitetura foram escolhidos 3 modelos que obtiveram os melhores resultados nos testes. São eles:

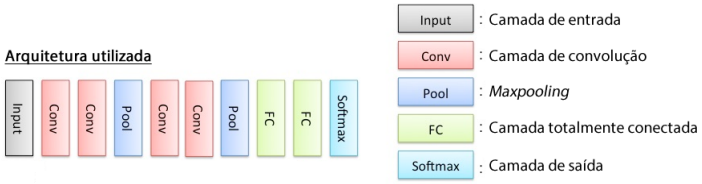


Fig. 10. Arquitetura utilizada para classificação das espécies de peixe

- **Modelo 1:** utiliza os 4 filtros 3×3 na Conv_1 e Conv_2, 8 filtros filtros 3×3 na conv_3 e conv_4, 96 neurônios na FC_1, 16 na FC_2 e 8 na camada de saída.
- **Modelo 2:** utiliza os 16 filtros 2×2 na Conv_1 e Conv_2, 32 filtros filtros 3×3 na conv_3 e conv_4, 96 neurônios na FC_1, 16 na FC_2 e 8 na camada de saída.
- **Modelo 1:** utiliza os 4 filtros 3×3 na Conv_1 e Conv_2, 8 filtros filtros 3×3 na conv_3 e conv_4, 144 neurônios na FC_1, 32 na FC_2 e 8 na camada de saída.

Em todos os modelos são utilizados *stride = 1* para as camadas convolucionais. Além disso, as mesmas possuem funções de ativação do tipo ReLU; Todos os *maxpoolings* são realizados com filtros 2×2 e com *stride = 2*, tanto na horizontal, quanto na vertical; O treinamento é realizado via SDG (*stochastic gradient descent*), com taxa de aprendizado igual 0.01, *weight decay* igual a 1×10^{-6}. *momentum* igual a 0.8, *dropout* com probabilidade igual a 0.5 e *mini-batch* igual a 24. Além disso, é utilizada a técnica de validação cruzada *k-folder* contendo 5 *folders*. Os modelos são treinados utilizando 100 épocas em cada *folder*. Todos esses valores foram obtidos de maneira empírica. Por fim, todos os modelos recebe imagens de 48×48 *pixels* em RGB.

A métrica de desempenho utilizada pelo *Kaggle* é a *multi-class log loss*, definida como:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} ln(p_{ij}) \quad (1)$$

onde N é o número de amostras, M é o número de classes do problema, y_{ij} é 1 se a amostra i pertence a classe j, caso contrário é 0, e p_{ij} é a probabilidade obtida pelo modelo da amostra i pertencer a classe j. Quanto menor o valor obtido pelo $logloss$, melhor é o modelo. Portanto, essa métrica pune classificadores que erra com muita certeza e beneficia aqueles que acertam com muita certeza.

Como descrito na seção anterior, a base de dados utilizada é bem desbalanceada. Isso é um problema para o treinamento de redes neurais em geral. Além disso, é de conhecimento dos organizadores e dos competidores que se houvesse mais imagens na base, facilitaria muito o trabalho de uma CNN. Todavia, essas restrições fazem parte do problema e fica a cargo dos competidores solucioná-los. Para auxiliar nestes problemas, foi retirado um conjunto de 150 imagens para teste, balanceando as amostras com o número de classes, e as imagens restantes foram replicadas aumentando o número de imagens das classes BET, DOL, LAG, Outros e Shark. Depois desta atualização, cada classe ficou com cerca de 400 imagens.

Na Tab. II são descritos os resultados de cada modelo. Primeiramente o modelo 1 é testado para a base de dados sem o balanceamento. Na sequência, todos os modelos são executados para a base com balanceamento. A base de validação contida na tabela é constituída das 150 imagens que foram separadas anteriormente e a base de teste é a disponibilizada pelo *Kaggle*, no qual não são conhecidos as respostas para cada amostra, somente o resultado do *logloss* para as 1000 amostras que as contém. As posições no raking do *Kaggle* descritas na tabela são referentes ao dia 12 de dezembro de 2016, como a competição se encerra em 4 meses, novos competidores estão submetendo seus resultados diariamente e esse ranking pode mudar. Vale a pena ressaltar, que até a presente data exitem 689 submissões de time ou competidores individuais no desafio.

Base desbalanceada				
Modelos	**Logloss treino**	**Logloss validação**	**Logloss Kaggle**	**Pos. Kaggle**
Modelo 1	0.4012	0.9817	1.6120	512
Base balanceada				
Modelos	**Logloss treino**	**Logloss validação**	**Logloss Kaggle**	**Pos. Kaggle**
Modelo 1	0.3055	0.7080	1.1716	167
Modelo 2	0.2644	0.8012	1.2628	318
Modelo 3	0.2124	0.7618	1.2437	287

Tab. II. Resultado da classificação da base de dados para cada um dos modelos. Cada modelo foi executado 10 vezes

Como pode ser observado na Tab. II, o balanceamento das imagens contribuiu para melhorar o desempenho das mesmas. Além disso, o modelo 1 foi o que obteve melhor resultado em relação aos demais modelos. Ao aumentar o número de neurônios da camada totalmente conectada a rede começa a sofrer de *overfitting*, começando a decorar os dados de treinamento diminuindo o *logloss* destes dados, mas aumentando o dos testes. De fato, este é um problema esperado, levando em consideração que a base não contém um acervo muito grande de imagens.

Na Tab. III é exibido uma matriz de confusão para o modelo 1 com a base balanceada. É possível notar que as maiores confusões de classe ocorrem entre a classe ALB e as demais. Isso ocorre, por que mesmo após o balanceamento, a classe ALB continua com muito mais imagens do que as demais.

	ALB	**BET**	**DOL**	**LAG**	**NoF**	**Outros**	**Shark**	**YET**
ALB	25	0	0	0	0	0	0	0
BET	1	15	0	0	1	0	0	3
DOL	0	0	15	0	0	0	0	0
LAG	0	0	0	10	0	0	0	0
NoF	5	0	0	0	15	0	0	0
Outros	2	0	0	0	0	18	0	0
Shark	2	0	0	0	0	0	18	0
YFT	2	0	0	0	0	0	0	18

Tab. III. Matriz de confusão para o modelo 1 utilizando a base de dados balanceada

V. Considerações finais

Neste trabalho uma rede neural convolucional foi utilizada pra classificar espécies de peixes disponibilizados no desafio *The Nature Conservancy Fisheries Monitoring*, da plataforma Kaggle. A base de dados é desafiadora, devido as características das imagens, que possuem ângulos diferentes, muito ruído, iluminação muito diferentes, etc; além da base ser muito desbalanceada e não contemplar um número muito grande de imagens. A arquitetura utilizada para atacar o problema, semelhante à AlexNet, foi dividida em três modelos. Seus resultados foram descritos e discutidos. O melhor modelo obteve posição 167ª dentre as 689 possíveis. O resultado obtido é razoável mas pode ser muito melhorado. Sendo assim, como trabalho futuro, pretende-se melhorar o balanceamento da base de dados, pois os resultados aqui apresentados mostram que isso é essencial para o desempenho da rede; utilizar um modelo pré-treinado com outras bases de dados para ensinar a rede o que é um peixe e em seguida, aplicá-lo para esta base de dados; realizar uma rede modulada, no qual primeiro ela classifica se o peixe é da classe ALB ou nao, e na sequência realiza a classificação para as demais classes; e por fim, utilizar um *ensemble* com outros classificadores para melhorar o desempenho da rede. Tendo em vista que a competição se encerra daqui a quatro meses, todos esses pontos podem ser implementados a fim de melhorar a classificação no ranking final.

References

[1] D. Williams and G. Hinton, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[2] O. Mohareri, R. Dhaouadi, and A. B. Rad, "Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks," *Neurocomputing*, vol. 88, pp. 54–66, 2012.

[3] D. D'Addona and R. Teti, "Image data processing via neural networks for tool wear prediction," *Procedia CIRP*, vol. 12, pp. 252–257, 2013.

[4] R. Hrasko, A. G. Pacheco, and R. A. Krohling, "Time series prediction using restricted boltzmann machines and backpropagation," *Procedia Computer Science*, vol. 55, pp. 990–999, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[7] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[8] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] L. Yann, C. Corinna, and B. Christopher, "The MNIST database of handwritten digits," disponível em:https://www.kaggle.com/c/the-nature-conservancy-fisheries-monitoring/details/evaluation, visitado em 8 de dezembro de 2016.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[12] I. Wallach, M. Dzamba, and A. Heifets, "Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery," *arXiv preprint arXiv:1510.02855*, 2015.

[13] E. Gibney, "Google ai algorithm masters ancient game of go," *Nature*, vol. 529, pp. 445–446, 2016.

[14] Kaggle, "The nature conservancy fisheries monitoring," disponível em: http://yann.lecun.com/exdb/mnist, visitado em 8 de dezembro de 2016.

[15] F.-F. Li, A. Karpathy, and J. Johnson, "Cs231n: Convolutional neural networks for visual recognition," 2015.

[16] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.