Classificação de imagens com uma rede neural convolucional

Matheus Henrique[1] , **Antônio Carlos**[2] and **Joice Cristina**[3]
Universidade Federal de São João del-Rei
matheusrickbatista@aluno.ufsj.edu.br[1], juniormanhaesfilho@aluno.ufsj.edu.br[2],
joyciinha_cristina@aluno.ufsj.edu.br[3]

Abstract

O problema de classificação de imagens consiste em realizar a análise e associação dos pixels de uma imagem a um rótulo, esse rótulo representa um objeto do mundo real. Nesse artigo, apresentamos uma solução para o problema utilizando uma rede neural convolucional (RNC). A rede foi treinada para realizar a classificação através de uma base de dados com imagens de gatos e cachorros. O objetivo é que, após o treinamento, a rede possa receber uma imagem e classificar como gato ou cachorro.

1 Introdução

O problema de classificação de imagens é um problema do campo da visão computacional que busca desenvolver algoritmos que possam atribuir um rótulo a imagens com base na análise dos seus pixels. Esse problema pode ser aplicado em diversos problemas reais como reconhecimento facial, diagnóstico de doenças e análise de imagens astronômicas.

Uma rede neural artificial é um método do campo do aprendizado de máquina que se inspira na estrutura neural de organismos inteligentes visando ensinar computadores a processar dados e adquirir conhecimento através da experiência. Esse método tem sido amplamente aplicado ao problema de classificação de imagens.

1.1 Problema

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) ou HIP (Human Interactive Proof) são desafios que protegem páginas web de serem visitadas por máquinas de modo que a intenção seja inibir ataques. Esses desafios tem como característica grande facilidade para resolução por humanos ao mesmo tempo que são tarefas muito difíceis para computadores.

ASSIRA (Animal Species Image Recognition for Restricting Access) é um dos desafios que solicita aos usuários que identifiquem fotografias de cães e gatos. Em [1] Philippe Golle apresentou uma técnica de classificação com acurácia de 82,7%, o que leva a uma probabilidade de 10,3% de resolução de um problema de 12 imagens ASSIRA. Por isso, ASSIRA passou a não ser considerado um método tão seguro.

Visto os avanços no estado da arte, a Kaggle, uma plataforma de competição de ciência de dados e comunidade on-line de cientistas e praticantes de aprendizado de máquina, lançou um desafio que consiste em classificar imagens de gatos e cachorros para identificar as abordagens mais recentes no campo da visão computacional. Utilizamos a base de dados disponibilizada pela plataforma para a aplicação de uma rede neural convolucional que pudesse criar um modelo de classificação dessas imagens.

1.2 RNCs aplicadas ao problema

As redes neurais convolucionais têm sido amplamente utilizadas no problema de classificação. Entre os trabalho de destaque mais recentes estão:

- **RNC no auxilio do diagnóstico de leucemia:** Os autores em [2] combinaram a utilização de uma RNC pré treinada, conhecida como AlexNet, com a técnica de data augmentation para conseguir uma acurácia impressionante de 99,50% no diagnóstico de imagens doentes e não doentes.
- **Identificação de pragas com RNC:** Em [3] o autor utilizou uma RNC para criar um aplicativo que realiza a identificação de pragas em plantações. A rede recebe uma imagem e em caso de classificação bem sucedida da imagem o aplicativo fornece informações que auxiliam no conhecimento da praga em questão. A rede obteve taxa de acerto de 92,5%.
- **RNC para detecção de incêndios:** Em [4] o autor apresenta um sistema que utiliza uma câmera para monitoramento e um aplicativo que sinaliza a ocorrência de um incêndio. A captação de frames pela câmera alimenta uma rede neural que é capaz de identificar a incidência de fogo ou não fogo no ambiente. A rede notifica o usuário via aplicativo se identificar fogo em 20 ou mais frames consecutivos.

1.3 Objetivo

O objetivo desse trabalho é realizar a classificação simples de imagens, para profundo entendimento da arquitetura convolucional. Acreditamos que esse problema é o passo base para aplicações mais complexas como as que foram apresentadas.

2 Fundamentação teórica

Antes de definirmos uma rede neural artificial é preciso fazer uma contextualização das áreas de estudo. As redes neurais

artificiais, são uma subárea do aprendizado profundo, que, por sua vez, é uma subárea do aprendizado de máquina.

2.1 Machine Learning

O Machine Learning, ou aprendizado de máquina, é uma subárea da inteligência artificial que se concentra na construção de sistemas que aprendem, ou melhoram o desempenho, com base nos dados que consomem. Em vez de fornecer instruções específicas para executar uma tarefa, o machine learning permite que os sistemas aprendam a partir de exemplos e experiências passadas, identifiquem padrões nos dados e tomem decisões ou façam previsões com base nesses padrões.

2.2 Deep Learning

O Deep Learning, ou aprendizado profundo, por sua vez é um ramo do machine learning que se concentra em técnicas avançadas de aprendizado por meio do uso de redes neurais profundas para aprimorar diversas aplicações, como reconhecimento de fala, visão computacional e processamento de linguagem natural. Essa abordagem está se tornando rapidamente uma das áreas mais estudadas e buscadas dentro da ciência da computação. É uma tecnologia usada por diversas empresas, como, por exemplo, o Google com o Google Tradutor, a Microsoft com a Cortana e a Amazon com a Alexa.

2.3 Rede neural artificial

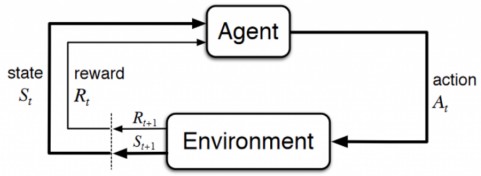
Uma rede neural ou rede neural artificial (RNA) tem duas características principais, a arquitetura que está relacionada a organização estrutural da rede e o algoritmo de aprendizagem. Existem várias arquiteturas de redes neurais, e cada uma delas possui características e propriedades específicas. Alguns exemplos comuns de arquiteturas de redes neurais incluem:

- **Redes Neurais Multilayer Perceptron (MLP):** É um algoritmo simples que consiste em uma camada de entrada para receber o sinal, uma ou mais camadas intermediárias (ou camadas ocultas) que são os mecanismos computacionais MLP e uma camada de saída que toma decisão ou previsão sobre a entrada.
- **Redes Neurais Recorrentes (RNN):** Estes conjuntos de algoritmo são especialmente úteis para processamento de dados sequenciais como som, dados de série temporais ou linguagem natural. Elas têm conexões recorrentes que permitem que as informações fluam em loops, permitindo que a rede tenha memória interna para lidar com informações anteriores.
- **Redes Neurais Generativas Adversariais (GAN):** Essa arquitetura é composta por duas redes que se colocam uma contra a outra: o gerador e o discriminador. O gerador gera novas amostras sintéticas, enquanto o discriminador tenta distinguir entre as amostras reais e as sintéticas. Essas redes são usadas para gerar dados sintéticos realistas, como imagens ou texto.
- **Redes Neurais convolucionais (CNN):** São redes neurais artificiais profundas que podem ser usadas para classificar imagens, agrupá-las por similaridade (busca de fotos) e realizar reconhecimento de objetos dentro de cenas. Será abordado na próxima sessão detalhes específicos desta arquitetura.

Essas são apenas algumas dentre várias arquiteturas de redes neurais artificiais. A escolha de utilização delas depende do problema que deseja resolver e dos tipos de dados envolvidos.

O algoritmo de aprendizagem é um conjunto de regras bem definidas para a solução de um problema de aprendizagem. Em uma rede neural é muito importante a habilidade de aprender a partir de seu ambiente e melhorar seu desempenho e isso acontece através de um processo iterativo de ajustes aplicados a seus pesos, o treinamento. Alguns algoritmos diferem entre si principalmente pelo modo como o treinamento é feito.

- **Aprendizado supervisionado:** Nesta abordagem, o modelo de IA aprende a partir de resultados pré-definidos, ou seja, os dados de entrada tem uma resposta correta associada a eles. O objetivo do aprendizado supervisionado é fazer com que o modelo aprenda a mapear corretamente os dados de entrada para suas respostas correspondentes.
- **Aprendizado não-supervisionado:** Ao contrário do aprendizado supervisionado, nesta abordagem o algoritmo lida com conjuntos de dados não rotulados, ou seja, os dados de entrada não têm respostas corretas associadas a eles.
- **Aprendizado por reforço:** São conhecidos também como modelos semi-supervisionados, nesta técnica é permitido que um agente interaja e tome ações com um ambiente a fim de maximizar sua recompensa acumulada.



Exemplo de Aplicação: Considere um bebê que tem um controle remoto da TV em sua casa como parte de seu ambiente. O bebê age como um agente que observa e constrói uma representação do ambiente, que é o estado atual. Ele decide realizar ações, como pressionar os botões do controle remoto, e observa as respostas da TV, que resultam em um próximo estado. Como a TV não responde e não é interessante, o bebê associa uma recompensa negativa a ela e passa a realizar menos ações que levam a esse resultado indesejado. O bebê repete esse processo, atualizando sua política de ações em diferentes circunstâncias, com o objetivo de encontrar uma política que esteja satisfeito, ou seja, maximizar as recompensas totais com desconto.

2.4 Rede neural artificial convolucional

Na matemática, a convolução é um operador linear que recebe duas funções como entrada e retorna uma terceira função com origem, a partir do somatório da multiplicação

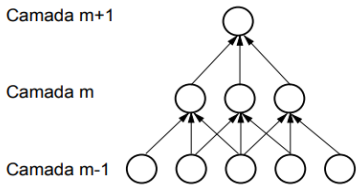
da função *kernel* na região superposta da função alvo pelo deslocamento da *kernel* sobre ela. A fórmula da convolução no domínio é a seguinte:

$$(f * g)(k) = h(k) = \sum_{i=1}^{k} f(i).g(k-i)$$

f e g são sequencias numéricas de tamanhos iguais ou variados e a função retorna o k-ésimo elemtno do somatório da multiplicação.

As *Conv Nets*, ou CNN, são um tipo específico de *ANN*, proposta pelo francês Yann LeCun (LECUN, et al.,1998) e se mostraram eficazes para resolver problemas de classificação. São arquiteturas biologicamente inspiradas, capazes de serem treinadas e aprenderem representações invariantes a escala, translação, rotação e transformações afins (LECUN; KAVUKCUOGLU; FARABET, 2010). Em um dos trabalhos de Hubel e Wiesel (1968), experimentos realizados em gatos e macacos revelaram que o córtex visual é organizado em um conjunto hierárquico de células sensíveis à pequenas sub regiões chamadas de campos receptivos. No estudo, Hubel categorizou as células como simples, complexas e super complexas de acordo com os estímulos que as ativava e a partir de então surgiu-se a hipótese de que uma boa representação interna deve respeitar uma hierarquia, onde os *pixels* formam arestas, as arestas formam padrões, os padrões combinados dão origem às partes, as partes combinadas formam os objetos e os objetos formam cenas (LECUN; KAVUKCUOGLU; FARABET, 2010). Isto sugere que para realizar o reconhecimento, é preciso vários estágios de treinamento empilhados uns nos outros, para cada nível de hierarquia. A RNC surge para representar este tipo de estrutura.

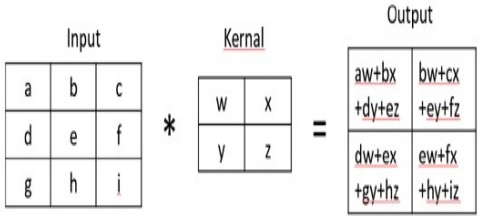
As RNC são arquiteturas multiestágios que podem ser treinadas. Os campos receptivos são fortemente correlacionados à localidade do estímulo na imagem de entrada, fazendo com que seja forçado um padrão de conectividade entre as camadas de neurônios artificiais. No exemplo abaixo vemos a organização dessas camadas. A camada $m-1$ representa a imagem de entrada. A camada superior m possui um campo receptivo de tamanho 3, onde cada neurônio recebe estímulo de três neurônios da camada anterior. Já a camada $m+1$ possui campo receptivo de tamanho 3 quando comparado com a camada m mas tem tamanho 5 quando comparado com a imagem de entrada.



A partir dessa analogia, cada campo receptivo é considerado um filtro não linear onde seus pesos devem ser aprendidos para que o neurônio só seja ativado quando um estímulo específico de sua natureza esteja presente na área onde o filtro foi aplicado. Cada filtro é aplicado em toda imagem de entrada de forma convolucional e o resultado da aplicação do filtro é chamado de *feature map*, mapa de características. Cada mapa compartilha os mesmos parâmetros, senão não seria possível realizar a convolução e preservar as características originais da imagem. As entradas de dados de cada estágio é um conjunto de *feature maps*. Caso a imagem de entrada seja colorida, a entrada do primeiro estágio são os três canais de cores da imagem. Cada vetor 2D começa a funcionar como um *feature map*. Na saída de cada estágio, cada mapa corresponde a convolução do mapa de entrada por um filtro. A função do filtro é destacar as diferentes características da imagem. Uma RCN pode ser composta de um ou mais estágios mas cada estágio executa as três etapas. Um *feature map* é obtido efetuando a convolução de uma imagem de entrada por um filtro linear, seguido da adição de um termo de *bias* e de uma função não linear.

2.5 Filtragem

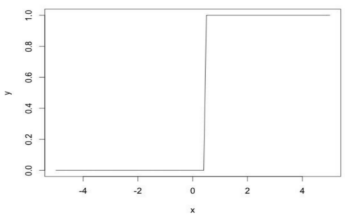
A filtragem, *filter bank layer*, é a primeira etapa do processo de convolução. Ela realiza a convolução dos filtros W_i^k, correspondente ao i-ésimo filtro da camada k de tamanho $l_1 \times l_2$ na imagem. Cada filtro detecta uma característica específica em todas as localizações na imagem de entrada. Na figura abaixo temos um exemplo de convolução, onde a máscara é colocada em diferentes posições da imagem, o que reduz o tamanho resultante pois algumas camadas da máscara são sobrepostas.



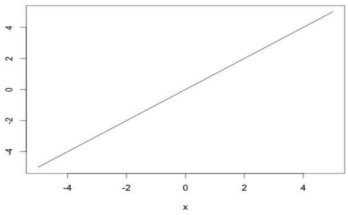
2.6 Não Linear

A etapa não linear, *non-linearity layer*, se trata da etapa onde se aplica uma função não linear em cada elemento do *feature map*. Essas funções são chamadas de função de ativação. Estes são alguns exemplos de função de ativação:

- **Função de etapa binária:** Os autores em [2] Essa função é comumente utilizada como um limiar que determina se o neurônio deve ou não deve ser ativado. Caso o valor Y esteja acima do limite determinado, é retornado 1, caso contrário retorna 0.

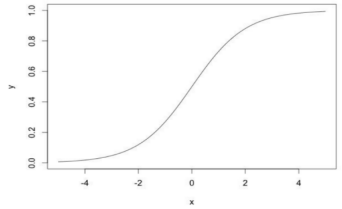


- **Função linear:** É a função mais básica porque não altera a saída de um neurônio. Geralmente utilizada nas camadas de saída em redes neurais de regressão.

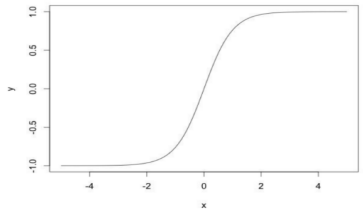


- **Função Sigmoide:** É comumente utilizada por redes neurais com propagação positiva (*Feedforward*), que precisam ter como saída apenas números positivos, em redes neurais multicamadas. A função Sigmoide consegue determinar se um valor é positivo ou negativo pois o resultado da função será 0 para qualquer valor negativo, enquanto para qualquer valor positivo estará entre o intervalo $]0, 1]$.

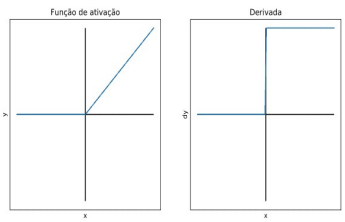
$$\phi(x) = 1/(1 + e^{-x})$$



- **Função Tangente Hiperbólica:** Tem utilização similar à função Sigmoide, pois produz resultados no intervalo $[-1, 1]$, entretanto, converge a zero mais rapidamente.

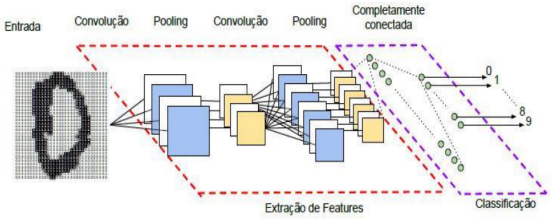


- **ReLU:** Unidade linear retificada, *rectified linear unit*, é uma função que produz resultados no intervalo $[0, \infty[$. Ela retorna 0 para todos os valores negativos e o próprio valor para os positivos. É uma das funções de ativação mais utilizadas atualmente, pois como tem resultado 0 para valores negativos, tende a não ativar alguns neurônios, aumentando a velocidade do treinamento, embora exista o risco do neurônio não aprender. Apesar de ser muito utilizada para o treinamento não costuma ser usada na camada de saída.



2.7 Redução

Também chamada de *subsampling*, calcula a média ou máximo de uma vizinhança pré-determinada em cada um dos *feature maps*. O resultado é outro mapa de menor resolução que proporciona invariação a pequenas translações. O método utilizado nesta etapa foi o *Max-Pooling* e será abordado com mais detalhes na próxima seção. Ao término da etapa de Redução é finalizado o processo convolucional, e os *feature maps* são transformados em vetores 1D, que por fim são utilizados no treinamento de classificadores em uma camada totalmente conectada, onde a entrada é a saída da camada anterior e a sua saída são N neurônios, sendo N a quantidade de classes do modelo.



3 Implementação

A implementação da rede neural convolucional passa por algumas etapas, demonstradas nos próximos subtópicos.

3.1 Base de dados

O primeiro passo para aplicação da rede é montar a base de dados. Conforme apresentamos na introdução, utilizamos os dados fornecidos pela plataforma Kaggle com mais de 25 mil imagens de cães e gatos. Essa base de dados deve passar por um pré-processamento visando aumentar a acurácia no treino da rede.

3.2 Pré-processamento da base de dados

Uma forma comum de dividir a base de dados para treinamento de modelos de aprendizado de máquina é dividir os dados em 3 conjuntos de imagens. Os conjuntos são: treino, validação e teste.

O conjunto de treino contém os primeiros dados com qual o modelo tem contato, trata-se de um conjunto com exemplos utilizados para ajustar os parâmetros do modelo, como pesos de conexões entre neurônios. A rede é treinada com esse conjunto por meio de um método de aprendizado supervisionado, as imagens da pasta treino tem um rótulo que informa para a rede que as características presentes naquela imagem pertencem aquele rótulo.

O conjunto de validação é utilizado durante o processo de treino da rede, ele fornece imagens que não foram apresentadas para o modelo anteriormente. Imagine que o modelo é um aluno que está estudando um livro, durante o processo de estudo o aluno pode começar a memorizar características em vez de realmente aprender, ele pode memorizar por exemplo que a página 20 do livro possui uma integral específica sem de fato aprender a identificar a integral. Dessa forma, o conjunto de validação apresenta ao longo do processo de treinamento imagens inéditas para ajudar a rede a ajustar seus

hiper-parâmetros, assim, a rede não fica ”viciada” no conjunto de dados de treino.

O conjunto de dados teste armazena dados que são usados exclusivamente para testar o desempenho do modelo, após o treinamento. Esses dados permitem realizar uma avaliação do desempenho do modelo, como ele se comporta para a entrada de novos dados e qual sua capacidade de realizar uma previsão assertiva. Durante o processo de avaliação do modelo a previsão dele é comparada com o rótulo da imagem, de posse das previsões é possível medir a acurácia da rede.

Das 25 mil imagens da base de dados, 70% foram destinadas para treino, 20% para teste e 10% para validação.

3.3 Ferramentas utilizadas

Alguns processos para a implementação de uma rede neural são demasiadamente complicados, por isso, especialistas da área já trabalharam para disponibilizar algumas ferramentas em bibliotecas para utilização geral. Para implementar o modelo utilizamos a biblioteca para aprendizado de máquina TensorFlow aliada a biblioteca focada em redes neurais Keras.

3.4 Implementação da rede neural convolucional

3.4.1 Criar *datasets*

O primeiro passo é criar *datasets* em um formato de objeto que o TensorFlow possa trabalhar. Para isso utilizamos a função `image_dataset_from_directory` passando três parâmetros: o nome do diretório de interesse, image_size e batch_size.

A função percorre todas as imagens do diretório de interesse e utiliza o parâmetro image_size para redimensionar as imagens para o padrão passado. Esse redimensionamento é importante para que a rede trabalhe com imagens de um mesmo tamanho quadrado. O parâmetro batch_size diz de quantas em quantas imagens os pesos da rede neural (hiperparâmetros) serão atualizados.

Os valores dos parâmetros utilizados nessa parta da implementação foram definidos de forma empírica como:

```
image_dataset_from_directory(diretório,
image_size = (180,180), batch_size = 32).
```

Para cada pasta da base de dados (treino, validação e testes) foram criados *datasets*.

3.4.2 Criar modelo

Para criar o modelo da rede neural optamos por utilizar o modelo sequencial do Keras, esse modelo permite inserir camadas de uma rede neural em série, onde o *output* da primeira camada serve como *input* da segunda, e assim por diante. Utilizamos alguns parâmetros para ajustes do modelo, são eles:

Rescaling: É um processo de normalização que irá representar cada cor da imagem em valores entre 0 e 1, isso facilita os cálculos para a rede neural e ajuda a convergir para uma melhor acurácia.

RandomFLip: É um parâmetro de *data augmentation* que irá rotacionar para a horizontal ou para a vertical uma imagem aleatória. Isso criará uma nova imagem para a rede neural ser treinada. *Data augmentation* é técnica de aumentar artificialmente o conjunto de treinamento criando cópias modificadas de um conjunto de dados existentes, isso inclui fazer pequenas alterações nas imagens.

RandomRotation: Também é um parâmetro de *data augmentation* que rotaciona uma porcentagem de uma imagem aleatória entre zero e o valor fornecido.

RandomZoom: Também é um parâmetro de *data augmentation* que da *zoom* em uma porcentagem da imagem aleatória entre zero e o valor fornecido.

Os valores dos parâmetros utilizados nessa parta da implementação foram definidos de forma empírica como:

```
keras.Sequential([Rescaling(scale =
1.0/255), RandomFlip("horizontal"),
RandomRotation(0.1), RandomZoom(0.2)])
```

3.4.3 Adicionar camadas convolucionais

Após criar o modelo e os parâmetros para treino, é possível adicionar camadas convolucionais ao modelo. Para adicionar uma camada convolucional utilizamos a função Conv2D da biblioteca Keras. A camada de convolução irá extrair a geometria das características do objeto de interesse através através dos parâmetros passados, que são os operadores de convolução. Utilizamos 3 parâmetros nesse trabalho, são eles:

kernel: kernel, ou filtros, atuam como um detector de características relevantes nas imagens, procurando padrões de interesse, eles percorrem a matriz da imagem realizando um calculo matemático com o valor do peso do filtro e o valor do pixel da imagem armazenando em uma saída que representa aquela característica da imagem.

kernel_size: uma tupla que representa a altura e a largura do filtro de convolução.

activation: é a função de ativação que calcula a saída da camada e decide se o próximo neurônio sera ativado ou não.

Uma única camada de convoluçãonão é suficiente para treinar um modelo, portanto, empiricamente decidimos utilizar quatro camadas de convolução com variação apenas do parâmetro *kernel*.

Os seguintes valores, também empíricos, para os parâmetros apresentados foram definidos:

```
Conv2D(32, kernel_size = (3, 3),
      activation = 'relu'))
Conv2D(64, kernel_size = (3, 3),
      activation = 'relu'))
Conv2D(128, kernel_size = (3, 3),
      activation = 'relu'))
Conv2D(256, kernel_size = (3, 3),
      activation = 'relu'))
```

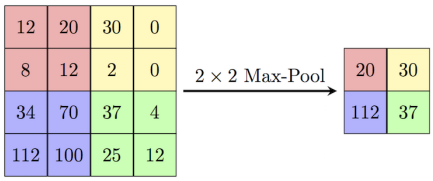
3.4.4 Aplicando o *Batch Normalization*

[5] *Batch normalization* é uma técnica para treinar redes neurais muito profundas que padroniza as entradas em uma camada para cada mini-lote. Isso tem o efeito de estabilizar o processo de aprendizado e reduzir drasticamente o número de épocas de treinamento necessárias para treinar redes profundas.

A função `BatchNormalization` da biblioteca Keras foi aplicada imediatamente após cada camada de convolução, nas 4 camadas implementadas.

3.4.5 Aplicando o *Max-Pooling*

O operação de Pooling recebe as imagens de borda da camada de convolução (normalizadas pelo Batch Normalization) e redimensiona as imagens a fim de reduzir a sobrecarga de informação, deixando apenas as características condensadas. No Max-Pooling o valor maximo de uma matriz determinada é salva como característica principal daquela região da imagem, isso condensa a característica máxima principal.



Essa técnica auxilia a reduzir o *overfitting*, reduz o grande número de informações e ensina a rede quais características podem ou não podem estar juntas. O *Deep Learning Book* [6] define o Max-Pooling como uma forma de a rede perguntar se um determinado recurso é encontrado em qualquer lugar de uma região da imagem e em seguida, eliminar a informação posicional exata. A intuição é que, uma vez que um recurso tenha sido encontrado, sua localização exata não é tão importante quanto sua localização aproximada em relação a outros recursos. Um grande benefício é que há muito menos recursos agrupados e, portanto, isso ajuda a reduzir o número de parâmetros necessários nas camadas posteriores.

Após cada *BatchNormalization* utilizamos a função *MaxPooling2D* da biblioteca Keras passando apenas o parâmetro *pool_size*. Esse parâmetro refere-se as dimensões da janela de *Pooling*. O valor do parâmetro foi definido como:

```
MaxPooling2D(pool_size = (2, 2))
```

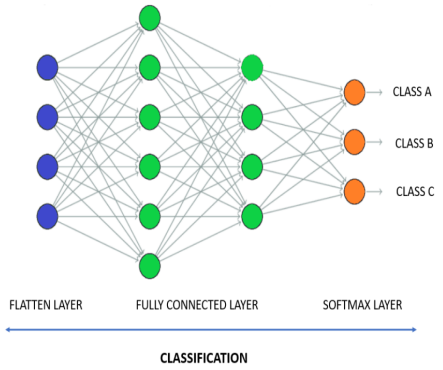
3.4.6 *Flaten*

A função `Flaten` da biblioteca Keras foi utilizada após o *Max-Pooling* da última camada para transformar uma matriz multidimensional em um vetor unidimensional, mantendo o conteúdo, mas removendo qualquer estrutura de dimensões adicionais.

A saída das camadas de convolução e *Pooling* é uma matriz tridimensional que representa as características extraídas das imagens de entrada. A intenção é vetorizar essas informações para utiliza-las em uma camada única totalmente conectada que será apresentada posteriormente.

3.4.7 Camada totalmente conectada

A camada *fully connected* é uma camada densa que vem após *Flaten*, cada neurônio nessa camada é conectado a todos os neurônios da camada anterior. A implementação dessa camada é útil para realizar a classificação final dos dados, é ela quem diz se a imagem contém características de gato ou de cachorro.



Para aplicação dessa camada utilizamos a função *Dense* da biblioteca Keras. Os paramêtros utilizados foram a quantidade de neurónios da camada e a função de ativação.

Os seguintes valores de parâmetros foram utilizados.

```
Dense(256, activation = 'relu')
```

3.4.8 Dropout

O *dropout* consiste em desligar alguns neurônios e maneira aleatória durante o treinamento. Isso pode parecer contraintuitivo a principio mas Geoffrey Hinton [7] demonstra que isso fortalece a rede no geral. Isso acontece porque alguns neurônios podem ficar especialistas e enviesados demais durante o treinamento, ao forçar o desligamento de alguns, todos os neurônios passam pelo processo de aprendizado de forma igualitária. A camada de dropout é aplicada imediatamente a camada anterior.

Utilizamos a função `Dropout` da biblioteca Keras na implementação. O único parâmetro passado é um valor entre 0 e 1 que diz respeito a uma faixa entre zero e o valor passado de neurônios que serão desligados.

O seguinte valor de Dropout foi utilizado:

```
Dropout(0.5)
```

3.4.9 Neurônio *sigmoid*

Na última camada, utilizamos um único neurônio com função de ativação *sigmoid* para classificação binária que irá realizar a classificação final entre gato ou cachorro.

```
Dense(1, activation = "sigmoid")
```

3.4.10 Compilação da rede

Utilizamos função *compile* da biblioteca Keras para compilar todo o modelo de rede neural. Três parâmetros foram utilizados:

loss: é a função de perda a ser utilizada durante o treinamento. Utilizamos a *binary_crossentropy*, que é comumente usada em problemas de classificação binária, como gatos ou cachorros.

optimizer: é o otimizador utilizado durante o treinamento, utilizamos *adam* que é amplamente utilizado na literatura.

metrics: especifica as métricas a serem avaliadas durante o treinamento e teste do modelo. Utilizamos a *accuracy*, que é uma métrica que mede a proporção de amostras classificadas corretamente em relação ao total de amostras, essa métrica também é amplamente utilizada em problemas de classificação.

Função e os parâmetros utilizados:

```
compile(loss = "binary_crossentropy",
    optimizer = "adam", metrics =
        ["accuracy"])
```

3.4.10 Salvar modelo em arquivo

Para salvar o treinamento da rede utilizamos a função `ModelCheckpoint` do Keras. Três parâmetros foram utilizados:

filepath: passa para a função o nome do arquivo de *output*.

save_best_only: para decidir se o melhor momento da rede será salvo.

monitor: é a métrica que será monitorada para determinar qual o momento de melhor desempenho do modelo.

Os valores de parâmetros utilizados foram:

```
ModelCheckpoint(filepath =
"modelo1.keras", save_best_only = True,
        monitor = "val_loss")
```

3.4.11 Treinamento

Por fim, resta descrever a função que irá dar inicio ao treinamento do modelo. A função *fit* da biblioteca Keras da inicio ao treinamento, basta passar o nome da pasta de treinamento, o número de épocas, o nome da pasta de validação e o checkpoint do modelo.

Utilizamos a função fit com os seguintes valores de parâmetros:

```
fit(dadosTrain, epochs = 30,
validation_data = dadosValidation,
    callbacks = modelo(...))
```

3.4.12 Máquina utilizada

Durante os testes e treinamento da rede utilizamos uma máquina com as seguintes especificações:

Sistema operacional Windows 10 Pro 64-bit.

Processador Intel Core i7-9700K 3.60GHz (8 CPUs).

Memória RAM 32GB.

Placa de vídeo NVIDIA GeForce RTX 3080.

4 Resultados e Análise

Nessa secção, apresentamos os resultados do treinamento da rede.

Inicialmente podemos perceber que o modelo aprendeu bastante sobre o conjunto de dados treino logo nas primeiras épocas, isso porque próximo da época 15 a taxa de erro no conjunto de dados treino já estava próxima de zero. Quanto ao conjunto de dados de validação a rede sofreu picos e próximo da época 15 houve uma taxa de erro significativa. No entanto, no geral, seguiu próximo a taxa de erro do treino.

A acurácia no treino cresceu significativamente e chegou próxima dos 95%. A acurácia na validação também sofreu picos em especial próximo a época 15 em que o modelo tomou um caminho ruim que foi corrigido posteriormente. No geral, a acurária na validação também alcançou a acurácia dos treinos.

Na acurácia dos testes gerais o sistema alcançou percentual de 94,72%. Taxa próxima dos 98,91% do líder do desafio na plataforma Kaggle.

Acreditamos que o que aconteceu próximo a época 15 do treinamento a rede reflete em uma escolha ruim da mesma que não apresentou problema para a acurácia do algoritmo.

Em testes com 10 imagens isoladas, dadas como *input* uma a uma, separadamente, para o modelo já treinado, houve acerto de todas as 10 imagens com *output* de cerca de 98% de certeza que a imagem analisada é um gato ou é um cachorro.

5 Conclusão

Nesse trabalho, apresentamos uma introdução as redes neurais convolucionais, com enfase no problema de classificação de imagens. Demonstramos uma implementação utilizando em sincronia as bibliotecas TensorFLow e Keras.

A rede neural treinada obteve acurácia de 94,72%, taxa que condiz com trabalhos relacionados e com resultados gravados na literatura.

É fato que as redes neurais convolucionais ganharam, nos últimos anos, grande destaque nos campos da visão computacional e do aprendizado de máquina. Esse destaque tem se mostrado positivo, visto a gama de possibilidades de aplicação da rede em problemas reais.

No campo da visão computacional, a rede tem levado vantagem em relação a outros métodos tradicionais, pois a arquitetura aproveita as características espaciais e de correlação presentes em dados de imagem.

A aplicação utilizada nesse artigo, tratou de um problema introdutório a teoria que envolve a rede, problemas mais complexos como identificação de objetos e pessoas em tempo real já vem sendo tratados com essa arquitetura.

Quanto a implementação da rede, alguns pontos merecem destaque:

- O tratamento da base de dados é imprescindível. O sucesso do modelo em parte está relacionado a subdivisão dos dados que permite que ele não só aprenda como verifique seu aprendizado, isso diminui o número de épocas e o uso de recursos computacionais.
- Deve existir um equilíbrio entre número de camadas e o tamanho da base de dados. Sabemos que no mundo real nem sempre dispomos de uma base de dados adequada para treinamento, em decorrência disso o sucesso da rede depende do equilíbrio do número de camadas.
- Operadores como *dropout*, *batch normalization* e *data augmentation* são valiosos para atingir uma acurácia ideal. Para lidar com a incerteza intrínseca dos dados esses operadores tem o poder de elevar a consistência do treinamento.

Portanto, a arquitetura convolucional demonstrou estar apta para lida com problemas relacionados a imagens e merece atenção para se aplicada junto a outras técnicas nos campos da visão computacional e do aprendizado de máquina.

6 Referências

[1] Machine Learning Attacks Against the Asirra CAPTCHA. Philippe Golle.

[2] Shafique, S. and Tehsin, S. (2018). Acute lymphoblastic leu-kemia detection and classification of its subtypes using pretrained deep convolutionalneural networks.Technology in Cancer Research Treatment, 17:1–7.

[3] Método de classificação de pragas por meio de rede neural convolucional profunda. Renann de Paula Rosa.

[4] Rede neural convolucional aplicada a visão computacional para detecção de incêndio. Andre Luiz Bertoni, Diego Vieira de Souza Feder.

[5] A Gentle Introduction to Batch Normalization for Deep Neural Networks - Jason Brownle.

[6] deeplearningbook.com.br/camadas-de-pooling-em-redes-neurais-convolucionais/

[7] Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[8] Segmentação de veias do pulso com uso de redes neurais convolucionais. Matheus Souza Cardoso.

[9] medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184

[10] Reconhecimento de produtos por imagem utilizando palavras visuais e redes neurais convolucionais. Guilherme Defreitas Juraszek.

[11] Laboratório iMobilis (UFOP). www2.decom.ufop.br/imobilis/redes-neurais-funcoes-de-ativacao

[12] Convolutional Networks and Applications in Vision. Yann Lecun, Koray Kavukcuogiu, Clement Farabet.