RECONHECIMENTO DE PADRÕES PARA DIFERENCIAÇÃO DE CACHORROS E GATOS

Aline Maria Gemelli

Bruno Afonso Cesca

Lucas Dos Santos Lemos

Mateus Henrique Calza

2021

Resumo

Neste trabalho é demonstrado na prática a utilização de redes neurais artificiais para o reconhecimento de padrões, onde serão aplicadas técnicas de programação e conceitos da literatura para realizar o reconhecimento de imagens de cachorros e gatos, fazendo a diferenciação entre os animais.

Palavras-chave: Inteligência Artificial, Redes neurais, Python, Keras, perceptron multicamadas.

1 Introdução

Para o desenvolvimento do trabalho foi utilizada a linguagem de programação Python, esta que é uma das linguagem mais utilizadas no mundo ficando na oitava posição, escolhe-se tal linguagem devido a grande gama de ferramentas e bibliotecas para de trabalhar com inteligência artificial e mais especificamente redes neurais que é o foco do trabalho em questão.

Para implementação do código se fez uso da biblioteca TensorFlow, devido ser uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações, análogo à forma como humanos aprendem e raciocinam.

1.1 Python

Esta foi a linguagem de programação utilizada para o desenvolvimento do trabalho, a mesma foi criada por Guido Van Rossum em 1991, focada em produtividade e legibilidade, produzindo assim um código de fácil entendimento para o programador. Esta linguagem oferece vários benefícios, como: gerenciamento de memória eficiente; baixo índice de palavras-chave; baixo uso de caracteres especiais no desenvolvimento do código. (BORGES, 2014)

Esta linguagem possui uma biblioteca padrão muito rica, na qual permite uma diversidade de abordagens já existentes para utilização, comportando trabalhos com números complexos, interfaces gráficas, entre muitas outras aplicações.
Como a linguagem é multiplataforma, trata-se de uma linguagem de código aberto, ou seja, a mesma pode ser modificada para compilar diversas outras linguagens, devido a isso, é uma linguagem muito utilizada para o trabalho com inteligência artificial, data mining e machine learn.

1.2 Multicamadas Perceptron

De acordo com Antunes et. al, Python e predição de dados usando redes neurais multicamadas, p. 2 apud Haykin (2001), as redes neurais artificiais são sistemas paralelos e distribuídos, constituídos de unidades simples (neurônios), que calculam determinadas funções matemáticas (principalmente não lineares) e apresentam capacidade de generalização, auto organização e processamento temporal. Possui camadas e conexões denominadas sinapses, onde é possível atribuir valores, que são responsáveis por ponderar as entradas de cada neurônio como forma de armazenamento do conhecimento de um determinado modelo.

Uma rede MLP(Multilayer Perceptron) ou PMC (Perceptron Multicamadas) é um tipo de rede neural artificial retroalimentada que gera uma série de saídas a partir de uma série de entradas, assim se diferenciando da rede Perceptron comum que gera uma só saída. Geralmente a rede MLP é utilizada em aplicações deep learning (MARSLAND, 2014)

1.3 Keras e Tensorflow

No desenvolvimento foi utilizado a API Keras na plataforma TensorFlow, proporcionando uma maior agilidade no desenvolvimento do código, pois a API Keras possui aprendizado profundo escrito em Python, executando na plataforma de aprendizado de máquina. Utilizamos a biblioteca Keras como nosso backend, incrementando as redes neurais convolucionais, onde as redes são baseadas em sequência e redes baseadas em grafos.

Já na plataforma de código aberto da Google, o TensorFlow, é uma plataforma robusta na qual possui código aberto para desenvolvimento de aprendizado de máquina.

1.4 Dataset Dogs vs Cats

Trata-se do dataset da plataforma Kaggle, na qual possui um arsenal com mais de 3(três) milhões de fotos de cães e gatos. Este surgiu por meio de uma parceria entre a Microsoft e a Petfinder.com.

O dataset foi usado originalmente como um CAPTCHA, porém foi intitulado de ASIRRA (Reconhecimento de Imagem de Espécie Animal para Restrição de acesso).

2 Metodologia

Partindo da definição das tecnológicas que seriam aplicadas buscou-se desenvolver a aplicação partindo do pressuposto que a aplicação em si fosse dinâmica, ou seja, funcionasse em tempo real. Com isso definido iniciou-se a implementação. Abaixo podemos observar o passo a passo e as tecnologias empregadas para realizar a diferenciação de cachorros e gatos utilizando a rede neural.

1 Preparo do dataset, organizando e dividindo para cerca de 25% para validação e 75% para treinamento;
2 Pré-processamento das imagens usando ImageDataGenerator do Tensorflow Keras;
3 Conversão das cores de 0 a 255, convertidas para 0 a 1;
4 Redimensionamento das imagens para uma dimensão de 200x200 pixels;
5 Leitura das imagens em modo de iteração não-cumulativa, para não incluir todas na memória de uma vez, evitando estouros;
6 Treinamento do modelo sequencial de rede neural convolucional utilizando um otimizador ADAM, com função de custo usando Entropia Cruzada Binária;
7 Camada utilizadas:
   - Convolucional 2D, com 32 filtros, com a função de ativação ReLU e kernel 3x3
   - Operação max pooling, com pool 2x2
   - Convolucional 2D, com 64 filtros, função de ativação ReLU, e kernel 3x3
   - Operação max pooling, com pool 2x2
   - Convolucional 2D, com 128 filtros, função de ativação ReLU, e kernel 3x3

- Operação max pooling, com pool 2x2
- Camada flatten tornando os dados em vetor 1D
- Camada densa de rede neural padrão, com 128 unidades, e função de ativação ReLU
- Camada densa de rede neural padrão, com 1 unidade, e função de ativação Sigmoid

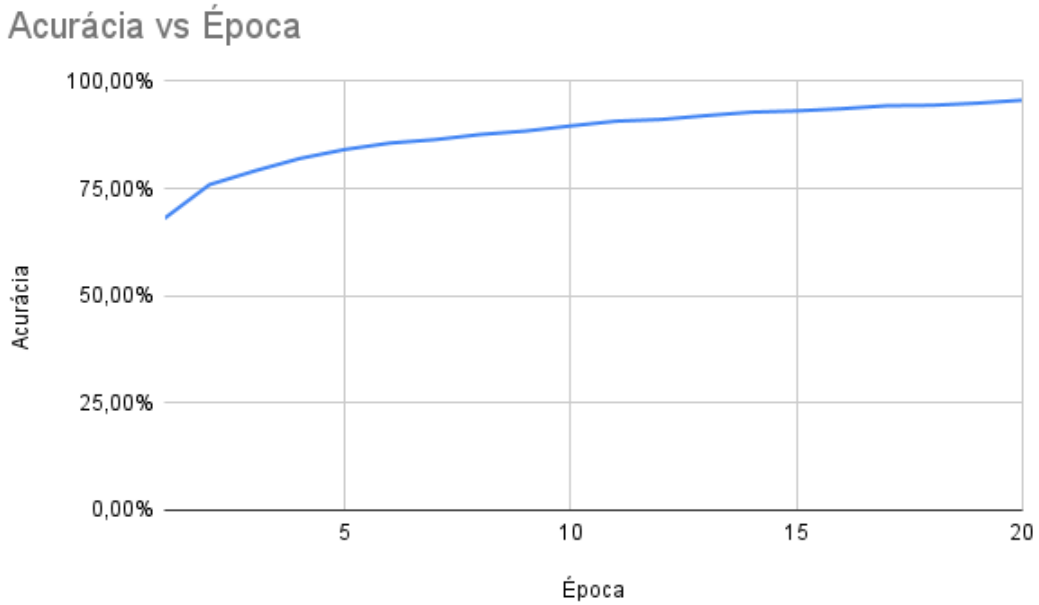
Com a primeira camada tem entrada de 200x200x3, isso significa que as imagens terão resolução de 200x200, em 3 canais de cores. A última camada tem 1 unidade de saída, para que a classificação seja binária.

O modelo foi inspirado no método VGG, que ao usar os múltiplos de 2 para a quantidade de filtros em sequência crescente nas camadas convolucionais 2D e kernel 3x3, sempre acompanhadas de camada Max pooling de pool 2x2, costuma trazer bons resultados para a rede.

Foi utilizada também uma otimização do tempo de treinamento usando GPU/CUDA. Ao mudar de uma CPU Intel Core i7 para processamento em GPU GeForce GTX 1660, o treinamento ficou 10x mais rápido, durando apenas cerca de 51 minutos para treinar, ao invés das quase 9 horas do CPU, para facilitar a implementação foi usado o Docker.

8 Após realização do treinamento o modelo foi validado usando as imagens separadas, obtendo 95,43% de precisão.

Neste modelo de treinamento, obteve a seguinte resposta ao longo das épocas:



Para a construção do código do modelo utilizou-se a biblioteca Keras, pois permite experimentação rápida com redes neurais profundas visto que roda em cima do TensorFlow, ela se concentra em ser fácil de usar, modular e extensível.

Para que o modelo fosse dinâmico, foi utilizada uma webcam para captura das imagens em tempo real, e tal resolução foi possibilitada pela biblioteca OpenCV. Então cada frame capturado através do OpenCV é adaptado com Numpy e PIL, e feita a predição usando Keras/Tensorflow no modelo já treinado.

Abaixo podemos observar o código final de treinamento, este que futuramente pode ser melhorado, visto que o mesmo está disponível no GitHub do curso.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator

modelo = Sequential()
modelo.add(Conv2D(32, (3, 3), activation='relu',
          kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
modelo.add(MaxPooling2D((2, 2)))
modelo.add(Conv2D(64, (3, 3), activation='relu',
          kernel_initializer='he_uniform', padding='same'))
modelo.add(MaxPooling2D((2, 2)))
modelo.add(Conv2D(128, (3, 3), activation='relu',
          kernel_initializer='he_uniform', padding='same'))
modelo.add(MaxPooling2D((2, 2)))
modelo.add(Flatten())
modelo.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
modelo.add(Dense(1, activation='sigmoid'))
modelo.compile(optimizer='adam', loss='binary_crossentropy',
               metrics=['accuracy'])

gerador_imagens = ImageDataGenerator(rescale=1.0/255.0,
                                     shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
iterador_imagens = gerador_imagens.flow_from_directory('./treinamento',
                                           class_mode='binary', batch_size=64, target_size=(200, 200))
iterador_imagens_validacao = gerador_imagens.flow_from_directory('./validacao',
                                           class_mode='binary', batch_size=64, target_size=(200, 200))
resultados = modelo.fit(iterador_imagens, steps_per_epoch=len(iterador_imagens),
                        validation_data=iterador_imagens_validacao, validation_steps=len(iterador_imagens_validacao),
                        epochs=20, verbose=2)

modelo.save('modelo.h5')

_, acc = modelo.evaluate(iterador_imagens_validacao, steps=len(iterador_imagens_validacao), verbose=0)
print('Precisão = %.2f' % (acc * 100.0))

```

Logo abaixo observamos a função que faz a predição, junta a imagem original, o mapa de calor, e sobrepõe com a classificação.

```python
from tf_explain.core.grad_cam import GradCAM
import numpy as np
import cv2

fonte = cv2.FONT_HERSHEY_SIMPLEX
fonte_escala = 0.8
texto_posicao = (10, 230)
texto_cor = (255, 0, 0)
texto_largura = 1
rodape = np.zeros((50,400,3), np.uint8)

def imagem_predicao_analise(modelo, imagem):
  imagem_array = imagem.reshape(1, 200, 200, 3)
  imagem_array = imagem_array.astype('float32')
  imagem_array = imagem_array / 255.0

  resultado = modelo.predict(imagem_array)
  if resultado[0][0] <= 0.5:
    acuracia = 1 - resultado[0][0]
    classificacao = 'Gato'
  else:
    acuracia = resultado[0][0]
    classificacao = 'Cachorro'
  texto = classificacao + ' {0:.2f}%'.format(acuracia * 100)

  dados_explainer = (imagem_array, None)
  explainer = GradCAM()
  imagem_analise = explainer.explain(dados_explainer, modelo, class_index=0)
  imagem_analise = cv2.cvtColor(imagem_analise, cv2.COLOR_RGB2BGR)

  resultado = np.hstack((imagem, imagem_analise))

  resultado = np.vstack((resultado, rodape))
  resultado = cv2.putText(resultado, texto, texto_posicao, fonte,
                          fonte_escala, texto_cor, texto_largura, cv2.LINE_AA)

  return resultado
```

Abaixo segue o código que faz a análise em tempo real com a webcam, reaproveitando a função da figura acima.

```python
import cv2
import numpy as np
from PIL import Image
from tensorflow.keras import models
from gui import imagem_predicao_analise

modelo = models.load_model('./modelo.h5')
video = cv2.VideoCapture(0)

while True:
    _, frame = video.read()
    imagem_pil = Image.fromarray(frame, 'RGB')
    imagem_pil = imagem_pil.resize((200, 200))
    imagem_array = np.array(imagem_pil)

    resultado = imagem_predicao_analise(modelo, imagem_array)
    cv2.imshow("Resultado", resultado)

    key = cv2.waitKey(1)
    if key == 27:
        break

video.release()
cv2.destroyAllWindows()
```

Logo abaixo segue o trecho de código responsável por realizar a predição de imagens armazenadas reaproveitando a função de predição.

```
from os import makedirs
from os import listdir
from shutil import copyfile
from random import seed
from random import random
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from tensorflow.keras import models
from gui import imagem_predicao_analise
import cv2

modelo = models.load_model('./modelo.h5')

pasta_predicao = './predicao'
pasta_predicao_saida = './predicaoSaida'


for file in listdir(pasta_predicao):
  arquivo_predicao = pasta_predicao + '/' + file

  if file.lower().endswith('.jpg') or file.lower().endswith('.jpeg'):
    arquivo_predicao_saida = pasta_predicao_saida + '/' + file

    imagem = load_img(arquivo_predicao, target_size=(200, 200))
    imagem = img_to_array(imagem)
    imagem = cv2.cvtColor(imagem, cv2.COLOR_RGB2BGR)

    resultado = imagem_predicao_analise(modelo, imagem)

    cv2.imwrite(arquivo_predicao_saida, resultado)

```

Como era necessário fazer a leitura e montagem das imagens, optou-se por utilizar OpenCV, por ser uma biblioteca multiplataforma, para o desenvolvimento de aplicativos na área de visão computacional .

Também se fez uso do TF Explain para visualizar no formato de heatmap o que a rede neural convolucional julgou relevante para classificar, usando algoritmo GradCAM, e PIL para auxiliar no redimensionamento. Para tornar a codificação de fácil utilização, entendimento e manutenção optou-se por usar a biblioteca Numpy, pois além de servir de base para as outras bibliotecas, tem facilitadores que auxiliam na montagem das imagens.

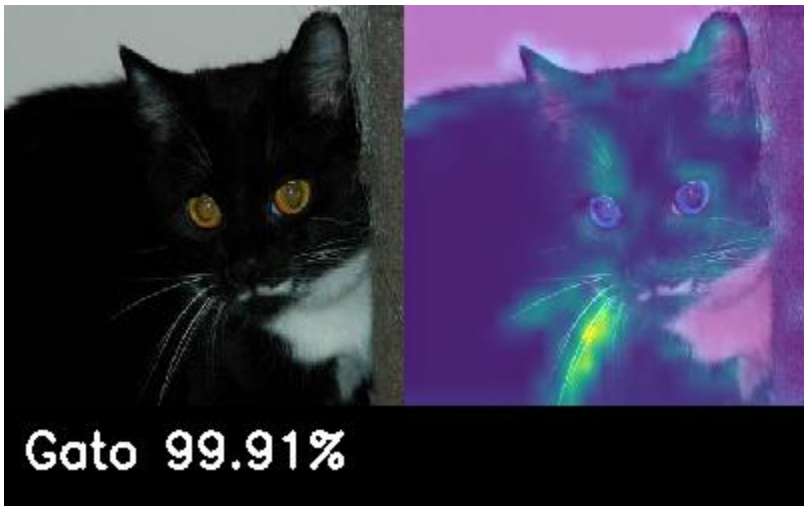
3 Resultados e discussões

Como principais resultados podemos destacar a alta acurácia do modelo em comparação com outros encontrados nas pesquisas realizadas, vale ressaltar que como o modelo foi treinado para realizar a detecção de cachorros e gatos, o modelo se torna ineficaz quando utilizado para predição de outros tipos de animais, porém com a devida modificação da base de dados e seguida de um treinamento, o modelo se torna muito eficaz.

Em nossos testes obteve-se um bom resultado, estes que podem ser observados nas imagens abaixo.


Gato 99.24%


Gato 81.31%


Gato 99.91%


Cachorro 67.92%


Cachorro 99.96%


Gato 99.99%

Abaixo observam-se imagens que mostram algumas predições incorretas.

4 Conclusão

Conclui-se então que o trabalho atingiu os objetivos previamente propostos, com isso foi possível entender o funcionamento e os principais conceitos de uma rede neural, bem como entender mais sobre o funcionamento de algumas bibliotecas e suas vantagens e desvantagens. Com o uso de webcam para resultados em tempo real os resultados são satisfatórios, porém inferiores ao uso com fotografias estáticas. Isso provavelmente se deve às imagens do dataset que raramente incluem movimentos e se baseiam em um estilo de captura diferente, a análise mais apurada dessas condições são uma sugestão de trabalho futuro.

Referências

[1] KERAS. Keras API reference. Disponível em: https://keras.io/api/>. Acesso em: 09/09/2021

[2] PLAYGROUND PREDICTION COMPETITION. Dogs vs. Cats. Disponível em: https://www.kaggle.com/c/dogs-vs-cats/data>. Acesso em: 11/09/2021

[3] PYTHON. Python 3.9.7 documentation. Disponível em: https://docs.python.org/3/>. Acesso em: 09/09/2021

[4] TENSORFLOW. Bibliotecas e extensões. Disponível em: https://www.tensorflow.org/resources/libraries-extensions?hl=pt-br>. Acesso em: 10/09/2021

[5] JASON BROWNLEE. How to Make Predictions with Keras. 2018. Disponível em: https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/>. Acesso em: 10/09/2021

[6] AVIJAYABHASKAR J . Tutorial on using Keras flow_from_directory and generators. 2018. Disponível em: https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720>. Acesso em: 12/09/2021

[7] RAMPRASAATH R. SELVARAJU · MICHAEL COGSWELL · ABHISHEK DAS · RAMAKRISHNA VEDANTAM · DEVI PARIKH · DHRUV BATRA. 2019. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Disponível em: https://arxiv.org/pdf/1610.02391.pdf >. Acesso em: 16/09/2021

[8] TF-EXPLAIN. tf-explain usage. Disponível em: https://tf-explain.readthedocs.io/en/latest/usage.html>. Acesso em: 13/09/2021

[9] MATE LAB. Everything you need to know about Neural Networks. 2017. Disponível em: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491?utm_source=pocket-chrome-readthedocs>. Acesso em: 10/09/2021

[10] JASON BROWNLEE. How to Choose Loss Functions When Training Deep Learning Neural Networks. 2019. Disponível em: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>. Acesso em: 13/09/2021

[11] JASON BROWNLEE. How to Classify Photos of Dogs and Cats (with 97% accuracy). 2019. Disponível em: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/>. Acesso em: 13/09/2021

[12] ARTUR KĘPKA. Detector-Classifier Neural Network Architecture with TensorFlow. Disponível em:

https://towardsdatascience.com/detector-classifier-neural-network-architecture-with-tensorflow-fc5dd9dce3e>. Acesso em: 14/09/2021

[13] ZAHRA ELHAMRAOUI. Training a CNN from scratch on a small dataset. 2020. Disponível em: https://medium.com/analytics-vidhya/training-a-cnn-from-scratch-on-a-small-dataset-59038686d6ac>. Acesso em: 13/09/2021

[14] JASON BROWNLEE. How to Develop a CNN From Scratch for CIFAR-10 Photo Classification. 2019. Disponível em: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>. Acesso em: 13/09/2021

[15] SIMONYAN JKAREN, ZISSERMAN ANDREW.. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. 2015. Disponível em: https://arxiv.org/pdf/1409.1556.pdf>. Acesso em: 16/09/2021