Intel Image Classification Using Machine Learning

Daniel Correia, nº mec. 90480, danielvalacorreia@ua.pt

***Resumo*—O reconhecimento rápido de diferentes tipos de imagens e a sua classificação é uma área de grande importância e com variadíssimas aplicações em diversos ramos distintos. No entanto, utilizando métodos de programação tradicionais é extremamente difícil e demoroso desenvolver um sistema que consiga classificar rapidamente diversos tipos de imagens. Contudo, a classificação de diversos tipos de imagens é um tema ideal para modelos de *machine learning* e *deep learning*. Bem como reconhecimento de expressões faciais, localização de objetos em tempo real que são outros temas mais profundos de reconhecimento de imagens. Neste projeto é abordado o tema de classificação de diferentes imagens que se inserem numa de seis possíveis categorias de paisagens diferentes.**

***Index Terms*—Support, Vector, Machine, Convolutional, Neural, Network, Keras, Python, Tensorflow, Intel, Image, Classification.**

I. Introdução

NO âmbito de aprendizagem automática, o tema de reconhecimento de imagens é um tópico muito abordado e estudado. Conseguir desenvolver um sistema que consiga distinguir rapidamente imagens, ou até objetos que se encontram numa imagem, com um grau de certeza elevado, é algo muito procurado em enumeras áreas. O tema abordado em específico, a distinção de diferentes tipos de paisagens e categorizá-las em seis tipos diferentes de labels: 'sea', 'forest', 'buildings', 'street', 'glacier', 'mountain'.

Para conseguir classificar estas paisagens, foram desenvolvidos e treinados dois modelos de aprendizagem automática diferentes, sendo eles avaliados e comparados, entre eles e com outros modelos desenvolvidos por outras pessoas em termos de precisão das classificações.

A razão da escolha do tema específico de classificação de imagens tem como motivo a sua aplicação em áreas comuns de aprendizagem automática, e por ter um enorme financiamente de diversos ramos para o seu desenvolvimento.

II. Estado da Arte

É normal recorrer a Redes Neurais Convolucionais[Geo21] no desenvolvimento de um modelo para este tipo de problemas, como pode ser verificado nas referências identificadas. Como habitual na área de *machine learning* e *deep learning*, foi utilizada a linguagem predominantemente escolhida, Python[Ros] na plataforma Jupyter Notebook[Pér]. Este desafio de desenvolvimento de um modelo de machine learning foi originalmente proposto pela Analytics Vidhya[Vid], uma comunidade prestigiada de Análise e Ciência de Dados. Porém, o conjunto de dados relativos aos diferentes tipos de paisagem bem como as suas classificações foram disponibilizados pela Intel[Int]. Sumariando, foi lançado um desafio à comunidade de construir uma Rede Neural que conseguisse classificar as imagens deste dataset com melhor a melhor precisão possível.

Lançado o desafio, vários projetos e artigos nasceram sustentados no mesmo, em que alguns deles foram investigados e estudados para uma melhor abordagem a este problema, sendo estes:

- Solução em primeiro lugar na Private Leaderboard Rank, publicada no Towards Data Science[Say19]
- Artigo publicado no Medium[Rog19]
- Um paper sobre uma novel arquitetura, Wise-SrNet[Moh21]
- Uma solução desenvolvida por Vincent Liu, publicada no Kaggle[Liu20]

III. Dataset

A. Estrutura

No âmbito do desenvolvimento deste projeto, foram desenvolvidos dois modelos diferentes, os quais foram comparados os resultados das suas precisão. Os dados destes dois modelos foram os mesmos. Foi utilizado o *dataset* da Intel[Int] disponibilizado no Kaggle[Ban], este já dividido em dois subconjuntos: treino e teste. O conjunto de treino foi então dividido em dois subconjuntos: treino e validação, sendo que estes dois incluem a resposta correcta para cada amostra.

Cada amostra consiste numa imagem de 150x150 pixeis, com três canais de cor (RGB), o que resulta num total de 67500 *features* por amostra.

- Dados de Treino: 14034 amostras
- Dados de Validação: 2806 pessoas
- Dados de Teste: 3000 amostras

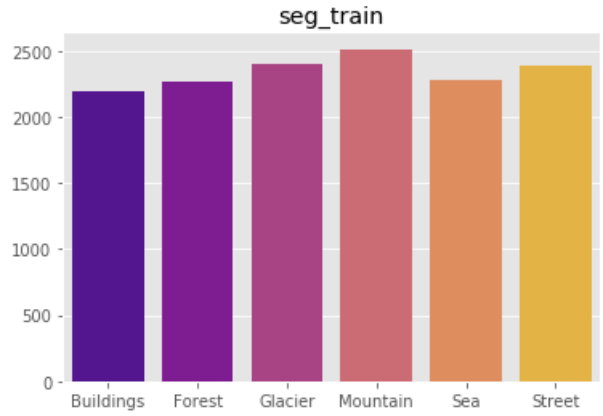

Figura 1. Frequência de classes dos dados de treino

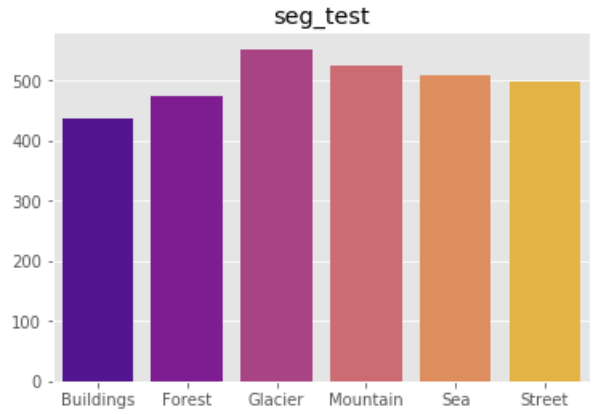

Figura 2. Frequência de classes dos dados de teste

Em termos de proporção de imagens por classe, é possível observar nas figuras 1 2 que os dados utilizados, tanto como os de treino quantos os de teste, se encontram relativamente balanceados. Cada classe conta com quase a mesma quantidade de amostras, sendo providenciado assim já pelo dataset[Ban] fornecido, um conjunto de dados já uniformizado.

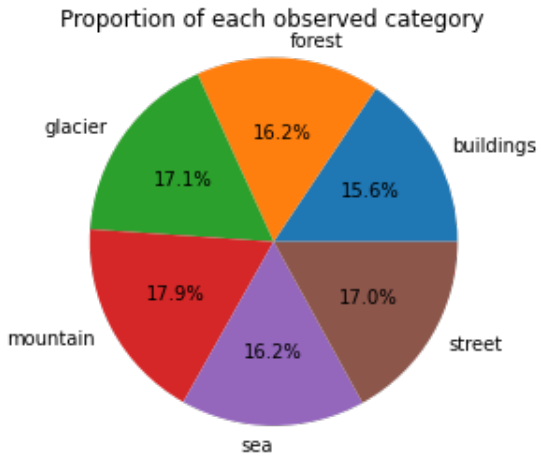

Figura 3. Proporção de classes

Como já foi referido anteriormente, cada imagem do dataset pode ser classificada em uma de seis classes: floresta, edifícios, glaciares, montanha, mar ou rua. Os dados de treino,validação (0.2 dos dados de treino) e teste, já se encontram organizados em subdirectórios, em que cada um correspondente à classe na qual se insere. Portanto, na leitura dos dados para memória é logo feita a salvaguarda das classes a que cada uma das amostras pertence.
Na figura 4 podemos ver exemplos das amostras fornecidas no dataset[Ban] com a classificação correspondente.

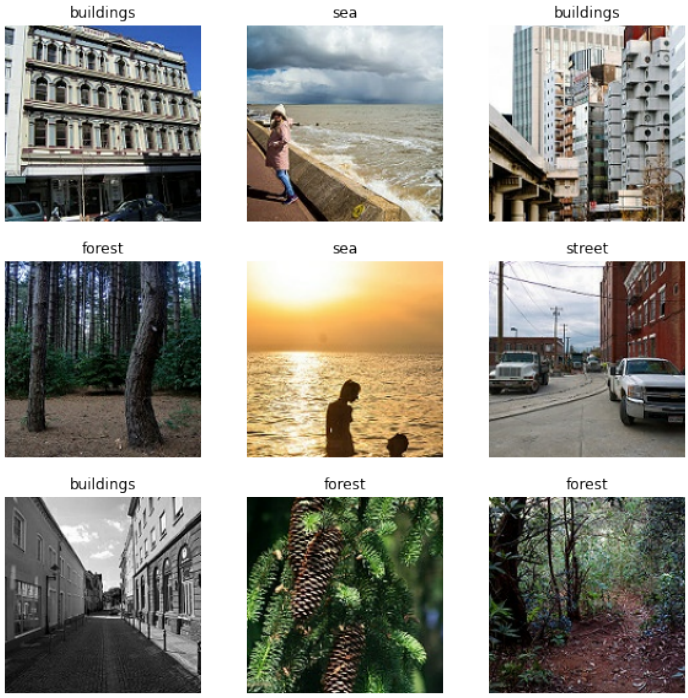

Figura 4. Amostra do Dataset

IV. Modelos

Os modelos utilizados foram *Support Vector Machine*, o qual foi implementado recorrendo à biblioteca *open-source* sklearn[Bos] e *Convolutional Neural Network*, implementada recorrendo à biblioteca Keras[Cho] também *open-source*, sendo ambas escritas em Python.

A. Support Vector Machine

O modelo de máquina de vetores de suporte é um modelo que se rege pelo aprendizado supervisionado, analisando dados e reconhecendo padrões. A sua maior utilização advém dos problemas de classificação, sendo por isso um dos modelos escolhidos para este problema.

B. Convolutional Neural Network

O modelo de *Convolutional Neural Network* é um modelo muito eficaz vagamente baseado no cérebro humano, utilizando diversas células dispostas em camadas.

V. Support Vector Machine

A. Leitura dos Dados

Os dados de treino, validação e de teste são armazenados em estrutura de dados do tipo *numpy.array*. São utilizados quatro arrays de dados, o primeiro é o X_train (14034, 22500) que contém as imagens utilizadas para criar o modelo, X_test (3000, 22500) que contém as imagens utilizadas para teste do modelo, y_train (14034,) que contém as labels das imagens utilizadas no treino, ou seja, qual das seis diferentes categorias cada imagem de treino pertence, y_test (3000,) que contém as labels das imagens utilizadas no teste.

B. Pré-Processamento

Nas especificações do *dataset*[Ban] é mencionado que as imagens já se encontram todas nas dimensões 150x150 px com três canais de cor (RGB). No entanto, foi lido durante a pesquisa e desenvolvimento do trabalho que algumas das imagens do dataset[Ban] não se encontravam nas dimensões a cima referidas, portanto para garantir uma uniformidade do conjunto de dados, foi feito o redimensionamento de todas as imagens do dataset utilizando a biblioteca [Cor].

C. Estrutura Geral

Foi utilizado como base do modelo o parâmetro imutável de 67500 features como entrada (150x150x3) e as 6 classes como saída: floresta, edifícios, glaciares, montanha, mar ou rua.
Foram testados os algoritmos *Linear Support Vector Machine* e *Nonlinear Support Vector Machine* com diferentes *kernel*.
Foram procurados os melhores híper-parâmetros C, gamma e sigma, de modo a que a métrica *precisão* seja maximizada. Foram feitos todas as combinações entre parâmetros e escolhidos os que obtiveram melhores resultados. Os valores utilizados foram os seguintes:

- **C:**
  - 0.01
  - 0.03
  - 0.1
  - 0.3
  - 1
  - 3
  - 10
  - 30
- **Gamma:**
  - 0.01
  - 0.03
  - 0.1
  - 0.3
  - 1
  - 3
  - 10
  - 30
- **Sigma:**
  - 0.01
  - 0.03
  - 0.1
  - 0.3
  - 1
  - 3
  - 10
  - 30

Depois de calcular os melhores híper-parâmetros foi calculada a matriz de confusão, da qual foram calculados os valores de, *precision*, *recall* e *f1 score* para cada uma das categorias de imagem.

Precision	52.8	60.2	53.9	61.1	32.3	55.5
Recall	37.3	87.8	80.7	84.9	25.2	35.4
F1 Score	42.2	73.5	64.0	68.6	28.4	42.0

Tabela I
Precision, Recall e F1 Score

VI. Convolutional Neural Network

A. Leitura dos Dados

Os dados de treino e dados de validação são lidos e armazenados numa estrutura de daodos definida pela biblioteca *tensorflow*. Foram denominados por X_train e X_validation correspondentemente, e lidos através de um método da mesma biblioteca que permite ler e organizar imagens em dados. Estes dados estão estruturados em subdirectórios, da mesma maneira que estavam distribuídos pelos subdiretórios no dataset original[Ban].

B. Pré-Processamento

Para melhorar o processamento das *features* das amostras, estas, que originalmente variam de 0 a 255, por se tratarem de valores de cor para o seu respectivo canal, são processados para valores entre 0 e 1.

C. Estrutura Geral

No desenvolvimento do modelo da rede neural convolucional, foram feitas várias iterações diferentes, nas quais se tentou ajustar os parâmetros por forma a melhorar os resultados, pelo que, a partir de uma arquitetura base da rede, foram feitas alterações e modificações conforme necessário. Esta arquitetura base foi a disponibilizada na aula sobre redes convolucionais.
A rede convolucional foi construída com base na biblioteca keras[Cho], pelo que as camadas introduzidas correpondem a tipos de dados da mesma.
Em ambos os modelos criados, os tempos de treino foram incrivelmente altos de modo a que a quantidade de testes e configurações diferentes não foram as desejadas. Apesar disso foram conseguidos resultados aceitáveis, mas que poderiam ser melhorados.

1) Modelo Base:

- **Input Layer**
- Rescaling Layer: camada de pré-processamento.
- **Conv2D:**
  - 32 filtros
  - kernel 5x5
  - strides: 1
- BatchNormalization
- Activation: 'relu'
- GlobalAveragePooling2D: kernel 2x2
- **Dense:** função softmax

Sobre este modelo base, foram usados os seguintes híper-parâmetros:

- Loss: função *sparse categorical crossentropy*
- Optimizer: adam
- Epochs: 50
- Batch Size: 32

Este modelo mostrou-se muito ineficaz, conseguindo uma precisão de treino de **60%** e uma precisão de teste de **40%**. Para além disso, devido ao elevado número de epochs, o modelo mostrou-se demasiado lento no seu treino, demorando uma enorme quantidade de tempo. Os valores de precisão foram contudo demasiado baixos e a precisão de validação é muito inferior à de treino, o que demonstra overfitting. Estes problemas foram abordados nas seguintes iterações.

2) Próxima iteração:

- **Input Layer**
- Rescaling Layer: camada de pré-processamento.
- Sequencial
- ZeroPadding2D: 3x3
- **Conv2D:**
  - 32 filtros
  - kernel 5x5
  - strides: 2
- BatchNormalization
- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Conv2D:**
  - 32 filtros
  - kernel 3x3
  - strides: 1
- BatchNormalization
- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Conv2D:**
  - 64 filtros
  - kernel 3x3
  - strides: 1
- BatchNormalization
- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Conv2D:**
  - 128 filtros
  - kernel 3x3
  - strides: 1
- BatchNormalization
- Activation: 'relu'
- **Flatten**
- **Dense:** função softmax

Os híper-parâmetros foram modificados, diminuindo a quantidade de epochs e aumentando o batch size:

- Loss: função *sparse categorical crossentropy*
- Optimizer: adam
- Epochs: 30
- Batch Size: 128

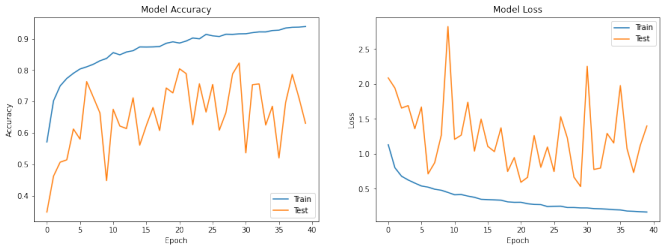

Figura 5. Precisão e Perda da Rede Neural Convolucional

Para este modelo, a precisão de treino foi **73%**, no entanto a precisão de teste manteve-se num valor mais baixo de **62%**. Neste modelo, como é possível observar em 5, continua a haver um overfitting. Foi diminuido o número de epochs e aumentado o batch size o que melhorou a quantidade de tempo de treino do modelo, cerca de **54s** por epoch.

3) Versão final:

- **Input Layer**
- Rescaling Layer: camada de pré-processamento.
- Sequencial
- ZeroPadding2D: 3x3
- **Conv2D:**
  - 32 filtros
  - kernel 5x5
  - strides: 2
- BatchNormalization
- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Conv2D:**
  - 64 filtros
  - kernel 3x3
  - strides: 1
- BatchNormalization

- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Conv2D:**
  - 128 filtros
  - kernel 3x3
  - strides: 1
- BatchNormalization
- Activation: 'relu'
- MaxPooling2D: kernel 2x2
- **Dropout:** rate 0.3
- **Flatten**
- **Dense:** função softmax

Para este modelo final, a precisão de treino foi **94%**, e uma precisão de teste de **84%**.. No entanto, para subir uma quantidade significativa, seria preciso um tempo de treino extremamente maior, dado que, devido ao volume de dados e features, uma única epoch demora consideravelmente a concluir.

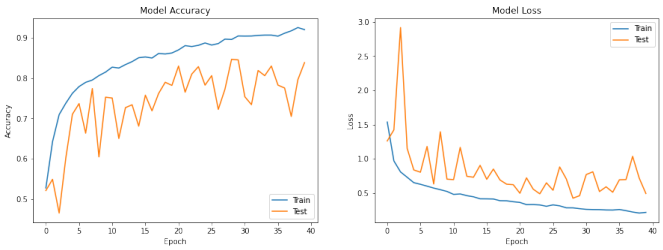

Figura 6. Precisão e Perda da Rede Neural Convolucional

Analisando o gráfico 6, é possível observar que os valores de teste são bastante variáveis o que é normal utilizando o otimizador **adam** e por serem poucas iterações (epochs), o que é compensado pelo tempo de treino mais reduzido.

VII. Resultados

Após analisar ambos os modelos, é evidente que o mais adequado é a Rede Neural Convolucional. Os melhores resultados obtidos foram de uma precisão de **94%** nos dados de treino e de **84%** nos dados de teste. Estes resultados vão em linha com a tendência na área de processamento de imagem, na qual se faz muito uso das Redes Neurais para diversos problemas. Neste projecto, a Rede Neural Convolucional final, embora tenha conseguido resultados satisfatórios, estes poderiam ser melhorados como é abordado na secção VIII.

VIII. Conclusão

Apesar de ter sido conseguida uma precisão de 84% nos dados de teste, o modelo da Rede Neural Convolucional obtido aquém dos valores de precisão obtidos atualmente. Uma das abordagens que poderia melhorar o modelo seria o uso da técnica de *data augmentation*, que a partir do *dataset* utilizado, seria possível criar mais exemplos através de transformações dos exemplos já obtidos.

Porém, isto também significaria um volume de dados consideravelmente superior, o que resultaria num maior tempo de treino necessário. Para além, disto, um estudo mais intensivo sobre os híper-parâmetros utilizados bem como as configurações das camadas usadas na rede, resultariam num desempenho melhor, após mais testes efectuados.
Poder-se-ia também ter recorrido a técnicas e modelos mais avançados, podendo ter sido feita uma pesquisa mais exaustiva e tendo sido feito uso de diferentes bibliotecas e frameworks mais especializadas, como FastAI ou mesmo até utilizando a técnica da solução vencedora em [Say19], onde é utilizada uma combinação de CNN e kNN.
Poderia ter sido utilizada também a técnica *Transfer Learning* em que é utilizada uma Rede Neural já treinada como por exemplo, Inception V3 ou ResNet50, nas quais são apenas treinadas as últimas camadas da Rede.
É de notar que todas estas técnicas implicariam que o tempo de teste do modelo e a quantidade de memória necessária fossem consideravelmente superiores, o que resultaria num maior tempo de treino necessário.

IX. Contribuições

O trabalho foi realizado por apenas um elemento.

Referências

[Rog19] Steve Rogers. "Image Classification using Keras: Intel Scene Classification Challenge". Em: *Medium* (2019). URL: https://medium.com/beovolytics/image-classification-using-keras-intel-scene-classification-challenge-b0b087374654. (accessed: 05.02.2022).

[Say19] Afzal Sayed. "1st Place Solution for Intel Scene Classification Challenge". Em: *Towards Data Science* (2019). URL: https://towardsdatascience.com/1st-place-solution-for-intel-scene-classification-challenge-c95cf941f8ed. (accessed: 05.02.2022).

[Liu20] Vincent Liu. "Intel Image Classification (CNN - Keras)". Em: *Kaggle* (2020). URL: https://www.kaggle.com/vincee/intel-image-classification-cnn-keras. (accessed: 05.02.2022).

[Geo21] Pétia Georgieva. "Deep Learning - Convolutional Neural Networks (ConvNets)". Em: *Slides Teóricos* (2021). (accessed: 05.02.2022).

[Moh21] Mohammad Reza Mohammadi. "Wise-SrNet: A Novel Architecture for Enhancing Image Classification by Learning Spatial Resolution of Feature Maps". Em: *Papers With Code* (2021). URL: https://paperswithcode.com/paper/wise-srnet-a-novel-architecture-for-enhancing. (accessed: 05.02.2022).

[Ban] Puneet Bansal. *Dataset Intel Image Classification*. URL: https://www.kaggle.com/puneet6060/intel-image-classification. (accessed: 05.02.2022).

[Bos] Joris Van den Bossche. *Sklearn*. URL: https://scikit-learn.org/stable/. (accessed: 05.02.2022).

[Cho] François Chollet. *Keras*. URL: https://keras.io. (accessed: 05.02.2022).

[Cor] Intel Corporation. *OpenCV*. URL: https://opencv.org/. (accessed: 05.02.2022).

[Int] *Intel*. URL: https://www.intel.com. (accessed: 05.02.2022).

[Pér] Fernando Pérez. *Jupyter Notebook*. URL: https://jupyter.org/. (accessed: 05.02.2022).

[Ros] Guido van Rossum. *Python*. URL: https://www.python.org. (accessed: 05.02.2022).

[Vid] *Analytics Vidhya*. URL: https://datahack.analyticsvidhya.com. (accessed: 05.02.2022).