LICENCIATURA EM ENGENHARIA INFORMÁTICA
2023/2024

Inteligência Computacional

Garbage Classification
Gravitational Search Algorithm
Meta 3

Bruno Oliveira - 2019136478
Micael Eid – 2019112744

Contents

Cap 1 – Descrição do Problema

Na cadeira de Inteligência Computacional foi-nos proposto fazer um trabalho de Classificação de imagens seguindo o tema da Classificação Sustentável. Escolhemos o Garbage Classification que essencialmente seria a classificação de lixo em 10 classes diferentes, das quais escolhemos 5 (Papel, Metal, Cartão, Plástico e Vidro).

Ao longo do semestre fomos aperfeiçoando o modelo para classificação das imagens nestas classes.

Nesta meta iremos recorrer ao VGG (rede neuronal convulacional), o GSA (algoritmo escolhido na meta anterior), GridSearch (encontrar parâmetros ótimos) e o RandomSearch (encontrar parâmetros ótimos).

Com recurso a estes métodos iremos obter o melhor resultado possível.

Cap 2 – Metodologias Utilizadas

Gravitational Search Algorithm

O Algoritmo de Busca Gravitacional (GSA) é uma técnica de otimização baseada em fenómenos gravitacionais que ocorrem na natureza. Desenvolvido por Rashedi, Nezamabadi-pour e Saryazdi em 2009, o GSA é inspirado no comportamento de corpos celestes que interagem entre si devido à força gravitacional.

GSA é inspirado nas leis físicas da gravidade descobertas por Isaac Newton. A escolha entre eles dependerá da natureza específica do problema e das características desejadas para a otimização

Vantagens

- GSA é relativamente simples de entender e implementar, tornando-o acessível para utilizadores menos experientes.
- Comparado a alguns algoritmos de otimização, como o PSO, o GSA possui menos parâmetros para ajustar., o que facilita na configuração do algoritmo.
- GSA tende a ser eficaz na exploração global do espaço de busca, graças à influência gravitacional entre as partículas. Isso pode ser útil para problemas onde encontrar a solução global é crucial.
- Inspiração nas leis físicas da gravidade, resultando em comportamentos interessantes e eficientes.

Desvantagens

- Por vezes o GSA pode convergir para a solução ótima de maneira mais lenta em comparação com outros algoritmos.
- Apesar de ter menos parâmetros que alguns algoritmos, a escolha dos mesmos pode influenciar significativamente o desempenho do algoritmo.
- A eficácia do GSA pode variar dependendo das características específicas do problema.
- O desempenho do GSA pode ser afetado por problemas de escala, especialmente em espaços de busca de alta dimensão. Isso pode tornar o GSA menos eficiente para problemas complexos.

VGG16

VGG16 é uma arquitetura de rede neural de convulocional(CNN) usada para reconhecimento de imagens. Utiliza 16 camadas e é considerado uma das melhores arquiteturas de modelos de visão até hoje.
K. Simonyan e A. Zisserman da Universidade de Oxford propuseram este modelo e publicaram-no num artigo chamado Very Deep Convolutional Networks for Large-Scale Image Recognition.
O modelo VGG16 pode atingir uma precisão de teste de 92,7% no ImageNet, um conjunto de dados que contém mais de 14 milhões de imagens de treinamento em 1.000 classes de objetos.

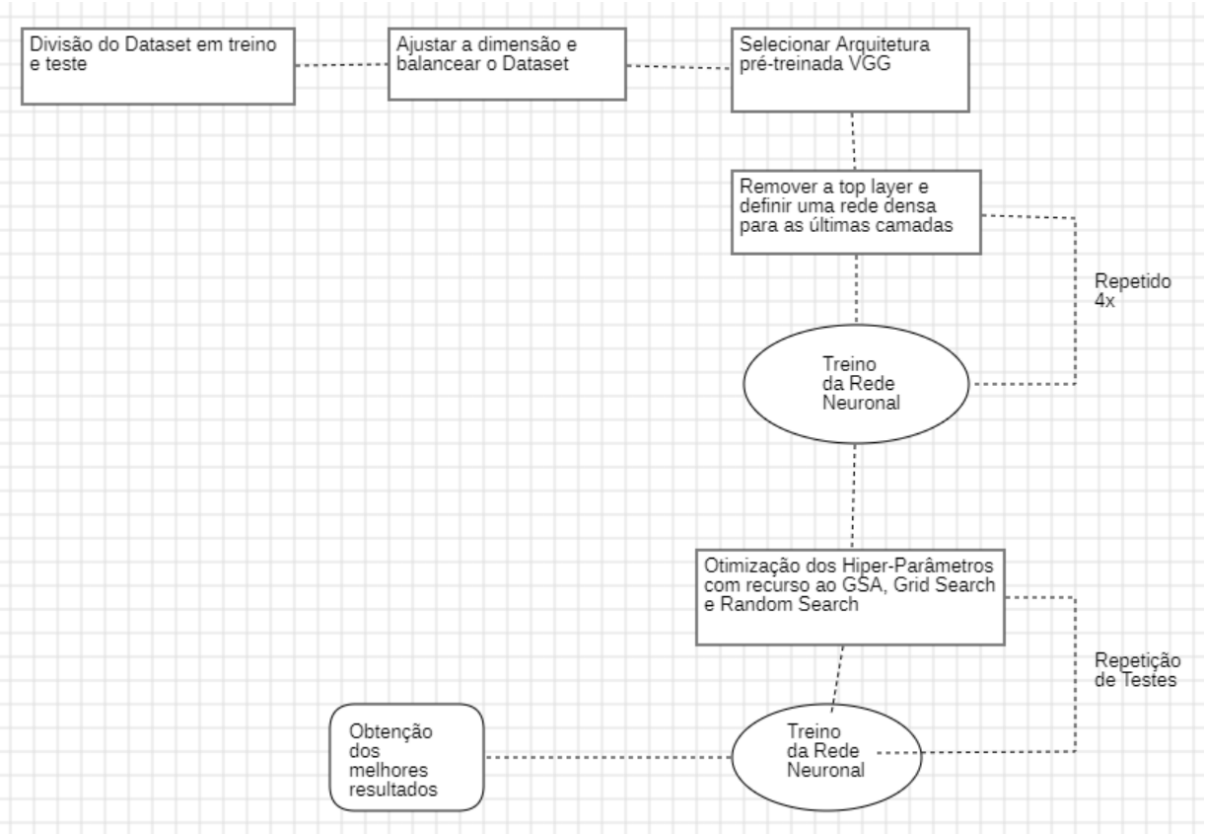
Grid Search

É uma pesquisa exaustiva para selecionar um modelo. No GridSearch configuramos uma grid de valores e hiperparâmetros e, para cada combinação, treinamos o modelo e validamos os dados de teste.
Nesta abordagem cada combinação de valores dos hiperparâmetros é testada, o que pode ser bastante ineficiente, consumindo muito tempo e recursos computacionais.

Random Search

À semelhança da GridSearch a RandomSearch cria uma grelha para os hiperparâmetros, mas, ao contrário da GridSearch, este escolhe uma combinação aleatória de valores e realiza os testes conforme.
Nesta abordagem o consumo computacional é claramente melhor e mais vantajoso que a GridSearch.

Cap 3 – Arquitetura de Código



Cap 4 – Descrição da Implementação de Algoritmos

Gravitaional Search Algorithm

Definimos os parâmetros do GSA:

- GO
- Dimension
- Num_Agents
- Iterations
- Ub
- Lb
- Fitness_function

O GSA está a ser usado da biblioteca SwarmPackagePy, definimos os hiperparâmetros a utilizar e os seus valores foram colocados no ub e lb da função.
Após esta definição corremos o GSA com a fitness_funcion criada por nós e retiramos os melhores valores para os hiperparâmetros que queremos otimizar.
Após recebermos estes valores testamos na função VGG e retiramos os melhores resultados.

VGG16

Para a utilização do VGG nós tiramos a camada superior da função:

```
# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)
base_model.trainable = False

# Flatten and dense layers
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(128, activation='relu')
dense_layer_2 = layers.Dense(100, activation='relu')
#dense_layer_3 = layers.Dense(194, activation='relu')
# dense_layer_4 = layers.Dense(53, activation='relu')
# dense_layer_5 = layers.Dense(53, activation='relu')
prediction_layer = layers.Dense(num_classes, activation='softmax')

# Build the model
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    #dense_layer_3,
    # dense_layer_4,
    # dense_layer_5,
    prediction_layer
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Depois treinamos e testamos o modelo obtendo valores bastante superiores aos das metas passadas.

RandomSearch

Para este tipo de pesquisa criamos uma grelha com valores dos hiperparâmetros que queremos otimizar.
Criamos um modelo para adição de camadas de densas e de otimização e ainda usamos o cross validation para a validação do problema.

```
# Define the model
def create_model(num_layers, num_neurons):
    model = Sequential()
    model.add(layers.Flatten(input_shape=input_shape))

    for _ in range(num_layers - 1):
        model.add(layers.Dense(num_neurons, activation='relu'))

    model.add(layers.Dense(num_classes, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=

    return model

# Create a KerasClassifier with your model-building function
keras_classifier = KerasClassifier(build_fn=create_model, verbose=0)

# Define parameters for the random search
param_dist = {
    'num_layers': [1,2,3,4,5],  # Number of layers
    'num_neurons': [10,50,100,200] # Number of neurons
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Random search
random_search = RandomizedSearchCV(estimator=keras_classifier, param_distributions=p
random_search.fit(images_train, labels_train_encoded)

# Get the best parameters and score from random search
best_params_random = random_search.best_params_
best_score_random = random_search.best_score_

print(f'Best Hyperparameters: {best_params_random}')
print(f'Best Cross-Validated Accuracy: {best_score_random}')
```

GridSearch

Para este tipo de pesquisa criamos uma grelha com valores dos hiperparâmetros que queremos otimizar. Este tipo de pesquisa é exaustiva portanto vai verificar todas as combinações possiveis de hiperparâmetros, devolvendo o melhor par.

Criamos um modelo para adição de camadas de densas e de otimização e ainda usamos o cross validation para a validação do problema.

```python
# Define the model
def create_model(num_layers, num_neurons):
    print(f'Creating model with {num_layers} layers and {num_neurons} neurons...')
    model = Sequential()
    model.add(layers.Flatten(input_shape=input_shape))

    for _ in range(num_layers - 1):
        model.add(layers.Dense(num_neurons, activation='relu'))

    model.add(layers.Dense(num_classes, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model

# Create a KerasClassifier with your model-building function
keras_classifier = KerasClassifier(build_fn=create_model, verbose=0)

# Define parameters for the grid search
param_grid = {
    'num_layers': list(range(1, 6)),  # Number of layers from 1 to 5
    'num_neurons': list(range(10, 201))  # Number of neurons from 10 to 200
}

# Define callbacks
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True)

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Grid search
grid_search = GridSearchCV(estimator=keras_classifier, param_grid=param_grid, scoring='accuracy', cv=cv, verbose=1)
grid_search.fit(images_train, labels_train_encoded, validation_split = 0.2 ,callbacks=[early_stopping, model_checkpoint])

# Get the best parameters and score from grid search
best_params_grid = grid_search.best_params_
best_score_grid = grid_search.best_score_

print(f'Best Hyperparameters: {best_params_grid}')
print(f'Best Cross-Validated Accuracy: {best_score_grid}')
```

Cap 5 – Análise de Resultados

Começando pela abordagem mais simples, nós realizámos testes só com o VGG e obtivémos resultados bastante positivos relativamente às metas anteriores.
O teste mais básico, com cerca de 200 imagens por classe, teve uma accuracy de 82% sendo que o máximo que tinhamos obtido até agora foi de 58%.

Relativamente à pesquisa com os melhores hiperparâmetros obtidos com o Gravitational Search Algorithm, obtivemos uma accuracy semelhante ao VGG mas com dimensões das imagens bastante inferiores, passando de 200x200 para 128x128, mesmo assim conseguimos obter uma accuracy praticamnete idêntica à do VGG, 81%, concluindo assim que se os nossos computadores o permitissem obteríamos um resultado significativamente superior com a mesma dimensão de imagens. Melhor resultado foi conseguido com 4 camadas e 53 neurónios (hiperparâmetros).

A pesquisa GridSearch foi a mais demorada, durando cerca de 15 horas com as dimensões bastante baixas, mesmo assim conseguimos uma accuracy superior, cerca de 83%. Este resultado foi obtido com 1 camada e 61 neurónios.

Surpreendentemente o melhor resultado optido foi com o RandomSearch, que, como o nome indica, faz uma pesquisa aleatória conforme os parâmetros dados. A melhor accuracy foi de 84%, obtida com 2 camadas e 194 neurónios.

Para finalizar nós realizámos um teste com 1000 imagens de cada classe mas só com o VGG para verificar se havia uma diferença significativa nos resultados. Tal aconteceu, sendo que conseguimos uma accuracy de 89%, um aumento de 7% relativamente aos testes que andámos a realizar com 200 imagens. Por isso podemos afirmar que se tivéssemos poder computacional requerido para tais testes, os 3 algoritmos aumentariam significativamente +- ~7%, atingindo assim os 88% no GSA, 90% no Grid Search e 91% no random Search.

Comparando os 3 processos de otimização, o GSA foi o que obteve piores resultados porém como foi Mencionado em cima, os testes foram realizados com a dimensão de imagens bastante reduzidas comparativamente aos outros processos de otimização.

Relembrar que foram utilizados em todos os testes uma amostra de 200 imagens por classe, número bastante reduzido para o tipo de problema, devido à falta de poder computacional.

Todas as imgens/gráficos serão enviados num ficheiro em anexo.

Cap 6 – Conclusões

Com a conclusão deste trabalho conseguimos adquirir novos conhecimentos sobre redes CNN, problemas de classificação e problemas computacionais.
Nesta meta ocorreram vários erros devido ao poder computacional necessário para a realização de testes e treino dos modelos pedidos.
Ao contrário da meta anterior, conseguimos realizar todas as etapas pedidas no trabalho.
Finalmente nesta meta conseguimos obter valores bastante satisfatórios, a rondar os 90%, algo que ainda não tinhamos conseguido.

Cap 7 – Bibliografia

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

https://www.linkedin.com/pulse/what-randomizedsearchcv-machine-learning-noor-saeed/

https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

https://chat.openai.com

https://www.geeksforgeeks.org/cross-validation-machine-learning/

https://keras.io/api/callbacks/early_stopping/

https://keras.io/api/applications/vgg/#vgg16-function

https://medium.com/geekculture/boost-your-image-classification-model-with-pretrained-vgg-16-ec185f763104

https://www.youtube.com/watch?v=YEkrjuyrtG0