META FINAL

Bruno Oliveira nº2019136478
Micael Eid nº2019112744

Indíce

- VGG16_____pg 3
- Implementação do VGG_____pg 4
- Grid Search_____pg 5
- Implementação do Grid Search_____pg 6
- Random Search_____pg 7
- Implementação do Random Search _____pg 8
- Análise de Resultados_____pg 9 – 13
- Conclusão e discussão de resultados_____pg 14

VGG16

- K. Simonyan e A. Zisserman da Universidade de Oxford
- Utiliza 16 camadas e é considerado uma das melhores arquiteturas de modelos de visão até hoje
- 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.

- Principais Limitações:
  - Treino muito lento (O modelo original foi treinado usando uma Nvidia Titan GPU por 2 - 3 semanas)
  - O tamanho dos 'Weights' ImageNet treinados com VGG16 é de 528mb, causando a necessidade de muito espaço em disco.

Implementação - VGG16

- Importar o VGG16 das bibliotecas tensorflow.keras

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
```

- Inicializar o VGG16 (removendo a 'top layer', Indicar o uso dos conjuntos pré-treinados do ImageNet).
- congelar os 'weights' do modelo, tornando-os não treináveis.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)
base_model.trainable = False
```

- Criação de uma rede densa.

```
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(128, activation='relu')
prediction_layer = layers.Dense(num_classes, activation='softmax')
```

- Criação e compilação de um novo modelo com o VGG16 e a nova rede densa.

```
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    prediction_layer
])
```

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

GRID Search

- O Grid Search é um método sistemático para encontrar a melhor combinação de hiperparâmetros de um modelo de Machine learning.
- Testa todas as combinações possíveis de valores para os hiperparâmetros especificados.
- Cada combinação é então avaliada usando uma métrica de desempenho, como a acurácia ou o erro médio quadrático. O objetivo é encontrar a combinação de hiperparâmetros que resulta no melhor desempenho do modelo.

- Principais Limitações:
  - Uma das principais limitações é o tempo de execução. O Grid Search testa todas as combinações possíveis de valores, o que pode levar muito tempo, especialmente se tivermos muitos hiperparâmetros e um grande número de valores para cada hiperparâmetro.
  - Outra limitação é que o Grid Search não leva em consideração a interação entre os hiperparâmetros. Ele testa todas as combinações possíveis de forma independente, o que pode não capturar as relações complexas entre os hiperparâmetros..

Implementação – Grid Search

- Importar o GridSearch da biblioteca sklearn.
- Importar o StratifiedKFold para o Cross-validation da biblioteca sklearn.

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
```

- Configuração do KerasClassifier para ser utilizado no processo de busca em grelha

```
keras_classifier = KerasClassifier(build_fn=create_model, verbose=0)
```

- Definir os '*Hyperparameters*', neste caso: Número de camadas e neurônios.

```
param_grid = {
    'num_layers': list(range(1, 6)),  #
    'num_neurons': list(range(10, 201))
}
```

- Definição da Cross-Validation, dividindo o conjunto de dados em 5 partes

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

- Inicializa o GridSearch ('*GridSearchCV*') e retira os resultados.

```
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])
```

```
best_params_grid = grid_search.best_params_
best_score_grid = grid_search.best_score_
```

Random Search

- Técnica de otimização de hiperparâmetros que difere da busca em grelha tradicional ao amostrar aleatoriamente conjuntos de hiperparâmetros para treinar e avaliar modelos.
- A principal vantagem da busca aleatória em relação à busca em grelha é que ela pode ser mais eficiente em termos de tempo computacional, especialmente quando o espaço de hiperparâmetros é grande.

- Principais Limitações:
  - Como a seleção de conjuntos de hiperparâmetros é aleatória, não há garantia de que todos os pontos do espaço de hiperparâmetros serão explorados.
  - A eficácia da busca aleatória pode depender da natureza do problema e do conjunto de dados. Em alguns casos, a busca em grelha pode superar a busca aleatória.
  - Como as escolhas são feitas aleatoriamente, não há uma sequência lógica ou coerente de experimentos, o que pode dificultar a interpretação dos resultados.

Implementação – Random Search

- Importar o RandomSearch da biblioteca sklearn.
- Importar o StratifiedKFold para o Cross-validation da biblioteca sklearn.

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold
```

- Configuração do KerasClassifier para ser utilizado no random search

```
keras_classifier = KerasClassifier(build_fn=create_model, verbose=0)
```

- Definir os '*Hyperparameters*', neste caso: Número de camadas e neurônios.

```
param_dist = {
    'num_layers': [1,2,3,4,5],  # 
    'num_neurons': [10,50,100,200]
}
```

- Definição da Cross-Validation, dividindo o conjunto de dados em 5 partes

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

- Inicializa o RandomSearch ('*RandomizedSearchCV*') e retira os resultados.

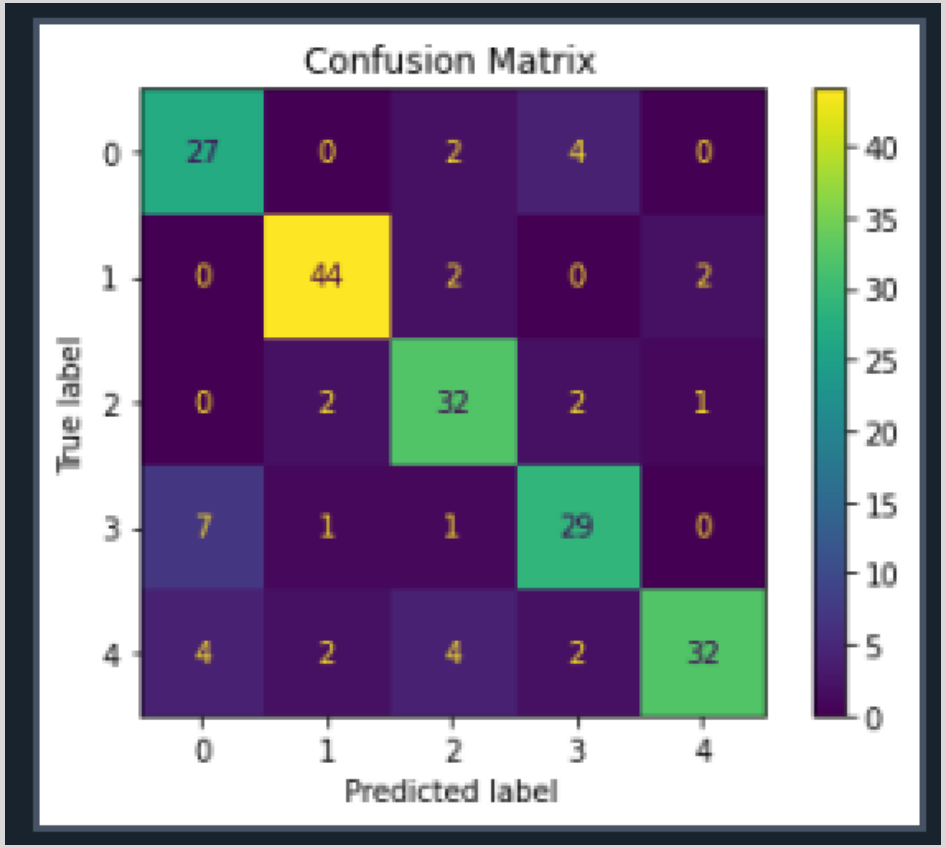
```
random_search = RandomizedSearchCV(estimator=keras_classifier, param_distributions=param_dist, scoring='accuracy', cv=cv, n_iter=25)
random_search.fit(images_train, labels_train_encoded)
```

```
best_params_random = random_search.best_params_
best_score_random = random_search.best_score_
```

Análise de Resultados

Apenas VGG e Rede densa simples.

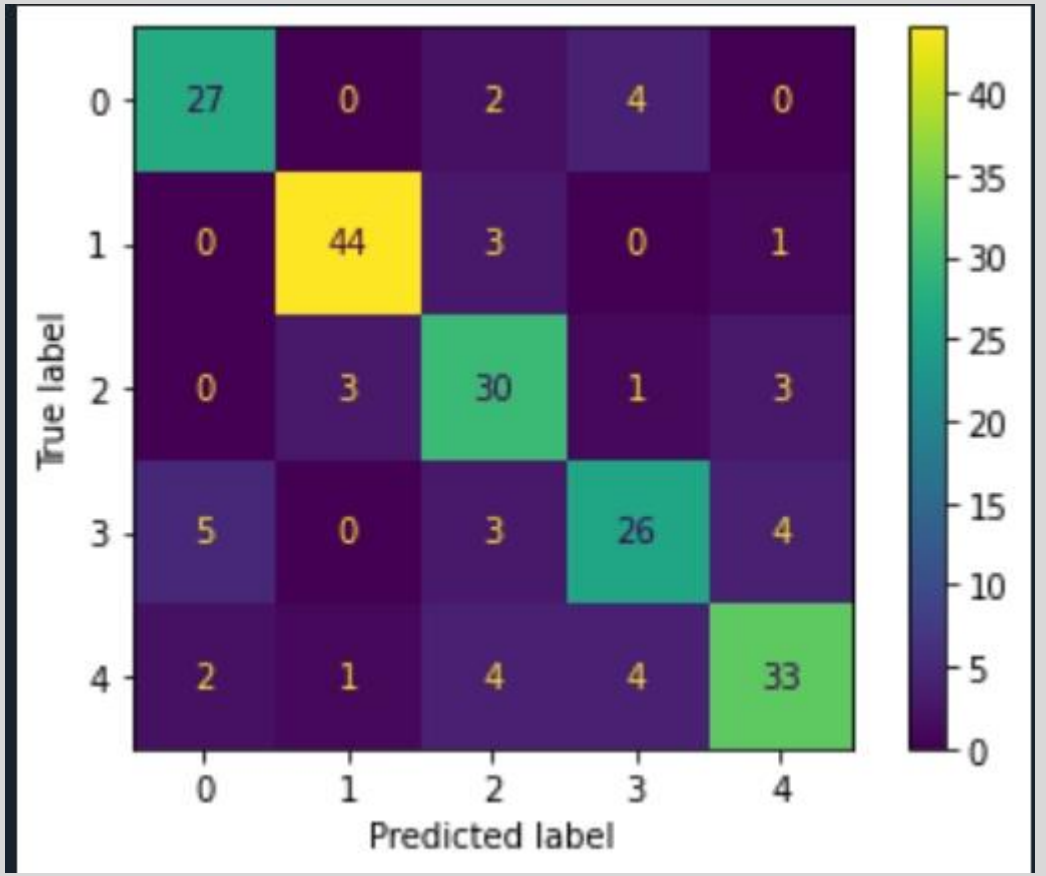
- Nós realizámos testes só com o VGG e obtivémos resultados bastante positivos relativamente às metas anteriores. O teste mais básico, com cerca de 200 imagens por classe, teve uma accuracy de 82% sendo que o máximo que tinhamos obtido até agora foi de 58%.



Análise de Resultados

VGG e Otimização com GSA.

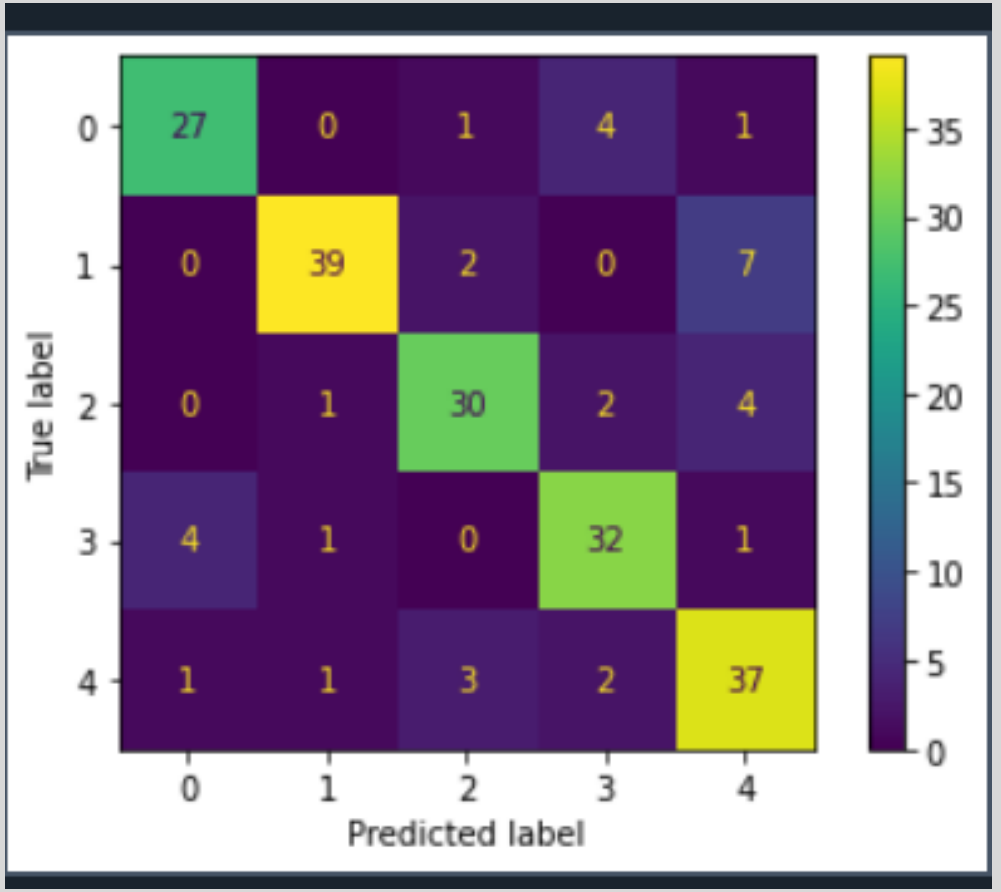
- Accuracy semelhante ao VGG "simples" mas utilizando imagens com dimensões inferiores, passando de 200x200 para 128x128.
- Conseguimos obter uma accuracy praticamnete idêntica à do VGG, 81%.
- Melhor resultado foi conseguido com 4 camadas e 53 neurónios na rede densa (hiperparâmetros).



Análise de Resultados

VGG e Otimização com Grid Search.

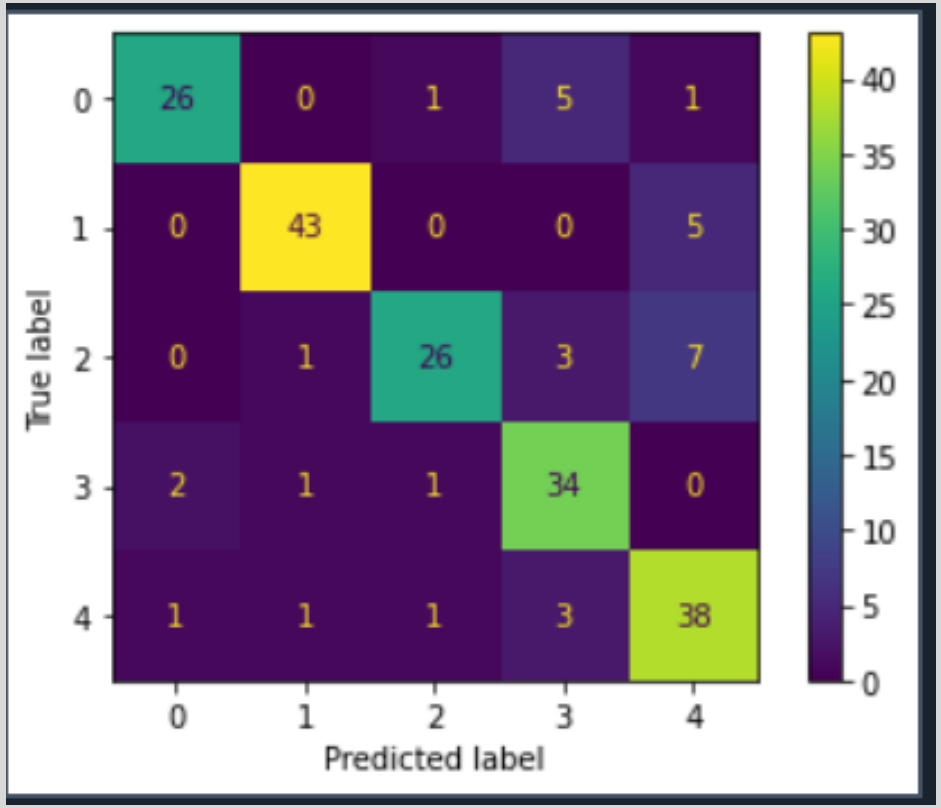
- A pesquisa GridSearch foi a mais demorada, durando cerca de 15 horas com as dimensões bastante baixas, mesmo assim conseguimos uma accuracy superior, cerca de 83%. Este resultado foi obtido com 1 camada e 61 neurónios.



Análise de Resultados

VGG e Otimização com Random Search.

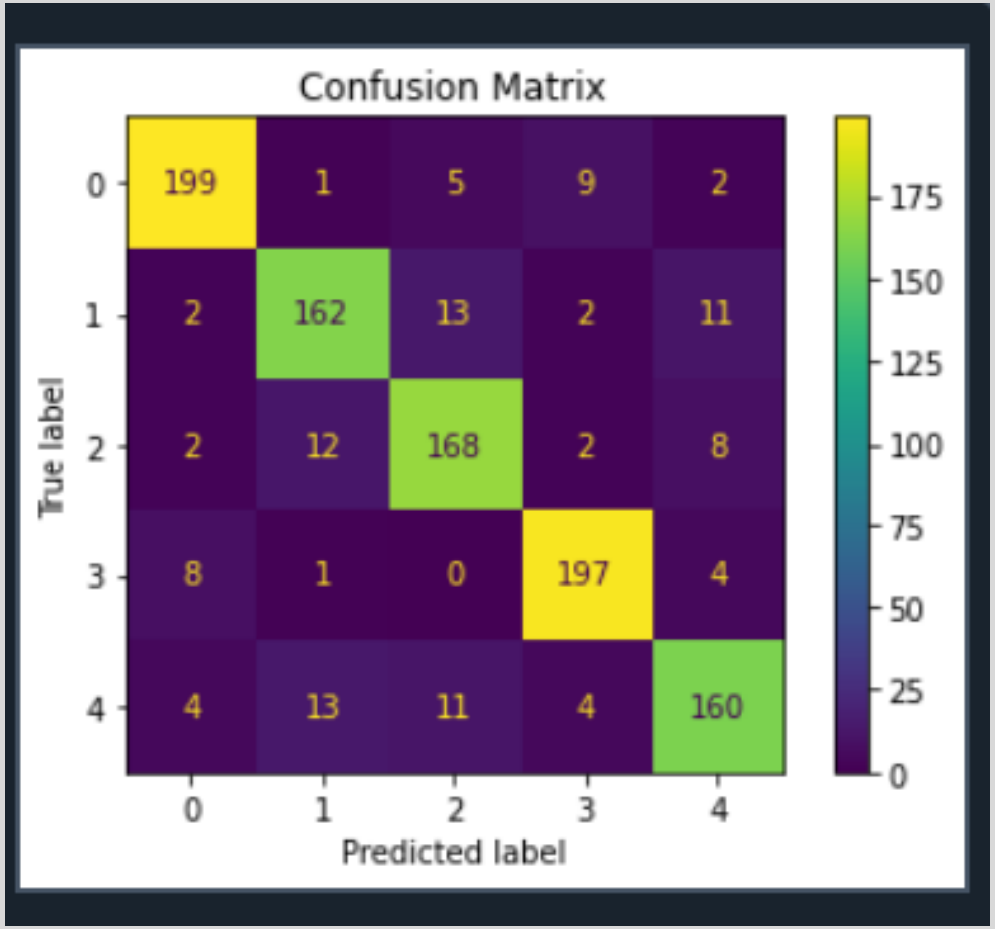
- Surpreendentemente o melhor resultado optido foi com o RandomSearch, que, como o nome indica, faz uma pesquisa aleatória conforme os parâmetros dados. A melhor accuracy foi de 84%, obtida com 2 camadas e 194 neurónios.



Análise de Resultados

VGG com rede densa simples (5000 imagens).

- Para finalizar nós realizámos um teste com 1000 imagens, 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 apenas.



Conclusão e discussão de resultados

- 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.
- 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.