Introduction to Machine Learning: OMPF Image Similarity Search Model

Ciro Beneduce - Alessia Meloni - Patrick Montanari - Roberta Peracchio

Index:

- ❑ Dataset Creation - Patrick & Ciro
- ❑ Model Evaluations - Roberta & Patrick
- ❑ Model Development - Alessia + Ciro + Roberta
- ❑ Model Training - Everyone
- ❑ Results Overview - Everyone

Link: https://github.com/alessia96/OMPF/

Introduction

Task: Design a image search engine which, given a query picture, returns the most N similar pictures from a gallery using an image similarity algorithm, based on metrics distance.

The dataset

Starting from the original dataset, we expanded the labels using collections found from Kaggle and KAIST's Data Mining Lab's Database for a total of 115 classes of animals, with a total of 77'000 pictures.

KAIST

kaggle

The model

What we need:

- classification models
- supervised learning (because we already have the labels)
- efficient in predictions
- efficient over time

Discarded Models - I

K-NN

pros

- easy to use and implement

cons

- Might be slow, unreliable.
- How do we find the optimal K?
  - iteratively, stop when optimal is reached (so when we found the maximum accuracy and increasing k decreases accuracy)
- not as good for new data

Support Vector Machines (SVM)

pros

- can be very efficient in guesses
- easily adapt to non-linear distributions

cons

- takes a lot of time and space

Discarded Models - II

Logistic regression, QDA, LDA

pros

- very easy to implement
- gives clear separation, dimensionality reduction

cons

- variance-overfitting
- lower model stability

Random forest and decision trees

pros

- High validation performance, stability due to CV.

cons

- very slow and our dataset might be too big for these models

Choosing the model

CNN (Convolutional Neural Network) reduces the number of parameters while maintaining the same quality of the model

→ perfect for image recognition!

CNN from scratch: limitations

However images have a high number of features, which means:

- long training time
- long wait to determine the highest accuracy it can reach

Solution → use resnet50

CNN: Our Model

```
model = tf.keras.models.Sequential([data_augmentation, keras.layers.Normalization(),
     tf.keras.layers.Conv2D(32,(3,3), padding="valid", activation = "relu" , input_shape = (img_width,
img_height)) ,
     tf.keras.layers.Conv2D(32,(3,3), padding="same",  activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),

     tf.keras.layers.Conv2D(64,(3,3), padding="valid", strides=2, activation = "relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),
     tf.keras.layers.Conv2D(64,(3,3), padding="same", activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),

     tf.keras.layers.Conv2D(128,(3,3), padding="valid", strides=2, activation = "relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),
     tf.keras.layers.Conv2D(128,(3,3), padding="same", activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),
     tf.keras.layers.Conv2D(128,(3,3), padding="same", activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),

     tf.keras.layers.Conv2D(256,(3,3), padding="valid", strides=2, activation = "relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),
     tf.keras.layers.Conv2D(256,(3,3), padding="same", activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),
     tf.keras.layers.Conv2D(256,(3,3), padding="same", activation="relu",
kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)) ,
     tf.keras.layers.BatchNormalization(),

     tf.keras.layers.GlobalAveragePooling2D(),
     tf.keras.layers.Flatten()
     tf.keras.layers.Dense(num_classes ,activation = "softmax") ])
```

CNN: final model

Despite working hard on the previous model, the one which performed the best was a transfer learning-based model.

- we loaded the model without the input layer and then we froze all the layers. In this way, during the training phase the original model would not be overwritten.
- we trained the model until convergence between train and validation sets. After that, we performed fine-tuning unfreezing the resnet layers and trained the whole model once more.

Results

Our final model had the following performance:

Score	Top 1	Top 5	Top 10
523.077	0.615	0.708	0.738

All things considered, this model could be expanded and trained further to improve, but it represents a solid starting point for image similarity metrics.

Ciro Beneduce
Alessia Meloni
Patrick Montanari
Roberta Peracchio

Thank you for your attention!