/Intro to ML: COMPETITION

A work by
Indecisive Trees

Antonio_Padalino / Martina_Arturi / Anna_Culpo / Davide_Longo

<Literature review>

Content-based Image Retrieval System

- The **performance** of the system depends on:
    - Features extracted
    - Similarity measurements

- Main objective → reducing the so-called **semantic-gap** existing between:
    - Low-level image features
    - High-level human perception

/Table of contents

/01 /Data preparation - Martina

* data loading
* data augmentation

/02 /The Neural Nets - Antonio

* ResNet152
* built-from-scratch

/03 /Feature extraction - Davide

* methods examined
* our feature extraction

/04 /The search engine - Anna

* making queries
* evaluating the model

<Data preparation>

> Preliminary operations on our dataset <

/01

<Data preparation>

How we dealt with our dataset before feeding it to the Neural Network

- Dataset() function

- **How to extend the original dataset?**
→ Implemented a script that automatically added more images to our classes

- **How to augment our dataset?**
→ Implemented the following function [5 augmentation per image, as the more we augment our dataset the more accurate it'll be]

- **Prepare our training data by splitting the** training folder into training and validation

```
# The transformer we used

data_aug = ImageDataGenerator(

        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'

)
```

<The Neural Nets>

> The actual attempt at building a CNN <

/02

/Overview on Neural Networks used

We considered 2 different approaches based on Convolutional NNs.

/Transfer Learning with a pre-trained ResNet

- using a ResNet backbone and adding fully connected layers
- the overall best solution within our means and aim

/A built-from-scratch convolutional model

- our attempt at building a CNN using Keras
- partially inspired to other popular architectures

/ResNet152

- we used ResNet152 as our backbone

```
# Initialize the ResNet152 backbone without classifier

resnet = ResNet152(weights='imagenet',
                   include_top=False,
                   input_shape=(224, 224, 3))
```

/ResNet152

- we used ResNet152 as our backbone
- freezed 4 of its convolutional blocks

```
# Freeze the first 4 convolutional blocks

for layer in resnet.layers:
    if 'conv5' in layer.name:
        break
    layer.trainable = False
```

/ResNet152

- we used ResNet152 as our backbone
- freezed 4 of its convolutional blocks
- added a classifier on top of it, with:
  - **a Flatten layer**
    turning the pooled feature map
    into a vector to feed to next layers
  - **a Dense layer**
    with 10 nodes (# classes) and Softmax
    as nonlinearity activation function
  - **a Dropout layer**
    randomly dropping out some
    nodes to reduce overfitting

```
# Create our classifier on top of it

inputs = tf.keras.Input(shape=(224, 224, 3))
x = resnet(inputs, training=False)
x = layers.Flatten(name='flatten')(x)
x = layers.Dense(10, activation='softmax',
kernel_regularizer='l1_l2')(x)
outputs = layers.Dropout(0.1, seed=999)(x)

model1 = tf.keras.Model(inputs, outputs)

model1.summary()
```

/Our built-from-scratch model

It is a quite simple one:
- it features 3 convolutional blocks
- increasing (yet small) numbers of kernels

We handled overfitting through:
- L1 and L2 regularizers
- dropout layers throughout the model

After training, we cut our model:
- so that, fed with an image, it will return its features after the conv.

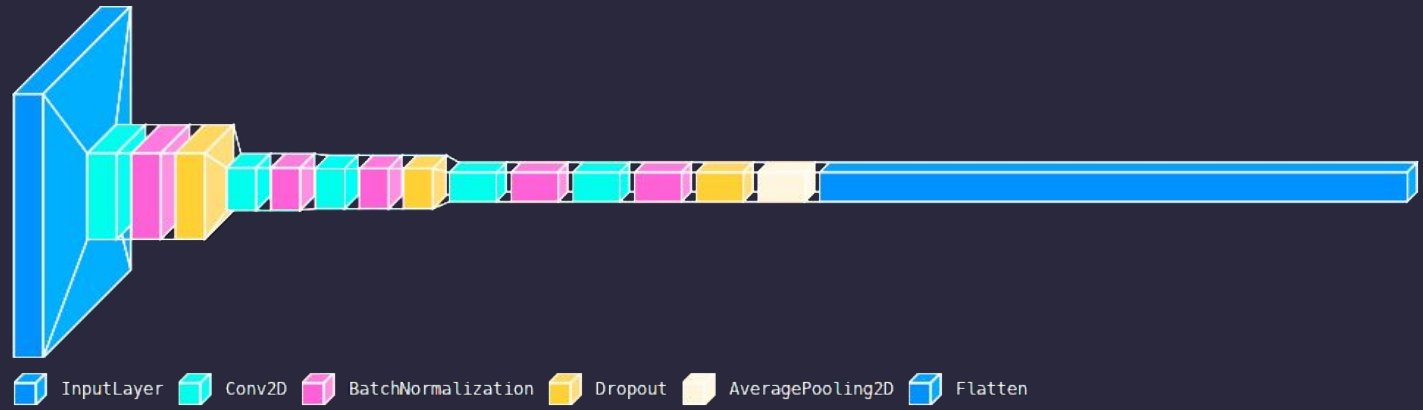
```
features = model2.get_layer("flatten_layer").output

model2 = tf.keras.Model(model2.inputs, features)
```

/Our built-from-scratch model

```
model2 = Sequential([

    layers.InputLayer(input_shape=(224,224,3)),

    layers.Conv2D(8,(7,7),strides=(3,3),activation='relu',kernel_regularizer='l1_l2'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),

    layers.Conv2D(16,(3,3),strides=(2,2),activation='relu',kernel_regularizer='l1_l2'),
    layers.BatchNormalization(),
    layers.Conv2D(16,(3,3),strides=(1,1),activation='relu',kernel_regularizer='l1_l2'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),

    layers.Conv2D(32,(3,3),strides=(2,2),activation='relu',kernel_regularizer='l1_l2'),
    layers.BatchNormalization(),
    layers.Conv2D(32,(3,3),strides=(1,1),activation='relu',kernel_regularizer='l1_l2'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),

    layers.AvgPool2D(2,2),
    layers.Flatten(name='flatten_layer'),

    layers.Dense(10, activation='softmax',kernel_regularizer='l1_l2'),
    layers.Dropout(0.2)

])
```

/Our built-from-scratch model

Here’s a quick look at our model.



<Training phase / ResNet152>

We tried running the model on 10, 50 and 100 epochs,
but later decided to introduce:

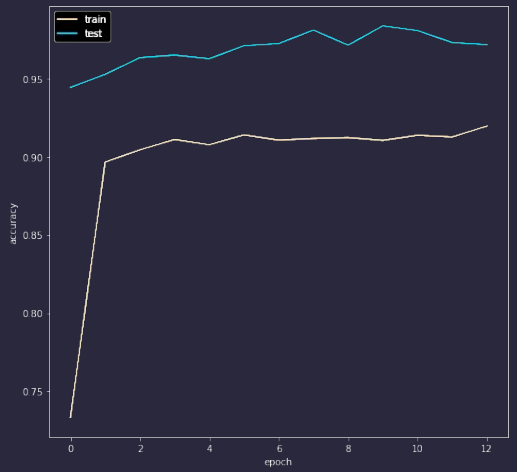
- an EarlyStopping callback to monitor ‘val_loss’:

  if there hasn’t been a change of at least 0.1 for 3 epochs,
  the model training is stopped

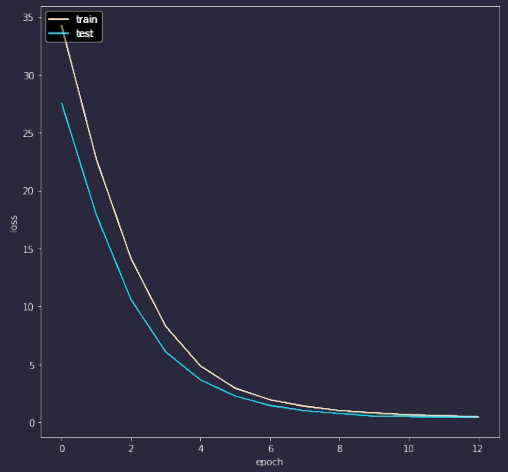
```
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=3)

history = model1.fit(train_ds,validation_data=val_ds, epochs=50, callbacks=[early_stop])
```

<Training phase / ResNet152>

Accuracy
(Sparse Categorical Accuracy)



Loss
(Sparse Categorical CrossEntropy)



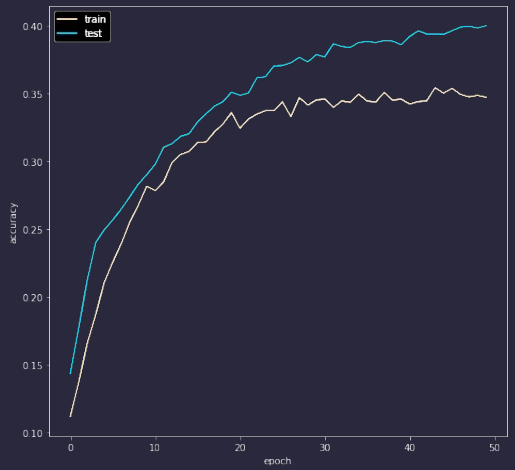
<Training phase / built-from-scratch>

- We tried with 200 epochs: too slow and our accuracy didn’t improve
- We introduced a ReduceLROnPlateau callback to adjust the lr
- We used once again the EarlyStopping callback

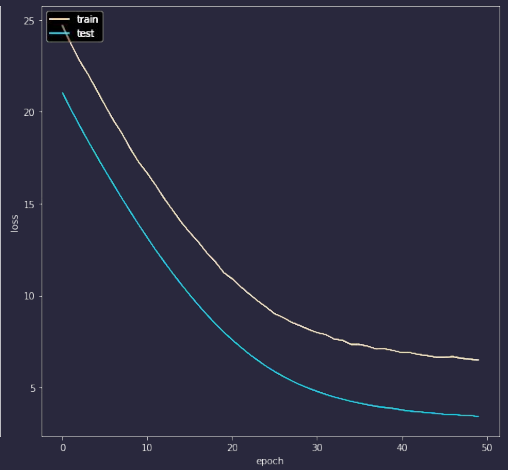
```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=5)

model2.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
               loss=tf.keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])


history = model2.fit(train_ds, validation_data=val_ds, epochs=50, callbacks=[reduce_lr, early_stop])
```

<Training phase / built-from-scratch>

Accuracy
(Sparse Categorical Accuracy)



Loss
(Sparse Categorical CrossEntropy)



<Feature extraction>

What’s feature extraction?

/03

/Features

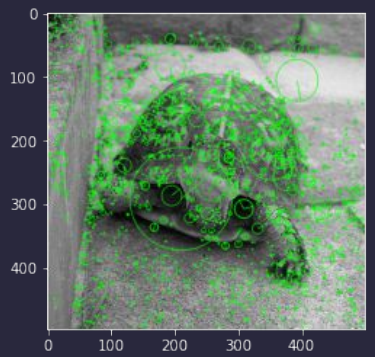
There are different types of features that can be extracted from an image:

1. **Global features**
    - **Color** → linear (RGB) vs non-linear
    - **Texture** → image structure, randomness, granulation, linearity, roughness, homogeneity
    - **Shape** → boundary-based and region-based
2. **Local features**
    - Describe **local information** using **key points** of some image parts
    - Invariant to **image scale** and **rotation**
    - Higher **accuracy**

/Local Features Extractors

1. Scale invariant feature transformation (**SIFT**)

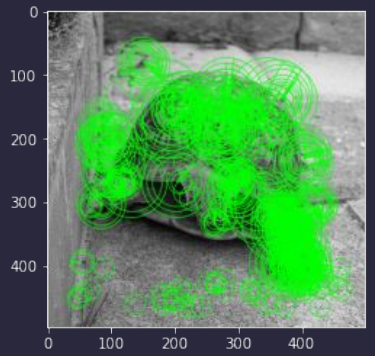
```python
img_rgb = cv2.imread('turtle.JPEG')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp, descs = sift.detectAndCompute(img_gray, None)

img_sift=cv2.drawKeypoints(img_gray, kp, img_rgb, (0,255,0),

flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.imshow(img_sift)
plt.show()
```



/Local Features Extractors

1. Scale invariant feature transformation (**SIFT**)
2. Oriented fast and rotated brief (**ORB**)

```python
img_rgb = cv2.imread("turtle.JPEG")
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

orb = cv2.ORB_create(nfeatures=500)
kp, descs = orb.detectAndCompute(img_gray, None)

img_orb = cv2.drawKeypoints(img_gray, kp, img_rgb, (0,255,0),

flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.imshow(img_orb)
plt.show()
```

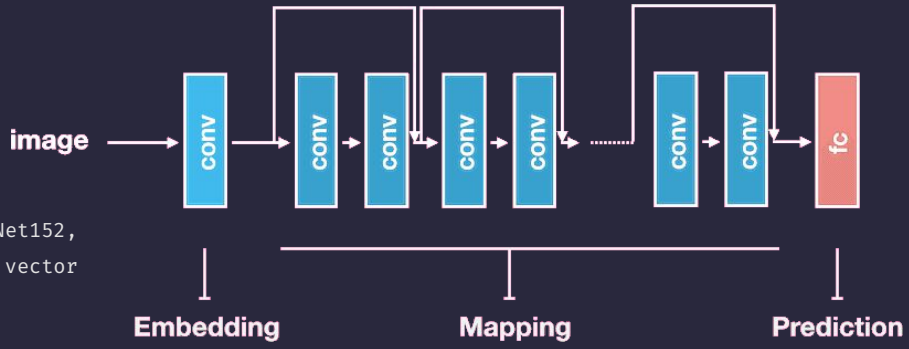


/Local Features Extractors

1. Scale invariant feature transformation (**SIFT**)
2. Oriented fast and rotated brief (**ORB**)
3. Speed up robust features (**SURF**)

/Local Features Extractors

1. Scale invariant feature transformation (**SIFT**)
2. Oriented fast and rotated brief (**ORB**)
3. Speed up robust features (**SURF**)
4. Convolutional Neural Networks (**CNN**)



the architecture of ResNet152,
ending with a feature vector

/Let’s extract features

```python
def extract_features(img_path, model):

    # prepare image before model.predict()
    input_shape = (224, 224, 3)
    img = image.load_img(img_path, target_size=(input_shape[0], input_shape[1]))
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img = preprocess_input(expanded_img_array)

    # now we can .predict()
    features = model.predict(preprocessed_img)
    flattened_features = features.flatten()
    normalized_features = flattened_features / norm(flattened_features)
    return normalized_features
```

<The search engine>

How our actual search engine works

/04

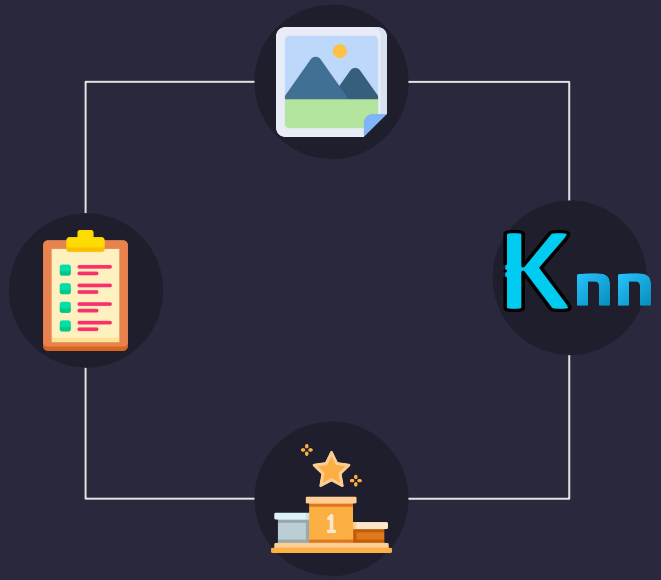
/Query-Gallery Search

The make_query() function takes the index i of a query image as input and returns the accuracies of the:

- Top1
- Top3
- Top5
- Top10

gallery images with similar features

To do that, it applies the k-Nearest-Neighbours algorithm to the list of gallery features gallery_feats.



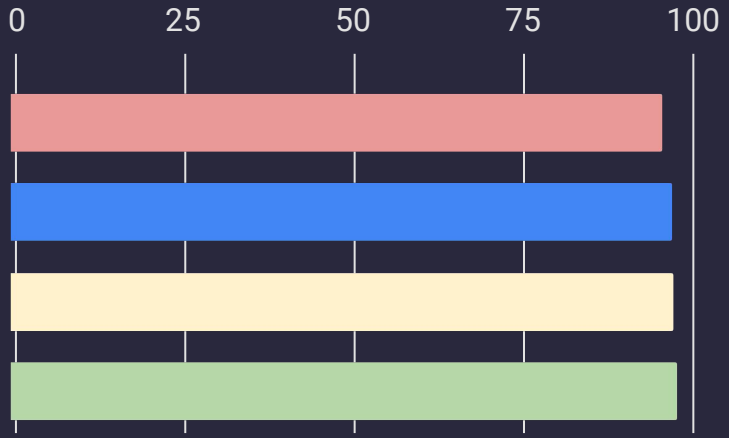
<Top ten images>

The make_query() function shows the query image and the ten most similar gallery images.

It prints also a list of names of the ten images with a **circle o** (if gallery label is equal to query label) or a **cross ×** (if the labels are different)

```
Queried img:
n02077923(sea_lion)

Retrieved imgs:
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o
n02077923(sea_lion) o

RESULTS
Top-1 accuracy is: 1.0
Top-3 accuracy is: 1.0
Top-5 accuracy is: 1.0
```

ResNet152 Evaluation

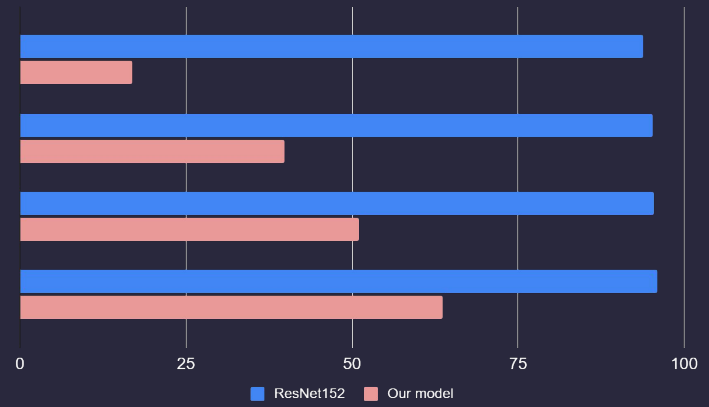



```
top-1 accuracy:  0.9383
top-3 accuracy:  0.9533
top-5 accuracy:  0.9550
top-10 accuracy: 0.9600
```

The evaluation() function applies the make_query() function to all the images in the query folder.

It sums the “True” values in the lists all_t1, all_t3, all_t5, all_t10 and it divides the sums by the number of query images.

ResNet152 VS our model performance



We extract query and gallery features using our model.

Its performance is significantly lower than the one of ResNet152.

```
top-1 accuracy:  0.1700
top-3 accuracy:  0.3983
top-5 accuracy:  0.5100
Top-10 accuracy: 0.6367
```

<Conclusion>

- Chose the Transfer Learning Approach and use a Residual Neural Network for the first model and Convolutional Neural Network for the second one.

- For both the models, we implemented Dataset class → augmented the dataset → fit our models → extracted features → `make_query()` [returns top-10 similar images to the query with the respective accuracy] & `evaluation()` [accuracy of our model overall]

- The first model is better as it has better accuracy

/Some words on evaluation

Why did our models score so differently?

- Our built-from-scratch model isn't trained on all ImageNet's images
  → our dataset isn't as rich
- It doesn't have 152 layers like ResNet
- The architecture is simple
- Our computers' computational power is limited

<The Competition>

How well did our models perform?

/to conclude

/Our results on competition

From a quick examination of the ground truth labels we found out that:

- images belongs to 67 classes, each with a greatly variable number of samples

- the largest amount of images (5335) are labeled as confusion_classes

- also, query and gallery do not contain images from exactly the same classes:

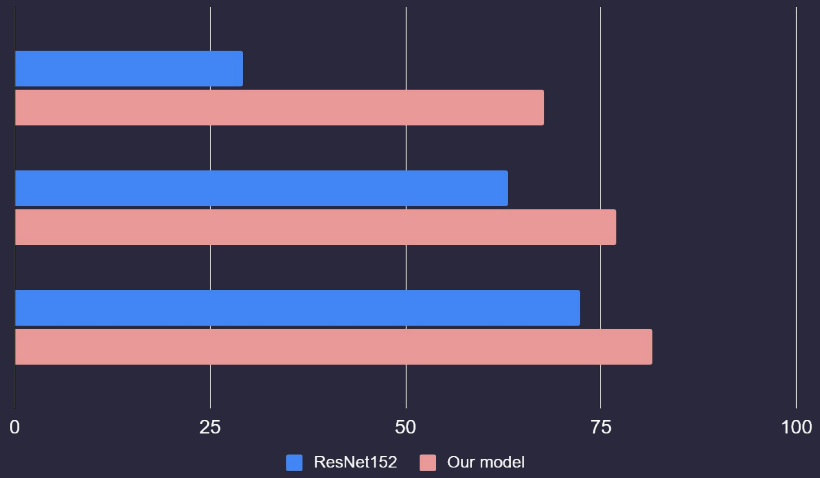
    - query contained 1 image per class, excluding confusion_classes
    - gallery contained all the remaining images, including confusion_classes
    - some classes (such as bobcat or panda) only appears in the gallery set

Looking at our accuracies, we can say that our model was actually pretty good at ignoring the (many) confounding images from the gallery.

/Our results on competition

We know that the competition dataset was also made up of sketches, and we are aware that they are hard to classify because they lack complex visual information in contrast to traditional images that contain an overabundance of visual information, one of the possible reason for the better performance of our built-from-scratch model could be that being less deep and complex than ResNet152 helped it to extract more simple features.

	Res-Net	Our Model
top-1 accuracy:	0.2920	0.6769
top-5 accuracy:	0.6308	0.7692
top-10 accuracy:	0.7231	0.8154



/References

ALZU'BI, Ahmad; AMIRA, Abbes; RAMZAN, Naeem.
Semantic content-based image retrieval: A comprehensive study.
Journal of Visual Communication and Image Representation, 2015, 32: 20-54.

CHEN, Wei, et al.
Deep image retrieval: A survey. *ArXiv*, 2021.

GUDIVADA, Venkat N.; RAGHAVAN, Vijay V.
Content based image retrieval systems. *Computer*, 1995, 28.9: 18-22.

HE, Kaiming, et al.
Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016. p. 770-778.

JAIN, Surbhi; DHAR, Joydip.
Image based search engine using deep learning. In: *2017 Tenth International Conference on Contemporary Computing (IC3)*. IEEE, 2017. p. 1-7.

MAHANTESH, K.; RAO, Shubha.
Content based image retrieval-inspired by computer vision & deep learning techniques. In: *2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*. IEEE, 2019. p. 371-377.

/THANKS!

/any questions?

The project work is fully available at synchroazel/img-search