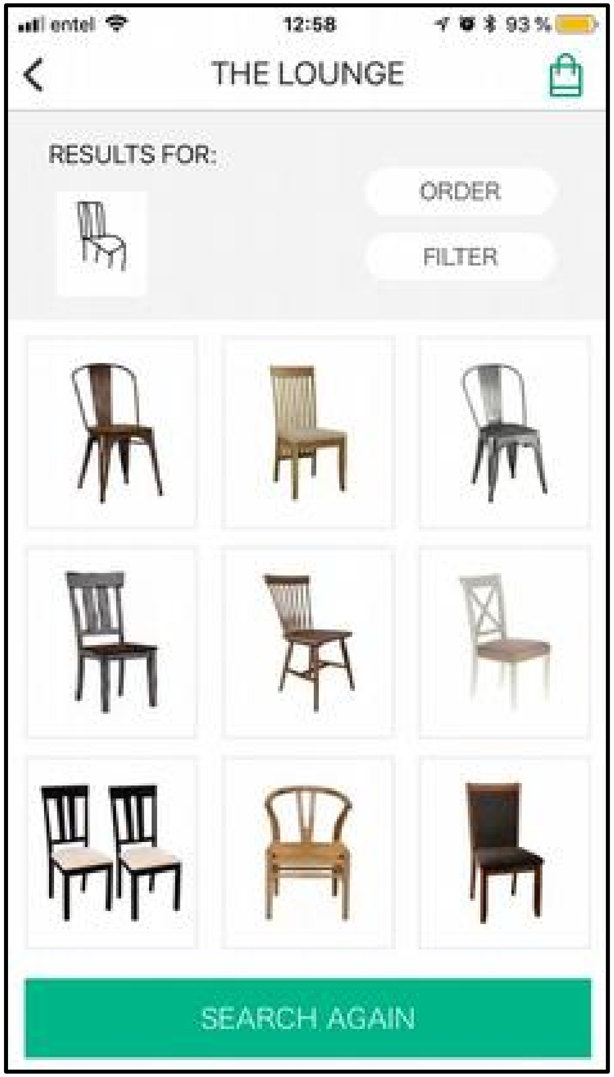
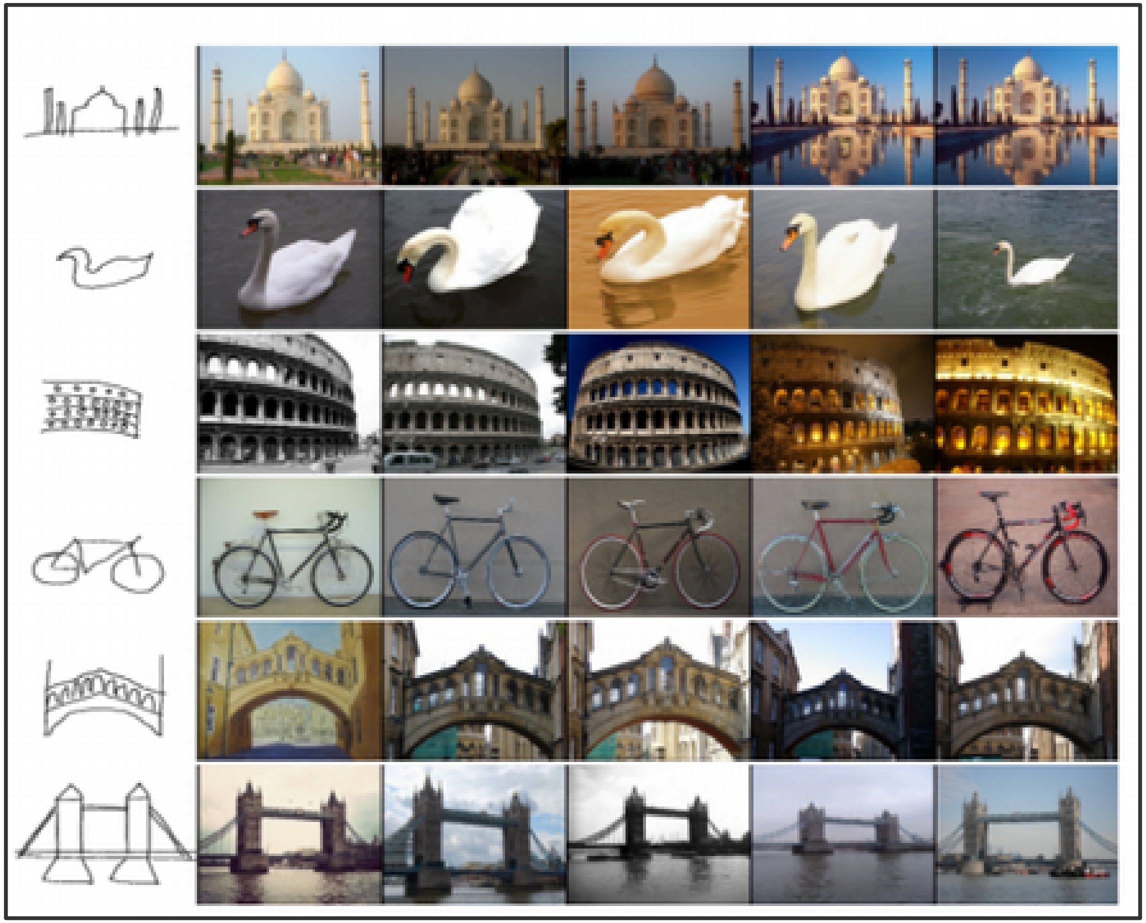
REDES NEURONALES CONVOLUCIONALES
Sketch Based Image Retrieval

José M. Saavedra R.

Sketch Based Image Retrieval



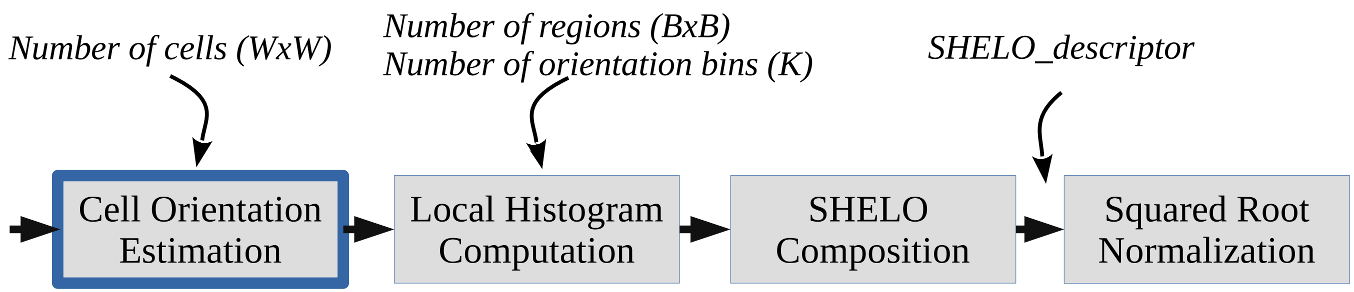


CC6204 José M. Saavedra

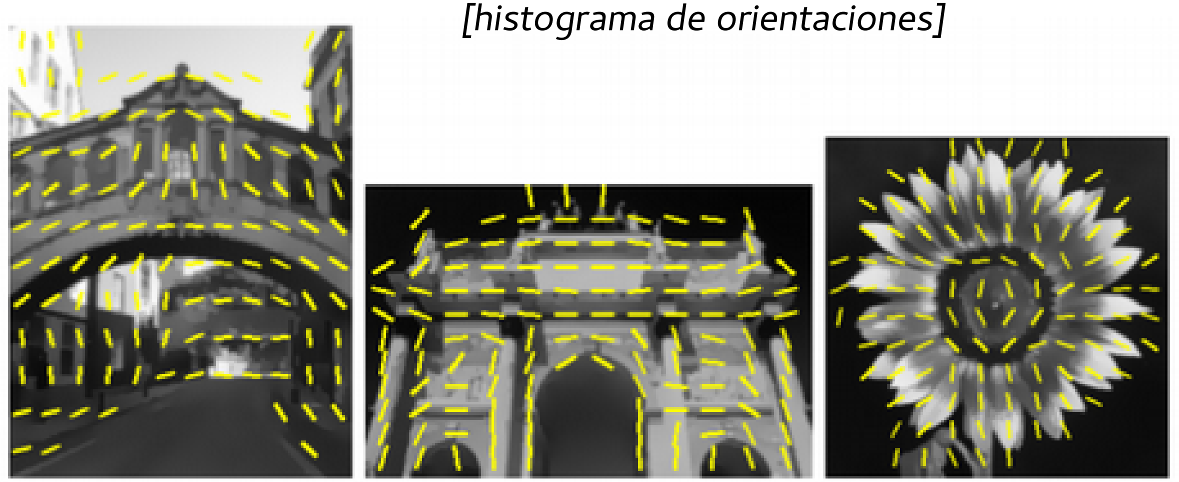
SBIR: Métodos Tradicionales

-Características de bajo nivel-

(a nivel de pixel)



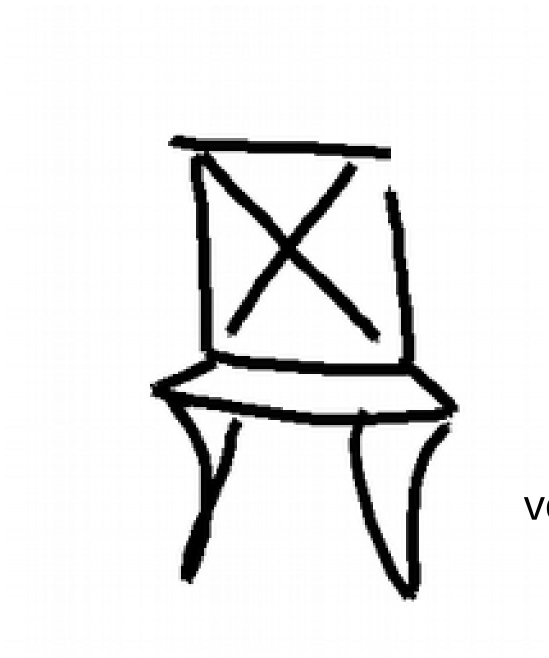
[histograma de orientaciones]



CC6204 José M. Saavedra

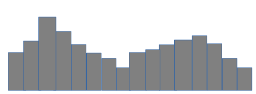
Jose M. Saavedra. RST-SHELO: Sketch-based image retrieval using sketch tokens and square root normalization Multimedia Tools and Applications, Springer, 2015

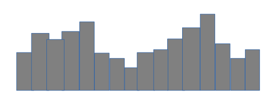
SBIR: Métodos Tradicionales



$$x \in \mathbb{R}^d$$

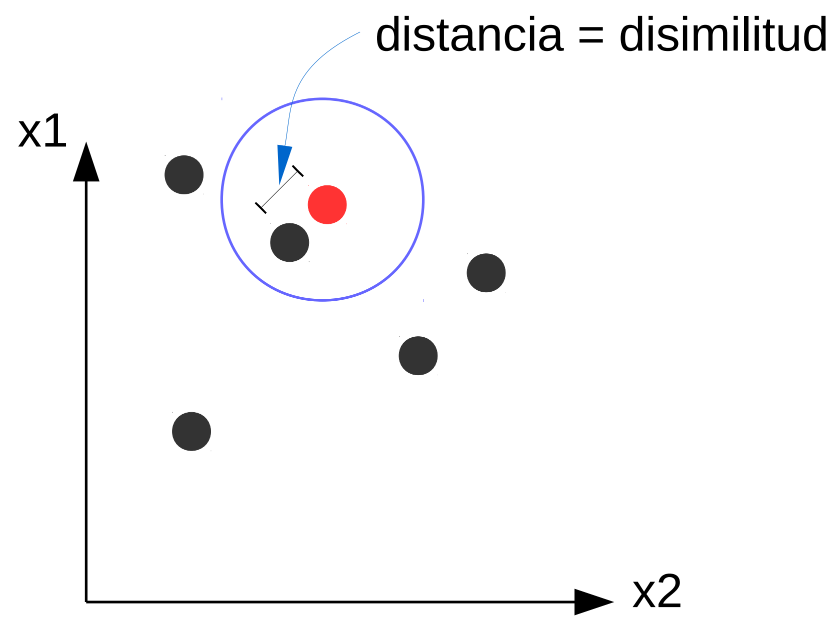
vector de características
[feature vector]

x^S 

x^I 

Comparación entre vectores de características

SBIR: Métodos Tradicionales



Para comparar vectores, usamos una función de distancia. Mientras menor sea la distancia, más similares serán los objetos subyacentes.

Funciones de Distancia

- Métricas para espacios vectoriales:
  - Distancias de Minkowski:

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}, p \geq 1$$

Manhattan (p=1)

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$$

Euclidiana (p=2)

$$L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{1/2}$$

Máximo (p=inf)

$$L_\infty(\mathbf{x}, \mathbf{y}) = max_{i=1}^{d} |x_i - y_i|$$

Funciones de Distancia

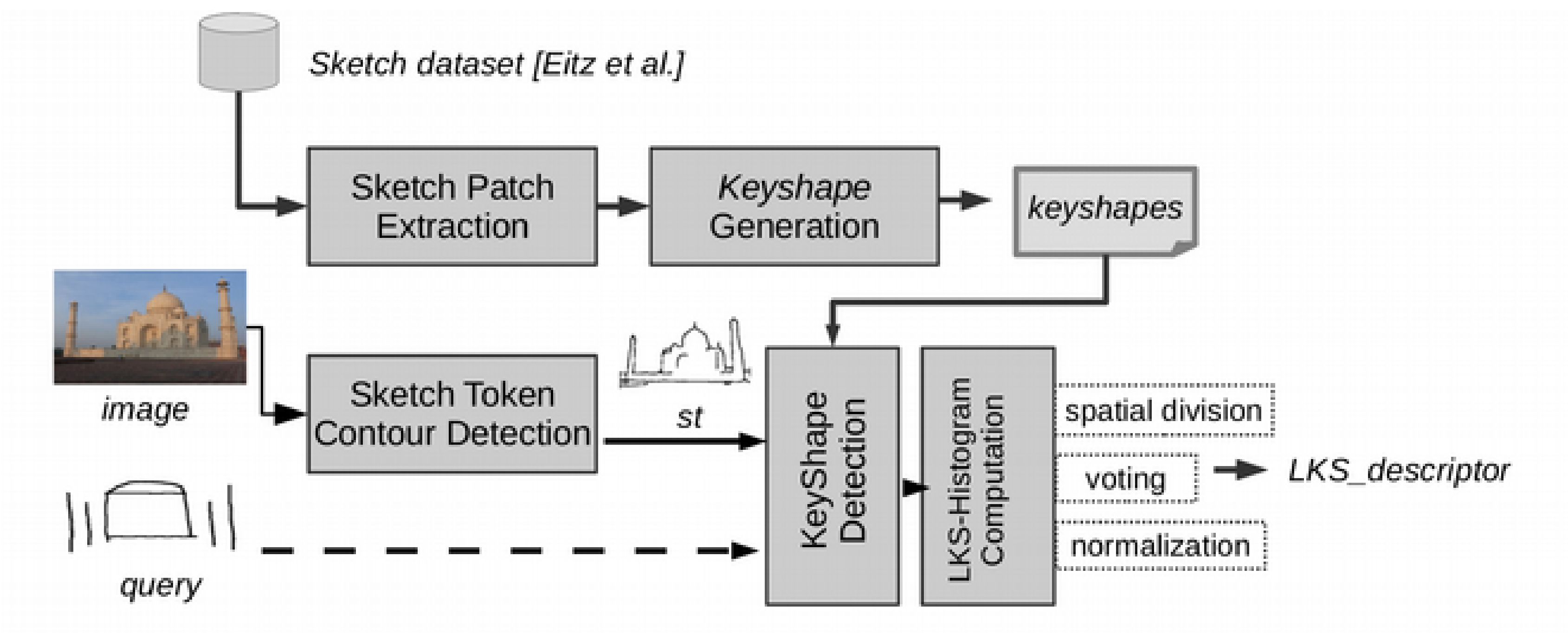
- Métricas para espacios vectoriales:
  - Bursting Effect: Grandes diferencias en muy pocas dimensiones producen un gran diferencia entre dos vectores.
  - **Solución Práctica:** Square-Root Normalization

$$sr\_norm(\mathrm{x}) = unit(sign(\mathrm{x}) \cdot \sqrt{|\mathrm{x}|})$$

SBIR: Métodos Tradicionales

-Características de más alto nivel-

KEYSHAPES

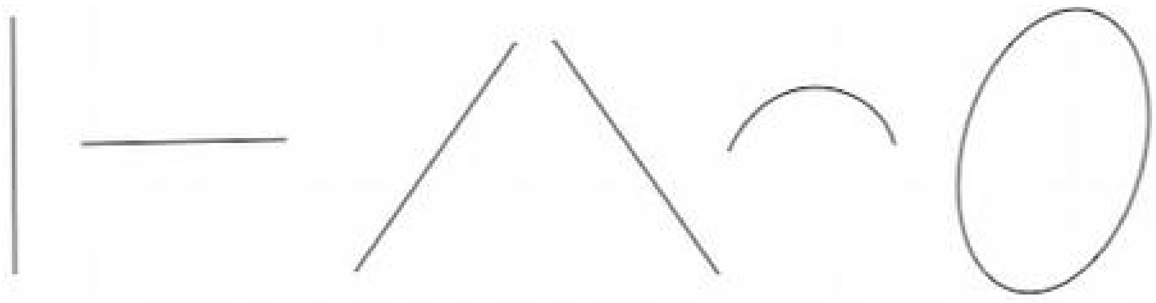


Jose M. Saavedra, Juan Manuel Barrios. Sketch based Image Retrieval using Learned KeyShapes (LKS) . In the proceedings of the 26th British Machine Vision Conference (BMVC), Swansea, UK, 2015

SBIR: Métodos Tradicionales

-Características de más alto nivel-

KEYSHAPES



Un conjunto de seis KEYSHAPES básicos

Jose M. Saavedra, Benjamin Bustos. Sketch-Based Image Retrieval using Keyshapes. Multimedia Tools and Applications, Springer (2013)

SBIR: Métodos Tradicionales

LKS

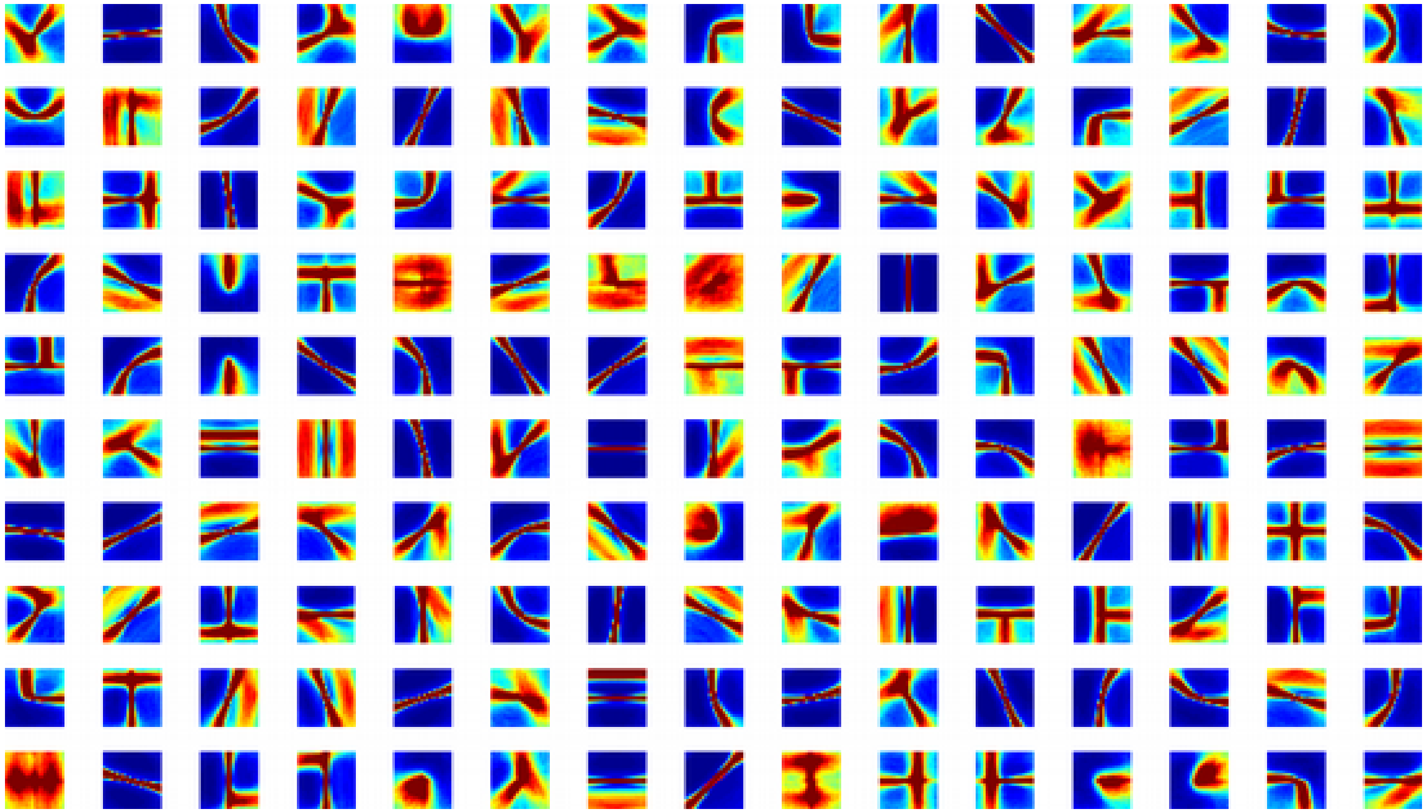
¿Cuáles son los KEYSHAPES?

~ 1 millón de *patches*

SBIR: Métodos Tradicionales

LKS

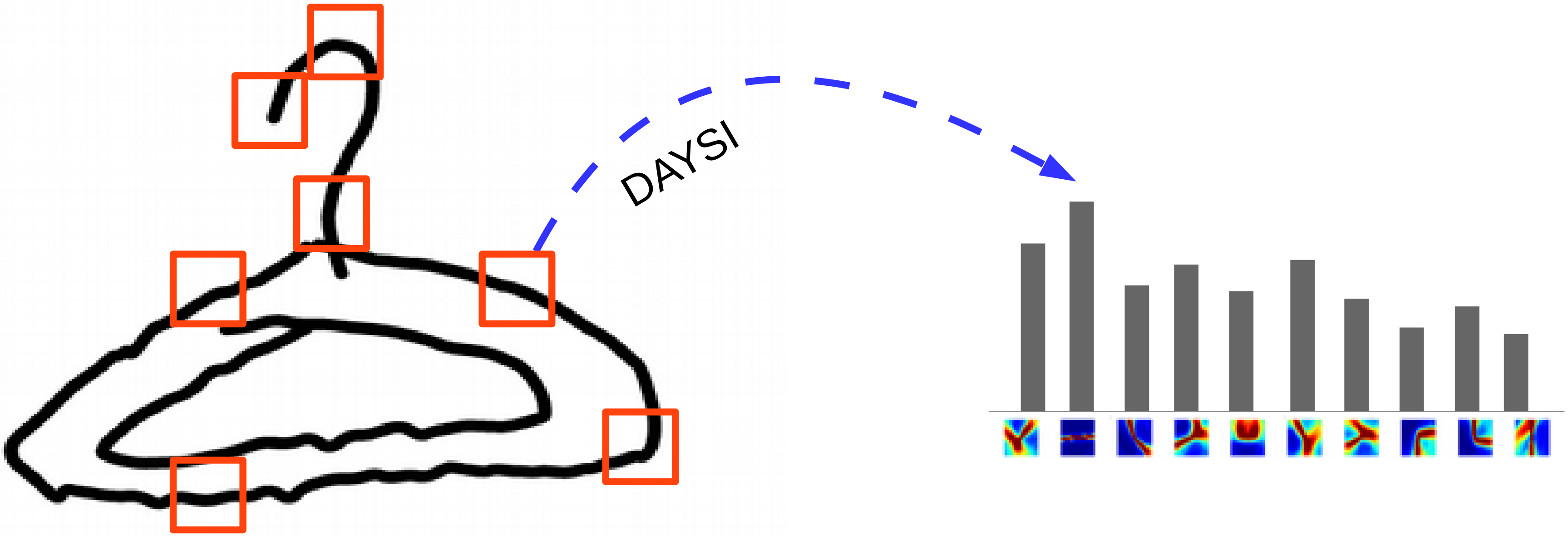
Clustering sobre los patches (e.g. K = 150)



SBIR: Métodos Tradicionales

LKS

Representación



SBIR: Métodos Tradicionales

Evaluación

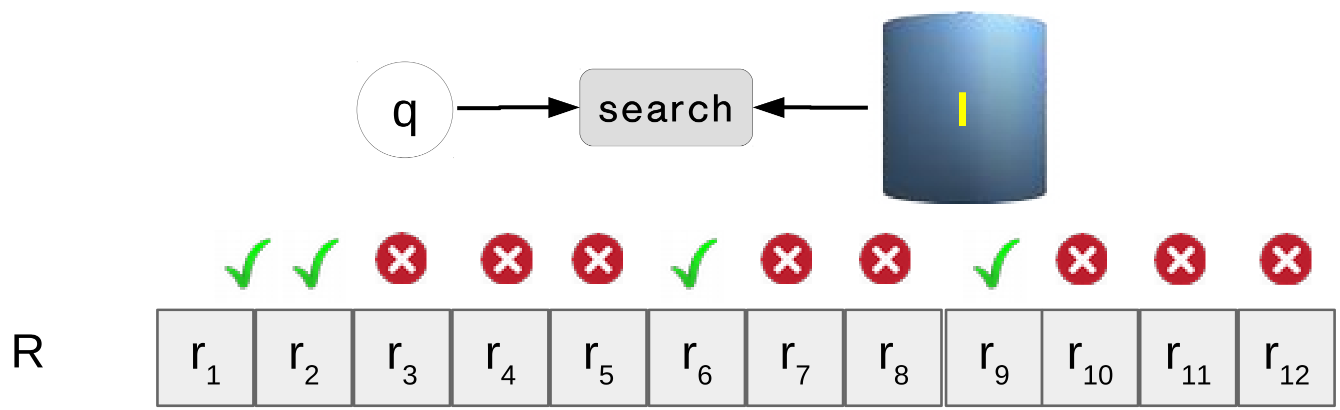
- **Using two different datasets**
  - *Saavedra (1326 images, 53 queries)*
  - *Flickr 15K (14660 images, 330 queries)*
- **Evaluation Metrics**
  - *mAP: Mean Average Precision*
  - *Recall-Precision Graphic*

SBIR: Métodos Tradicionales

Mean Average Precision mAP

I : Conjunto de imágenes a buscar.
Q : Conjunto de consultas, sketches.
R : Ranking, conjunto de impagenes ordenadas.

$$\Gamma_q = \{x \in I, q \in Q | x \text{ es relevante para } Q\}$$



SBIR: Métodos Tradicionales

Mean Average Precision mAP

I : Conjunto de imágenes a buscar.
Q : Conjunto de consultas, sketches.
R : Ranking, conjunto de impagenes ordenadas.

$$\Gamma_q = \{x \in I, q \in Q | x \text{ es relevante para } Q\}$$

Función característica, x es relevante para q

$$f_{\Gamma_q}(x) = \begin{cases} 1 & x \in \Gamma_q \\ 0 & \text{en otro caso} \end{cases}$$

Precisión

$$pr_q(i; R) = \frac{\sum_{k=1}^{i} f_{\Gamma_q}(r_k)}{i} \cdot f_{\Gamma_q}(r_i)$$

Precisión indica pureza, varía entre 0 y 1 indicando mínima y máxima pureza, respectivamente.

precisión se mide solamente en relevantes

SBIR: Métodos Tradicionales

Mean Average Precision mAP

Precisión promedio dada una consulta q

$$AP_q(R) = \frac{\sum_{i=1}^{|R|} pr_q(i,R)}{\sum_{i=1}^{|R|} f_{\Gamma_q}(r_i)}$$

$$AP_q(R) = \frac{\sum_{i=1}^{|R|} pr_q(i,R)}{|\Gamma_q|}$$

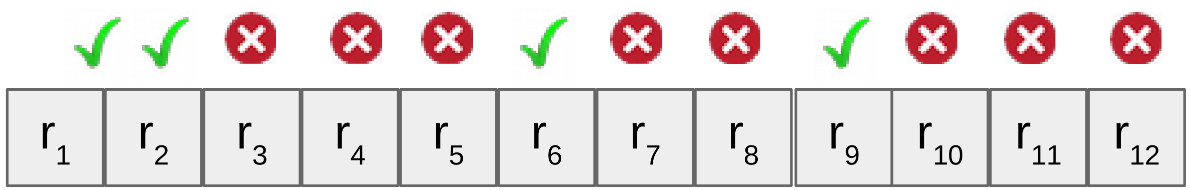
mean Average Precision

$$mAP = \frac{\sum_{q \ inQ} AP_q(R)}{|Q|}$$

SBIR: Métodos Tradicionales

Mean Average Precision mAP

R

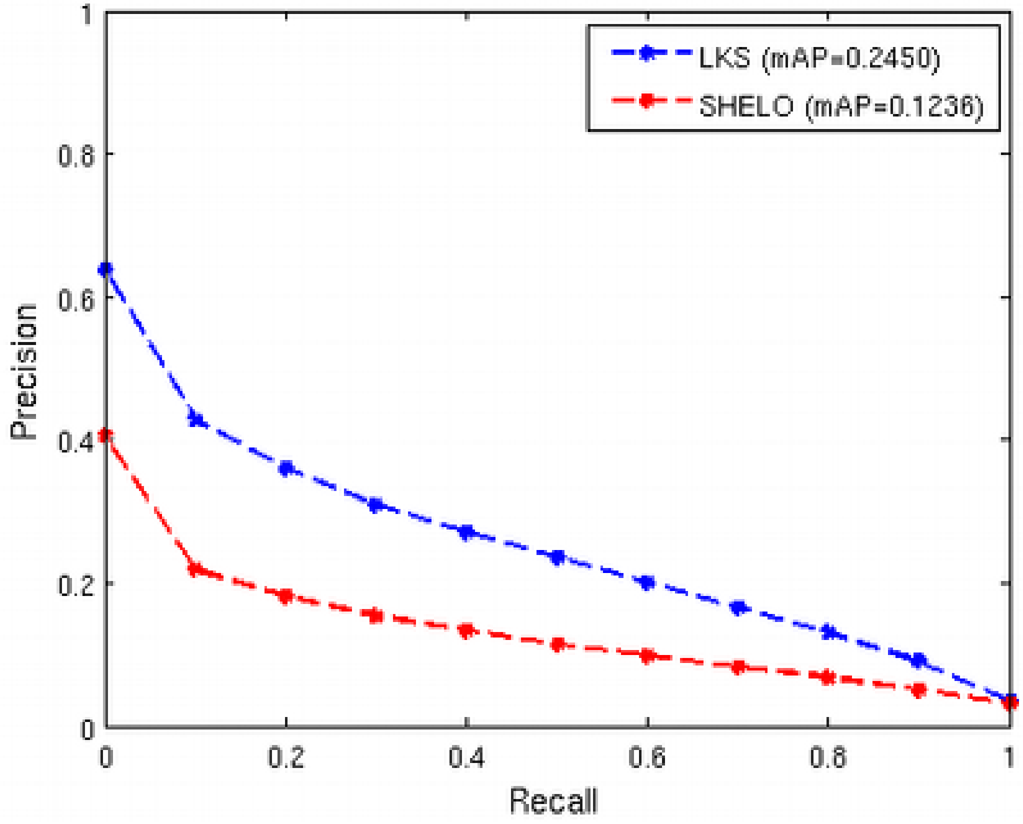


¿Precisión Promedio?

Mean Average Precisión para LKS

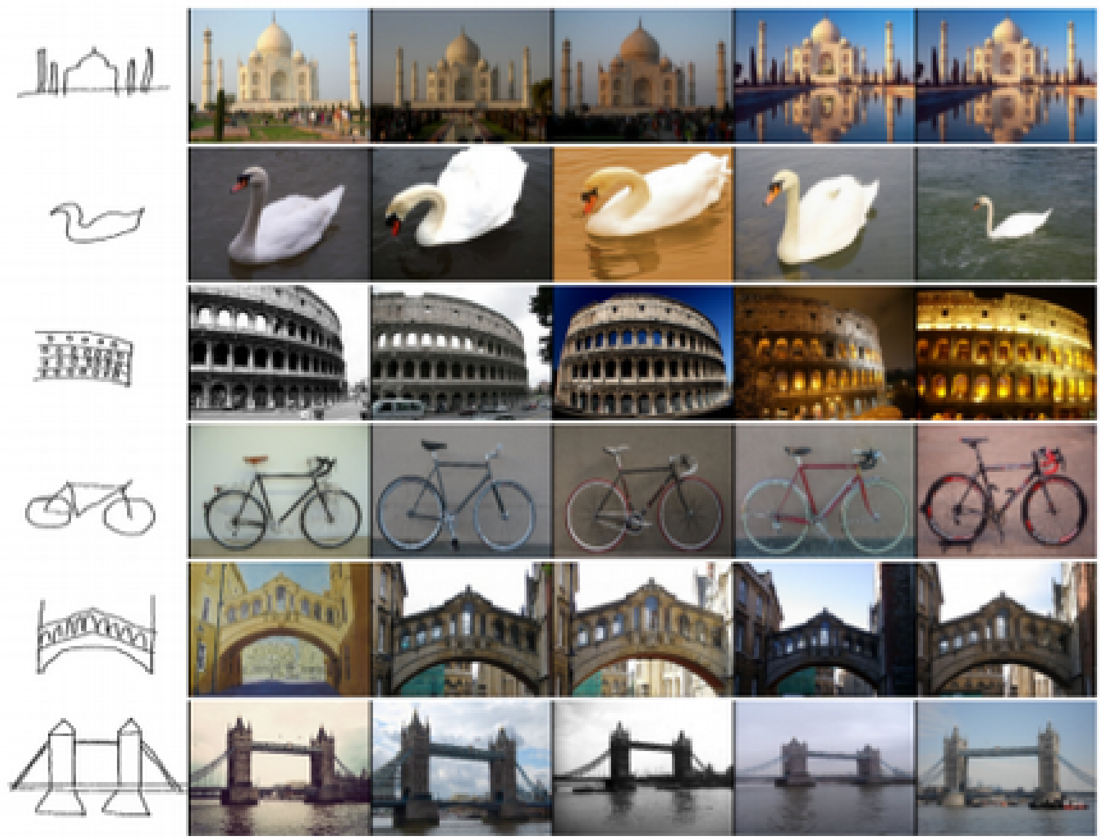
	HOG	GF-HOG[15]	SHELO[25]	**LKS**	gain
Saavedra's	0.2355	*unreported*	0.2766	**0.3251**	17.5%
Flickr15K	0.0771	0.1222	0.1236	**0.2450**	98.2%

Recall-Precision



LKS

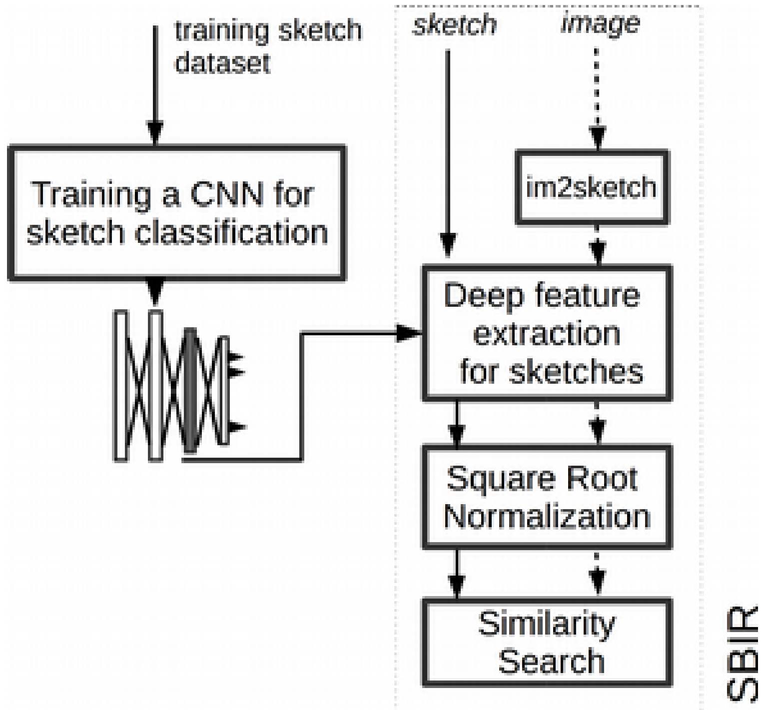
Resultados



SBIR: Deep Features

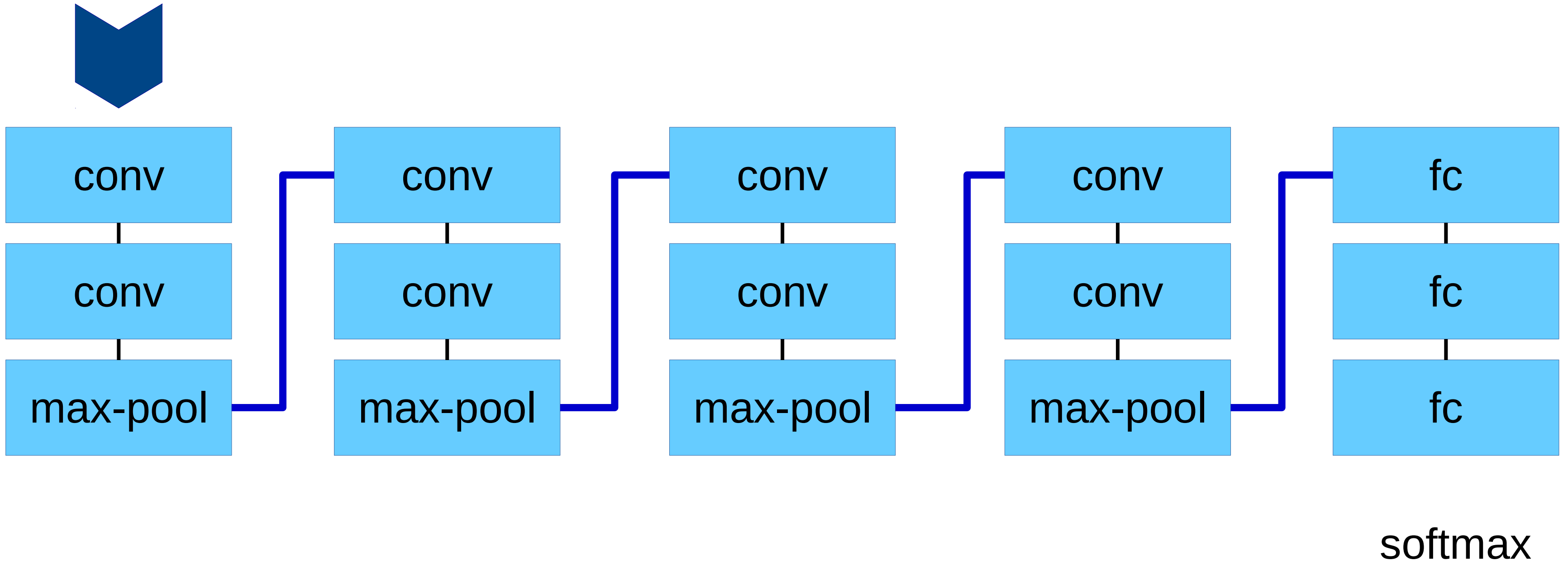
[dejemos que las características sean aprendidas]

SBIR: Deep Features

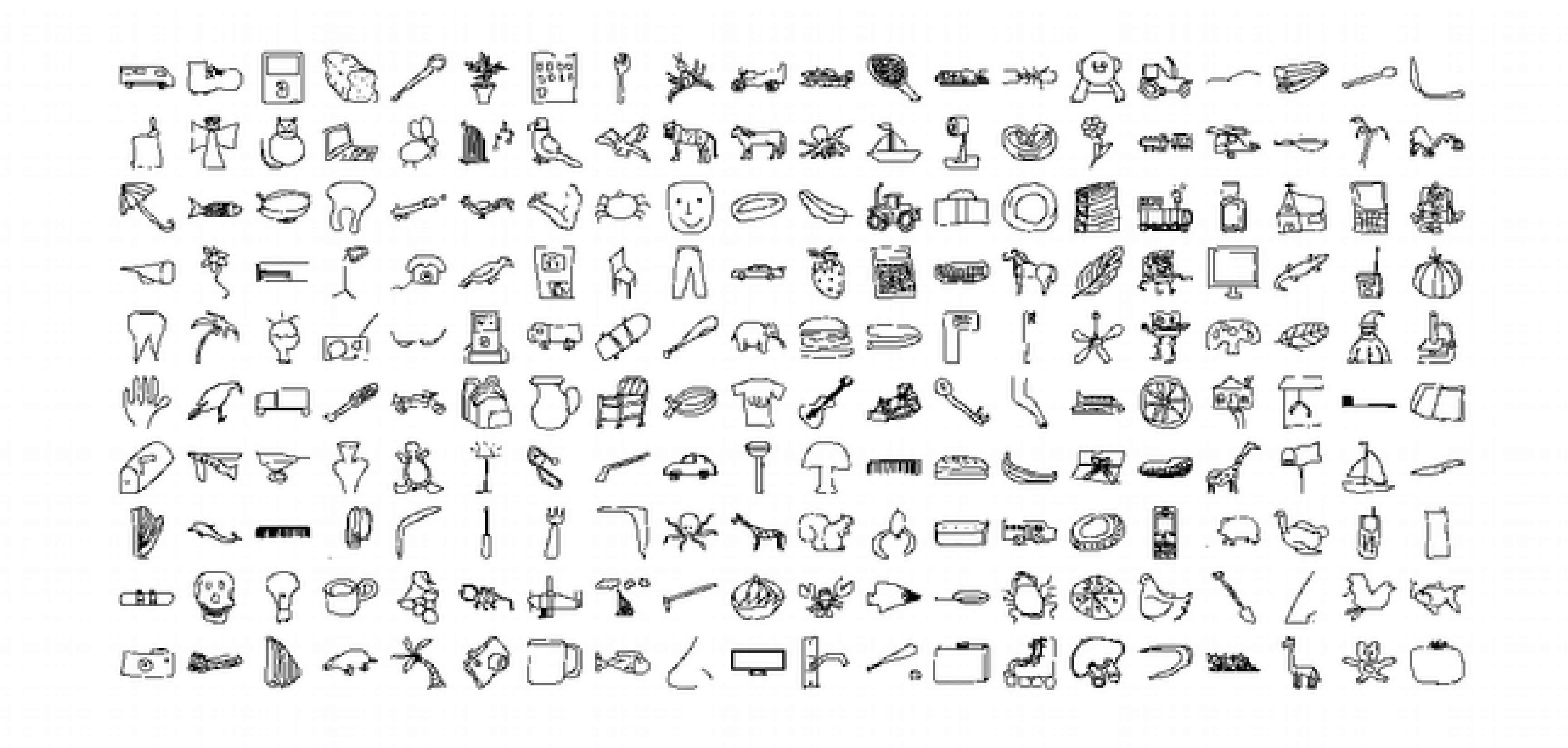


Jose M. Saavedra, Camila Alvarez. DeepSBIR:
Sketch Based Image Retrieval using Deep Features.
In DLPR-ICPR, 2016

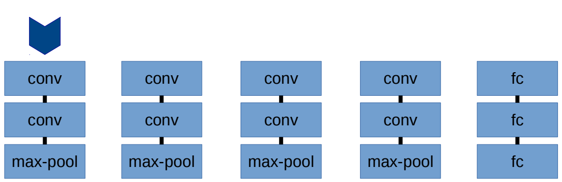
SBIR: Deep Features



SBIR: Deep Features



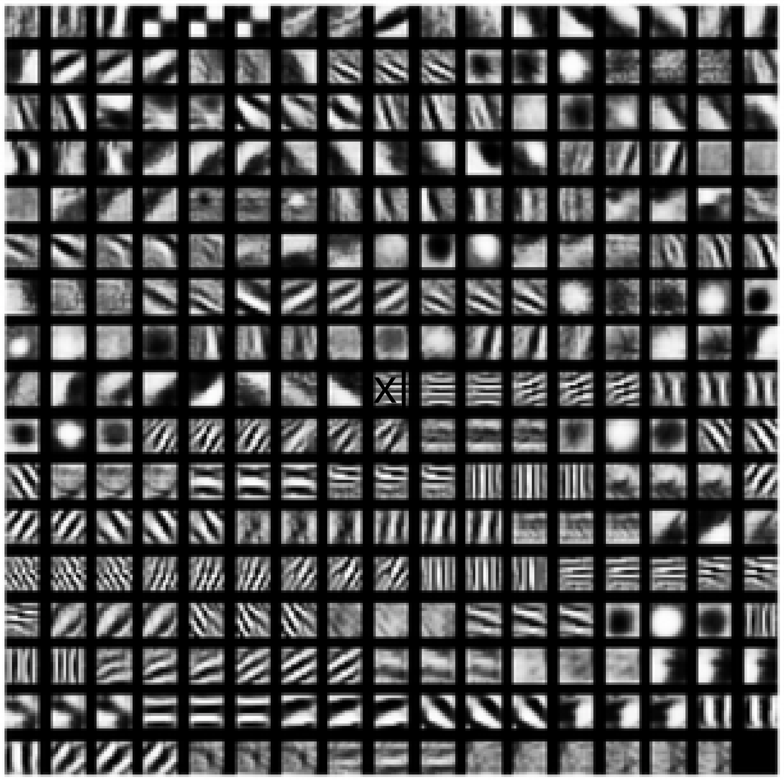
20.000 imágenes/250 clases



Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects

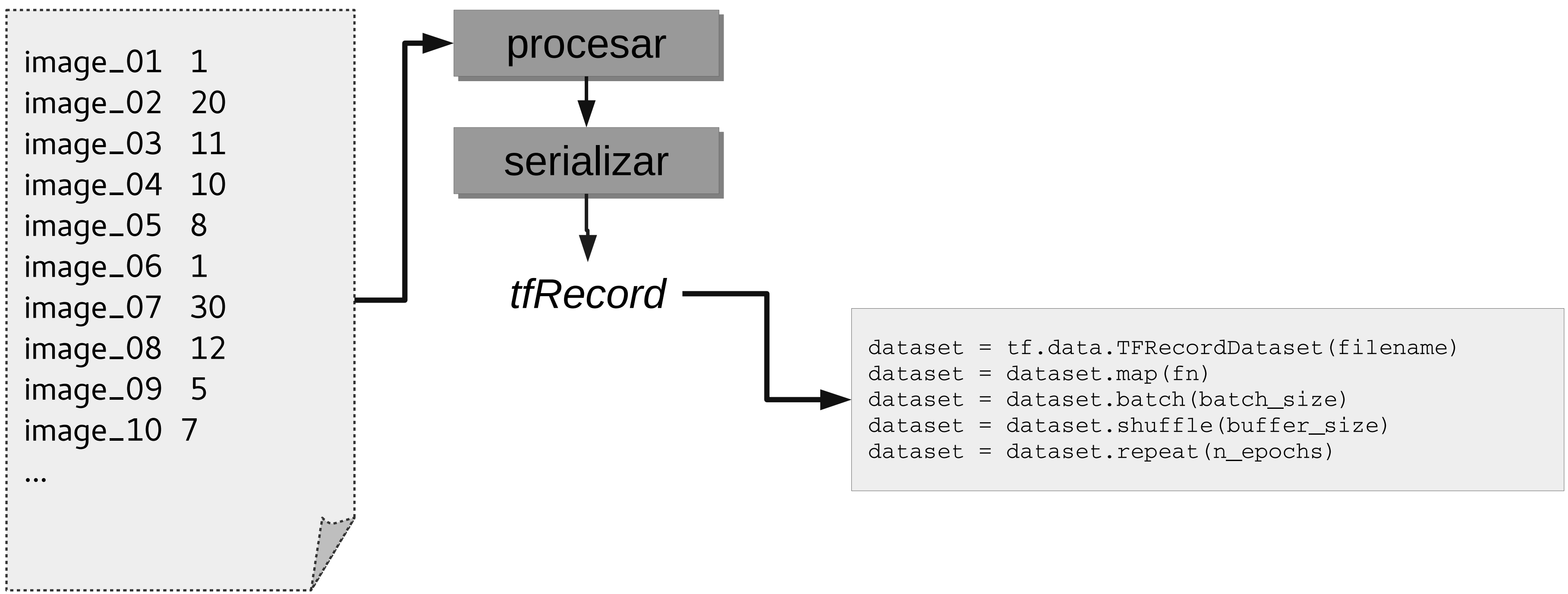
SBIR: Deep Features

Visualización de filtros luego de entrenar en el contexto de sketches



Dataset and Estimators

- Datasets [módulo tf.data]



lista de imágenes a procesar

Dataset and Estimators

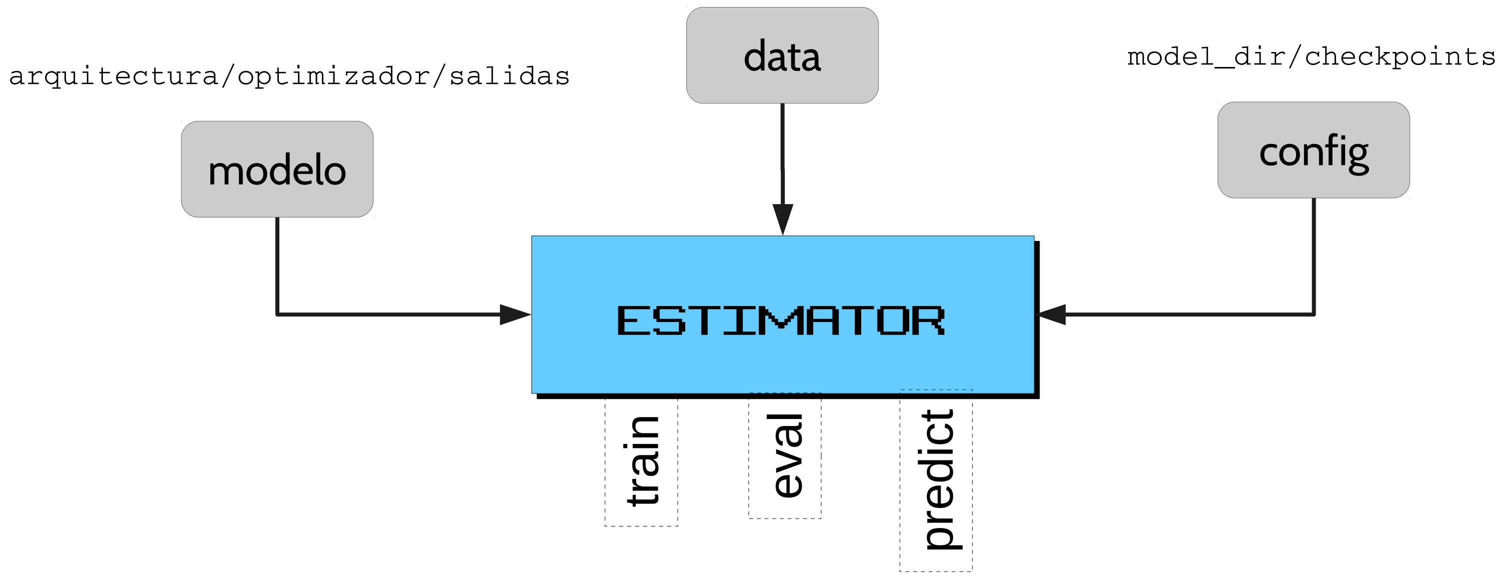
- Datasets [módulo tf.data]

```
def input_fn(filename, image_shape, mean_img, is_training, configuration):
    dataset = tf.data.TFRecordDataset(filename)
    dataset = dataset.map(lambda x: data.parser_tfrecord_sk(x,
                                                            image_shape[0],
                                                            mean_img,
                                                            configuration.getNumnberOfClasses()))
    dataset = dataset.batch(conf.getBatchSize())
    if is_training:
        dataset = dataset.shuffle(configuration.getNumberOfBatches())
        dataset = dataset.repeat(configuration.getNumberOfEpochs())
    # for testing shuffle and repeat is not required
    return dataset
```

Dataset and Estimators

- Estimators

  Permite relizar entrenamiento y pruebas en base a un modelo definido por el usuario.



Dataset and Estimators

- Definir un modelo
  - Aquí se definen la arquitectura y el modelo de optimización

```python
def net_fn(features, input_shape, n_classes, is_training = True):

    #reshape input to fit a  4D tensor
    x_tensor = tf.reshape(features, [-1, input_shape[0], input_shape[1],  1 ] )
    #conv_1
    conv_1 = layers.conv_layer(x_tensor, shape = [7,7, 1, 64],  stride =1, name='conv_1', is_training = is_training)
    conv_1_1 = layers.conv_layer(conv_1, shape = [5,5, 64, 128], stride=2, name='conv_1', is_training = is_training)
    conv_1_1 = layers.max_pool_layer(conv_1_1, 3, 2) # 64 x 64
    print(" conv_1: {} ".format(conv_1_1.get_shape().as_list()))
    #conv_2
    conv_2 = layers.conv_layer(conv_1_1, shape = [3, 3, 128, 128], name = 'conv_2', is_training = is_training)
    conv_2_1 = layers.conv_layer(conv_2, shape = [3, 3, 128, 128], name = 'conv_2', is_training = is_training)
    conv_2_1 = layers.max_pool_layer(conv_2_1, 3, 2) # 32 x 32
    print(" conv_2: {} ".format(conv_2_1.get_shape().as_list()))
    #conv_3
    conv_3 = layers.conv_layer(conv_2_1, shape = [3, 3, 128, 256], name = 'conv_3', is_training = is_training)
    conv_3_1 = layers.conv_layer(conv_3, shape = [3, 3, 256, 256], name = 'conv_3', is_training = is_training)
    conv_3_1 = layers.max_pool_layer(conv_3_1, 3, 2) # 16 x 16
    print(" conv_3: {} ".format(conv_3_1.get_shape().as_list()))
    #conv_4
    conv_4 = layers.conv_layer(conv_3_1, shape = [3, 3, 256, 256], name = 'conv_4', is_training = is_training)
    conv_4_1 = layers.conv_layer(conv_4, shape = [3, 3, 256, 256], name = 'conv_4', is_training = is_training)
    conv_4_1 = layers.max_pool_layer(conv_4_1, 3, 2) # 8x8
    print(" conv_4: {} ".format(conv_4_1.get_shape().as_list()))
    # fc 1
    fc5 = layers.fc_layer(conv_4_1, 1024, name = 'fc5')
    #fc5 = layers.dropout_layer(fc5, 0.8)
    print(" fc5: {} ".format(fc5.get_shape().as_list()))
    #fc 6
    fc6 = layers.fc_layer(fc5, 1024, name = 'fc6')
    #fc6 = layers.dropout_layer(fc6, 0.8)
    print(" fc6: {} ".format(fc6.get_shape().as_list()))
    #fully connected
    fc7 = layers.fc_layer(fc6, n_classes, name = 'fc7', use_relu = False)
    print(" fc7: {} ".format(fc7.get_shape().as_list()))
    #gap = layers.gap_layer(conv_5) # 8x8
    #print(" gap: {} ".format(gap.get_shape().as_list()))
    return {"output": fc7, "deep_feature": fc6}
```

Dataset and Estimators

- Definir un modelo

```
##
def model_fn (features, labels, mode, params):
    #instance of the cnet
    if mode == tf.estimator.ModeKeys.TRAIN:
        is_training = True
    else :
        is_training = False

    net = net_fn(features, params['image_shape'], params['number_of_classes'], is_training)
    train_net = net["output"]
    #----------------------------------------
    idx_predicted_class = tf.argmax(train_net, 1)
    #---------------------------------------
    # If prediction mode, early return
    predictions = { "idx_predicted_class": idx_predicted_class,
                    "predicted_probabilities": tf.nn.softmax(train_net, name="pred_probs"),
                    "deep_feature" : net["deep_feature"]
                   }
    if mode == tf.estimator.ModeKeys.PREDICT:
        estim_specs = tf.estimator.EstimatorSpec(mode, predictions=predictions)
    else :
        idx_true_class = tf.argmax(labels, 1)
        # Evaluate the accuracy of the model
        acc_op = tf.metrics.accuracy(labels=idx_true_class, predictions=idx_predicted_class)
        # Evaluate losss through an optimizer
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits = train_net, labels = labels)
        loss = tf.reduce_mean(cross_entropy)
        update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS) # to allow update is_training variable
        with tf.control_dependencies(update_ops) :
            optimizer = tf.train.AdamOptimizer(learning_rate= params['learning_rate'])
            train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
        # TF Estimators requires to return a EstimatorSpec, that specify
        # the different ops for training, evaluating, ...
        estim_specs = tf.estimator.EstimatorSpec(
            mode=mode,
            predictions=idx_predicted_class,
            loss=loss,
            train_op=train_op,
            eval_metric_ops={'accuracy': acc_op})

    return  estim_specs
```

Dataset and Estimators

- Entrenar y Evaluar

```
with tf.device(device_name):
    estimator_config = tf.estimator.RunConfig(model_dir = conf.getSnapshotPrefix(),
                                              save_checkpoints_steps=conf.getSnapshotTime(),
                                              keep_checkpoint_max=10)

    classifier = tf.estimator.Estimator(model_fn = sknet.model_fn,
                                        config = estimator_config,
                                        params = {'learning_rate' : conf.getLearningRate(),
                                                  'number_of_classes' : conf.getNumnberOfClasses(),
                                                  'image_shape' : image_shape
                                                 }
                                        )
    #
    tf.logging.set_verbosity(tf.logging.INFO) # Just to have some logs to display for demonstration
    #training
    if run_mode == 'train':
        train_spec = tf.estimator.TrainSpec(input_fn = lambda: input_fn(filename_train,
                                                                        image_shape,
                                                                        mean_img,
                                                                        is_training = True,
                                                                        configuration =  conf),
                                  steps=conf.getNumberOfIterations()
                                  )
        eval_spec = tf.estimator.EvalSpec(input_fn = lambda: input_fn(filename_test,
                                                                      image_shape,
                                                                      mean_img,
                                                                      is_training = False,
                                                                      configuration =  conf),
                                          start_delay_secs = conf.getTestTime(),
                                          throttle_secs = conf.getTestTime()*2)

        tf.estimator.train_and_evaluate(classifier, train_spec, eval_spec)
```

Dataset and Estimators

- Predicción

```
with tf.device(device_name):

    classifier = tf.estimator.Estimator(model_fn = sknet.model_fn,
                                        model_dir = "trained_sk_256_thick_simple/",
                                        params = {'learning_rate' : 0,
                                                  'number_of_classes' : 250,
                                                  'image_shape' : image_shape
                                                  })
    #
    tf.logging.set_verbosity(tf.logging.INFO) # show log
    #training
    filename = pargs.image
    input_image =  input_fn(filename, image_shape, mean_img)
    predict_input_fn = tf.estimator.inputs.numpy_input_fn(
            x=input_image,
            num_epochs=1,
            shuffle=False
            )

    predicted_result = list(classifier.predict(input_fn = predict_input_fn))
    for prediction in predicted_result:
        print("idx_predicted_class: {}".format(prediction["idx_predicted_class"]))
        deep_features = prediction["deep_feature"]
        print("deep_feature: {}".format(deep_features.shape))
        print("deep-features")
        print("-------------")
        print(deep_features)
```