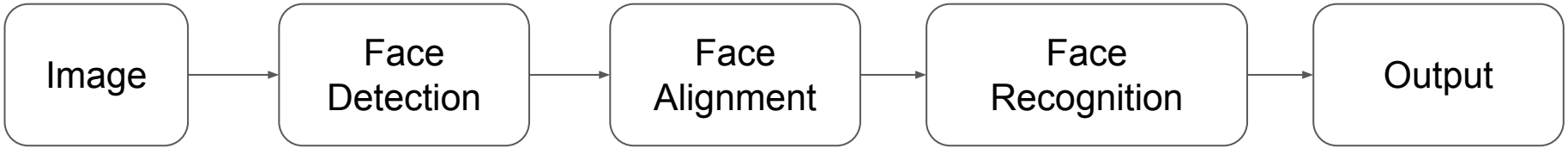
Tema 26: Teoría DeepFace

Pipeline (Inferencia)



Resize image (480, 640)

SSD-Mobilenet

68-Facial Landmarks (dlib)

VGGFace2 - Resnet

Bounding Boxes

Main program (bin/face.py)

```python
# detector_ssd_mobilenet_v2
def run(self, detector='detector_ssd_mobilenet_v2', recognizer=FaceRecognizerResnet.NAME, image='./samples/blackpink/blackpink1.jpg',
        visualize=True):

    self.set_detector(detector)                                    # Inicializar modelo de deteccion
    self.set_recognizer(recognizer)                                # Inicializar modelo de reconocimiento

    npimg = cv2.imread(image, cv2.IMREAD_COLOR)                    # Leer imagen

    if npimg is None:                                              # Verificar lectura de imagen
        sys.exit(-1)

    t = time.time()
    faces = self.detector.detect(npimg)                            # Correr detector de faces
    print("Detecta {} faces en {} segundos".format(len(faces), time.time()-t))

    if recognizer:
        faces = self.run_recognizer(npimg, faces, recognizer)      # Correr Reconocimiento de rostros

    img = draw_bboxs(np.copy(npimg), faces)                        # Dibujar los bounding boxes
    cv2.imshow('DeepFace', img)                                    # Mostrar imagen
    cv2.waitKey(0)

    return faces
```

Face Detection (deepface/detectors/detector_ssd.py)

```
def detect(self, npimg, resize=(480, 640)):
    """
    :param npimg:
    :param resize: False or tuple
    :return:
    """
    height, width = npimg.shape[:2]
    if not resize:
        infer_img = npimg
    else:
        infer_img = cv2.resize(npimg, resize, cv2.INTER_AREA)

    dets, scores, classes = self.session.run([self.tensor_boxes, self.tensor_score, self.tensor_class], feed_dict={
        self.tensor_image: [infer_img]
    })
```

Face detection (deepface/detector/detector_ssd.py)

```python
class FaceDetectorSSD(FaceDetector):
    NAME = 'detector_ssd'

    def __init__(self, specific_model):
        super(FaceDetectorSSD, self).__init__()
        self.specific_model = specific_model
        graph_path = os.path.join(
            os.path.dirname(os.path.realpath(__file__)),
            DeepFaceConfs.get()['detector'][self.specific_model]['frozen_graph']            # Pesos congelados del modelo
        )
        self.detector = self._load_graph(graph_path)                                        # Cargar Tensorflow graph

        self.tensor_image = self.detector.get_tensor_by_name('prefix/image_tensor:0')       # Tensor referente a imagen
        self.tensor_boxes = self.detector.get_tensor_by_name('prefix/detection_boxes:0')    # Tensor de bounding boxes
        self.tensor_score = self.detector.get_tensor_by_name('prefix/detection_scores:0')   # Tensor de scores
        self.tensor_class = self.detector.get_tensor_by_name('prefix/detection_classes:0')  # Tensor de clases detectadas

        predictor_path = os.path.join(
            os.path.dirname(os.path.realpath(__file__)),
            DeepFaceConfs.get()['detector']['dlib']['landmark_detector']
        )
        self.predictor = dlib.shape_predictor(predictor_path)                               # Inicializar predictor de landmarks

        config = tf.ConfigProto(gpu_options=tf.GPUOptions(allow_growth=True))               # Uso eficiente gpu
        self.session = tf.Session(graph=self.detector, config=config)
```

Face Alignment (deepface/utils/common.py)

```python
def landmark_to_pose(landmark, image_shape):
    image_points = np.array([
        landmark[33],  # (359, 391),  # Nose tip
        landmark[8],  # (399, 561),  # Chin
        landmark[36],  # (337, 297),  # Left eye left corner
        landmark[45],  # (513, 301),  # Right eye right corne
        landmark[48],  # (345, 465),  # Left Mouth corner
        landmark[54],  # (453, 469)  # Right mouth corner
    ], dtype='double')
```

```python
def get_roi(img, face, roi_mode):
    """
    :return: Cropped & Aligned Face Image
    """
    rpy, node_point = landmark_to_pose(face.face_landmark, img.shape)
    roll = rpy[0]
    if abs(roll) > math.pi / 2.0:
        roll = 0.0  # TODO ?

    height, width = img.shape[:2]
    new_w, new_h = (abs(math.sin(roll) * height) + abs(math.cos(roll) * width),
                    abs(math.sin(roll) * width) + abs(math.cos(roll) * height))
    new_w = roundint(new_w)
    new_h = roundint(new_h)
    mat = cv2.getRotationMatrix2D((height / 2, width / 2), -1 * roll * 180.0 / math.pi, 1.0)
    (tx, ty) = (roundint((new_w - width) / 2), roundint((new_h - height) / 2))
    mat[0, 2] += tx
    mat[1, 2] += ty
    dst = cv2.warpAffine(img, mat, dsize=(new_w + tx * 2, new_h + ty * 2))
```

Face Recognition (deepface/recognizer/recognizer_resnet.py)

```
def detect(self, rois=None, npimg=None, faces=None):
    probs, feats = self.extract_features(npimg=npimg,
                                         rois=rois,
                                         faces=faces)

    if self.db is None:
        names = [[(str(self.class_names[idx].encode('utf8')), prop[idx]) for idx in
                  prop.argsort()[-5:][::-1]] for prop in probs]
    else:
        names = []
        for feat in feats:
            scores = []
            for db_name, db_feature in self.db.items():
                similarity = np.dot(feat / np.linalg.norm(feat, 2), db_feature / np.linalg.norm(db_feature, 2))
                scores.append((db_name, similarity))
            scores.sort(key=lambda x: x[1], reverse=True)
            names.append(scores)

    return {
        'output': probs,
        'feature': feats,
        'name': names
    }
```

Face Recognition (extract features)

```
def extract_features(self, npimg=None, rois=None, faces=None):
    if not rois and faces:
        rois = faces_to_rois(npimg=npimg,
                             faces=faces,
                             roi_mode=FaceRecognizerResnet.NAME)

    if rois:
        new_rois = self.get_new_rois(rois=rois)
        for face, roi in zip(faces, new_rois):
            face.face_roi = roi
    else:
        return np.array([]), np.array([])

    probs = []
    feats = []
    for roi_chunk in grouper(new_rois, self.batch_size, fillvalue=np.zeros((224, 224, 3), dtype=np.uint8)):
        prob, feat = self.persistent_sess.run([self.network['out'], self.network['feat']],
                                              feed_dict={self.input_node: roi_chunk})
        feat = [np.squeeze(x) for x in feat]
        probs.append(prob)
        feats.append(feat)
    probs = np.vstack(probs)[:len(rois)]
    feats = np.vstack(feats)[:len(rois)]

    return probs, feats
```

Resource

- Deepface
  https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Taigman_DeepFace_Closing_the_2014_CVPR_paper.pdf?fbclid=IwAR2pWtIuPH-iJ7dXnb8C19icRtkjCL1VdQviRSbxPnWsn6OpVnTml-msR-E