



FACE MASK DETECTION

Custom Vision

Martín Alexis Samán Arata
martin.saman@vallegrande.edu.pe

Contenido

Contexto	2
Objetivos	2
Implementar el modelo	2
Entrenar el modelo	2
Prerequisitos	2
Crear el proyecto	2
Elegir imágenes de entrenamiento	3
Cargar y etiquetar imágenes	4
Entrenar el modelo	6
Evaluar el modelo	8
Precision	8
Recall	8
AP	8
Publicar el modelo	8
Publicar	8
Obtener el API	9
Visualizar las predicciones	10
Consumir el modelo	11
Spring Boot	11
Angular	12
Primefaces	15
Conclusiones	17
Consideraciones	17
Anexos	18
Referencias	18

Contexto

El [Decreto Supremo N° 057-2020-PCM](#) establece como obligatorio el uso de mascarilla para circular por las vías de uso público. Tomando medidas de sanidad y seguridad en entes estatales y empresas a nivel nacional (Perú), con el fin de evitar propagar el virus y disminuir la cantidad de infectados.

Objetivos

Automatizar el procedimiento de identificar si una persona lleva puesta mascarilla o no, mediante el uso de un modelo de Machine Learning haciendo uso del servicio “Custom Vision” de Microsoft.

Implementar el modelo

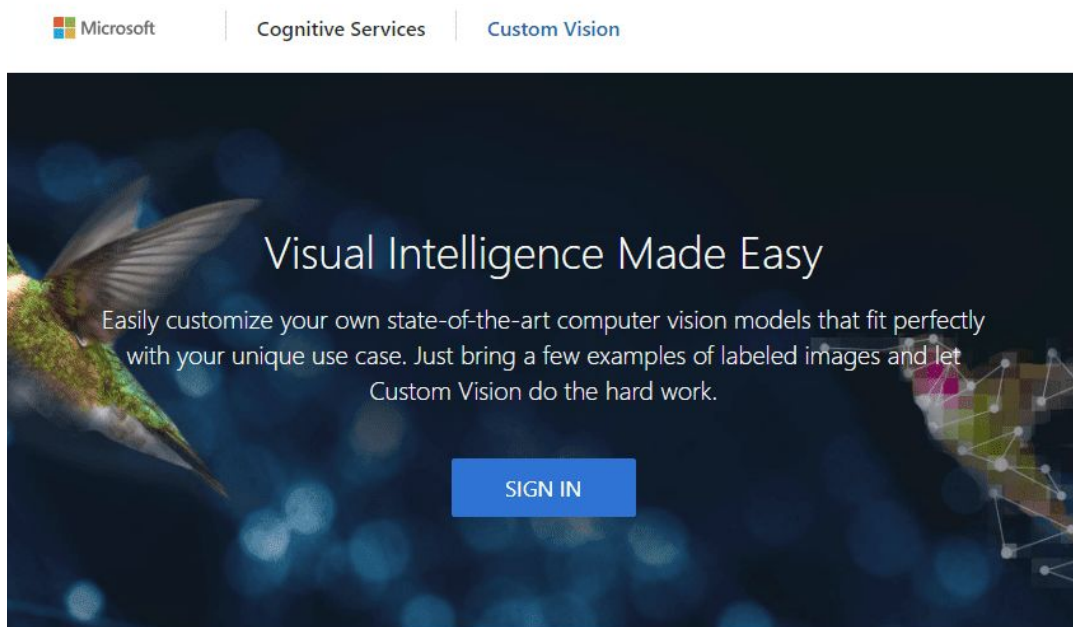
Entrenar el modelo

Prerequisitos

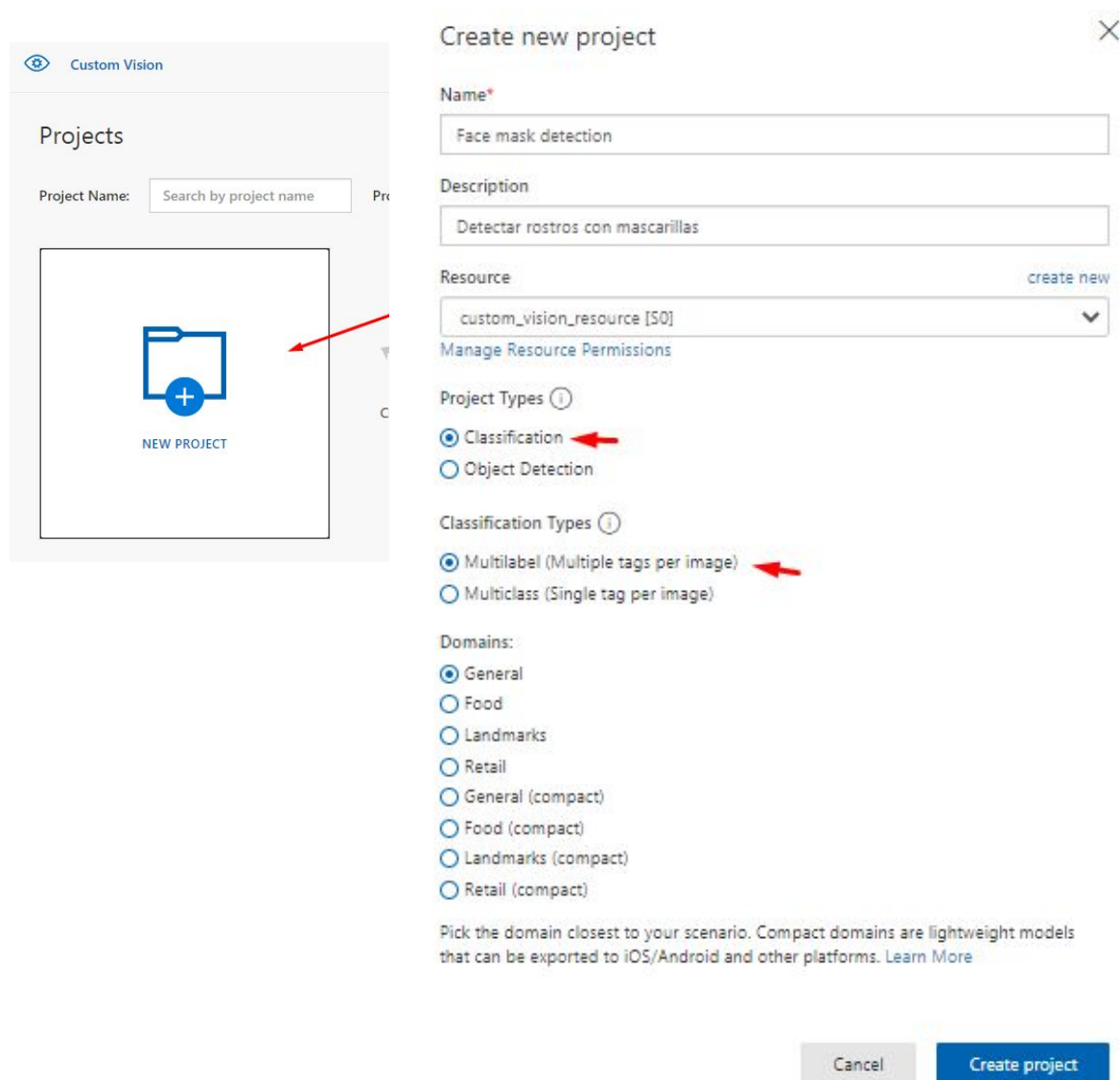
Dataset de imágenes, en este caso usaremos: [FaceMaskDataset](#).

Crear el proyecto

Nos dirigimos a la [página web de Custom Vision](#) y seleccione **Iniciar sesión**. Inicie sesión con la misma cuenta de Azure.



Seleccionamos en **New Project**, configuramos y creamos el proyecto.



Create new project

Name*
Face mask detection

Description
Detectar rostros con mascarillas

Resource [create new](#)
custom_vision_resource [S0]

[Manage Resource Permissions](#)

Project Types ⓘ
☒ Classification
☐ Object Detection

Classification Types ⓘ
☒ Multilabel (Multiple tags per image)
☐ Multiclass (Single tag per image)

Domains:
☒ General
☐ Food
☐ Landmarks
☐ Retail
☐ General (compact)
☐ Food (compact)
☐ Landmarks (compact)
☐ Retail (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

[Cancel](#) [Create project](#)

Elegir imágenes de entrenamiento

Como mínimo, es recomendable utilizar al menos 30 imágenes por etiqueta en el conjunto de formación inicial. Se puede recopilar las imágenes cuando se hagan predicciones: mejora continua del modelo.

Para entrenar su modelo de manera efectiva, use imágenes con variedad visual. Seleccione imágenes que varíen según:

- Ángulo de la cámara.
- Con distintas proporciones de iluminación.
- Estilo visual.
- Tamaño.
- Peso.

Además, las imágenes de entrenamiento deben cumplir con los siguientes criterios:

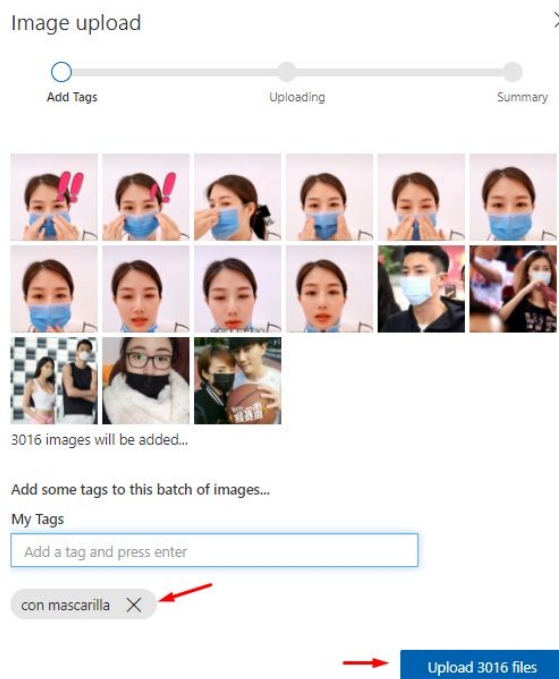
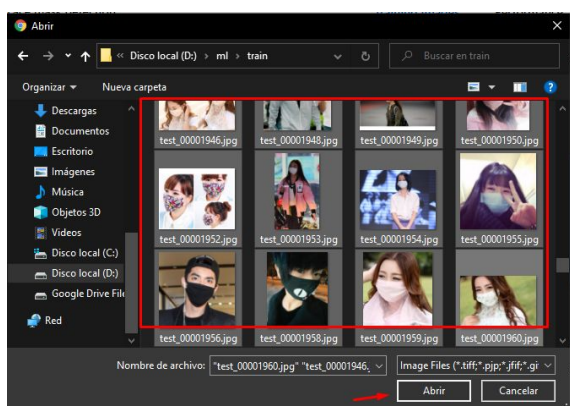
- Formato .jpg, .png, .bmp o .gif
- no más de 6 MB de tamaño (4 MB para imágenes de predicción)
- no menos de 256 píxeles en el borde más corto; Cualquier imagen más corta que esta será escalada automáticamente por el servicio Custom Vision.

Cargar y etiquetar imágenes

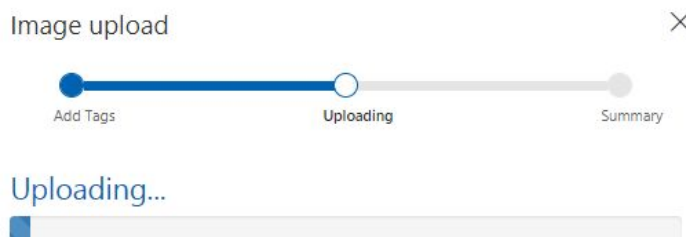
Para subir imágenes seleccionamos **Add images**.



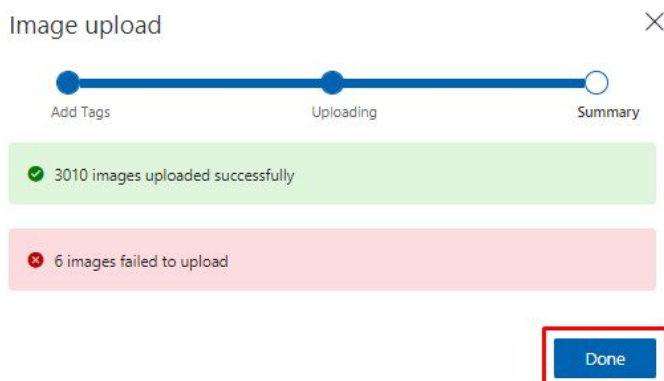
Seleccionamos las imágenes que subiremos y seleccionamos **Abrir**.



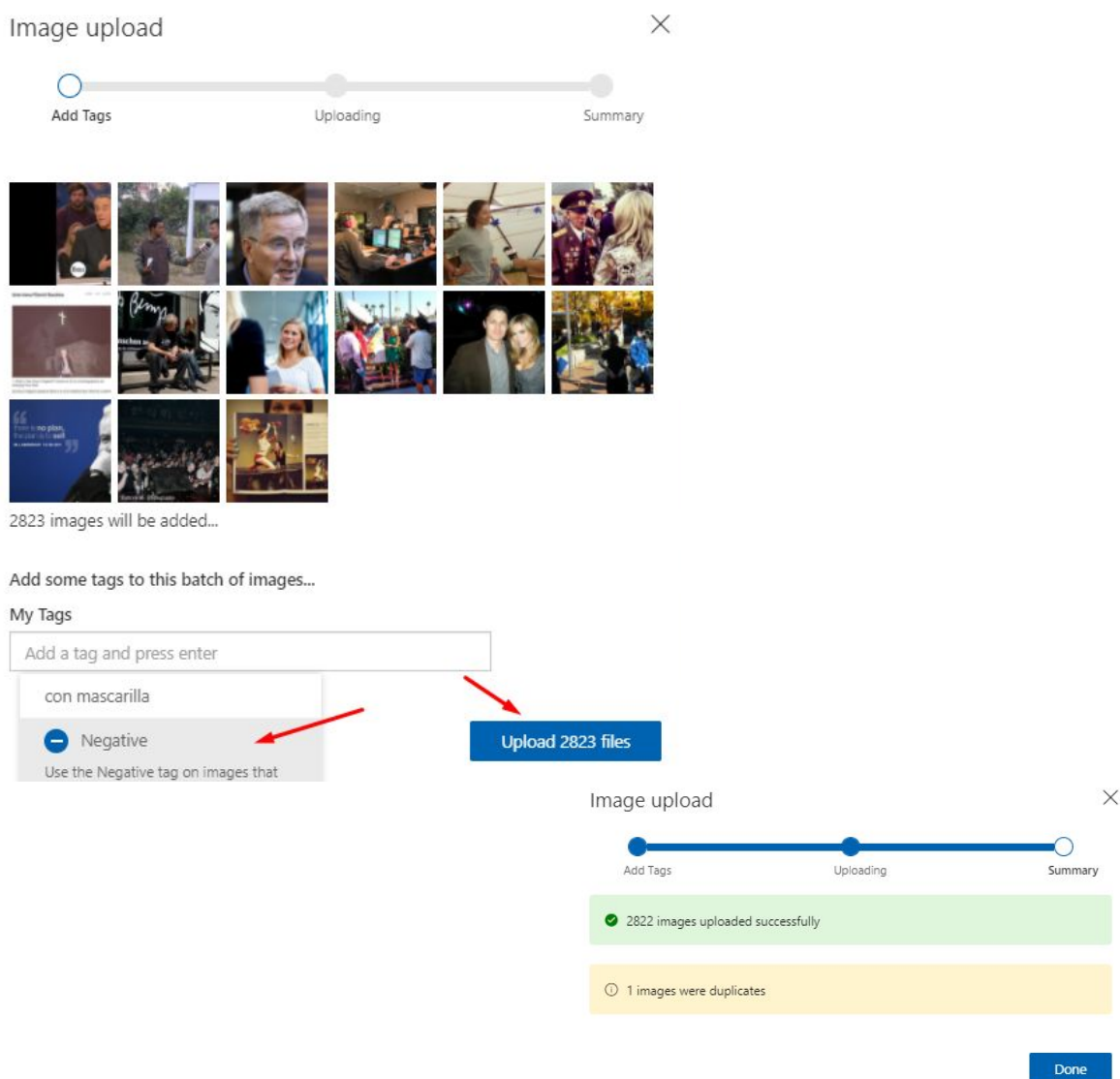
Posteriormente se le agrega la etiqueta y subimos las imágenes seleccionando **Upload xx files** y esperamos a que se suban las imágenes.



Después nos saldrá un resumen, le damos en **Done**.



Repetimos el proceso, pero ahora con las imágenes que no contiene lo que queremos clasificar (negativas):



Entrenar el modelo

Seleccionamos las etiquetas dándole **check**.

Showing:

con mascarilla X

— Negative X

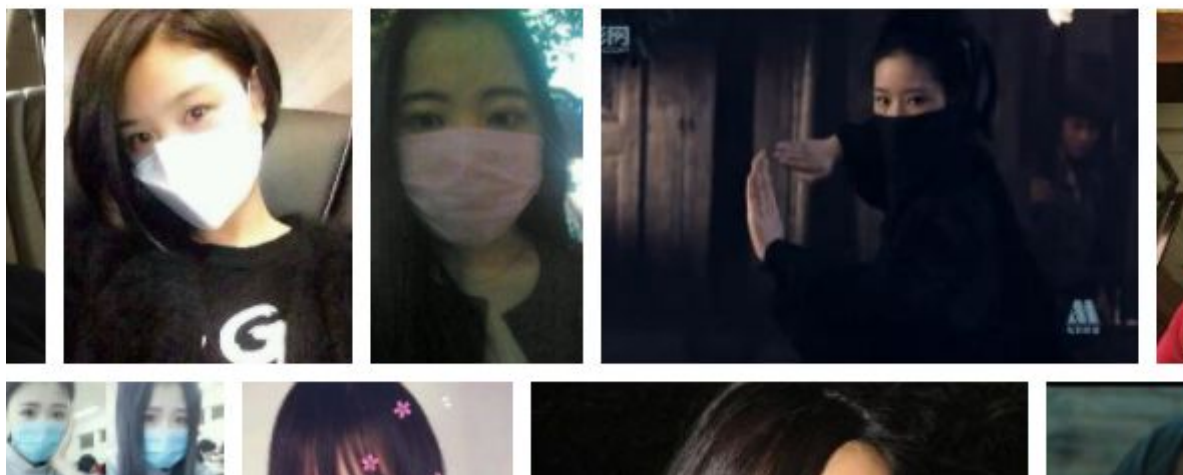
Search For Tags:

- ☒ Negative 2822 ...
- ☒ con mascarilla 3010 ...

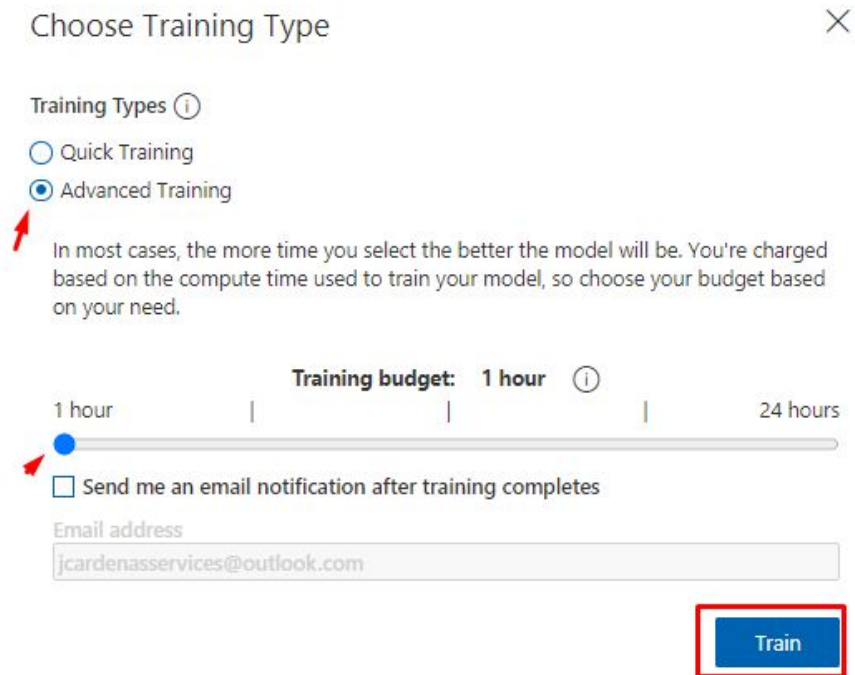
Para entrenar al clasificador, seleccione el botón **Entrenar**. El clasificador usa todas las imágenes actuales para crear un modelo que identifica las cualidades visuales de cada etiqueta.



Tag images Select all



Seleccionamos el **tiempo de entrenamiento**, mientras más tiempo mejor se entrenará el modelo y nos costará más dinero.



Choose Training Type

Training Types ⓘ

☐ Quick Training

☒ Advanced Training

In most cases, the more time you select the better the model will be. You're charged based on the compute time used to train your model, so choose your budget based on your need.

Training budget: 1 hour ⓘ

1 hour | | 24 hours

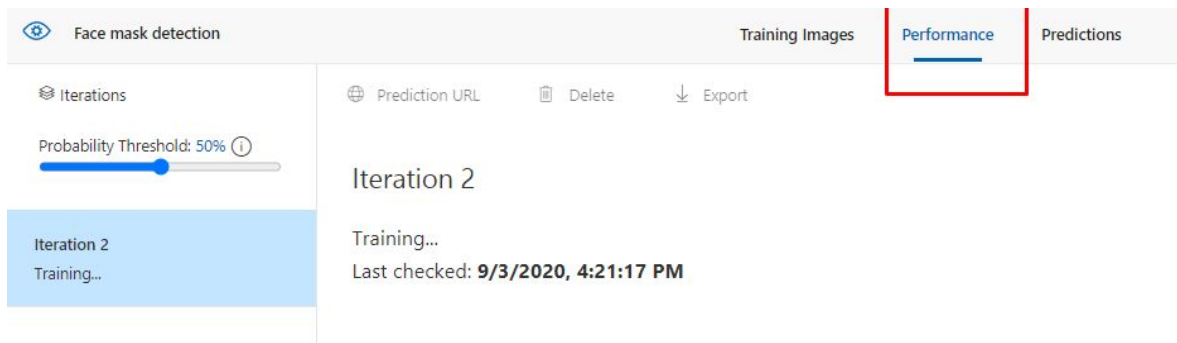
☐ Send me an email notification after training completes

Email address

jcardenasservices@outlook.com

Train

Podemos visualizar el proceso en **Performance**, esperamos.



Face mask detection

Training Images Performance Predictions

Iterations

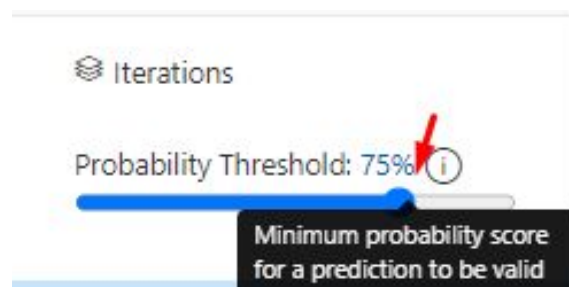
Probability Threshold: 50% ⓘ

Iteration 2

Training...

Last checked: 9/3/2020, 4:21:17 PM

Podemos cambiar el **umbral de probabilidad**.



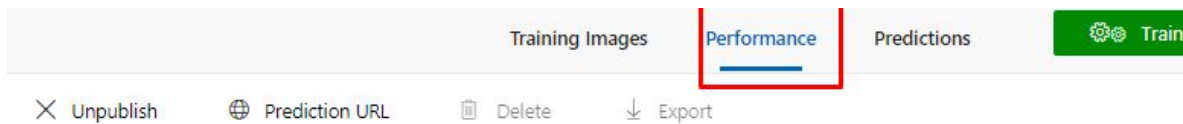
Iterations

Probability Threshold: 75% ⓘ

Minimum probability score for a prediction to be valid

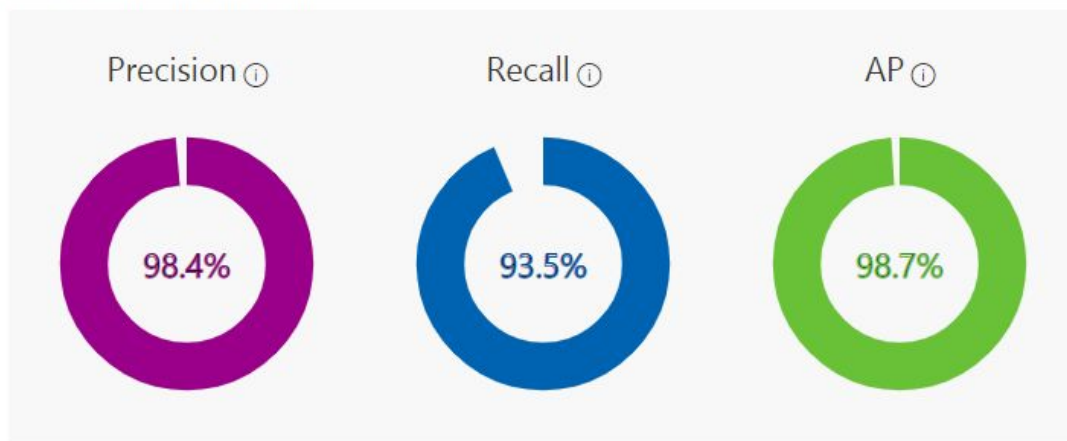
Evaluar el modelo

Podemos ver las métricas de nuestro modelo desde **Performance**



Iteration 2

Finished training on **9/3/2020, 4:25:02 PM** using **General** domain
Iteration id: **f03e2e81-07f0-4ab8-a23e-fad8aa7a2e28**
Classification type: **Multilabel (Multiple tags per image)**
Published as: **Face mask detection**



Precision

¿Qué proporción de identificaciones positivas fue correcta?

Recall

¿Qué proporción de positivos reales se identificó correctamente?

AP

Resume los 2 anteriores de acuerdo al umbral escogido

Publicar el modelo

Publicar

Seleccionamos **Publish**



Asignamos un **nombre** a nuestro modelo y le damos **Publish**, recomiendo no poner espacios en el nombre del modelo para evitar errores al consumirlo.

×

Publish Model

We only support publishing to a prediction resource in the same region as the training resource the project resides in.

Please check if you have a prediction resource and if the prediction resource is in the same region as the training resource.

Model name

Face_mask_detection

Prediction resource

custom_vision_resource

Publish

Cancel

Obtener el API

Seleccionamos **Prediction URL**

×

Unpublish

Prediction URL

🗑 Delete

↓ Export

Las peticiones HTTP son mediante el método POST.

How to use the Prediction API

If you have an image URL:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/a2t
```

Set **Prediction-Key** Header to : 19458cc3f34343b8ada735c605165471

Set **Content-Type** Header to : application/json

Set Body to : {"url": "https://example.com/image.png"}

If you have an image file:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/a2t
```

Set **Prediction-Key** Header to : 19458cc3f34343b8ada735c605165471

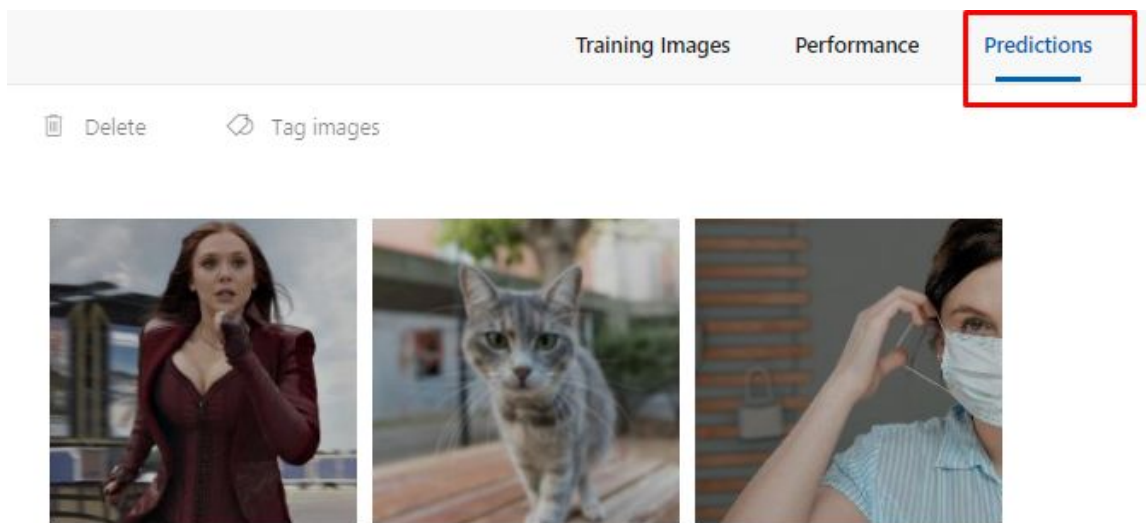
Set **Content-Type** Header to : application/octet-stream

Set Body to : <image file>

Got it!

Visualizar las predicciones

Para ver las predicciones que está haciendo nuestro modelo seleccionamos **Predictions**, esto recopila todas las imágenes que enviamos al API.



Podemos ver el porcentaje de similitud posando el cursor sobre las imágenes.



Se puede reutilizar las imágenes que mandamos al API para seguir mejorando nuestro modelo, volviéndolas a etiquetar.

Consumir el modelo

Spring Boot

Creamos el cliente con OpenFeign para el API de Microsoft Custom Vision.

```
@FeignClient(
    url = "https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/" +
        "a26b276f-3a57-4921-82f2-b678c4b8f07a/classify/iterations/face_mask_detection_model",
    name = "customvision"
)
public interface CustomVisionClient {
    @PostMapping("/url")
    CustomVisionReturned fromUrl(@RequestHeader("Prediction-Key") String predictionKey,
        @RequestBody CustomVisionSended customVisionSended);

    @PostMapping(value = "/image", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    CustomVisionReturned fromImage(@RequestHeader("Prediction-Key") String predictionKey,
        @RequestBody MultipartFile image);
}
```

Creamos el servicio e ingresamos el API Key.

```
@Service
public class CustomVisionService {
    @Autowired
    private CustomVisionClient client;

    private static final String PREDICTION_KEY = "19458cc3f34343b8ada735c605165471";

    public CustomVisionReturned fromUrl(CustomVisionSended customVisionSended) {
        return client.fromUrl(PREDICTION_KEY, customVisionSended);
    }

    public CustomVisionReturned fromImage(MultipartFile image) {
        return client.fromImage(PREDICTION_KEY, image);
    }
}
```

Exponemos nuestro servicio.

```
@PostMapping("/url")
public CustomVisionReturned fromUrl(@RequestBody CustomVisionSended customVisionSended) {
    return service.fromUrl(customVisionSended);
}

@PostMapping(value = "/image")
public CustomVisionReturned fromImage(@RequestParam("imagen") MultipartFile image) {
    return service.fromImage(image);
}
```

Devolveremos lo siguiente:

```
@Data
public class CustomVisionReturned {
    private String id;
    private String project;
    private String iteration;
    private String created;
    private List<PredictionDto> predictions;
}
```

```
@Data
public class PredictionDto {
    private Double probability;
    private String tagId;
    private String tagName;
}
```

Angular

Creamos el servicio que consumirá el API de Spring Boot.

```
export class ServicioService {
    apiUrl = environment.api;

    constructor(private http: HttpClient) {
    }

    public fromUrl(urlImagen: string): Observable<CustomVisionReturned> {
        return this.http.post<CustomVisionReturned>({ url: `${this.apiUrl}/url`, body: {
            url: urlImagen
        }});
    }
}
```

Seteamos el modelo con lo que nos devuelve el API.

```
predecirUrl(): void {
    this.clear();
    this.cliente.fromUrl(this.urlImagen).subscribe( next: data => {
        this.modelo = data;
        console.log(data);
    });
}

clear(): void {
    this.modelo = new CustomVisionReturned();
}
```


Creamos la interfaz.

```
<mat-form-field>
  <mat-label>Imagen</mat-label>
  <input matInput placeholder="Ingrese la url de la imagen" [(ngModel)]="urlImagen">
  <button mat-button *ngIf="urlImagen" matSuffix mat-icon-button aria-label="Clear" (click)="urlImagen='';clear()">
    <mat-icon>close</mat-icon>
  </button>
</mat-form-field>

<mat-list *ngIf="modelo.predictions">
  <div mat-subheader>Datos</div>
  <mat-list-item>
    <mat-icon mat-list-icon>masks</mat-icon>
    <div mat-line *ngIf="modelo.predictions[0].probability>=0.9; else sinMask">SI</div>
    <ng-template #sinMask>
      <div mat-line>NO</div>
    </ng-template>
  </mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item>
    <mat-icon mat-list-icon>timeline</mat-icon>
    <div mat-line>{{modelo.predictions[0].probability}}</div>
  </mat-list-item>
  <mat-divider></mat-divider>
</mat-list>

</mat-card-content>
```

Lo probamos:

Face Mask Detection - Martín Alexis Samán Arata

Microsoft Custom Vision



Imagen

<https://diariocorreo.pe/resize> x

Datos

 NO

 0.483794183

Predecir

Face Mask Detection - Martín Alexis Samán Arata

Microsoft Custom Vision



Imagen

<https://ep01.epimg.net/elpai:> x

Datos



SI



0.9999992

Face Mask Detection - Martín Alexis Samán Arata

Microsoft Custom Vision



Imagen

<https://okdiario.com/img/20> x

Datos



NO



0.000007883426

Primefaces

Creamos el cliente que se conectará con el API de Custom Vision.

```
@Stateless
public class CustomVisionClient {
    private static final String URL_CUSTOM_VISION = "https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/detect";
    private static final String PREDICTION_KEY = "19458cc3f34343b8ada735c605165471";

    public CustomVisionReturned fromUrl(String url) throws IOException {
        OkHttpClient client = new OkHttpClient().newBuilder()
            .build();
        MediaType mediaType = MediaType.parse("application/json");
        RequestBody body = RequestBody.create(mediaType, "{\r\n  \"url\": \"\" + url + \"\"\r\n}");
        Request request = new Request.Builder()
            .url(URL_CUSTOM_VISION + "/url")
            .method("POST", body)
            .addHeader("Prediction-Key", PREDICTION_KEY)
            .addHeader("Content-Type", "application/json")
            .build();
        Response response = client.newCall(request).execute();
        Gson gson = new Gson();
        return gson.fromJson(Objects.requireNonNull(response.body()).string(), CustomVisionReturned.class);
    }
}
```

Creamos el view que consume el cliente con datos de la vista.

```
public void predecirUrl() throws IOException {
    this.customVisionReturned = this.client.fromUrl(this.urlImagen);
    this.containsMask();
}

public void containsMask() {
    String resultado = "";
    if (this.customVisionReturned.getPredictions().get(0).getProbability() >= 0.9) {
        resultado = "Con Mascarilla";
    } else {
        resultado = "Sin Mascarilla";
    }
    mostrarMensaje(resultado, detail: "Probabilidad: " +
        this.customVisionReturned.getPredictions().get(0).getProbability().floatValue());
}

public void mostrarMensaje(String summary, String detail) {
    FacesContext context = FacesContext.getCurrentInstance();
    context.addMessage(null, new FacesMessage(summary, detail));
}
```

Creamos la vista.

```
<h:head>
    <title>Face Mask Detection</title>
</h:head>

<h:body>
    <h1>Face Mask Detection</h1>
    <h2>Custom Vision - Martin Alexis Samán Arata</h2>
    <h:form id="frmInput">
        <p:growl life="10000" showDetail="true" showSummary="true"/>
        
        <p:separator/>
        <p:inputText value="#{customVisionBean.urlImagen}"/>
        <p:commandButton id="btnPredecir" actionListener="#{customVisionBean.predecirUrl()}" value="Predecir"
            update="frmInput"/>
        <p:blockUI block="frmInput" trigger="btnPredecir"/>
    </h:form>
</h:body>
</html>
```


Lo probamos:

Face Mask Detection

Custom Vision - Martín Alexis Samán Arata



<https://estaticos.muyinteres.com>

Predecir



Con Mascarilla

Probabilidad: 0.9999998

Face Mask Detection

Custom Vision - Martín Alexis Samán Arata



<https://www.purina.es/gato/f>

Predecir



Sin Mascarilla

Probabilidad: 4.264071E-6

Face Mask Detection

Custom Vision - Martín Alexis Samán Arata



https://depor.com/resizer/_C

Predecir



Sin Mascarilla

Probabilidad:
0.0036598796

Conclusiones

Es más rápido y fácil implementar modelos de Machine Learning con servicios externos, a costa de eso tenemos que pagar: hay que hacer un trade-off y ver si nos conviene.

Hay que tener una gran cantidad de imágenes para que nuestro modelo de Machine Learning sea de calidad.

Para clasificar imágenes se tiene que etiquetar las imágenes de acuerdo a lo que queremos identificar.

En la plataforma de Custom Vision nos es indiferente los algoritmos que usan por detrás ya que automáticamente eligen el mejor, la contra es que no sabemos cual es para poder replicarlo manualmente.

Para clasificar imágenes es necesario tener un porcentaje del dataset en donde aparezca lo que queremos identificar y otro en donde no aparezca.

Si es que contamos con pocas imágenes es recomendable preprocesarlas antes: transformarlas, distorsionarlas; con el fin de tener diferentes perspectivas y aumentar la cantidad.

En Custom Vision se puede reutilizar las imágenes que le mandamos al API para aumentar nuestro dataset, para mejorar el modelo actual o crear uno nuevo.

El modelo entrenado tiene muchos campos de aplicación, más orientado a IoT, con el fin de evitar la propagación del virus ergo infectados o hasta muertos.

Para afrontar problemas de Machine Learning es importante recolectar la data y por eso la colaboración es vital, en este caso el dataset usado es de China.

Consideraciones

El API en Spring Boot soporta url de imágenes y archivos de imágenes, en el frontend solo se implementó el de url.

En Primefaces sólo se implementó por url.

El modelo ha sido entrenado con 5832 imágenes, las cuales 3010 son positivas y 2822 son negativas.

El servicio solo devuelve la probabilidad, no los ejes de donde se ha identificado.

El modelo tardó 4 minutos en entrenarse.

Anexos

[Repositorio de GitHub.](#)

Referencias

[What is Custom Vision?](#)

[Quickstart: How to build a classifier with Custom Vision](#)

[Test and retrain a model with Custom Vision Service](#)

[Use your model with the prediction API](#)