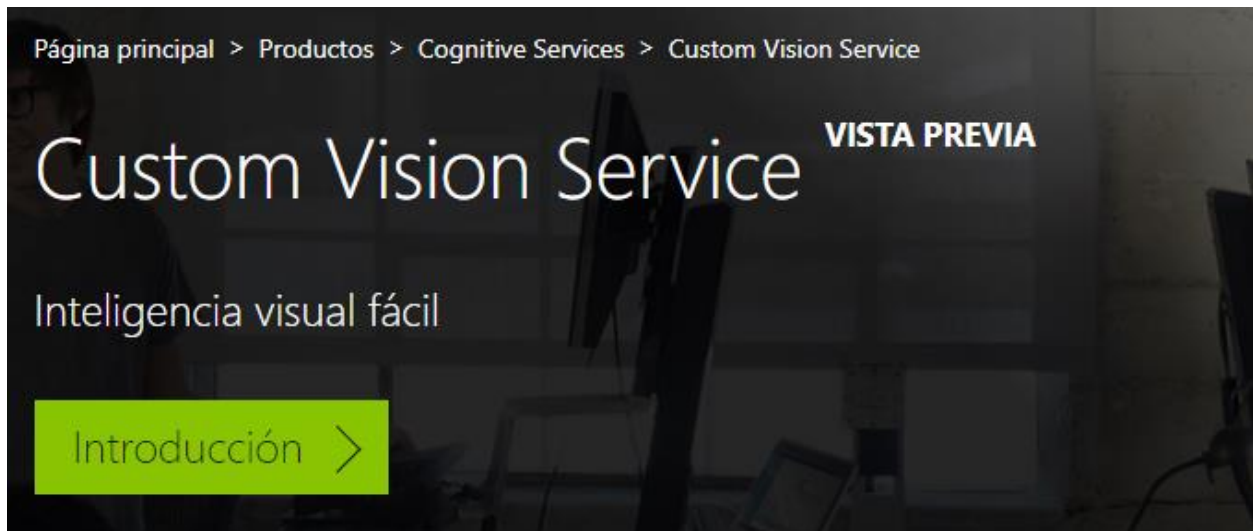
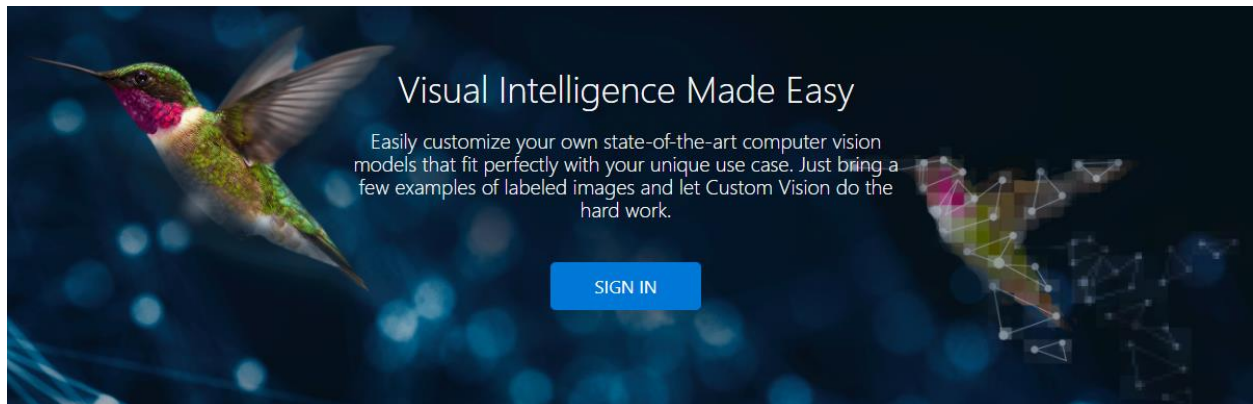


Práctica de Custom Vision API y Xamarin

Parte 1. Custom Vision API




Paso 1. Ingresa a <https://www.customvision.ai/> y da clic en **Sign In** para comenzar el registro al servicio.



Paso 2. Regístrate en el servicio con tu cuenta de Outlook/Hotmail/Live


Microsoft Custom Vision

Which account do you want to use?



Luis Antonio Beltran Prieto
luisantonio.beltranprieto@studentp...
Signed in

...

 Use another account

Paso 3. Proporciona los permisos solicitados dando clic en Aceptar.

Microsoft Custom Vision

App publisher website: microsoft.onmicrosoft.com

Microsoft Custom Vision needs permission to:

- View your basic profile ?
- View your email address ?
- Sign in as you ?

You're signed in as:

luisantonio.beltranprieto@studentpartner.com

[Show details](#)

Accept

Cancel

Paso 4. Acepta los términos del servicio.

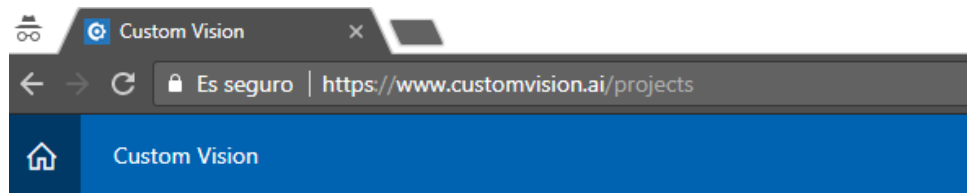
Terms of Service

Please note that Microsoft may retain copies of images uploaded for service improvement purposes. We won't publish your images or let other people use them.

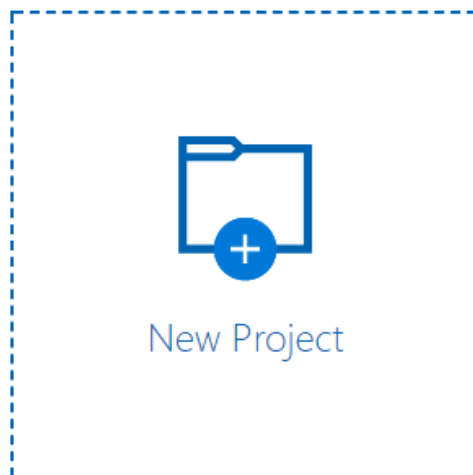
☒ I agree to the [Microsoft Cognitive Services Terms](#) and [Microsoft Privacy Statement](#)

I agree

Paso 5. Una vez configurado el entorno de trabajo, da clic en **New Project**.



My Projects



You have no projects yet, create one!

Paso 6. Ingresa los datos mostrados en la imagen siguiente (**Name, Description y Domain**) y da clic en **Create Project**.

New project

Name

Pokedex

Description

¿Quién es ese Pokémon?

Domains ⓘ

- ☒ General
- ☐ Food
- ☐ Landmarks
- ☐ Retail
- ☐ Adult
- ☐ General (compact)

Cancel

Create project

El clasificador que vamos a construir básicamente consiste en 3 pasos: **Subir imágenes, entrenar el proyecto y consumir el servicio.**

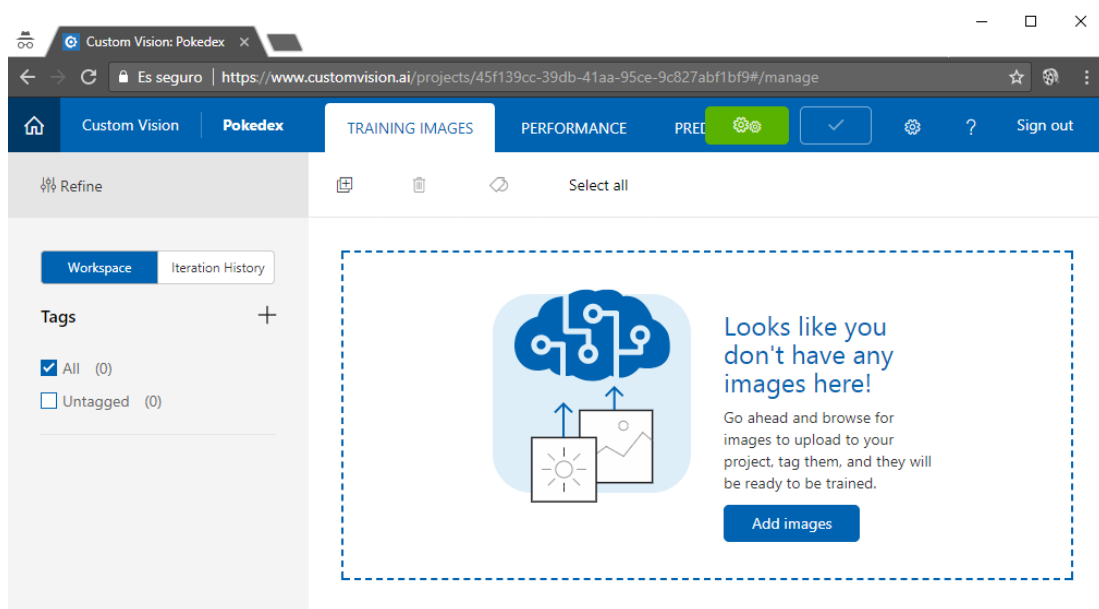


Improve your classifier!

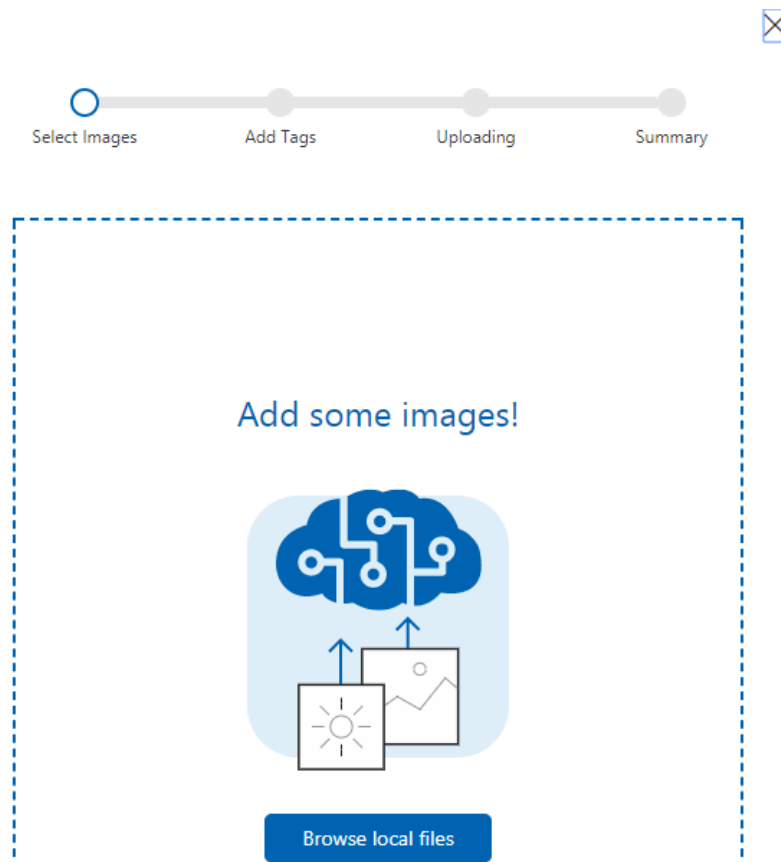
To get started follow these steps:

- 1 Upload your images**
First step is to add some images to your project and tag them. You'll need a minimum of 5 tagged images.
[Go](#)
- 2 Train your project**
Before you can setup your prediction endpoint, you will need to train your project first.
[Train](#)
- 3 Start using your prediction endpoint**
Once your project is trained, start using your prediction endpoint which will allow you to send images directly to your project for prediction

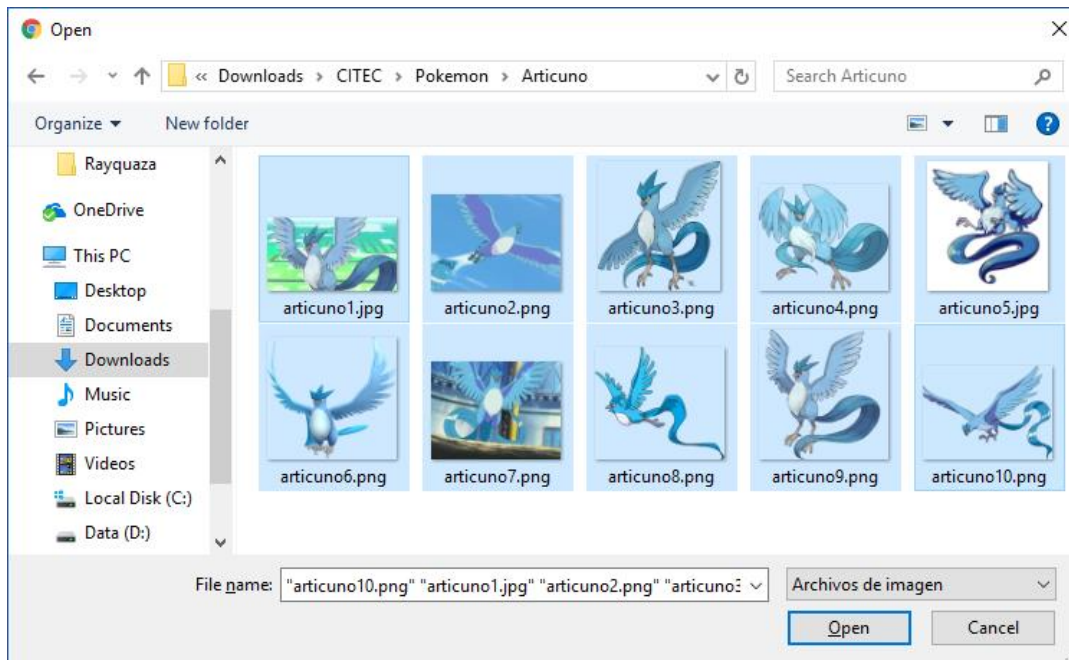
Paso 7. Da clic en el botón **Add images**.



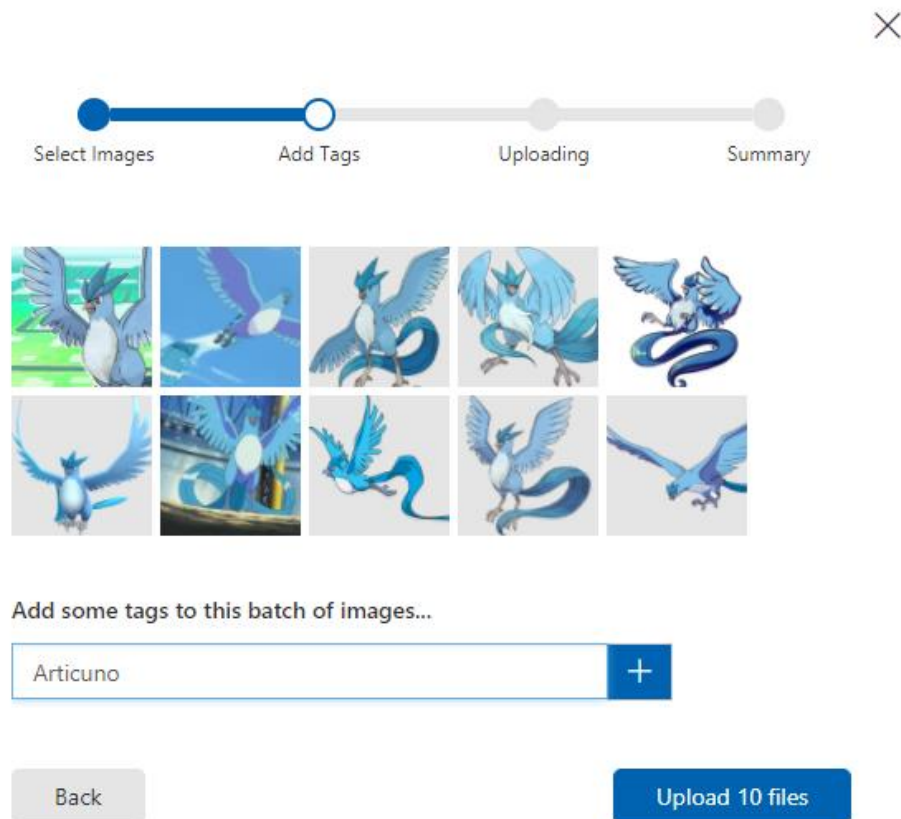
Paso 8. Da clic en el botón **Browse local files**.



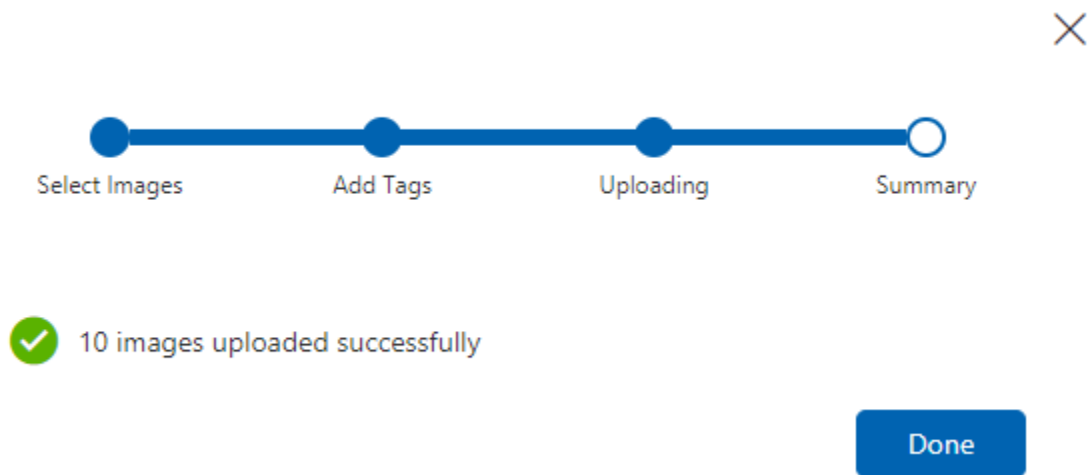
Paso 9. Selecciona las 10 imágenes proporcionadas en la carpeta **Articuno**.



Paso 10. Agrega la etiqueta (tag) **Articuno** y da clic en **Upload 10 files**.

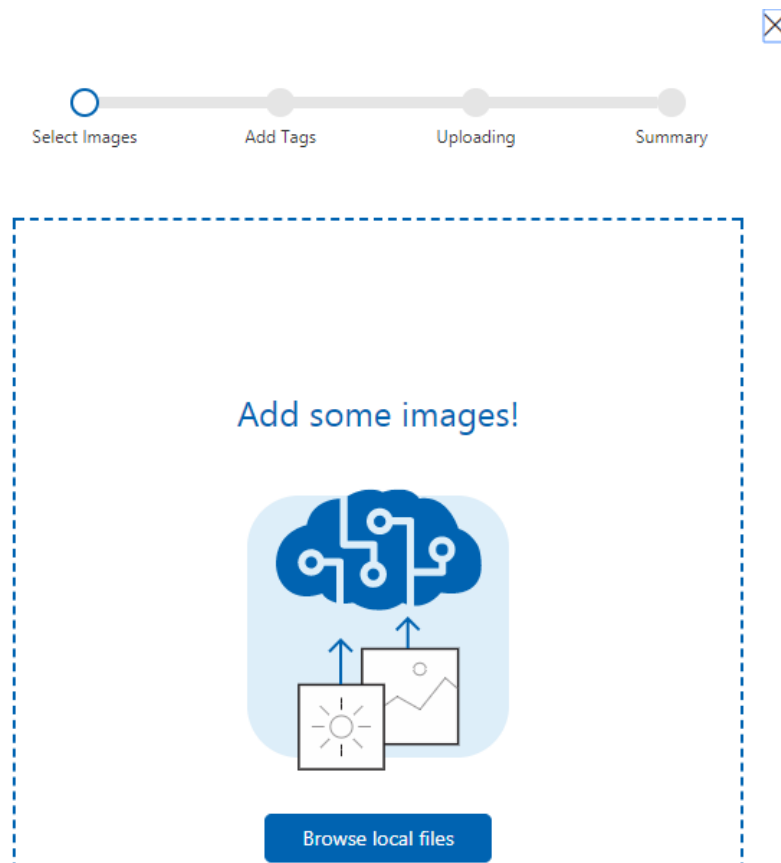


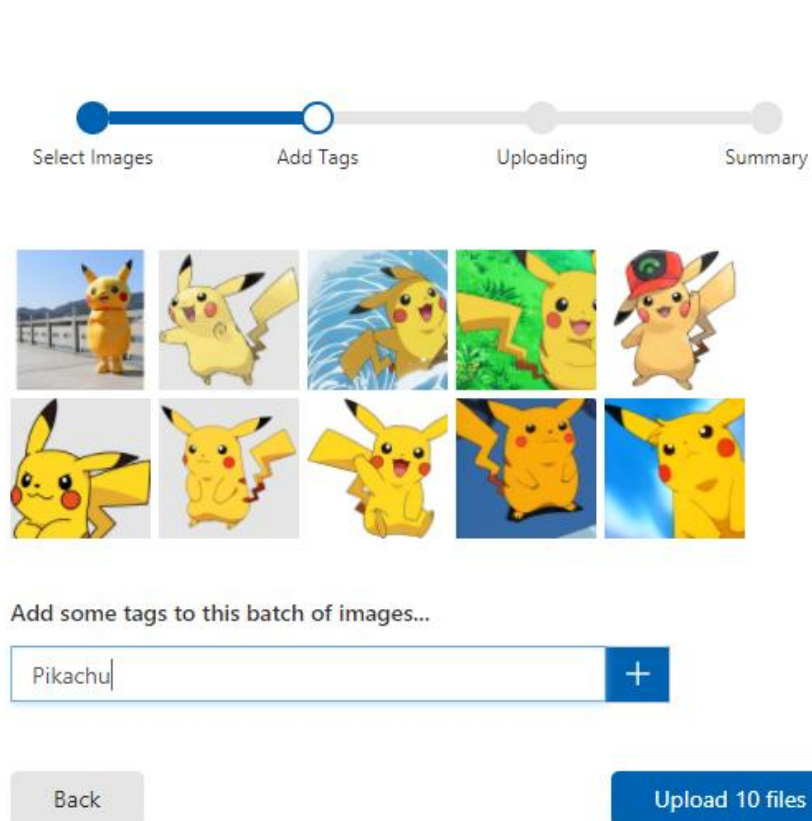
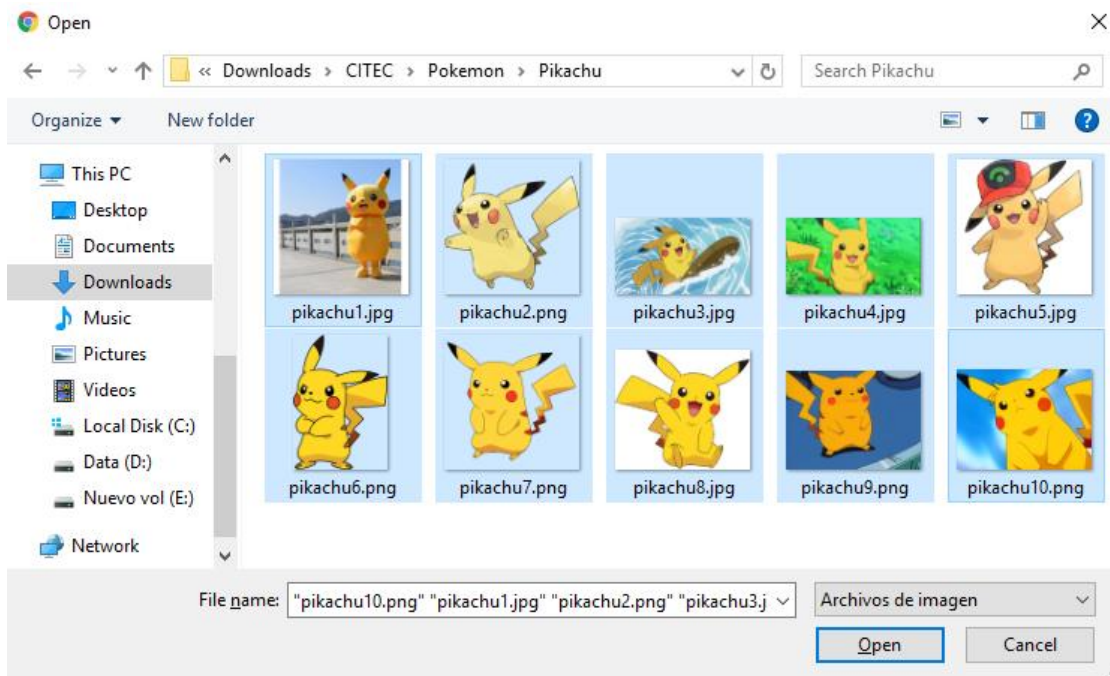
Una vez cargadas las imágenes en el Proyecto, recibirás el siguiente mensaje de éxito.




Paso 11. Repite el proceso (pasos 7 al 10) para las carpetas **Pikachu** y **Rayquaza** con sus etiquetas respectivas. Observa las siguientes imágenes:

Pikachu:








Select Images

Add Tags


Uploading

Summary

 10 images uploaded successfully

Done

Rayquaza:



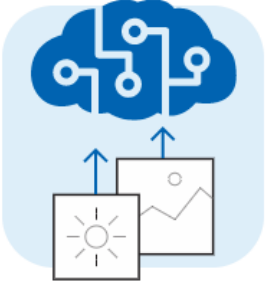
Select Images

Add Tags

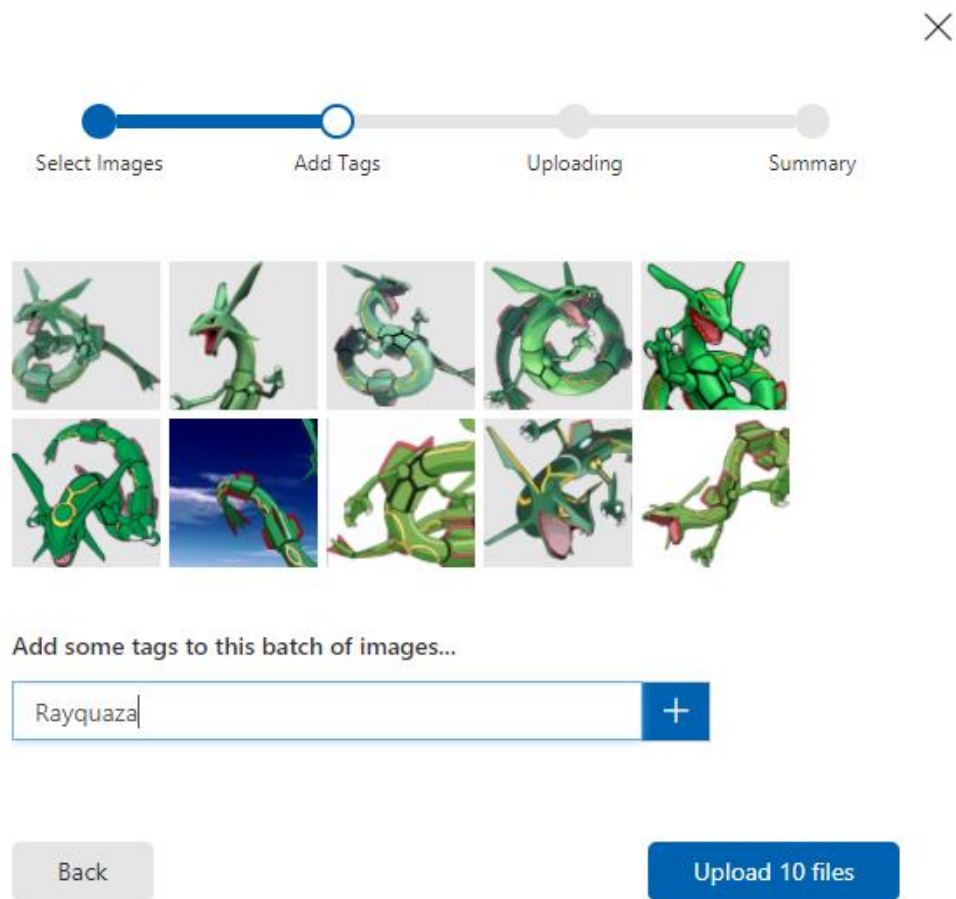
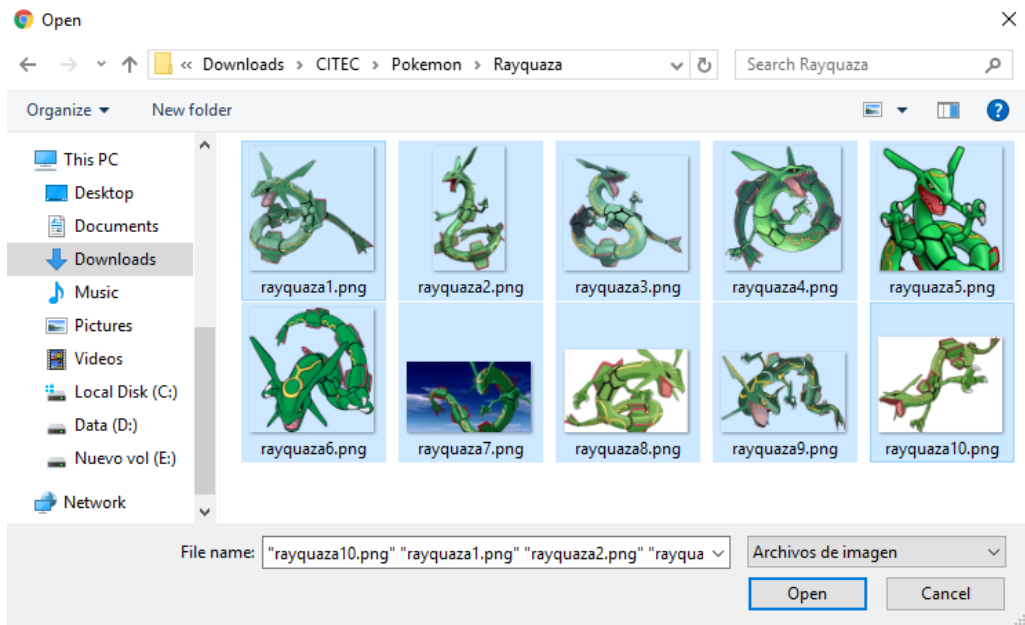
Uploading

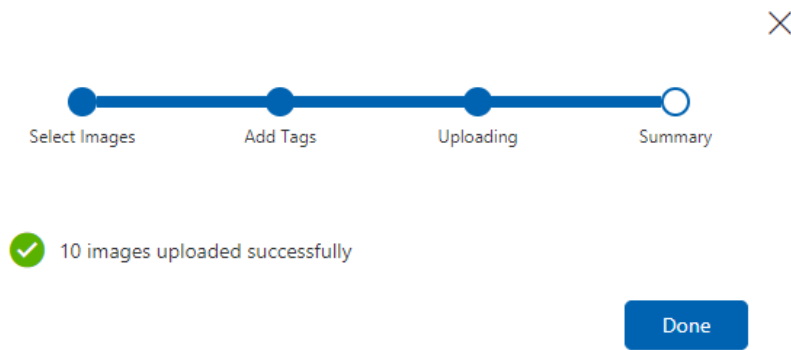
Summary

Add some images!



Browse local files





Una vez concluida la carga de imágenes, observa el detalle por etiqueta:

Workspace

Iteration History

Tags

+

☐ All (30)

☒ Articuno (10) ...

☒ Pikachu (10) ...

☒ Rayquaza (10) ...

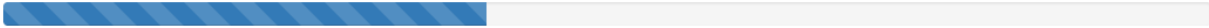
☐ Untagged (0)

Paso 12. Da clic en el botón **Train**.

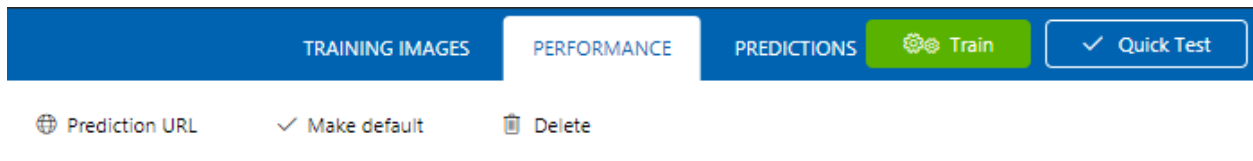


Observa el proceso de entrenamiento

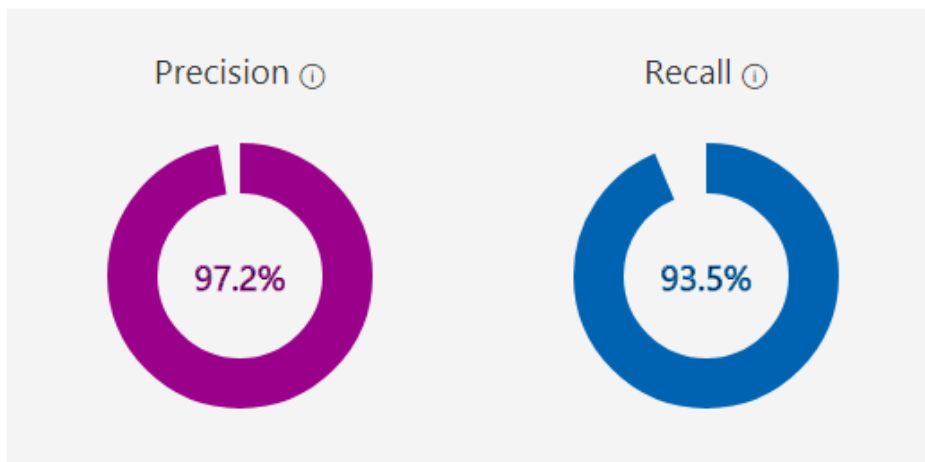
Iteration 1



Una vez finalizado, aparecerá la siguiente pantalla de resumen del proceso:



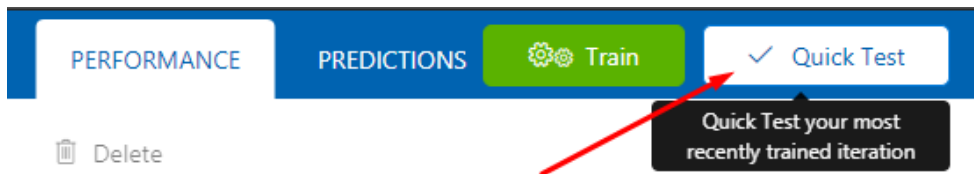
Iteration 1



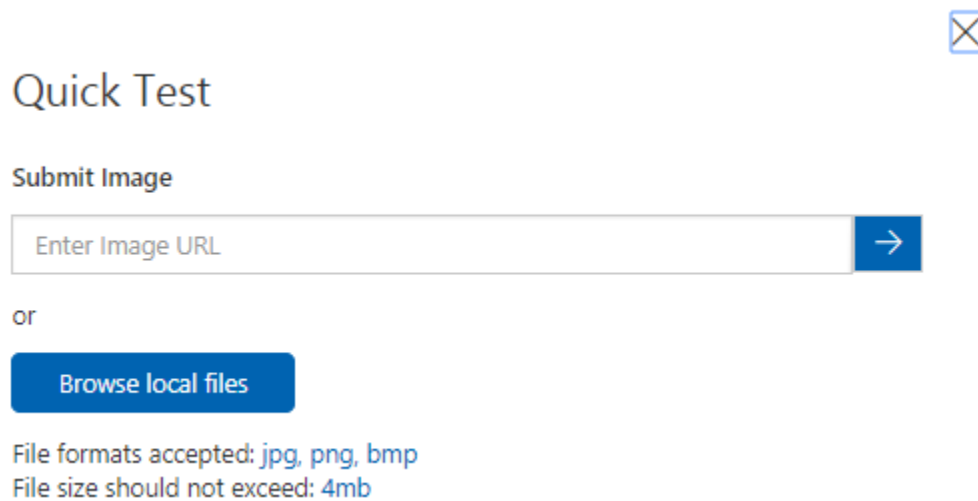
Performance Per Tag

Tag	Precision	Recall
Rayquaza	100.0%	100.0%
Articuno	93.3%	88.9%
Pikachu	100.0%	91.7%

Paso 13. Para verificar el modelo creado, da clic en el botón **Quick Test**.



Paso 14. Da clic en el botón **Browse local files**.



Paso 15. De la carpeta **Test**, selecciona **una a una** las imágenes correspondientes a los primeros 6 tests. Observa los resultados.

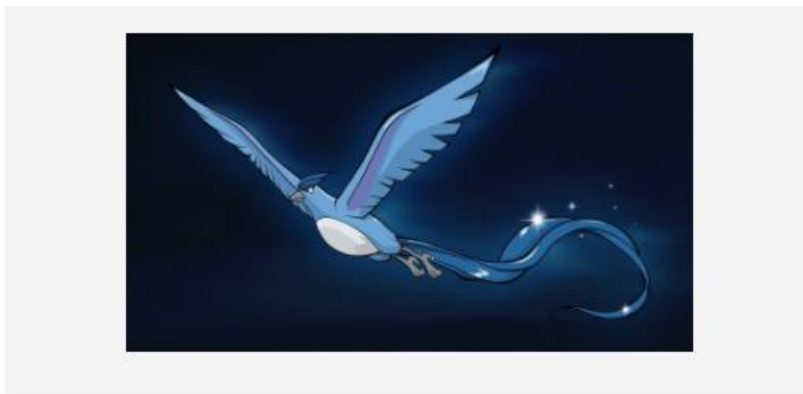
Test 1: OK



Results

Tag	Probability
Pikachu	99.6%
Rayquaza	0%
Articuno	0%

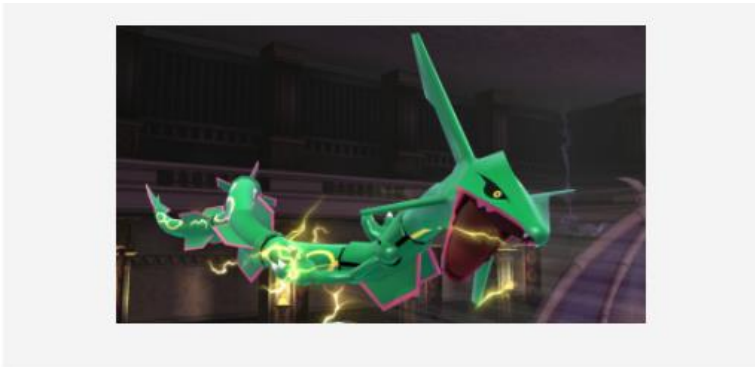
Test 2: OK



Results

Tag	Probability
Articuno	99.9%
Pikachu	0%
Rayquaza	0%

Test 3: OK



Results

Tag	Probability
Rayquaza	99.6%
Pikachu	0%
Articuno	0%

Test 4: OK



Results

Tag	Probability
Articuno	0%
Rayquaza	0%
Pikachu	0%

Test 5: Incorrecto



Results

Tag	Probability
Pikachu	98.2%
Articuno	0%
Rayquaza	0%

Test 6: Incorrecto



Results

Tag	Probability
Rayquaza	99.9%
Pikachu	0%
Articuno	0%

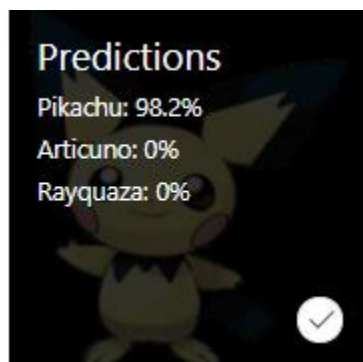
Paso 16. Vamos a introducir otra categoría para mejorar el modelo. Da clic en **Predictions**.



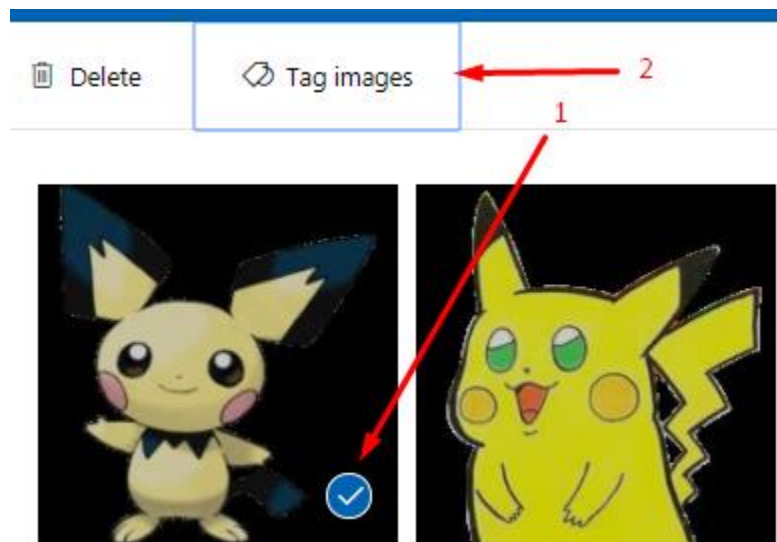
Observa que aparecen los tests realizados en el paso anterior.



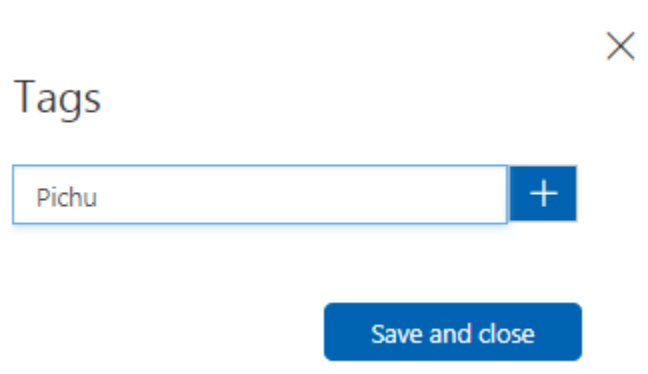
Si das clic en alguna imagen, aparece el resultado de la predicción



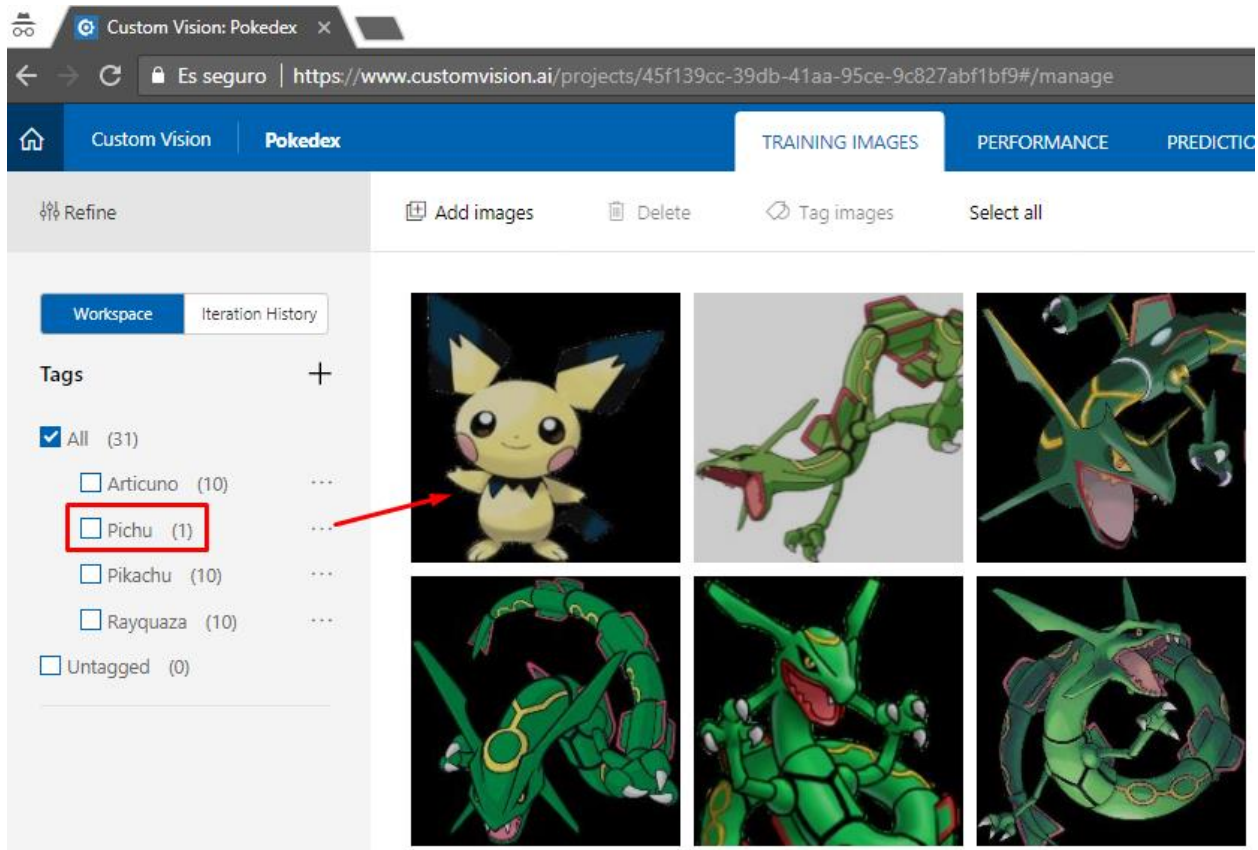
Paso 17. Selecciona la imagen mostrada en la figura y da clic en el botón **Tag images**.




Paso 18. Coloca el Tag **Pichu** y da clic en **Save and close**.

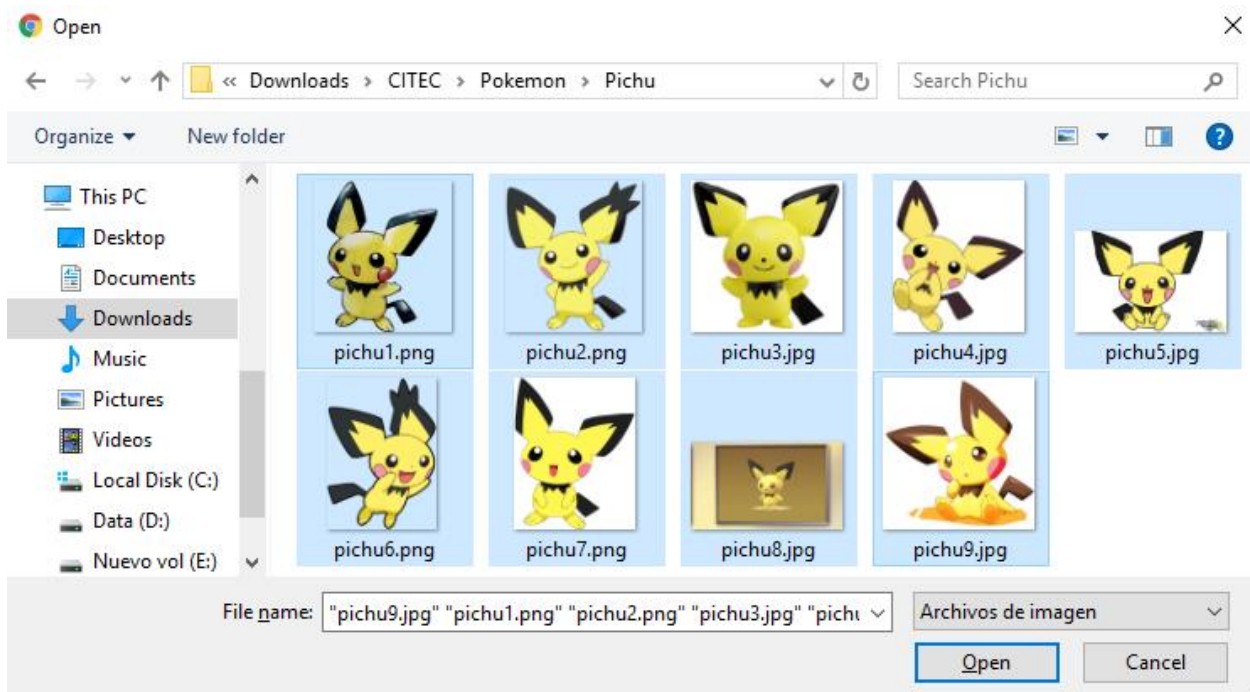
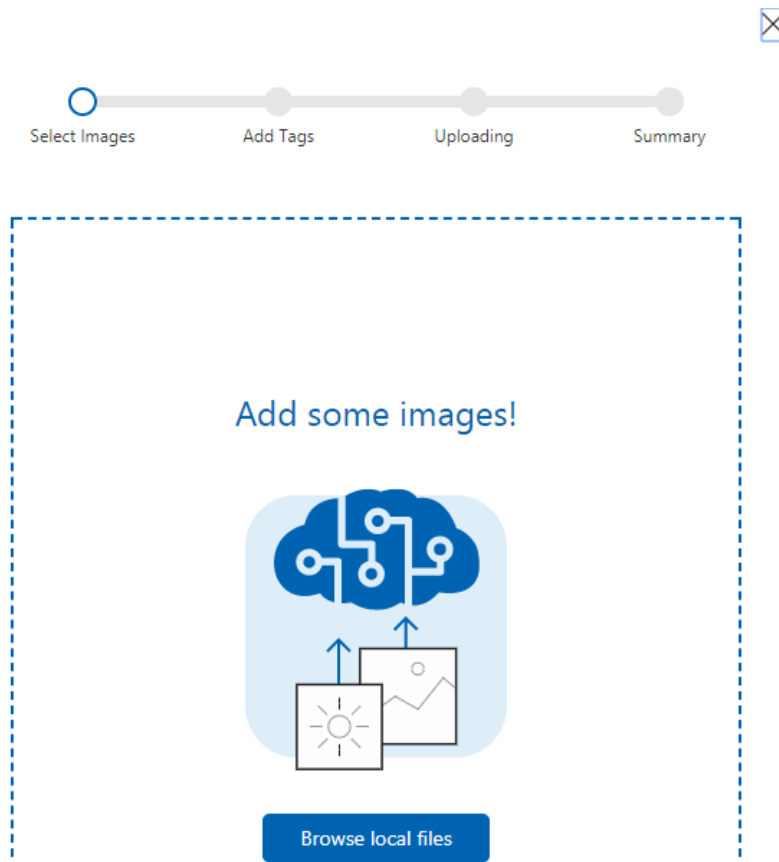


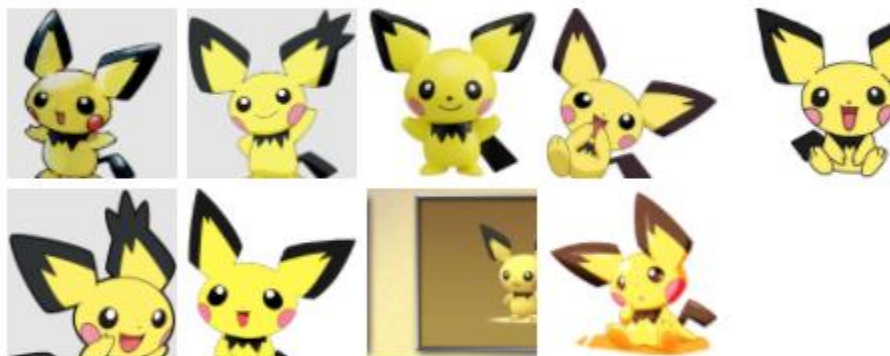
Observa que la imagen ha sido movida a la sección **Training images**.



Paso 19. Da clic en el botón **Add images** y repite el proceso que ya conoces para agregar las imágenes de la carpeta **Pichu** incluyendo dicha etiqueta. Observa las imágenes.

 Add images





Add some tags to this batch of images...

Pichu



Pichu

Back

Upload 9 files



✓ 9 images uploaded successfully

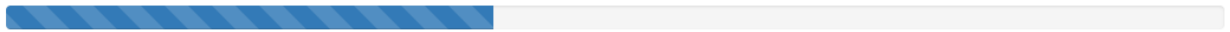
Done

Paso 20. Vuelve a dar clic en el botón **Train**.



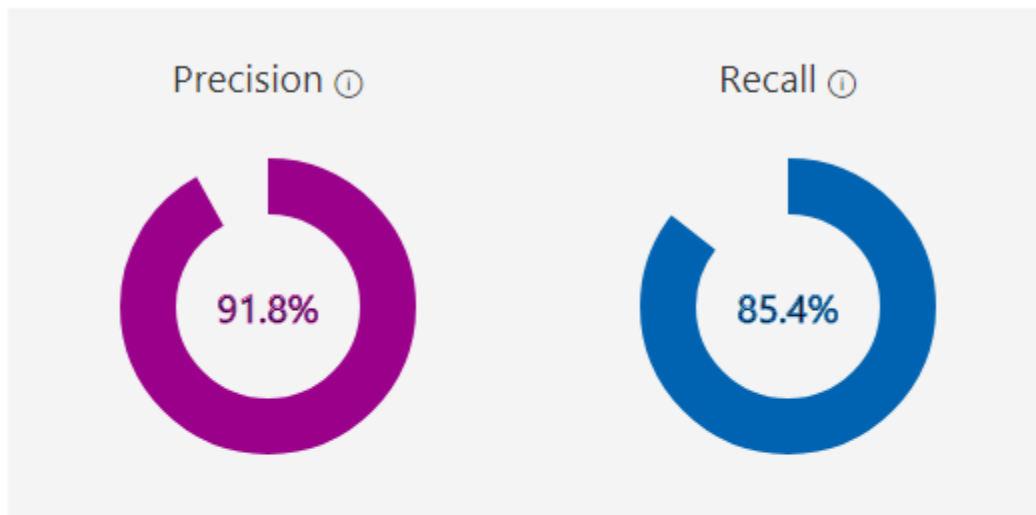
Observa que una nueva iteración (experiment) es creada.

Iteration 2



Al finalizar el entrenamiento, aparecerá nuevamente el resumen del proceso.

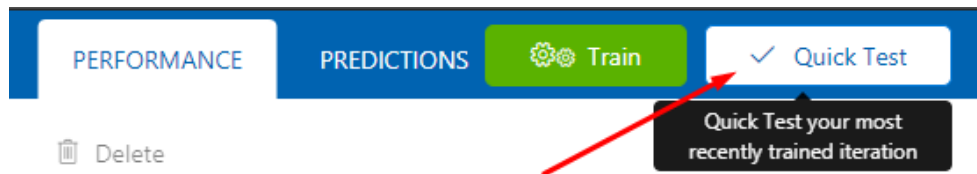
Iteration 2



Performance Per Tag

Tag	Precision	Recall
Rayquaza	100.0%	100.0%
Pichu	83.3%	91.7%
Articuno	91.7%	100.0%
Pikachu	100.0%	50.0%

Paso 21. Da clic de nuevo en **Quick Test**.



Paso 22. Utiliza el botón **Browse local files** y selecciona de la carpeta **Test** las imágenes 7 y 8. Observa los resultados.

A screenshot of the 'Quick Test' interface. At the top right is a close button (X in a square). The title 'Quick Test' is on the left. Below it is the section 'Submit Image'. This section contains a text input field with the placeholder 'Enter Image URL' and a blue button with a right-pointing arrow. Below the input field is the word 'or'. Underneath is a blue button labeled 'Browse local files'. At the bottom, there is text indicating 'File formats accepted: jpg, png, bmp' and 'File size should not exceed: 4mb'.

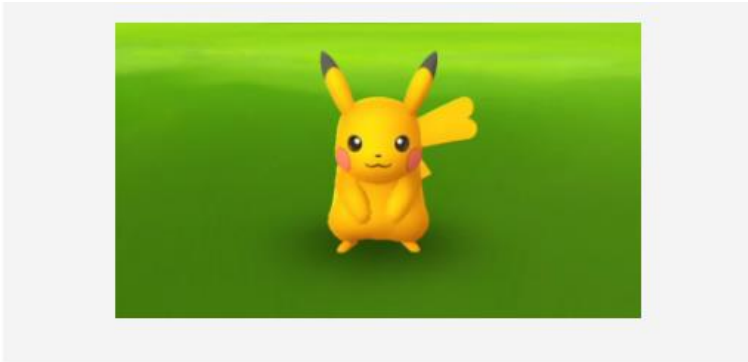
Test 7: OK



Results

Tag	Probability
Pichu	97.3%
Rayquaza	0%
Pikachu	0%
Articuno	0%

Test 8: OK

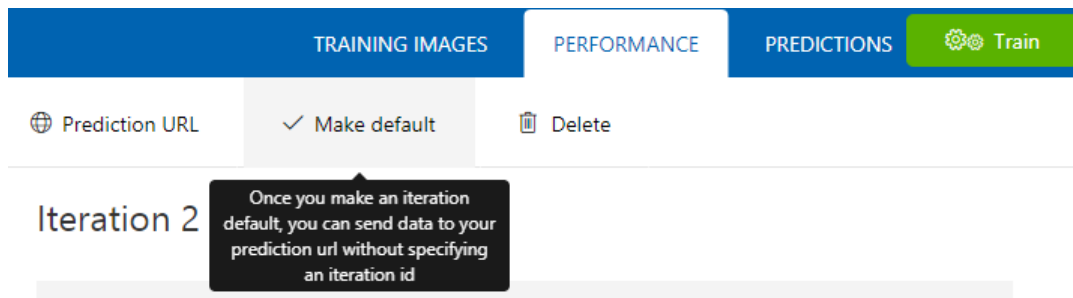


Results

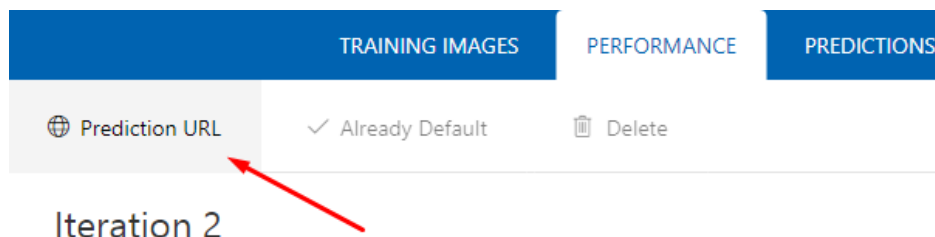
Tag	Probability
Pikachu	99.7%
Pichu	0%
Rayquaza	0%
Articuno	0%

Para poder consumir desde una aplicación el modelo generado, se necesita obtener la URL del servicio.

Paso 23. Da clic en el botón **Make default** de la pestaña **Performance**.



Paso 24. Da clic en el botón **Prediction URL**.



Paso 25. Copia la URL que aparece en la sección image file, así como la **Prediction Key** (los usaremos en la Parte 2) y toma nota de las consideraciones de cómo debe hacerse la petición URL (encabezados del request).

×

How to use the Prediction API

If you have an image URL:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/Predictic
```

Set **Prediction-Key** Header to : **1c4156479aff4627bf4749b8756e81b7**
Set **Content-Type** Header to : **application/json**
Set Body to : **{"Url": "<image url>"}**

If you have an image file:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/Predictic
```

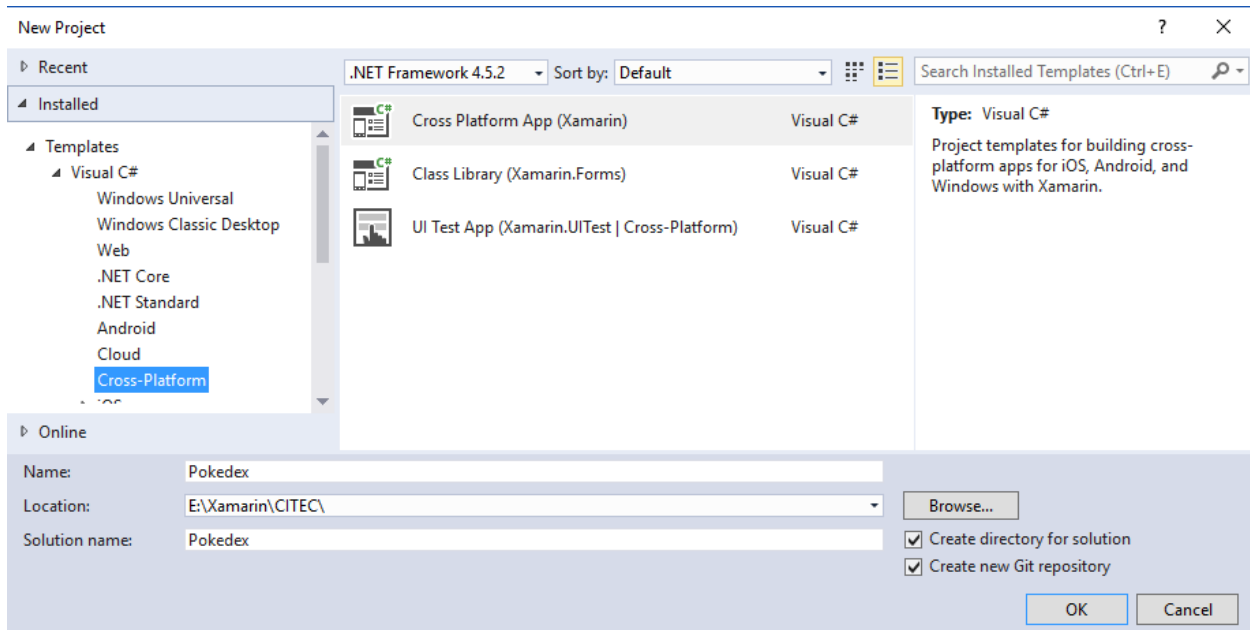
Set **Prediction-Key** Header to : **1c4156479aff4627bf4749b8756e81b7**
Set **Content-Type** Header to : **application/octet-stream**
Set Body to : **<image file>**

This iteration is marked as Default. If you mark another iteration as Default, the urls shown above will point to that iteration instead.

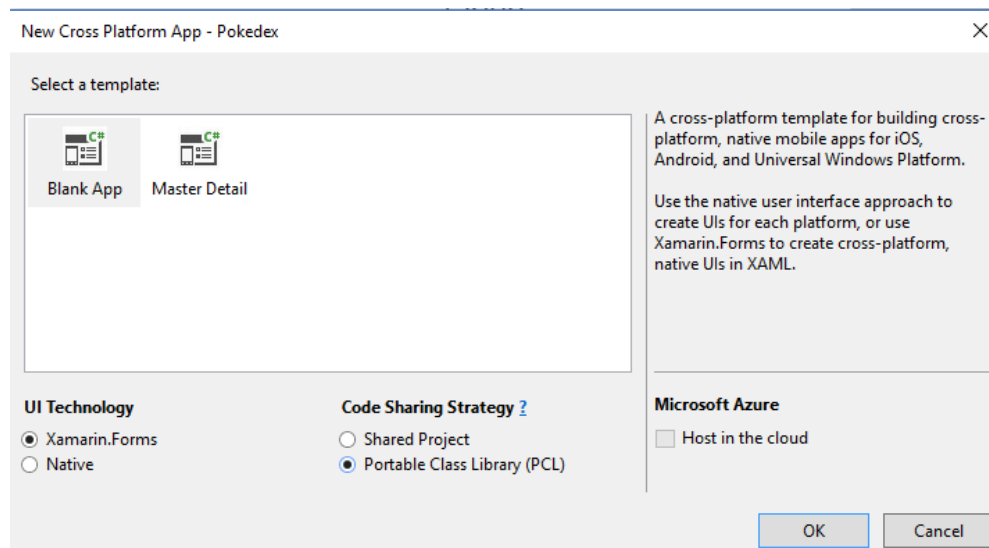
Got it!

Parte 2. Integración de Xamarin con Custom Vision API

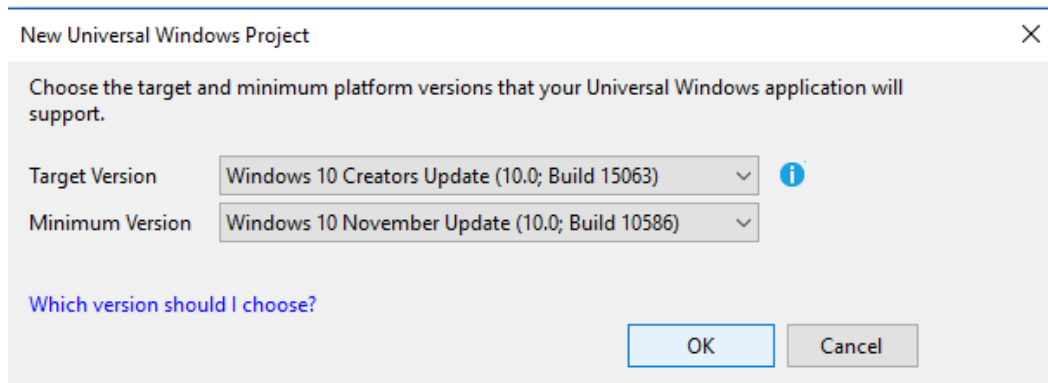
Paso 1. Abre **Visual Studio** y da clic en **Archivo → Nuevo Proyecto**. Selecciona la categoría **Cross-Platform → Cross Platform App (Xamarin)** y coloca el nombre de proyecto **Pokedex**.



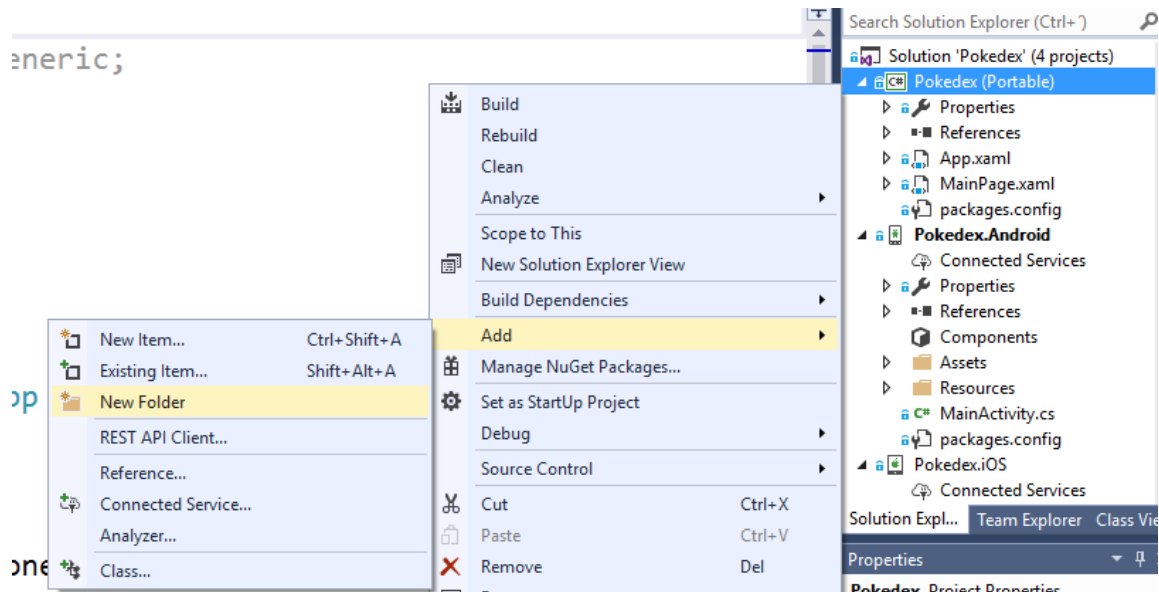
Paso 2. Selecciona **Portable Class Library (PCL)** para la estrategia de código compartido.



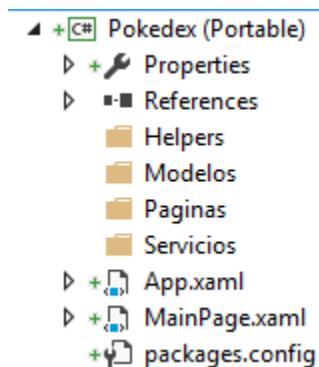
Paso 3. Si tienes instalado el SDK de Windows 10, simplemente da clic en Aceptar. En caso de que tengas una versión anterior, da clic en Cancelar.



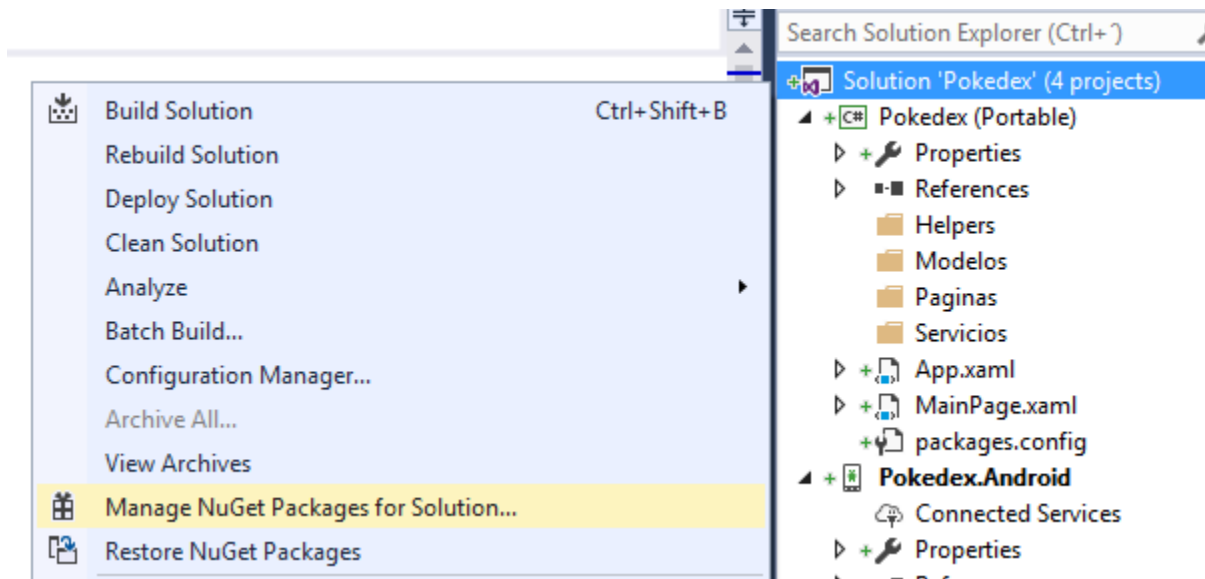
Paso 4. En el proyecto **Pokedex (Portable)**, da clic derecho y selecciona **Agregar → Nueva carpeta**.



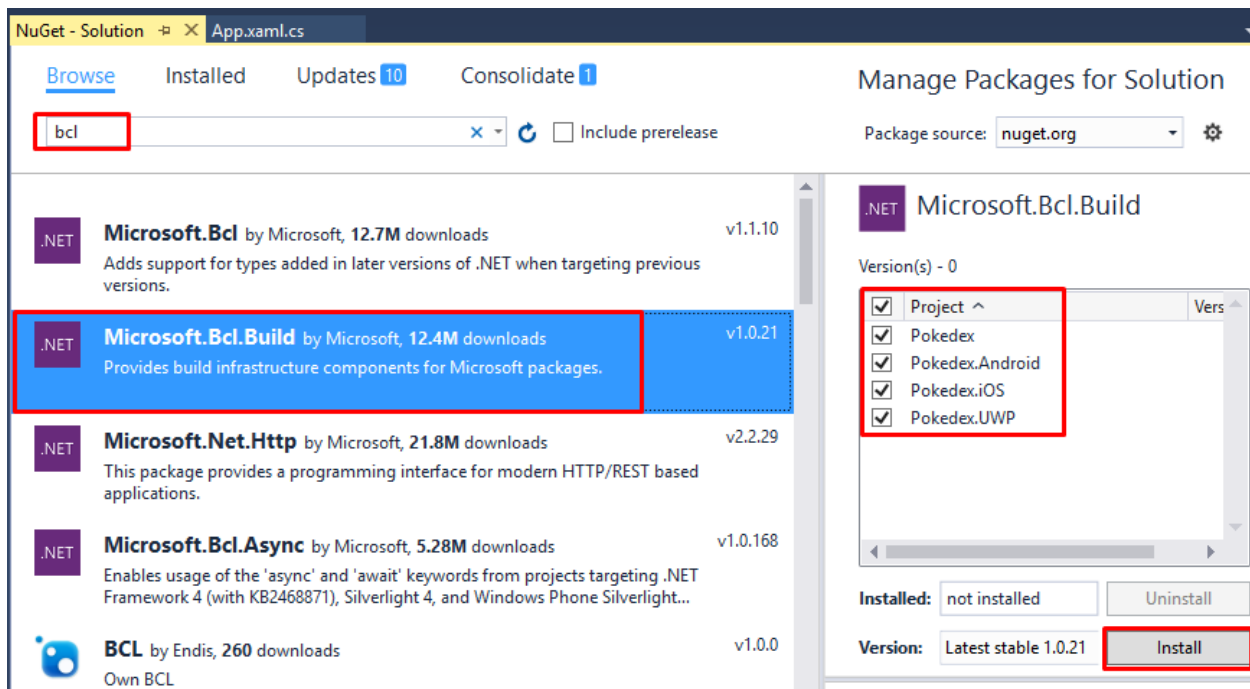
Paso 5. Agrega las siguientes carpetas al proyecto:



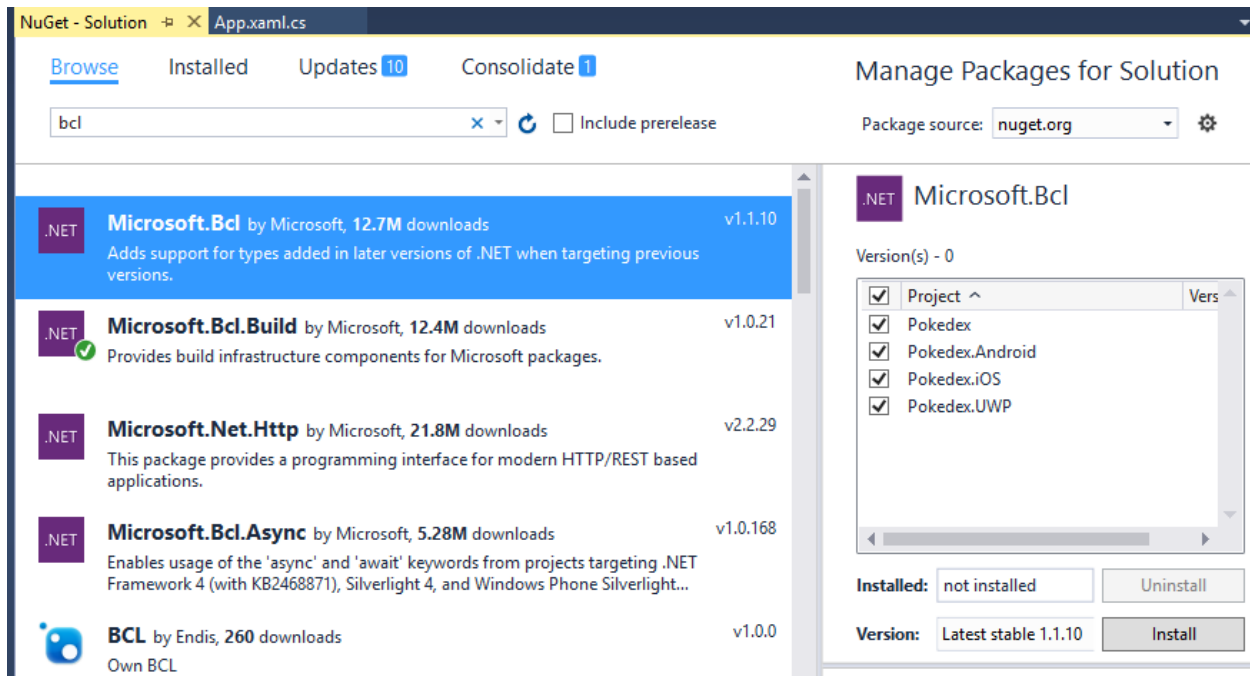
Paso 6. Da clic derecho en la solución **Pokedex** y elige **Administrar Paquetes Nuget para la solución**.



Paso 7. En la sección **Explorar**, escribe **bcl** e instala el paquete **Microsoft.Bcl.Build** para todos los proyectos (si te muestra un acuerdo de licencia de uso, acepta)



Paso 8. Ahora repite el proceso para instalar el paquete **Microsoft.Bcl**



The screenshot shows the NuGet Package Manager interface. The search bar contains 'bcl'. The results list includes:

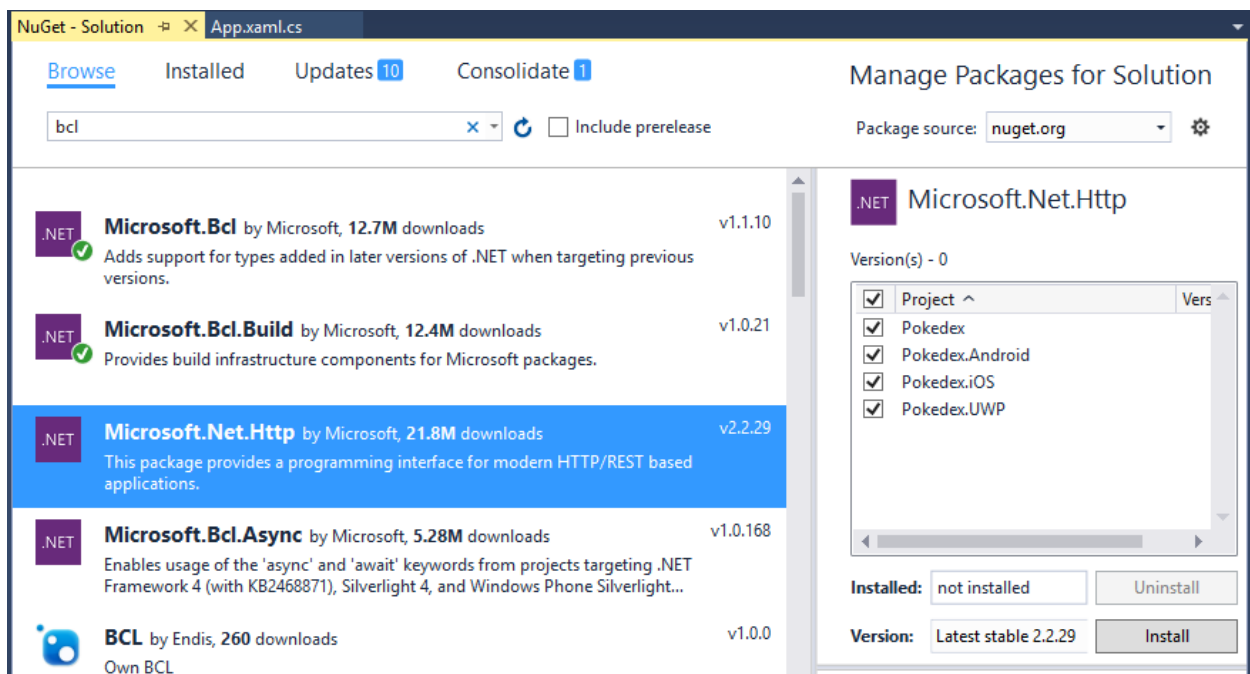
- Microsoft.Bcl** by Microsoft, 12.7M downloads, v1.1.10. Description: Adds support for types added in later versions of .NET when targeting previous versions.
- Microsoft.Bcl.Build** by Microsoft, 12.4M downloads, v1.0.21. Description: Provides build infrastructure components for Microsoft packages.
- Microsoft.Net.Http** by Microsoft, 21.8M downloads, v2.2.29. Description: This package provides a programming interface for modern HTTP/REST based applications.
- Microsoft.Bcl.Async** by Microsoft, 5.28M downloads, v1.0.168. Description: Enables usage of the 'async' and 'await' keywords from projects targeting .NET Framework 4 (with KB2468871), Silverlight 4, and Windows Phone Silverlight...
- BCL** by Endis, 260 downloads, v1.0.0. Description: Own BCL.

The right pane shows the details for **Microsoft.Bcl**. The version list is empty. The 'Project' list includes:

- ☒ Project ^
- ☒ Pokedex
- ☒ Pokedex.Android
- ☒ Pokedex.iOS
- ☒ Pokedex.UWP

The 'Installed' status is 'not installed' and the 'Version' is 'Latest stable 1.1.10'. Buttons for 'Uninstall' and 'Install' are visible.

Paso 9. Posteriormente, instala el paquete **Microsoft.Net.Http**

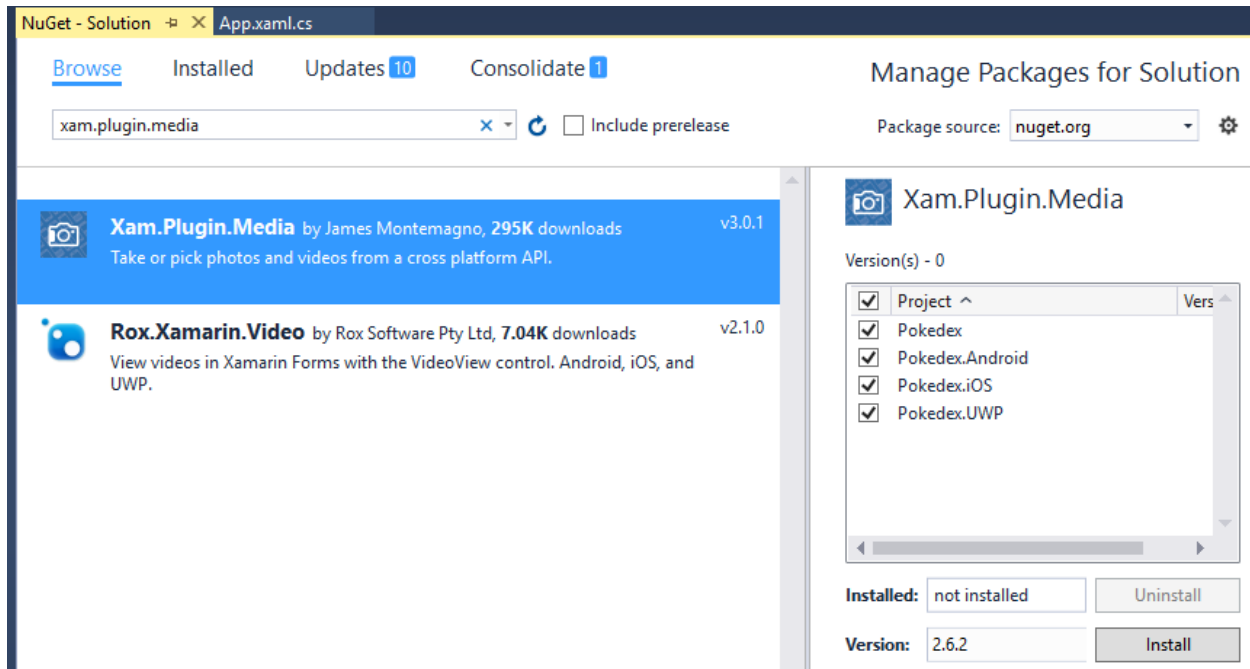


The screenshot shows the NuGet Package Manager interface. The search bar contains 'bcl'. The results list is the same as in the previous screenshot. The right pane shows the details for **Microsoft.Net.Http**. The version list is empty. The 'Project' list includes:

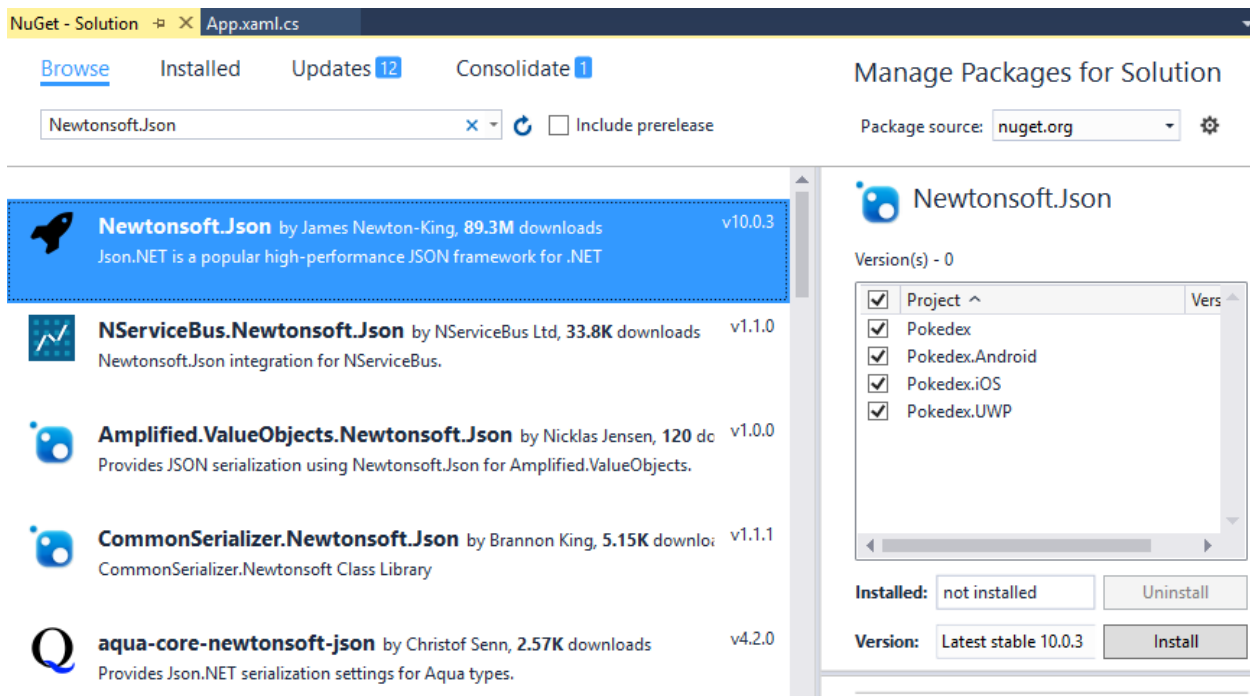
- ☒ Project ^
- ☒ Pokedex
- ☒ Pokedex.Android
- ☒ Pokedex.iOS
- ☒ Pokedex.UWP

The 'Installed' status is 'not installed' and the 'Version' is 'Latest stable 2.2.29'. Buttons for 'Uninstall' and 'Install' are visible.

Paso 10. Ahora busca el paquete **Xam.Plugin.Media**, selecciona todos los proyectos de la solución e instala la versión **2.6.2**



Paso 11. El último paquete por instalar (de momento) es **Newtonsoft.Json**. Búscalo en la lista y procede a instalarlo en todos los proyectos



Paso 12. Da clic derecho en la carpeta **Modelos**, selecciona **Agregar → Clase** y agrega la clase **Prediction**, con el código siguiente:

```
namespace Pokedex.Modelos
{
    public class Prediction
    {
        public string TagId { get; set; }
        public string Tag { get; set; }
        public double Probability { get; set; }
    }
}
```

Paso 13. En la misma carpeta, agrega la clase **CustomVisionResult** con el código siguiente

```
using System;
using System.Collections.Generic;

namespace Pokedex.Modelos
{
    public class CustomVisionResult
    {
        public string Id { get; set; }
        public string Project { get; set; }
        public string Iteration { get; set; }
        public DateTime Created { get; set; }
        public List<Prediction> Predictions { get; set; }
    }
}
```

Paso 14. Ahora, en la carpeta **Helpers**, agrega una clase llamada **Constantes**. Asigna los valores **PredictionKey** y **PredictionURL** de acuerdo a la información mostrada en el **paso 25** de la **Parte 1. Custom Vision API** de la práctica.

```
namespace Pokedex.Helpers
{
    public static class Constantes
    {
        public const string PredictionKey = "aqui-va-tu-llave";
        public const string PredictionURL = "aqui-va-tu-url";
    }
}
```

Paso 15. Agrega una nueva clase en la carpeta **Servicios** con el nombre **ServicioClasificador**. El código es el siguiente:

```
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Pokedex.Helpers;
using System;
using Pokedex.Modelos;
using Newtonsoft.Json;
using System.Linq;

namespace Pokedex.Servicios
{
    public static class ServicioClasificador
    {
        public async static Task<string> ClasificarImagen(MemoryStream stream)
        {
            try {
                var url = Constantes.PredictionURL;

                using (var cliente = new HttpClient())
                {
                    cliente.DefaultRequestHeaders.Add("Prediction-Key",
Constantes.PredictionKey);

                    using (var content = new ByteArrayContent(stream.ToArray()))
                    {
                        content.Headers.ContentType = new
MediaTypeHeaderValue("application/octet-stream");
                        var post = await cliente.PostAsync(url, content);
                        var resultado = await post.Content.ReadAsStringAsync();
                        var cv =
JsonConvert.DeserializeObject<CustomVisionResult>(resultado);

                        if (cv.Predictions.Count > 0)
                        {
                            var prediccion = ObtenerPrediccion(cv);
                            return prediccion.Probability > 0.5 ? prediccion.Tag :
"Pokémon no identificado en la base de datos";
                        }
                        else
                            return "Error al realizar la predicción. Intenta de nuevo.";
                    }
                }
            }
            catch (Exception ex) { return "Ocurrió una excepción: " + ex.Message; }
        }

        static Prediction ObtenerPrediccion(CustomVisionResult cv)
        {
            return cv.Predictions.OrderByDescending(x => x.Probability).Take(1).First();
        }
    }
}
```

Paso 16. Crea una nueva clase llamada **ServicioImágenes** en la carpeta **Servicios**. El código es el siguiente:

```
using Plugin.Media;
using Plugin.Media.Abstractions;
using System.Threading.Tasks;

namespace Pokedex.Servicios
{
    public class ServicioImágenes
    {
        public static async Task<MediaFile> TomarFoto(bool usarCamara)
        {
            await CrossMedia.Current.Initialize();

            if (usarCamara)
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakePhotoSupported)
                {
                    return null;
                }
            }

            var file = usarCamara
                ? await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions
                {
                    Directory = "Clasificador",
                    Name = "test.jpg"
                })
                : await CrossMedia.Current.PickPhotoAsync();

            return file;
        }
    }
}
```

Paso 17. Ahora, en la carpeta **Paginas** da clic derecho y selecciona **Agregar → Nuevo elemento**. De la lista, selecciona la categoría **Xamarin.Forms** y elige **Content Page**. El nombre de esta página será **PaginaPokedex**.

PaginaPokedex.xaml (diseño)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Pokedex.Paginas.PaginaPokedex"
             BackgroundColor="#DC0A2D">
    <ContentPage.Content>
        <StackLayout Padding="10">
            <StackLayout Orientation="Horizontal">
                <Button x:Name="btnCamara" x:Id="btnCamara" Text="Cámara"
                    Clicked="btnCamara_Clicked" Margin="10" WidthRequest="100" BackgroundColor="#000000"
                    TextColor="White"/>
                <Button x:Name="btnGaleria" x:Id="btnGaleria" Text="Galería"
                    Clicked="btnGaleria_Clicked" Margin="10" WidthRequest="100" BackgroundColor="#000000"
                    TextColor="White"/>
            </StackLayout>
            <Image x:Name="imgFoto" WidthRequest="150" HeightRequest="150"
                Aspect="AspectFit" HorizontalOptions="CenterAndExpand" Margin="5"/>
            <Button x:Name="btnClasificar" Text="¿Quién es ese Pokémon?"
                Clicked="btnClasificar_Clicked" HorizontalOptions="CenterAndExpand" Margin="5"
                BackgroundColor="#000000" TextColor="White"/>
            <Label x:Name="lblResultado" Text="---" TextColor="#BAF73C" Margin="5"
                FontSize="Large"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

PaginaClasificador (code-behind)

```
using System;
using System.Collections.Generic;
using Xamarin.Forms;
using System.IO;
using Pokedex.Servicios;
using Xamarin.Forms.Xaml;
using System.Threading.Tasks;

namespace Pokedex.Paginas
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PaginaPokedex : ContentPage
    {
        static MemoryStream streamCopy;

        public PaginaPokedex()
        {
            InitializeComponent();
        }

        async Task ObtenerImagen(bool camara)
        {
            var archivo = await ServicioImagenes.TomarFoto(camara);
            lblResultado.Text = "---";
            imgFoto.Source = ImageSource.FromStream(() => {
                var stream = archivo.GetStream();
                streamCopy = new MemoryStream();
                stream.CopyTo(streamCopy);
                stream.Seek(0, SeekOrigin.Begin);
                archivo.Dispose();
                return stream;
            });
        }

        private async void btnCamara_Clicked(object sender, EventArgs e)
        {
            await ObtenerImagen(true);
        }

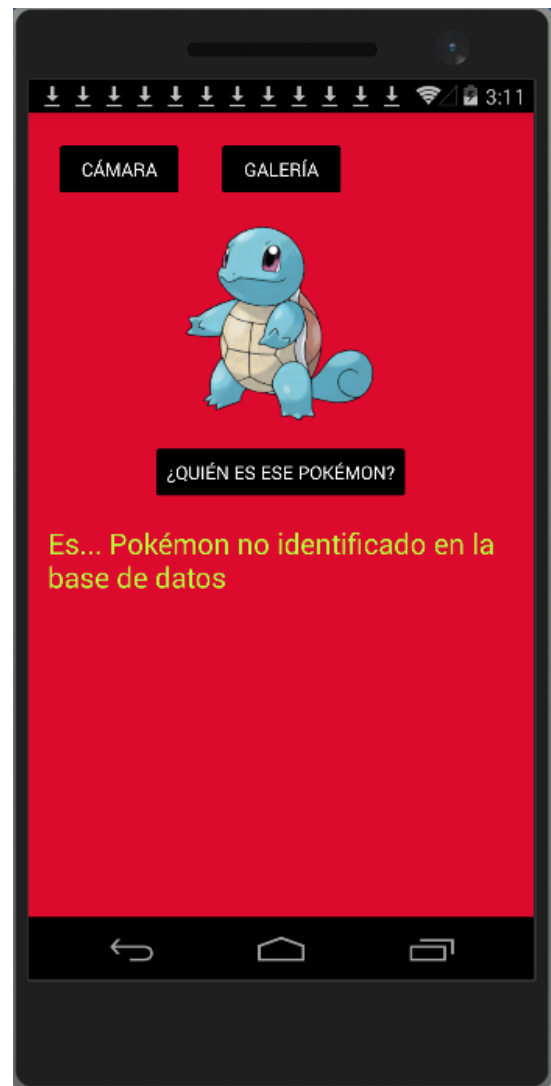
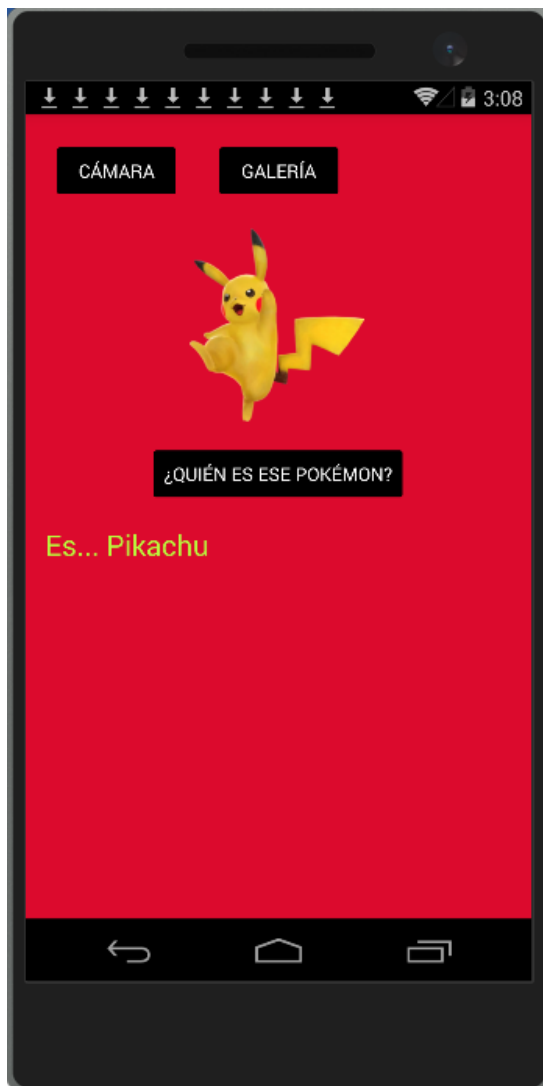
        private async void btnGaleria_Clicked(object sender, EventArgs e)
        {
            await ObtenerImagen(false);
        }

        private async void btnClasificar_Clicked(object sender, EventArgs e)
        {
            if (streamCopy != null) {
                streamCopy.Seek(0, SeekOrigin.Begin);
                var resultado = await ServicioClasificador.ClasificarImagen(streamCopy);
                lblResultado.Text = $"Es... {resultado}";
            }
            else
                lblResultado.Text = "---No has seleccionado una imagen---";
        }
    }
}
```

Paso 18. Modifica el constructor de la clase **App** para que la página inicial sea **PaginaPokedex**:

```
public App()  
{  
    InitializeComponent();  
  
    MainPage = new Pokedex.Paginas.PaginaPokedex();  
}
```

Paso 19. Compila y ejecuta la aplicación, verificando el correcto funcionamiento de la misma



Parte 3. Mejorando la aplicación:

- Obteniendo información adicional del Pokémon
- Reproduciendo sonidos