



“SISTEMA PARA EL MONITOREO, DETECCIÓN Y ALERTA DE
SOMNOLENCIA DEL CONDUCTOR MEDIANTE VISIÓN ARTIFICIAL,
COMUNICACIÓN INALÁMBRICA Y GEOLOCALIZACIÓN”

Primer Reporte Parcial

Lista de actividades

- Definir rutas del frontend
- Diseño de rutas del backend
- Conexión Backend con Mongo DB
- Sistema de acceso con credenciales
- Creación de la base de datos no relacional
- Investigación de modelos de Redes Neuronales Convolucionales
- Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos

Autores:

Alan Eduardo Gamboa Del
Ángel
Maite Paulette Díaz Martínez

Asesores:

M.en C. Niels Henrik Navarrete
Manzanilla
Dr. Rodolfo Vera Amaro

13 de Marzo 2023

Índice

1. Definir rutas del frontend	4
1.1. Objetivo	4
1.2. Descripción	4
1.3. Resultados	7
2. Definir rutas del backend	8
2.1. Objetivo	8
2.2. Descripción	8
2.3. Resultados	10
3. Creación de la base de datos No Relacional	12
3.1. Objetivo	13
3.2. Descripción	13
3.3. Resultados	13
4. Conexión Backend con Mongo DB	14
4.1. Objetivo	14
4.2. Descripción	14
4.3. Resultados	14
5. Sistema de acceso con credenciales	16
5.1. Objetivo	16
5.2. Descripción	16
5.3. Resultados	18
6. Investigación de modelos de Redes Neuronales Convolucionales	20
6.1. Objetivo	20
6.2. Descripción	20
6.3. Resultados	23
7. Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos	24
7.1. Objetivo	24
7.2. Descripción	24
7.3. Resultados	28
8. Conclusiones	31
9. Bibliografía	32

Índice de figuras

1.	Creación proyecto de react	4
2.	Instalación React Router DOM	4
3.	Función RequireAuth	5
4.	Componente Login	5
5.	Ruta protegida del componente Home	6
6.	Directorio de rutas	6
7.	Página de Login	7
8.	Instalación Amplify CLI	8
9.	Configuración del proyecto	8
10.	Implementación del SDK de Amplify	8
11.	Creación de aplicación de Express	9
12.	Amplify add api	9
13.	Configuración de parámetros de GraphQL	9
14.	Definición de Schemas para el manejo de datos	10
15.	Generación de Endpoint de GraphQL	11
16.	Tablas en DynamoDB	12
17.	Consola de AWS	13
18.	Tablas generadas mediante los schemas definidos	13
19.	Funcionamiento de Appsync	14
20.	Generación de endpoint de GraphQL	15
21.	Funcionamiento de Amazon Cognito	16
22.	Implementación de Amazon Cognito en el proyecto	17
23.	Menu de configuración de Amazon Cognito	17
24.	Implementación completada	18
25.	Grupos de usuario	18
26.	Grupos de usuario	19
27.	Grupos de usuario	19
28.	Documentación de Tensorflow referente a las aplicaciones con keras[5].	20
29.	Modelos de aprendizaje profundo disponibles en Keras.[6]	22
30.	Entorno de trabajo Google Colab.	24
31.	Dataset a trabajar desde Google Colab.	25
32.	Librerías utilizadas y repositorio en drive del dataset.	25
33.	Preparación de los datos de entrada	26
34.	Exportación del modelo MobileNet	26
35.	Modificación del modelo Mobilnet	27
36.	Entrenamiento del modelo MobileNet	27
37.	Entrenamiento del modelo EfficientNet	27
38.	Gráfica de precisión en cada epoca del modelo MobileNet.	28
39.	Gráfica de precisión en cada epoca del modelo EfficientNet.	28
40.	Gráfica de precisión en cada epoca del modelo MobileNet.	28
41.	Gráfica de precisión en cada epoca del modelo MobileNet.	29
42.	Gráfica de precisión en cada epoca del modelo EfficientNet.	29
43.	Gráfica de precisión en cada epoca del modelo EfficientNet.	30

Índice de tablas

1. Definir rutas del frontend

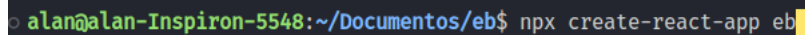
1.1. Objetivo

Definir e implementar las rutas que tendrá la aplicación, así como si serán públicas o privadas y la información que se desplegará en cada una de las mismas.

1.2. Descripción

Rutas públicas vs rutas protegidas

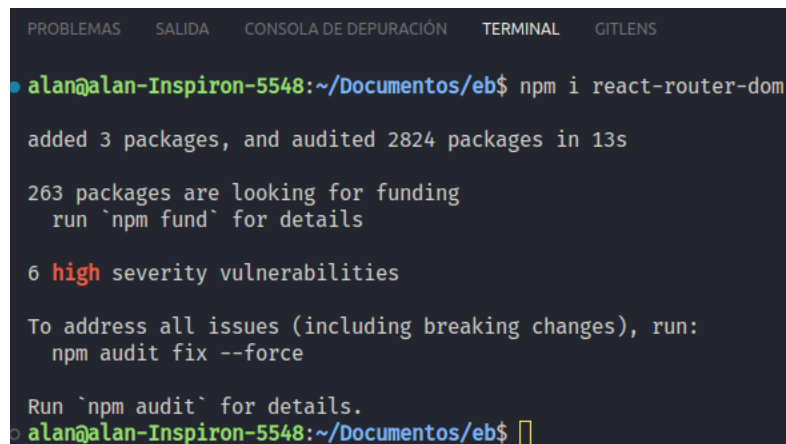
Cuando se habla de una ruta protegida en React, se refiere a programar un bloqueo en ciertas rutas a la cual se le restringe el acceso al usuario. Esto comunmente se realiza para la validación de inicio de sesión de usuarios. Sí el usuario no tiene una sesión iniciada, no podrá acceder a las rutas protegidas de la aplicación. Por otro lado, las rutas públicas son todas aquellas las cuales no requieren contar con una sesión iniciada, y pueden ser accesadas por cualquier tipo de usuario[1]. Como primer paso, se necesita crear un proyecto de React utilizando el siguiente comando:



```
alan@alan-Inspiron-5548:~/Documentos/eb$ npx create-react-app eb
```

Figura 1: Creación proyecto de react

Posteriormente, se realiza la instalación del moudelo *React Router Dom* utilizando el siguiente comando:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS

alan@alan-Inspiron-5548:~/Documentos/eb$ npm i react-router-dom

added 3 packages, and audited 2824 packages in 13s

263 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
alan@alan-Inspiron-5548:~/Documentos/eb$
```

Figura 2: Instalación React Router DOM

Utilizando la librería *useAuthenticator* ofrecida por el paquete de React Dom, crearemos un archivo de nombre *RequireAuth.js* el cuál contendrá una función la cual se encargará de validar si existe una sesión iniciada previamente.

```
src > RequireAuth.js > RequireAuth
1  import { useLocation, Navigate } from 'react-router-dom';
2  import { useAuthenticator } from '@aws-amplify/ui-react';
3
4  export function RequireAuth({ children }) {
5    const location = useLocation();
6    const { route } = useAuthenticator((context) => [context.route]);
7    if (route !== 'authenticated') {
8      return <Navigate to="/login" state={{ from: location }} replace />;
9    }
10   return children;
11 }
```

Figura 3: Función RequireAuth

Posteriormente, se necesita contar con una página de Login, la cuál permitirá validar que se encuentre una sesión iniciada por parte del usuario, para así poder acceder a las rutas protegidas.

```
src > Login.js > ...
1  import { useEffect } from "react";
2  import { Authenticator, useAuthenticator, View } from '@aws-amplify/ui-react';
3  import '@aws-amplify/ui-react/styles.css';
4  import { useNavigate, useLocation } from 'react-router';
5
6  export function Login() {
7    const { route } = useAuthenticator((context) => [context.route]);
8    const location = useLocation();
9    const navigate = useNavigate();
10   let from = location.state?.from?.pathname // '/';
11   useEffect(() => {
12     if (route === 'authenticated') {
13       navigate(from, { replace: true });
14     }
15   }, [route, navigate, from]);
16   return (
17     <View className="auth-wrapper">
18       <Authenticator></Authenticator>
19     </View>
20   );
21 }
```

Figura 4: Componente Login

Para indicar a React, que se desea implementar una ruta protegida, se necesita ir al componente de dicha ruta e ingresar el siguiente código:

```

src > Home.js > Home
1  import { useAuthenticator, Authenticator } from "@aws-amplify/ui-react";
2
3  export default function Home() {
4    const { route } = useAuthenticator((context) => [context.route]);
5    const message =
6      route === 'authenticated' ? 'FIRST PROTECTED ROUTE!' : 'Loading ...';
7    return (
8      <Authenticator>
9        ({ { signOut, user } }) => (
10          <main>
11            <h1>Bienvenido a Home</h1>
12
13            <button onClick={signOut}>Sign out</button>
14          </main>
15        )
16      </Authenticator>
17    );
18
19  }
20

```

Figura 5: Ruta protegida del componente Home

En dicho componente, se hace uso de las librerías *useAuthenticator* y *Authenticator* las cuales son ofrecidas por los servicios de Amazon Amplify. Se tendrá que hacer esto para todos los componentes que deseemos mantener como rutas protegidas.

Finalmente, dentro de nuestro componente **App.js**, crearemos una función que contendrá el directorio de rutas tanto públicas como protegidas:

```

src > App.js > MyRoutes
37
38  export function MyRoutes(){
39    return (
40      <BrowserRouter>
41        <Routes>
42          <Route path="/" element={<Layout />} />
43          <Route index element={<Home />} />
44          <Route
45            path="/conductor"
46            element={
47              <RequireAuth>
48                <Conductor />
49              </RequireAuth>
50            }
51          />
52          <Route
53            path="/incidencia"
54            element={
55              <RequireAuth>
56                <Incidencia />
57              </RequireAuth>
58            }
59          />
60
61          <Route
62            path="/ubicacion"
63            element={

```

Figura 6: Directorio de rutas

Las rutas protegidas, estarán dentro de las etiquetas `<RequireAuth>`/`</RequireAuth>`, mientras que las públicas, irán dentro de las etiquetas `<Route>`/`</Route>`.

1.3. Resultados

Como resultado de todo lo anterior, tendremos una página de Login que utilizando los servicios de AWS Amplify y Cognito, permitirá iniciar sesión así como registrar a nuevos usuarios.

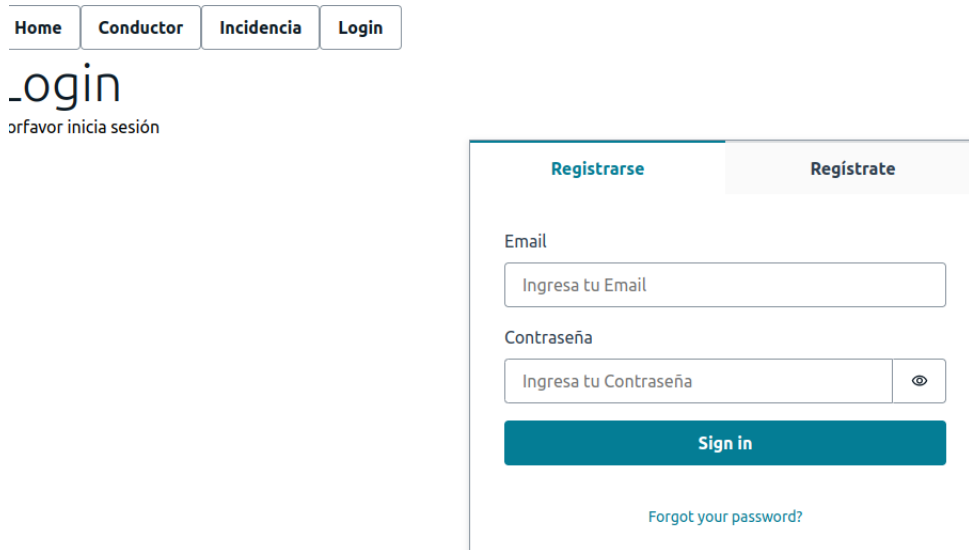


Figura 7: Página de Login

Si intentamos ingresar a las paginas de Conductores, Incidencias o Ubicacion, nos redigirá a la página de Login, debido a que estas páginas fueron definidas como rutas protegidas. Por lo tanto el usuario debe haber iniciado sesión para poder acceder a las mismas.

- **Path:** /
Descripción: En esta dirección se encontrará la el formulario para poder iniciar sesión o registrarse
- **Path:** /home
Descripción: Esta dirección será la página principal de la aplicación dónde se mostrarán las incidencias más recientes así como una lista de todos los conductores
- **Path:** /conductor/
Descripción: Esta dirección mostrará el perfil del conductor de id correspondiente
- **Path:** /detalle_incidencia
Descripción: Esta dirección mostrará cada incidencia mostrando detalles como hora, fecha, coordenadas
- **Path:** /conductor/id/ubicacion
Descripción: En esta vista se mostrará la ubicación en tiempo real de cada conductor
- **Path:** /conductor/id/incidencias
Descripción: En esta vista se mostrará todas las incidencias registradas por cada conductor


2. Definir rutas del backend

2.1. Objetivo

Crear las rutas mediante las que el cliente realizará las peticiones y tendrá acceso a las operaciones, así como su funcionamiento en cuanto a obtención de datos y comunicación con el resto de la aplicación.

2.2. Descripción

Para poder utilizar los servicios de Amazon Amplify, necesitamos dirigirnos al directorio root de nuestro proyecto y ejecutar el siguiente comando:

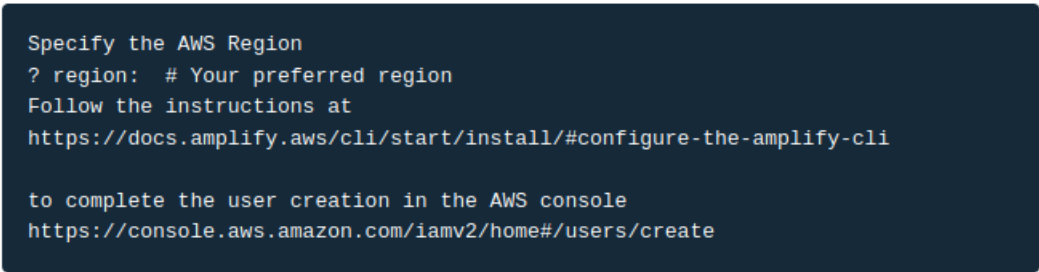


```
npm install -g @aws-amplify/cli
```

[copy](#)

Figura 8: Instalación Amplify CLI

Posteriormente, se necesita especificar la región en la cual queremos alojar nuestra aplicación web:




```
Specify the AWS Region
? region: # Your preferred region
Follow the instructions at
https://docs.amplify.aws/cli/start/install/#configure-the-amplify-cli

to complete the user creation in the AWS console
https://console.aws.amazon.com/iamv2/home#/users/create
```

Figura 9: Configuración del proyecto

Utilizando un editor de código, se necesita especificar que estará utilizando el *SDK* de Amazon Amplify:



```
const AWS = require('aws-sdk')
const awsServerlessExpressMiddleware = require('aws-serverless-express/middleware')
const bodyParser = require('body-parser')
const express = require('express')

AWS.config.update({ region: process.env.TABLE_REGION });
```

Figura 10: Implementación del SDK de Amplify

Posteriormente, declaramos una aplicación de ExpressJs la cuál nos permitirá ejecutar entre otras cosas, peticiones HTTP para la comunicación con la base de datos.

```
// declaracion de una app de express
const app = express()
app.use(bodyParser.json())
app.use(awsServerlessExpressMiddleware.eventContext())
```

Figura 11: Creación de aplicación de Express

Finalmente, se necesitan definir las rutas de las APIs que se estarán utilizando en el proyecto, las cuales serán dos, la primera se encargará de la aplicación web, y la segunda de realizar la comunicación con el Módulo Central de Procesamiento.

Para agregar una api, se necesita ejecutar el siguiente comando desde el directorio raíz del proyecto.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add api
```

Figura 12: Amplify add api

La consola de AWS Amplify requerirá introducir parámetros con los cuales serán construida nuestra API, para el presente proyecto se estará utilizando una arquitectura mediante GraphQL, por lo cual seleccionaremos dicha opción.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add api
? Select from one of the below mentioned services: GraphQL
? Here is the GraphQL API that we will create. Select a setting to edit or continue Continue
? Choose a schema template: One-to-many relationship (e.g., "Blogs" with "Posts" and "Comments")

⚠ WARNING: your GraphQL API currently allows public create, read, update, and delete access to all models via a
n API Key. To configure PRODUCTION-READY authorization rules, review: https://docs.amplify.aws/cli/graphql/autho
rization-rules

✔ GraphQL schema compiled successfully.

Edit your schema at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema.graphql or place .graphql files in
a directory at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema
✔ Do you want to edit the schema now? (Y/n) - yes
```

Figura 13: Configuración de parámetros de GraphQL

Antes de poder publicar nuestra API en AWS Amplify, se necesita definir el Schema con el cuál se hará el manejo de datos. A contiunación se muestran dichos schemas:

```
amplify > backend > api > ebase > ✨ schema.graphql
1  # This "input" configures a global authorization rule to enable
2  # all models in this schema. Learn more about authorization rule
3  input AMPLIFY { globalAuthRule: AuthRule = { allow: public } } #
4
5  type Conductor @model {
6    id: ID
7    nombre: String
8    apellido: String
9    incidencias: [Incidencia] @hasMany
10   num_incidencias: Int
11 }
12
13
14 type Incidencia @model {
15   id: ID
16   title: String
17   estado: Boolean
18   conductor: Conductor @belongsTo
19   detalles: [Detalles] @hasOne
20   fecha_hora: Date
21 }
22
23 type Detalles @model {
24   id: ID
25   incidencia: Incidencia @belongsTo
26   ubicacion: String
27   url_video: String
28 }
29
```

Figura 14: Definición de Schemas para el manejo de datos

Finalmente, ejecutaremos el comando *amplify push*, el cuál publicará la API hacia AWS amplify, generando el endpoint correspondiente a dicha API

2.3. Resultados

Como resultado, tenemos el endpoint de nuestro modelo de GraphQL y nuestra API Key generada por Amplify

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS

Deployment completed.
Deploying root stack ebase [ ===== ] 1/3
  amplify-ebase-dev-175126    AWS::CloudFormation::Stack  UPDATE_IN_PROGRESS
  apiebase                    AWS::CloudFormation::Stack  CREATE_IN_PROGRESS
  authebase35f92a6b          AWS::CloudFormation::Stack  UPDATE_COMPLETE
Deployed api ebase [ ===== ] 9/9
  GraphQLAPI                  AWS::AppSync::GraphQLApi    CREATE_COMPLETE
  GraphQLAPINONEDS95A13CF0    AWS::AppSync::DataSource    CREATE_COMPLETE
  GraphQLAPIDefaultApiKey215A6D... AWS::AppSync::ApiKey        CREATE_COMPLETE
  GraphQLAPITransformerSchema3C... AWS::AppSync::GraphQLSchema  CREATE_COMPLETE
  Blog                        AWS::CloudFormation::Stack  CREATE_COMPLETE
  Comment                     AWS::CloudFormation::Stack  CREATE_COMPLETE
  Post                        AWS::CloudFormation::Stack  CREATE_COMPLETE
  ConnectionStack             AWS::CloudFormation::Stack  CREATE_COMPLETE
  CustomResourcesjson         AWS::CloudFormation::Stack  CREATE_COMPLETE

✓ Generated GraphQL operations successfully and saved at src/graphql
Deployment state saved successfully.

GraphQL endpoint: [REDACTED].appsync-api.us-east-1.amazonaws.com/graphql
GraphQL API KEY: ([REDACTED])jm

GraphQL transformer version: 2

alan@alan-Inspiron-5548:~/Documentos/eb$
```

Figura 15: Generación de Endpoint de GraphQL

3. Creación de la base de datos No Relacional

Debido a problemas de integración junto con los servicios de autenticación y de despliegue de Amplify, se decidió utilizar el sistema de gestión de bases de datos DynamoDB.

DynamoDB es un servicio de base de datos NoSQL Ofrecido por Amazon Web Services. DynamoDB trabaja con tablas. Estas a su vez, contienen parámetros importantes que se mencionarán a continuación.

- **Primary Key:** Se trata de una clave primaria simple, compuesta por un solo atributo denominado clave de partición. Una clave primaria puede ser una clave de partición o una combinación de clave de partición y clave de ordenación. La clave primaria debe ser única en toda la tabla.
- **Partition Key:** Es la llave principal por la cual se agruparán los datos, y determina cómo se particiona la información.
- **Sort Key:** Es llave de ordenamiento de los datos.

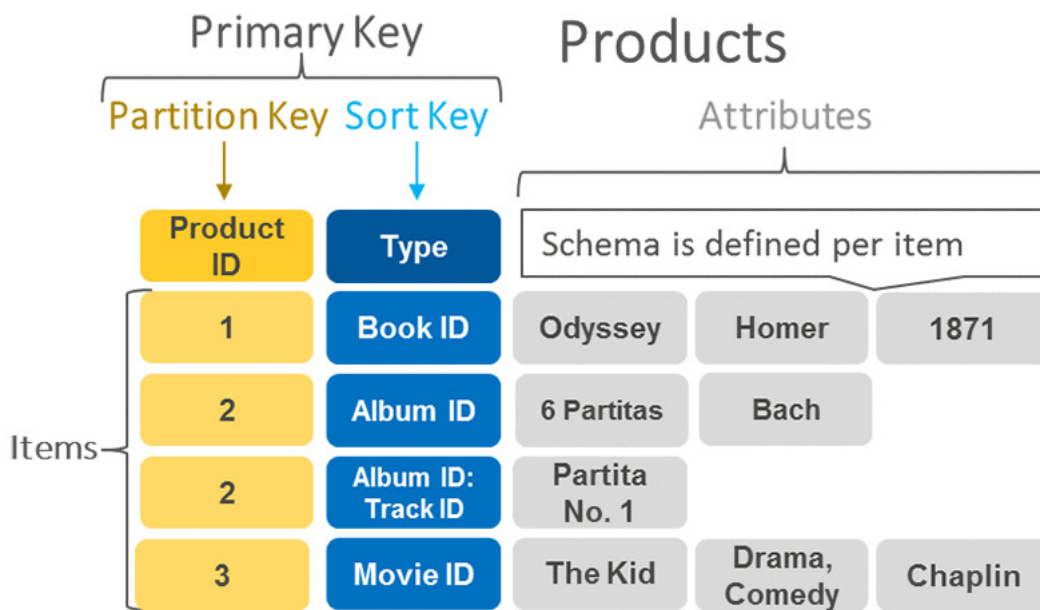


Figura 16: Tablas en DynamoDB

DynamoDB almacena los datos como grupos de atributos, conocidos como elementos. Los elementos son similares a las filas o registros de otros sistemas de bases de datos. DynamoDB almacena y recupera cada elemento en función del valor de la clave principal, que debe ser único.

DynamoDB utiliza el valor de la clave de partición como parámetro de entrada para una función hash interna. El resultado de la función hash determina la partición en la que se almacena el elemento. La ubicación de cada elemento viene determinada por el valor hash de su clave de partición.

Todos los elementos con la misma clave de partición se almacenan juntos y, para las claves de partición compuestas, se ordenan por el valor de la clave de ordenación. DynamoDB divide las particiones por clave de ordenación si el tamaño de la colección crece más de 10 GB[2].

3.1. Objetivo

Crear la base de datos en MongoDB.

3.2. Descripción

Para crear nuestras tablas de DynamoDB, se debe ingresar a la consola de AWS.



Figura 17: Consola de AWS

3.3. Resultados

Una vez dentro, ingresamos a la sección del servicio de DynamoDB

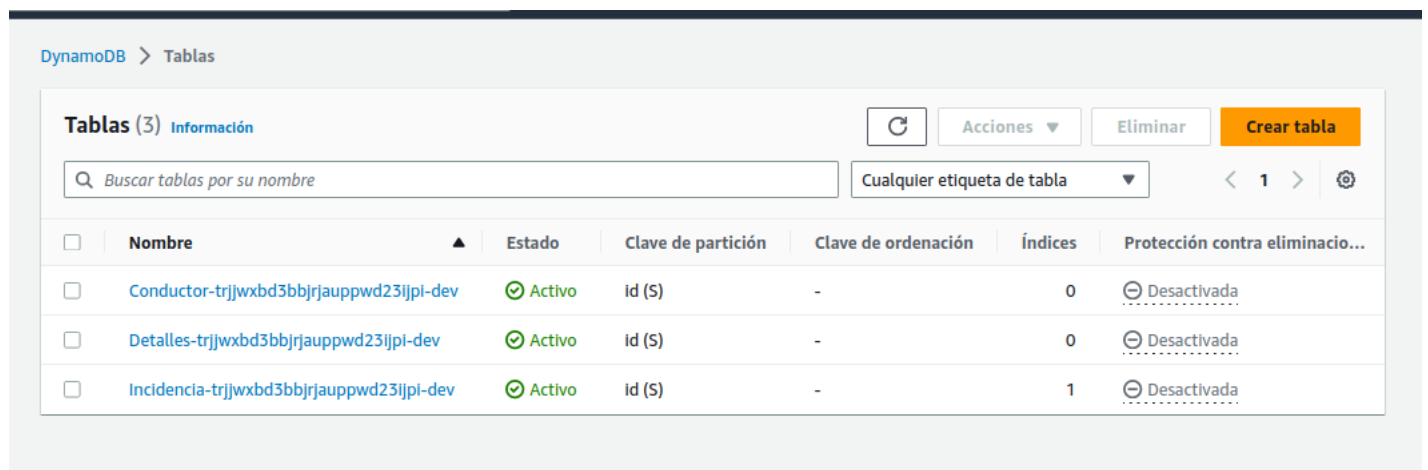


Figura 18: Tablas generadas mediante los schemas definidos

Como se puede observar, gracias a los pasos realizados en la sección 2, DynamoDB crea automáticamente las tablas creadas en base a los schemas definidos previamente.

4. Conexión Backend con Mongo DB

4.1. Objetivo

Realizar la conexión de NodeJs con la base de datos MongoDB.

4.2. Descripción

Para realizar la conexión y la integración de los servicios de DynamoDB hacia nuestra API, se hará uso de AppSync. Las API de GraphQL creadas con AWS AppSync brindan a los desarrolladores frontend la capacidad de consultar varias bases de datos, microservicios y API desde un único punto de conexión de GraphQL.



Figura 19: Funcionamiento de Appsync

AWS AppSync crea las API sin servidor de GraphQL y de publicación o suscripción que simplifican el desarrollo de aplicaciones a través de un único punto de conexión para consultar, actualizar o publicar datos.

4.3. Resultados

Después de haber realizado los pasos de la sección 2, si ejecutamos el comando *amplify status*, la consola de Amplify nos retornara el endpoint de GraphQL junto con AppSync correspondiente, el cuál se utilizará para realizar todas las operaciones con respecto al almacenamiento de datos

```
• alan@alan-Inspiron-5548:~/Documentos/eb$ amplify status

Current Environment: dev



| Category | Resource name | Operation | Provider plugin   |
|----------|---------------|-----------|-------------------|
| Auth     | ebase35f92a6b | No Change | awscloudformation |
| Api      | ebase         | No Change | awscloudformation |



GraphQL endpoint: https://ckqxuivcobc4rgh72skzudaixy.appsync-a
GraphQL transformer version: 2

○ alan@alan-Inspiron-5548:~/Documentos/eb$
```

Figura 20: Generación de endpoint de GraphQL

5. Sistema de acceso con credenciales

5.1. Objetivo

Establecer los roles de cada tipo de usuario con sus respectivos permisos de acceso a la aplicación web utilizando Amazon Cognito

5.2. Descripción

Amazon Cognito funciona utilizando *pools* de usuarios. Un pool de usuarios es un directorio almacenado en los servicios de Amazon[4]. Los beneficios que ofrece estar registrado en un pool de usuarios de Amazon Cognito son los siguientes:

- Servicio de registro e inicio de sesión
- Gestión del directorio de usuarios
- Servicios de seguridad tales como verificación de dos pasos
- Acceso a servicios de la suite de AWS tales como S3 o Dynamodb

Funcionamiento

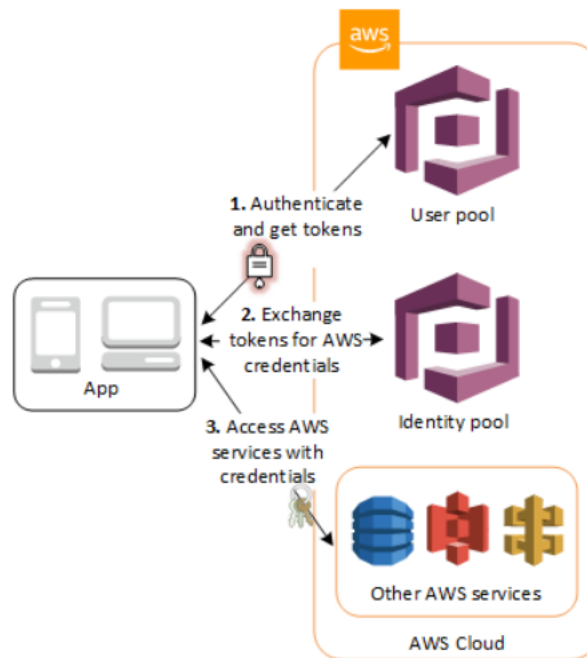


Figura 21: Funcionamiento de Amazon Cognito

- Como primer paso el usuario inicia sesión a través de un grupo de usuarios y recibe tokens del grupo de usuarios después de una autenticación exitosa.
- Posteriormente la aplicación intercambia los tokens del grupo de usuarios por las credenciales de AWS a través de un grupo de identidades.

- Finalmente, el usuario puede usar esas credenciales de AWS para acceder a otros servicios de AWS, como Amazon S3 o DynamoDB.

Implementación

Para poder hacer uso de Amazon Cognito en nuestra aplicación debemos de introducir el siguiente comando:

```
amplify add auth
```

[copy](#)

Figura 22: Implementación de Amazon Cognito en el proyecto

Obteniendo el siguiente menú:

```
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify update auth
Please note that certain attributes may not be overwritten if you choose to use defaults settings.

You have configured resources that might depend on this Cognito resource. Updating this Cognito resource could
have unintended side effects.

Using service: Cognito, provided by: awscloudformation
What do you want to do? Walkthrough all the auth configurations
Select the authentication/authorization services that you want to use: (Use arrow keys)
> User Sign-Up, Sign-In, connected with AWS IAM controls (Enables per-user Storage features for images or other
content, Analytics, and more)
  User Sign-Up & Sign-In only (Best used with a cloud API only)
  I want to learn more.
```

Figura 23: Menu de configuración de Amazon Cognito

Dejamos seleccionado la primera opción, la cual nos permitirá utilizar los servicios de autenticación ofrecidos por Amazon Cognito, además de otros servicios de la suite de AWS.

Finalmente utilizamos el comando *amplify push* para desplegar los cambios a nuestra aplicación.

5.3. Resultados

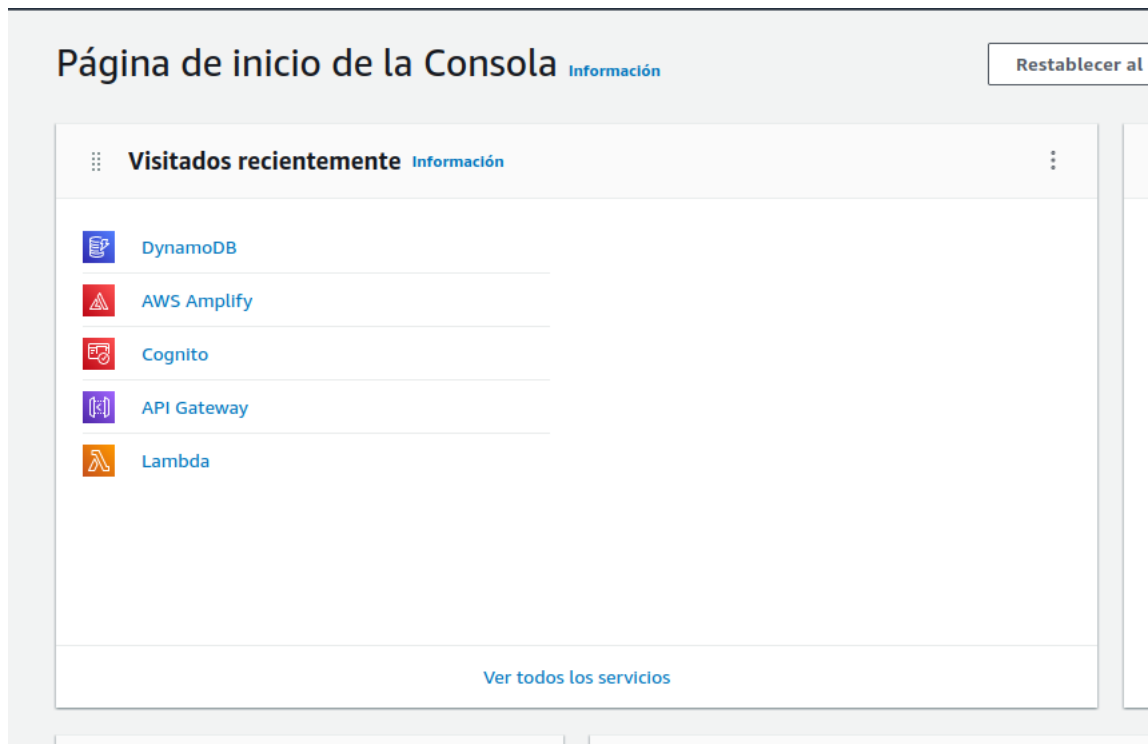


Figura 24: Implementación completada

Entrando a nuestra consola de AWS, en la sección de Amazon Cognito, se puede observar dos grupos de usuarios, uno de tipo Administrador, el cual puede realizar cambios a la configuración de la aplicación, y otro de tipo de Usuario, el cual sólo puede hacer uso del sistema de inicio de sesión y registro ofrecido por Cognito.

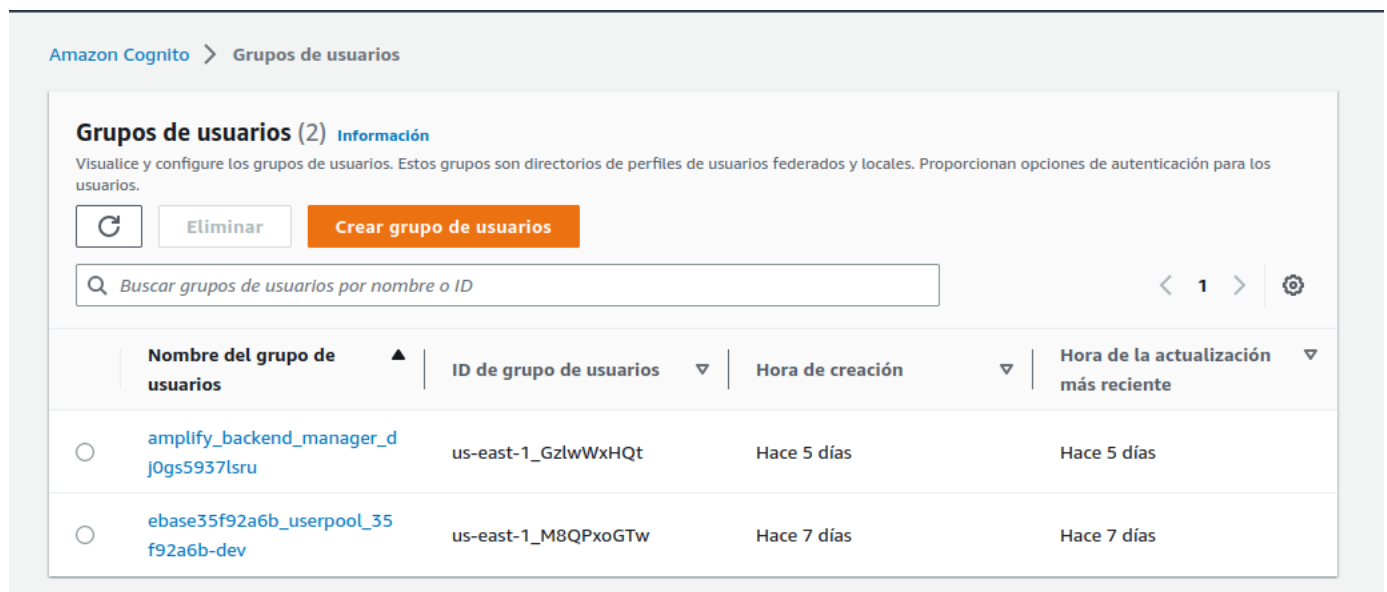


Figura 25: Grupos de usuario

Información general sobre el grupo de usuarios

Nombre del grupo de usuarios
amplify_backend_manager_dj0gs5937lsru

ID de grupo de usuarios
us-east-1_GzlwWxHQ

ARN
arn:aws:cognito-idp:us-east-1:387678068467:userpool/us-east-1_GzlwWxHQ

Número estimado de usuarios
1

Hora de creación
2 de marzo de 2023, 13:46 GMT-6

Hora de la actualización más reciente
2 de marzo de 2023, 13:46 GMT-6

Introducción

Usuarios

Grupos

Experiencia de inicio de sesión

Experiencia de inscripción

Mensajería

Integración de aplicaciones

Propiedades del grupo de usuarios

Usuarios (1) Información

Vea, edite y cree usuarios en el grupo de usuarios. Los usuarios habilitados y confirmados podrán iniciar sesión en el grupo de usuarios.

Nombre de usuario

Buscar usuarios por atributo

< 1 >

Nombre de usuario	Dirección de correo...	Correo electrónico ...	Estado de la confirmación	Estado
<input type="radio"/> aws-amplify-admin	-	No	Confirmado	Habilitado

Figura 26: Grupos de usuario

<

Usuarios

Grupos

Experiencia de inicio de sesión

Experiencia de inscripción

Mensajería

Integración de aplicaciones

Propied...

>

Usuarios (5) Información

Vea, edite y cree usuarios en el grupo de usuarios. Los usuarios habilitados y confirmados podrán iniciar sesión en el grupo de usuarios.

Eliminar usuario

Crear usuario

Nombre de usuario

Buscar usuarios por atributo

< 1 >

Nombre de usuario	Dirección de correo elec...	Correo electrónico verifi...	Estado de la confirmación	Estado
<input type="radio"/> 19bd77dc-441b-4d3e-8...	maite.diazm98@gmail.com	Sí	Confirmado	Habilitado
<input type="radio"/> 473bad4c-b959-45e4-8...	alangam97@gmail.com	No	No confirmado	Habilitado
<input type="radio"/> 849ca9f3-6fd3-4750-a8...	mayte_dm@hotmail.com	No	No confirmado	Habilitado
<input type="radio"/> d1df2a48-8374-4fd2-be...	mayte_dm@hotmail.es	No	No confirmado	Habilitado
<input type="radio"/> e6bf3d4f-1807-4325-ba...	alangam97@gmail.com	Sí	Confirmado	Habilitado

Figura 27: Grupos de usuario

Proyecto Terminal 2

19

6. Investigación de modelos de Redes Neuronales Convolucionales

6.1. Objetivo

Determinar distintos modelos de redes neuronales convolucionales que ofrezcan mejor eficiencia al clasificar imágenes.

6.2. Descripción

Para realizar la investigación de los modelos de redes neuronales convolucionales se consultó la documentación que ofrece Tensorflow y Keras. Dentro de las aplicaciones en Tensorflow se encuentran disponibles distintos modelos de aprendizaje profundo pre entrenados, los cuales se pueden utilizar para predicción y extracción de características, lo cual nos permitirá realizar una clasificación dentro de una imagen. Además de estas aplicaciones pre-entrenadas, TensorFlow.keras también proporciona una variedad de capas de redes neuronales, funciones de pérdida, optimizadores y métricas para construir y entrenar modelos personalizados.

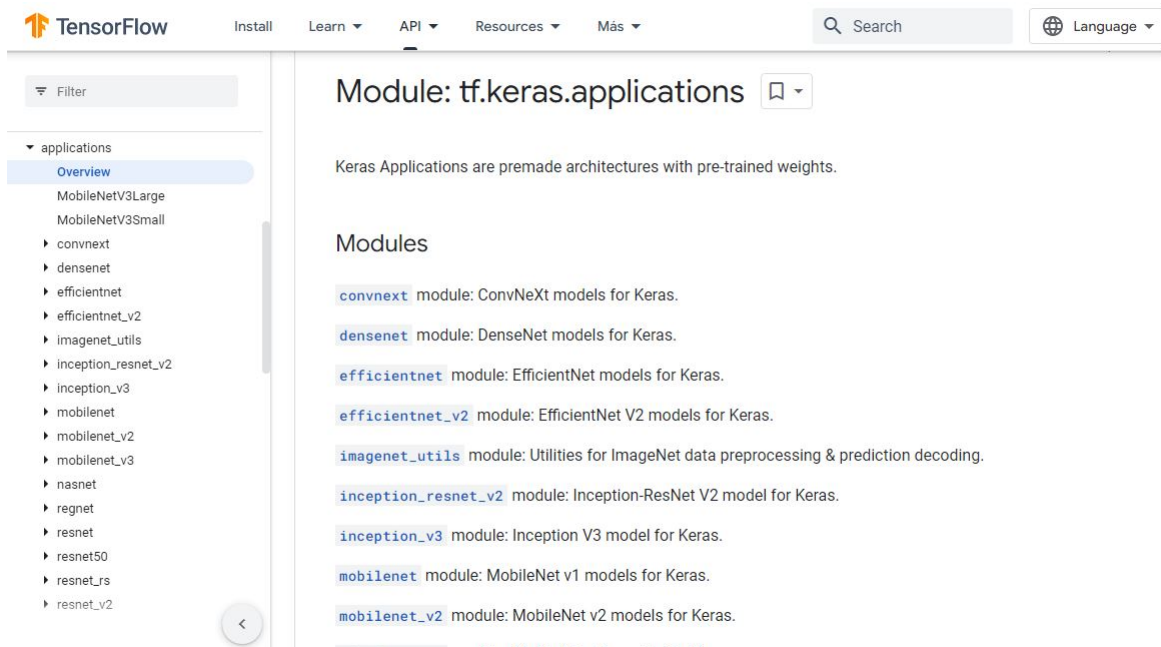


Figura 28: Documentación de Tensorflow referente a las aplicaciones con keras[5].

Existen diversas familias de modelos de aprendizaje preentrenados, cada uno de ellos diseñado para resolver diferentes problemas en el aprendizaje profundo. Algunas de las familias más populares de modelos de redes neuronales convolucionales para la clasificación de imágenes son las siguientes:

- VGG: La familia de modelos VGG, lanzada por el Visual Geometry Group de la Universidad de Oxford en 2014, es conocida por su simplicidad y efectividad en tareas de clasificación de imágenes. Los modelos de esta familia se utilizan comúnmente como modelos base en tareas de transferencia de aprendizaje.

- Inception: La familia de modelos Inception, lanzada por Google en 2014, se enfoca en mejorar la eficiencia y la precisión de los modelos convolucionales mediante la utilización de módulos de convolución en paralelo y convolución 1x1. Los modelos de esta familia se utilizan comúnmente en tareas de clasificación de imágenes y detección de objetos.
- DenseNet: La familia de modelos DenseNet, lanzada por la Universidad de California, Berkeley en 2016, se enfoca en mejorar la propagación de características a través de la red convolucional mediante conexiones densas entre capas. Los modelos de esta familia se utilizan comúnmente en tareas de clasificación de imágenes.
- MobileNet: La familia de modelos MobileNet, lanzada por Google en 2017, se enfoca en reducir el tamaño y la complejidad de los modelos convolucionales para que sean más eficientes en términos de recursos computacionales. Los modelos de esta familia se utilizan comúnmente en aplicaciones móviles y en dispositivos con recursos limitados.
- ConvNeXt: La familia de modelos ConvNeXt, lanzada por Facebook en 2017, se enfoca en mejorar la eficiencia y la precisión de los modelos convolucionales al utilizar módulos de convolución multi-ramificados. Los modelos de esta familia se utilizan comúnmente en tareas de clasificación de imágenes y detección de objetos.
- NasNet: La familia de modelos NasNet, lanzada por Google en 2018, se enfoca en automatizar el diseño de arquitecturas de redes neuronales convolucionales utilizando búsqueda de arquitectura neuronal automatizada (NAS). Los modelos de esta familia se utilizan comúnmente en tareas de clasificación de imágenes y detección de objetos.
- EfficientNet: La familia de modelos EfficientNet, lanzada por Google en 2019, es una extensión de los modelos MobileNet que se enfoca en maximizar el desempeño del modelo mientras se mantiene el tamaño y la complejidad reducidos. Los modelos de esta familia han superado consistentemente el desempeño en comparación con otros modelos en cuanto a la clasificación de imágenes.

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-
ConvNeXtSmall	192.29	82.3%	-	50.2M	-	-	-
ConvNeXtBase	338.58	85.3%	-	88.5M	-	-	-
ConvNeXtLarge	755.07	86.3%	-	197.7M	-	-	-
ConvNeXtXLarge	1310	86.7%	-	350.1M	-	-	-

Figura 29: Modelos de aprendizaje profundo disponibles en Keras.[6]

La tabla hace referencia al nombre del modelo y el tamaño del modelo en MB, por otro lado la precisión top-1 y top-5 se refiere al rendimiento del modelo en el conjunto de datos de validación de ImageNet (base de datos de imágenes etiquetadas para el entrenamiento y evaluación de modelos de visión por computadora). La profundidad se refiere a la profundidad topológica de la red incluyendo capas de activación, capas de normalización por lotes, etc. El tiempo por paso de inferencia es el promedio de 30 lotes y 10 repeticiones, se refiere al tiempo que tarda un modelo de aprendizaje en procesar una única muestra de entrada y producir una salida.

El tiempo de inferencia es importante porque puede afectar la capacidad del modelo para procesar datos en tiempo real y afectar la experiencia del usuario en aplicaciones en línea. Dos de las arquitecturas de redes neuronales convolucionales que se enfocan a reducir el tiempo de inferencia son MobileNet y EfficientNet. Estas arquitecturas permiten ser ejecutadas en dispositivos con recursos limitados como teléfonos móviles y dispositivos integrados, debido a que pueden procesar imágenes más rápidamente que otras arquitecturas más complejas.

Aprendizaje por transferencia

El Transfer Learning, o aprendizaje transferido en español, se refiere al conjunto de métodos que permiten transferir conocimientos adquiridos gracias a la resolución de problemas para resolver otros problemas. La utilización de métodos de Transfer Learning en Deep Learning consiste principalmente en explotar redes neuronales pre-entrenadas. Generalmente, estos modelos corresponden a algoritmos de alto rendimiento que han sido desarrollados y entrenados sobre grandes bases de datos y que son hoy de libre acceso [7].

El aprendizaje por transferencia nos permite hacer uso de los modelos pre-entrenados y modificar la estructura de estos modelos en base a los objetivos deseados, en lugar de entrenar una red neuronal desde cero para cada tarea, se utilizan los pesos de una red neuronal pre-entrenada como punto de partida y se ajustan con el objetivo de resolver una nueva tarea.

La idea detrás del transfer learning es que muchas tareas de aprendizaje profundo tienen características similares y comparten patrones comunes en los datos. Al transferir el conocimiento de una tarea a otra, se pueden aprovechar los conocimientos previamente adquiridos para mejorar la eficiencia y la precisión del modelo en la nueva tarea.

6.3. Resultados

Como resultado de la investigación, los modelos MobileNet y EfficientNet serán propuestos como base para realizar la clasificación de imágenes.

7. Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos

7.1. Objetivo

Diseñar y realizar pruebas de los modelos de redes neuronales convolucionales previamente investigados para determinar el rendimiento y la precisión de cada uno.

7.2. Descripción

Para realizar el entrenamiento de los modelos EfficientNet y MobileNet se hizo uso de google colab el cual nos permite ejecutar código de Python en el navegador.

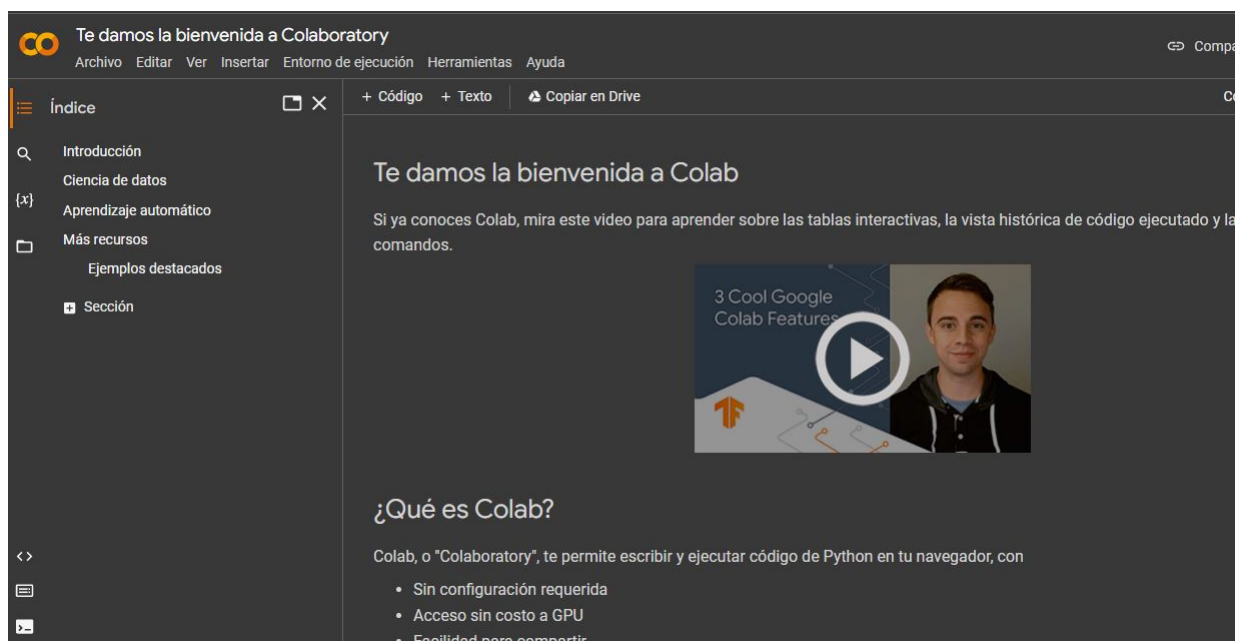


Figura 30: Entorno de trabajo Google Colab.

Se requirió de un dataset con imágenes de ojos abiertos y cerrados para entrenar los modelos, el dataset que se utilizó es el siguiente: MRL Eye Dataset[8].

Parte de los datos del data set original se separaron en dos carpetas, en la primera carpeta se colocaron las imágenes con ojos cerrados y en la segunda carpeta las imágenes de ojos abiertos. Posteriormente se subieron las carpetas a un repositorio en drive para trabajar desde Google Colab. El dataset con el cual se trabajó contiene 3242 imágenes de las cuales 1840 son imágenes de ojos cerrados y 1402 son de con ojos abiertos.

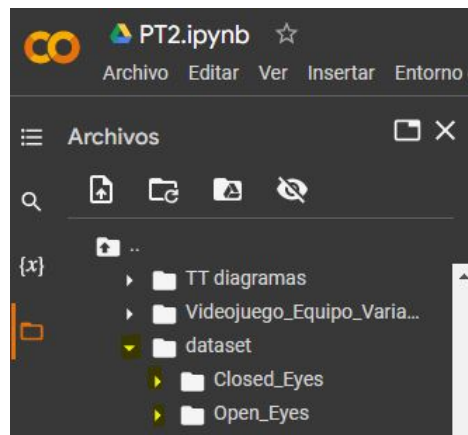


Figura 31: Dataset a trabajar desde Google Colab.

Se importaron las librerías a utilizar y el repositorio de drive para hacer uso del dataset, así mismo se definieron 2 clases, la primera para ojos abiertos y la segunda para ojos cerrados. Ya que los modelos pre entrenados con los que se trabajó son en base a imágenes con dimensión de 244 x 244, se definió el tamaño de la imagen en 224.

A screenshot of a Jupyter notebook cell in Google Colab. The code imports tensorflow, cv2, os, matplotlib.pyplot, numpy, and gc. It then mounts Google Drive at '/content/drive'. The output shows the drive is already mounted. The cell also defines a Datadirectory and classes for 'Closed_Eyes' and 'Open_Eyes' with an image size of 224.

```
import tensorflow as tf
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
import gc
gc.collect()

0

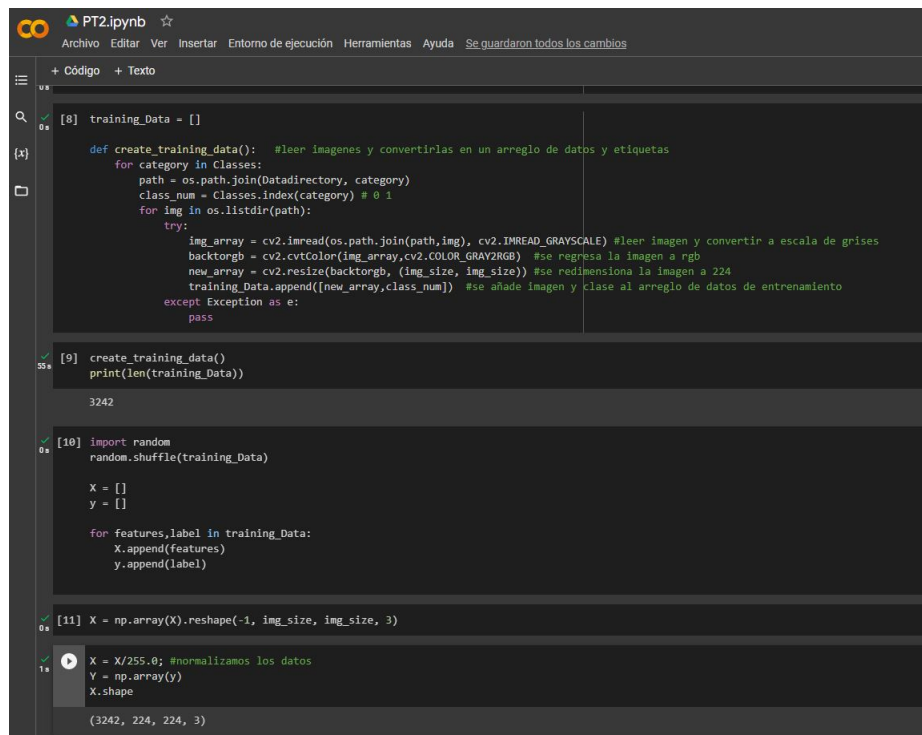
[6] from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[7] Datadirectory = "/content/drive/MyDrive/dataset" #entrenamiento del dataset
Classes = ["Closed_Eyes", "Open_Eyes" ] #Clases
img_size = 224
```

Figura 32: Librerías utilizadas y repositorio en drive del dataset.

Es necesario convertir las imágenes en un arreglo de datos con su respectiva clase por lo que se creo una función con dicho proposito, posteriormente se etiquetaron los datos y se normalizaron para obtener las entradas de datos, donde X contiene los pixeles de las imágenes en tamaño 224x224 en 3 planos (rgb) y Y contiene el arreglo de etiquetas 0 y 1, que corresponden a la clase a la que pertenece cada imagen.



```

PT2.ipynb
Archivo  Editar  Ver  Insertar  Entorno de ejecución  Herramientas  Ayuda  Se guardaron todos los cambios

+ Código  + Texto

[8] training_Data = []

def create_training_data(): #leer imagenes y convertirlas en un arreglo de datos y etiquetas
    for category in Classes:
        path = os.path.join(Datadirectory, category)
        class_num = Classes.index(category) # 0 1
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE) #leer imagen y convertir a escala de grises
                backtorgb = cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB) #se regresa la imagen a rgb
                new_array = cv2.resize(backtorgb, (img_size, img_size)) #se redimensiona la imagen a 224
                training_Data.append([new_array,class_num]) #se añade imagen y clase al arreglo de datos de entrenamiento
            except Exception as e:
                pass

[9] create_training_data()
print(len(training_Data))

3242

[10] import random
random.shuffle(training_Data)

X = []
y = []

for features,label in training_Data:
    X.append(features)
    y.append(label)

[11] X = np.array(X).reshape(-1, img_size, img_size, 3)

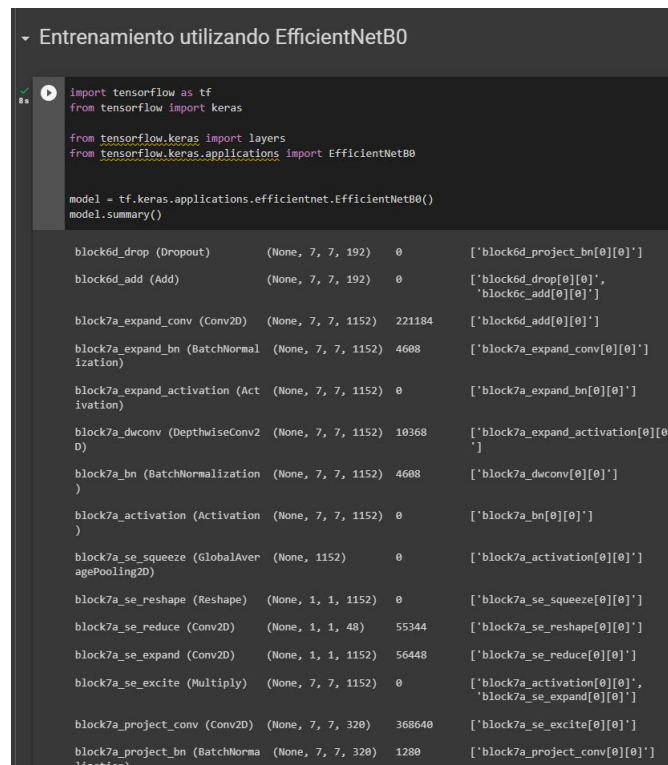
X = X/255.0; #normalizamos los datos
Y = np.array(y)
X.shape

(3242, 224, 224, 3)

```

Figura 33: Preparación de los datos de entrada

Posteriormente se exporto el modelo y se visualizó la arquitectura de la red.



```

Entrenamiento utilizando EfficientNetB0

import tensorflow as tf
from tensorflow import keras

from tensorflow.keras import layers
from tensorflow.keras.applications import EfficientNetB0

model = tf.keras.applications.efficientnet.EfficientNetB0()
model.summary()

block6d_drop (Dropout) (None, 7, 7, 192) 0 ['block6d_project_bn[0][0]']
block6d_add (Add) (None, 7, 7, 192) 0 ['block6d_drop[0][0]', 'block6c_add[0][0]']
block7a_expand_conv (Conv2D) (None, 7, 7, 1152) 221184 ['block6d_add[0][0]']
block7a_expand_bn (BatchNormal (None, 7, 7, 1152) 4688 ['block7a_expand_conv[0][0]']
ization)
block7a_expand_activation (Act (None, 7, 7, 1152) 0 ['block7a_expand_bn[0][0]']
ivation)
block7a_dwconv (DepthwiseConv2 (None, 7, 7, 1152) 10368 ['block7a_expand_activation[0][0]']
D)
block7a_bn (BatchNormalization (None, 7, 7, 1152) 4688 ['block7a_dwconv[0][0]']
)
block7a_activation (Activation (None, 7, 7, 1152) 0 ['block7a_bn[0][0]']
)
block7a_se_squeeze (GlobalAver (None, 1152) 0 ['block7a_activation[0][0]']
agePooling2D)
block7a_se_reshape (Reshape) (None, 1, 1, 1152) 0 ['block7a_se_squeeze[0][0]']
block7a_se_reduce (Conv2D) (None, 1, 1, 48) 55344 ['block7a_se_reshape[0][0]']
block7a_se_expand (Conv2D) (None, 1, 1, 1152) 56448 ['block7a_se_reduce[0][0]']
block7a_se_excite (Multiply) (None, 7, 7, 1152) 0 ['block7a_activation[0][0]', 'block7a_se_expand[0][0]']
block7a_project_conv (Conv2D) (None, 7, 7, 320) 368640 ['block7a_se_excite[0][0]']
block7a_project_bn (BatchNorma (None, 7, 7, 320) 1280 ['block7a_project_conv[0][0]']
lization)

```

Figura 34: Exportación del modelo MobileNet

Se aplicaron modificaciones a partir de la cuarta última capa, se agregaron nuevas capas y la función de activación sigmoid.

```

base_input = model.layers[0].input #Capa de entrada
base_output = model.layers[-2].output

Flat_layer = layers.Flatten()(base_output)
final_output = layers.Dense(1)(Flat_layer)
final_output = layers.Activation('sigmoid')(final_output)

new_model = keras.Model(inputs=base_input, outputs=final_output)
new_model.summary()

```

Layer (type)	Output Shape	Param #	Connected to
block6c_add (Add)	(None, 7, 7, 1152)	221184	['block6d_add[0][0]']
block7a_expand_conv (Conv2D)	(None, 7, 7, 1152)	4608	['block6d_add[0][0]']
block7a_expand_bn (Batch Normalization)	(None, 7, 7, 1152)	0	['block7a_expand_conv[0][0]']
block7a_expand_activation (Activation)	(None, 7, 7, 1152)	0	['block7a_expand_bn[0][0]']
block7a_dwconv (Depthwise Conv2D)	(None, 7, 7, 1152)	10368	['block7a_expand_activation[0][0]']
block7a_bn (Batch Normalization)	(None, 7, 7, 1152)	4608	['block7a_dwconv[0][0]']
block7a_activation (Activation)	(None, 7, 7, 1152)	0	['block7a_bn[0][0]']
block7a_se_squeeze (Global Average Pooling2D)	(None, 1152)	0	['block7a_activation[0][0]']
block7a_se_reshape (Reshape)	(None, 1, 1, 1152)	0	['block7a_se_squeeze[0][0]']
block7a_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	['block7a_se_reshape[0][0]']
block7a_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	['block7a_se_reduce[0][0]']
block7a_se_excite (Multiply)	(None, 7, 7, 1152)	0	['block7a_activation[0][0]', 'block7a_se_expand[0][0]']
block7a_project_conv (Conv2D)	(None, 7, 7, 320)	368640	['block7a_se_excite[0][0]']
block7a_project_bn (Batch Normalization)	(None, 7, 7, 320)	1280	['block7a_project_conv[0][0]']
top_conv (Conv2D)	(None, 7, 7, 1280)	409600	['block7a_project_bn[0][0]']
top_bn (Batch Normalization)	(None, 7, 7, 1280)	5120	['top_conv[0][0]']
top_activation (Activation)	(None, 7, 7, 1280)	0	['top_bn[0][0]']

Figura 35: Modificación del modelo Mobilenet

Finalmente se entrena el modelo MobileNet

```

[14] #ajustes para la clasificación binaria (abierto cerrado)
#model.compile(loss = "binary_crossentropy", optimizer="adam", metrics=["accuracy"])
#hist = model.fit(X,Y, epochs = 1, validation_split=0.2) #entrenamiento

new_model.compile(loss = "binary_crossentropy", optimizer="adam", metrics=["accuracy"])
hist = new_model.fit(X,Y, epochs = 5, validation_split=0.2) #entrenamiento transfer learning

```

Epoch	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 1/5	0.1598	0.9487	2.7340	0.5932
Epoch 2/5	0.0598	0.9753	1.2245	0.7596
Epoch 3/5	0.0592	0.9734	0.0797	0.9661
Epoch 4/5	0.0220	0.9907	0.0564	0.9753
Epoch 5/5	0.0376	0.9838	0.2774	0.9091

Figura 36: Entrenamiento del modelo MobileNet

Para el modelo EfficientNet se realizó el mismo procedimiento que en el modelo EfficientNet.

```

[18] #entrenamiento
new_model.compile(loss = "binary_crossentropy", optimizer="adam", metrics=["accuracy"])
hist = new_model.fit(X,Y, epochs = 5, validation_split=0.2) #entrenamiento transfer learning

```

Epoch	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 1/5	0.1273	0.9476	3.7610	0.5609
Epoch 2/5	0.0741	0.9707	1.4369	0.5609
Epoch 3/5	0.0668	0.9784	2.2189	0.5609
Epoch 4/5	0.0519	0.9769	1.3085	0.5609
Epoch 5/5	0.0462	0.9853	2.4268	0.5609

Figura 37: Entrenamiento del modelo EfficientNet

La gráfica muestra el las épocas y la precisión obtenida en cada una de ellas, así como la precisión de los datos ocupados para la validación dentro del entrenamiento.

MobileNet

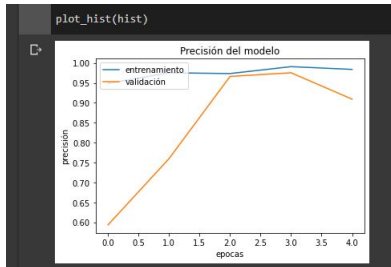


Figura 38: Gráfica de precisión en cada época del modelo MobileNet.

EfficientNet

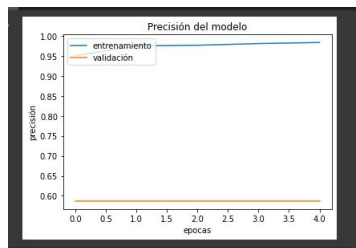


Figura 39: Gráfica de precisión en cada época del modelo EfficientNet.

7.3. Resultados

Una vez entrenados los modelos se realizaron las pruebas de los modelos EfficientNet y MobileNet para ojos cerrado y abiertos. Se obtuvieron los siguientes resultados:

MobileNet

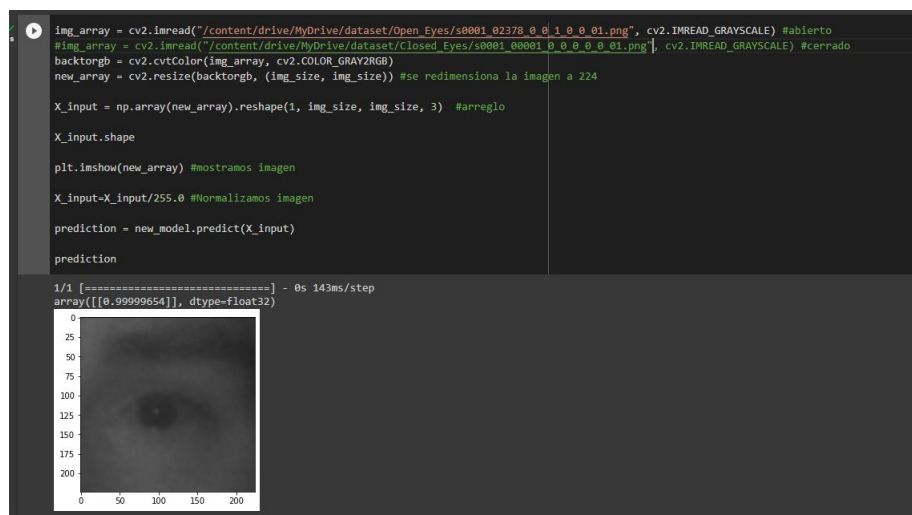


Figura 40: Predicción 1 para del modelo MobileNet.

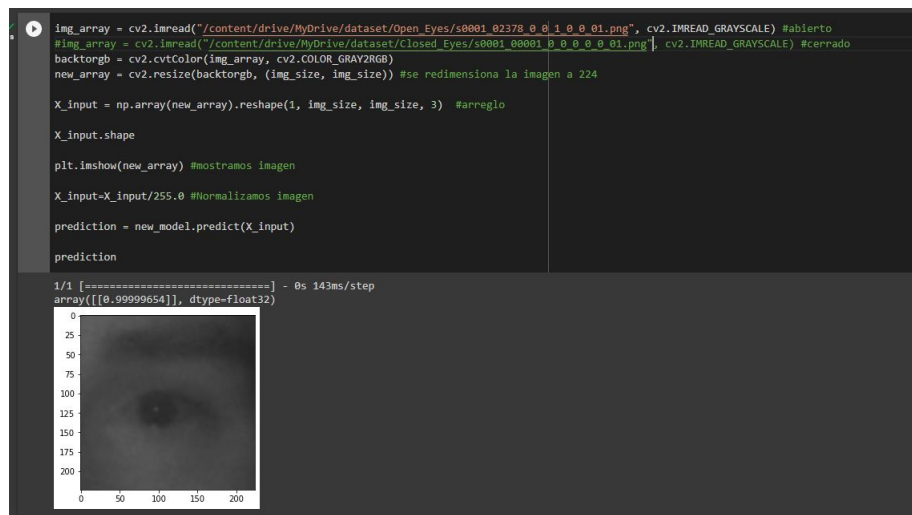


Figura 41: Predicción 2 para del modelo MobileNet.

EfficientNet

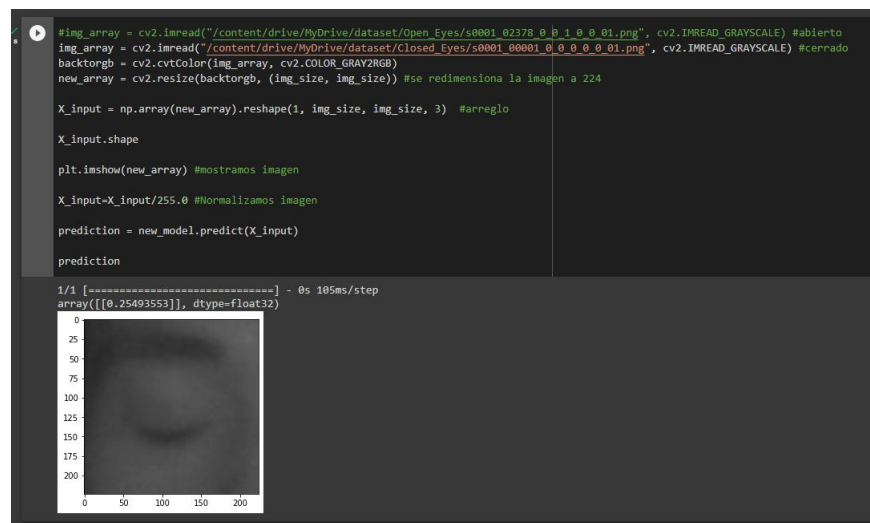


Figura 42: Predicción 1 para del modelo EfficientNet.

```
img_array = cv2.imread("/content/drive/MyDrive/dataset/Open_Eyes/s0001_02378_0_0_1_0_0_01.png", cv2.IMREAD_GRAYSCALE) #abierto
#img_array = cv2.imread("/content/drive/MyDrive/dataset/Closed_Eyes/s0001_00001_0_0_0_0_0_01.png", cv2.IMREAD_GRAYSCALE) #cerrado
backtorgb = cv2.cvtColor(img_array, cv2.COLOR_GRAY2RGB)
new_array = cv2.resize(backtorgb, (img_size, img_size)) #se redimensiona la imagen a 224

X_input = np.array(new_array).reshape(1, img_size, img_size, 3) #arreglo

X_input.shape

plt.imshow(new_array) #mostramos imagen

X_input=X_input/255.0 #Normalizamos imagen

prediction = new_model.predict(X_input)

prediction

1/1 [=====] - 2s 2s/step
array([[0.25561202]], dtype=float32)
```

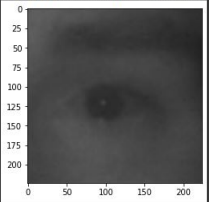


Figura 43: Predicción 2 para del modelo EfficientNet.

8. Conclusiones

Se implementó una página de Login utilizando los servicios de AWS Amplify y Cognito, lo que permite la autenticación y registro de usuarios en la aplicación. También se han definido rutas protegidas para garantizar que solo los usuarios autenticados puedan acceder a ciertas secciones de la aplicación. Además, se ha utilizado DynamoDB para almacenar los datos de la aplicación y se ha creado una API sin servidor de GraphQL utilizando AWS AppSync para simplificar el desarrollo de la aplicación. Los modelos MobileNet y EfficientNet han sido investigados y propuestos como base para la clasificación de imágenes de ojos abiertos y cerrados. Aunque se han realizado pruebas con los modelos MobileNet y EfficientNet para la clasificación de imágenes de ojos abiertos y cerrados, se necesita seguir trabajando en la mejora del sistema y la realización de pruebas adicionales para mejorar su eficiencia y precisión. Es posible que el sistema aún presente problemas de sobreajuste o sesgo, lo que puede afectar su capacidad para generalizar a nuevos datos. El proyecto ha logrado implementar una aplicación escalable utilizando los servicios de AWS, pero se requiere seguir trabajando en la optimización y evaluación de los modelos de aprendizaje automático para mejorar la eficiencia y precisión del sistema.

9. Bibliografía

Referencias

- [1] F. Martinez. *Protección de rutas con React Router Dom*, DEV Community. <https://dev.to/franklin030601/proteccion-de-rutas-con-react-router-dom-144j> (accedido el 7 de marzo de 2023).
- [2] *¿Qué es Amazon DynamoDB?*, Amazon Docs. <https://docs.aws.amazon.com/es-es/amazondynamodb/latest/developerguide/Introduction.html> (accedido el 2 de marzo de 2023).
- [3] *API sin servidor de GraphQL y de publicación o suscripción – AWS AppSync – Amazon Web Services*. Amazon Web Services, Inc. <https://aws.amazon.com/es/appsync/> (accedido el 12 de marzo de 2023).
- [4] *AWS — Gestión de identidades y autenticación de usuario en la nube*, Amazon Web Services, Inc. <https://aws.amazon.com/es/cognito/> (accedido el 8 de marzo de 2023).
- [5] TensorFlow. (2021). tf.keras.applications.mobilenet. TensorFlow API documentation. [Online]. Disponible en: https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet [Accedido el 13 de marzo de 2023].
- [6] Keras. (2021). Keras Applications. [Online]. Disponible en: <https://keras.io/api/applications/> [Accedido el 13 de marzo de 2023].
- [7] M. Arroyo, "¿Qué es el Transfer Learning?," Data Scientist, 2019. [Online]. Available: <https://datascientest.com/es/que-es-el-transfer-learning#:~:text=%E2%80%9CEl%20Transfer%20Learning%2C%20o%20aprendizaje,el%20crecimiento%20del%20Deep%20Learning>. [Accessed: 13-Mar-2023].
- [8] Marek Ruzicka Lab. (s.f.). Eye Dataset. [Online]. Disponible en: <http://mrl.cs.vsb.cz/eyedataset/> [Accedido el 13 de marzo de 2023].