



# Instituto Politécnico Nacional

ESCUELA SUPERIOR DE CÓMPUTO

## Trabajo Terminal

2019-A009

**Ingeniería en Sistemas Computacionales**

**PROTOTIPO DE SISTEMA PARA  
RECONOCER TEXTO EN IMÁGENES Y  
TRADUCIRLO A LATEX**

PRESENTAN:

**Carlos Tonatiuh Barrera Pérez**

**Juan Carlos García Medina**

**Ian Mendoza Jaimes**

DIRECTORES:

**Dr. Jorge Cortés Galicia**



Ciudad de México, 9 de junio de 2020

# Índice general

---

<b>Índice de figuras</b>	<b>III</b>
<b>Índice de tablas</b>	<b>v</b>
<b>1. Desarrollo del sistema</b>	<b>1</b>
1.1. Android . . . . .	1
1.1.1. Arquitectura de la aplicación . . . . .	1
1.1.1.1. Capa de datos . . . . .	2
1.1.1.2. Capa de dominio . . . . .	9
1.1.1.3. Capa de presentación . . . . .	15
1.2. Web . . . . .	16
1.2.1. Arquitectura de la aplicación . . . . .	16
1.2.1.1. Modelo . . . . .	16
1.2.1.2. Vista . . . . .	16
1.2.1.3. Template . . . . .	16
1.3. Conjuntos de entrenamiento . . . . .	17
1.3.1. CROHME . . . . .	17
1.3.1.1. Formato del conjunto de datos . . . . .	17
1.3.1.2. Conversión a imagen . . . . .	18
1.3.1.3. Generador de secuencia de tokens . . . . .	21
1.3.2. Harvard 100k . . . . .	23
1.3.3. Normalización . . . . .	23
1.4. Desarrollo módulo de análisis de imágenes . . . . .	25
1.4.1. Algoritmo de Otsu . . . . .	26
1.4.2. Algoritmo de Sauvola . . . . .	27
1.4.3. Resultados . . . . .	27
1.5. Reconocimiento de expresiones matemáticas . . . . .	35
1.5.1. Modelo . . . . .	35
1.5.1.1. Encoder . . . . .	37
1.5.1.2. Decoder . . . . .	38
1.5.1.3. Implementación . . . . .	40
1.5.1.4. Entrenamiento . . . . .	44
1.5.1.5. Resultados . . . . .	47

**ÍNDICE GENERAL**

---

1.5.1.6. Experimentos Previos . . . . .	49
<b>Bibliografía</b>	<b>55</b>

# Índice de figuras

---

1.1.	Tres capas que se tienen al utilizar la arquitectura Clean [1] . . . . .	2
1.2.	Capa de datos [2] . . . . .	4
1.3.	Capa de dominio [2] . . . . .	9
1.4.	Capa de presentación [3] . . . . .	15
1.5.	Ejemplo de una imagen del conjunto de entrenamiento Harvard 100k . . . . .	23
1.6.	Imagen de ejemplo del conjunto de entrenamiento generado . . . . .	25
1.7.	Fotografía procesada con algoritmo de Sauvola . . . . .	25
1.8.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	28
1.9.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	28
1.10.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	29
1.11.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	29
1.12.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	30
1.13.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	30
1.14.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	31
1.15.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	31
1.16.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	32
1.17.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	32
1.18.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	33
1.19.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	33
1.20.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	34
1.21.	Resultados de la aplicación del algoritmo de Otsu y Sauvola . . . . .	34
1.22.	Modelo <i>Encoder-Decoder</i> de la red neuronal. . . . .	36
1.23.	Función de pérdida del modelo entrenado con CROHME. . . . .	47
1.24.	(a) Atención de una secuencia corta junto con la secuencia predicha por el modelo. (b) Atención en una secuencia larga junto con la secuencia predicha por el modelo. . . . .	48
1.25.	(a) Atención de una secuencia larga renderizada por computadora junto con la secuencia predicha por el modelo. (b) Función de perdida del modelo. . . . .	49
1.26.	(a) Función de perdida del sistema, el modelo se entrena en varias ocasiones, siempre obteniendo una gráfica parecida. (b) Se muestra la atención obtenida por este modelo, se observa un aprendizaje nulo. . . . .	51

## ÍNDICE DE FIGURAS

# Índice de tablas

---

## Capítulo 1

# Desarrollo del sistema

---

A continuación, se explica el desarrollo del sistema, dicha explicación se encuentra dividida en la sección del desarrollo de la aplicación móvil y por otra parte se encuentra el desarrollo de la aplicación web.

## 1.1. Android

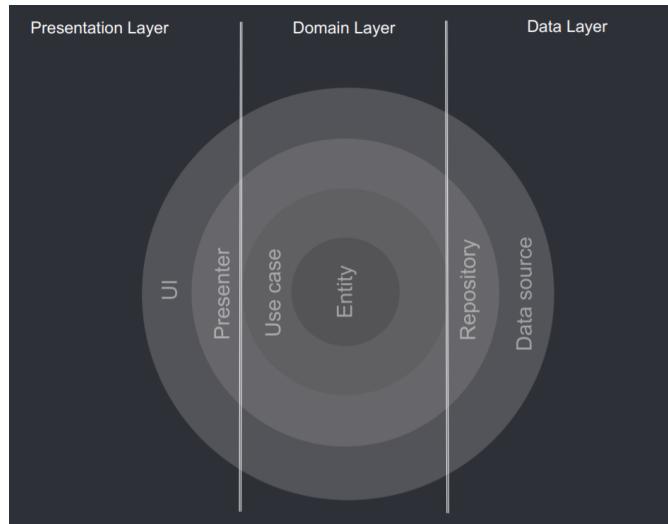
Esta sección tiene objetivo presentar las principales características en el desarrollo de la aplicación para Android.

### 1.1.1. Arquitectura de la aplicación

Para el desarrollo de la aplicación se implemento la arquitectura Clean, la cual como ya se ha mencionado antes se ha vuelto muy popular en el desarrollo de aplicaciones móviles para android debido a que es una solución que produce sistemas que presentan las siguientes características.

- Escalables, por lo que se pueden agregar más funcionalidades de forma sencilla.
- Presentan modularidad.
- Presentan independencia en cuanto a frameworks, interfaz de usuario y bases de datos.
- El proyecto es más fácil de mantener por lo que es más sencillo hacer cambios.

Al utilizar esta arquitectura el proyecto queda separado en tres capas como se observa en la figura 1.1 con lo cual cada una de ellas tiene su propósito definido.



**Figura 1.1:** Tres capas que se tienen al utilizar la arquitectura Clean [1]

### 1.1.1.1. Capa de datos

La información que se utiliza en el resto de capas proviene de esta capa. Esta capa a su vez se encuentra dividida en la capa de repositorio y en la capa de fuente de datos.

**Capa de repositorio** En esta capa se utiliza el patrón de repositorio como se muestra en la figura 1.2. Gracias a este patrón se puede tener acceso a diferentes fuentes de datos que se encuentran en la capa más baja de nuestra arquitectura, esto nos permite un acceso a los datos de forma transparente para el usuario bajo las condiciones que se presenten.

La forma de utilizar este patrón en la aplicación desarrollado crear una clase en la cual se hace uso de la interfaz que se tiene para el acceso a la fuente de datos. En el siguiente código se puede apreciar el como se crea una instancia de APIService que es nuestra interfaz para fuente de datos.

Después, en nuestro método findAllProyectosByUser se recupera la información necesaria para mandarla a las capas superiores.

```

1 public class ProjectRepositoryImpl implements ProjectRepository {
2     private APIService service = ServiceGenerator.createService(APIService.
3         class);
4     private static final String TAG = ProjectRepositoryImpl.class .
5         getCanonicalName();
6
7     @Override
8     public MutableLiveData<BusinessResult<ProyectoModel>>
9         findAllProyectosByUser(Integer id, String key) {

```

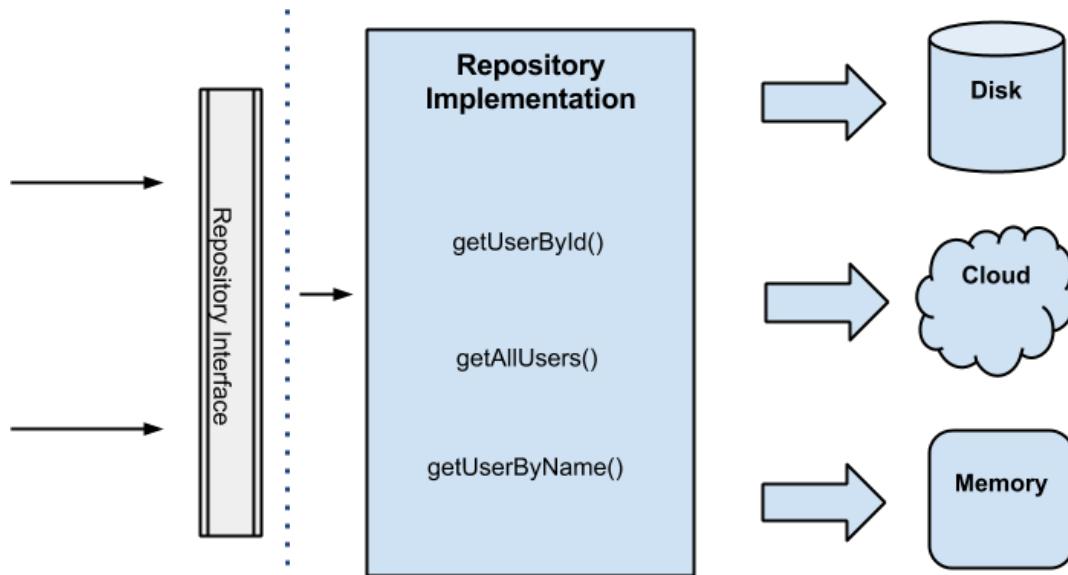
```

7     MutableLiveData<BusinessResult<ProyectoModel>>
8     proyectoDataMutableLiveData = new MutableLiveData<>();
9
10    try {
11        service.getProyectosByUsuario(id, key).enqueue(new Callback<List<
12            ProyectoData>>() {
13            @Override
14            public void onResponse(Call<List<ProyectoData>> call,
15            Response<List<ProyectoData>> response) {
16                BusinessResult<ProyectoModel> model = new BusinessResult
17                <>();
18                List<ProyectoModel> modelos = new ArrayList<>();
19                if (response.isSuccessful()) {
20                    for (ProyectoData data : response.body()) {
21                        ProyectoModel proyectoModel = new ProyectoModel()
22                        ;
23                        proyectoModel.setRate(data.getCalificacion());
24                        proyectoModel.setId(data.getId());
25                        proyectoModel.setName(data.getNombre());
26                        proyectoModel.setTextDate(data.getFecha());
27                        modelos.add(proyectoModel);
28                    }
29                    model.setResults(modelos);
30                    model.setCode(ResultCodes.SUCCESS);
31                }
32                proyectoDataMutableLiveData.setValue(model);
33            }
34            @Override
35            public void onFailure(Call<List<ProyectoData>> call,
36            Throwable t) {
37                BusinessResult<ProyectoModel> model = new BusinessResult
38                <>();
39                proyectoDataMutableLiveData.setValue(model);
40            }
41        });
42    }

```

```

35     } catch (NetworkOnMainThreadException e) {
36         Log.e(TAG, "findAllProyectosByUser ", e);
37     }
38     return proyectoDataMutableLiveData;
39 }
40 }
```



**Figura 1.2:** Capa de datos [2]

**Capa de fuente de datos** En este trabajo, la fuente de datos que se tiene es un API REST, sin embargo si se requiere acceder a información que se persista en el teléfono se puede agregar otra fuente de datos. Se utiliza retrofit para poder realizar la comunicación con el API REST.

La forma de utilizar retrofit es crear una interfaz con todos los métodos para recuperar o enviar información al API REST, en esta interfaz cada método tiene la URL a la cual se realizará la petición con alguno de los métodos que tiene HTTP, se tienen los parámetros que se envían y cada método nos regresa una llamada asíncrona que se trabaja en la capa de repositorio. Esto se puede apreciar en el siguiente código.

```

1 public interface APIService {
2     // Crear proyectoModel
3     @POST("/proyectos")
4     Call<ProyectoData> createProyecto(@Body ProyectoData proyectoModel,
5                                         @Header("Authorization") String token);
```

```
5 // Edicion de un proyectoModel
6 @PUT("/proyectos/{idProyecto}")
7 Call<ProyectoData> editProyecto(@Path("idProyecto") Integer idProyecto,
8 @Body ProyectoData proyectoModel, @Header("Authorization") String token);
// Elimina un proyecto
9 @DELETE("/proyectos/{idProyecto}")
10 Call<ProyectoData> deleteProyecto(@Path("idProyecto") Integer idProyecto,
11 @Header("Authorization") String key);
// Obtiene las traducciones asociadas a un proyecto
12 @GET("/proyectos/{idProyecto}/traducciones")
13 Call<List<TraduccionData>> getTraduccionesByProyecto(@Path("idProyecto") Integer idProyecto, @Header("Authorization") String key);
// Creacion de una traduccionModel
14 @POST("/traducciones")
15 Call<TraduccionData> createTraduccion(@Body TraduccionData traduccionData);
// Edicion de un traduccionModel
16 @PUT("/traducciones/{idTraducion}")
17 Call<TraduccionData> editTraduccion(@Path("idTraducion") Integer idTraducion, @Body TraduccionData traduccionData);
// Elimina una traduccion
18 @DELETE("/traducciones/{idTraducion}")
19 Call<TraduccionData> deleteTraduccion(@Path("idTraducion") Integer idTraducion, @Header("Authorization") String token);
// Manda a crear un usuarioData
20 @POST("/usuarios")
21 Call<UsuarioData> createUsuario(@Body UsuarioData usuarioData);
// Para hacer login
22 @POST("/users/login")
23 Call<UsuarioData> loginUsuario(@Body UsuarioData usuarioData);
// Para hacer la recuperacion de contra
24 @POST("/usuarios/recuperar")
25 Call<UsuarioData> recuperarUsuario(@Body UsuarioData usuarioData);
// Para editar usuarioData
26 @PUT("/users/{idUsuario}")
27
28
29
30
31
32
33
```

```

34     Call<UsuarioData> editUsuario(@Path("idUsuario") Integer id, @Body
35     UsuarioData userData, @Header("Authorization") String key);
36     // Obtiene los proyectos asociados a un usuario
37     @GET("/usuarios/{idUsuario}/proyectos")
38     Call<List<ProyectoData>> getProyectosByUsuario(@Path("idUsuario") Integer
39     idUsuario, @Header("Authorization") String key);
}

```

Para poder hacer uso de esta interfaz se tiene que configurar bajo ciertas características específicas como lo son la URL a la cual hará peticiones, el logger que se utilizará para poder observar las peticiones que se realizan y brindar una retroalimentación a la hora de hacer pruebas y por último el parser que se utilizará para trabajar y pasar de clases a datos que el API REST entienda y pueda utilizar, en este caso se utilizó el formato JSON. La definición de estas características se tiene en el siguiente código.

```

1 public class ServiceGenerator {
2     private static final String BASE_URL = "http://10.100.72.207:8000/";
3     private static Retrofit.Builder builder = new Retrofit.Builder().baseUrl(
4         BASE_URL)
5             .addConverterFactory(GsonConverterFactory.create());
6
7     private static Retrofit retrofit = builder.build();
8     private static OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
9
10    private static HttpLoggingInterceptor loggingInterceptor = new
11        HttpLoggingInterceptor();
12
13    public static <S> S createService(Class<S> serviceClass) {
14        if (!httpClient.interceptors().contains(loggingInterceptor)) {
15            loggingInterceptor.level(HttpLoggingInterceptor.Level.BODY);
16            httpClient.addInterceptor(loggingInterceptor);
17            builder.client(httpClient.build());
18            retrofit = builder.build();
19        }
20        return retrofit.create(serviceClass);
21    }
22}

```

Finalmente, en esta capa se tienen clases Java que después se mapean a objetos JSON y viceversa, para realizar esto se crea un POJO con los atributos que se necesitan además de agregar anotaciones de retrofit para que el parser pude hacer la conversión necesaria. Un ejemplo de esto es en la siguiente clase de java.

```
1 public class UsuarioData {
2     @SerializedName("id")
3     private Integer id;
4     @SerializedName("nombre")
5     private String nombre;
6     @SerializedName("apellido")
7     private String apellidos;
8     @SerializedName("username")
9     private String email;
10    @SerializedName("password")
11    private String password;
12    @SerializedName("responseCode")
13    private Integer responseCode;
14    @SerializedName("keyAuth")
15    private String keyAuth;
16    @SerializedName("currentPassword")
17    private String currentPassword;
18
19    public String getEmail() {
20        return email;
21    }
22    public void setEmail(String email) {
23        this.email = email;
24    }
25    public String getPassword() {
26        return password;
27    }
28    public void setPassword(String password) {
29        this.password = password;
30    }
31    public Integer getId() {
```

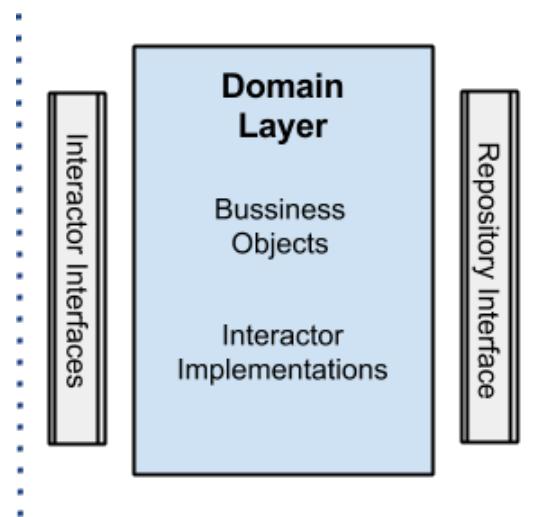
```
32         return id;
33     }
34     public void setId(Integer id) {
35         this.id = id;
36     }
37     public String getNombre() {
38         return nombre;
39     }
40     public void setNombre(String nombre) {
41         this.nombre = nombre;
42     }
43     public String getApellidos() {
44         return apellidos;
45     }
46     public void setApellidos(String apellidos) {
47         this.apellidos = apellidos;
48     }
49     public Integer getResponseCode() {
50         return responseCode;
51     }
52     public void setResponseCode(Integer responseCode) {
53         this.responseCode = responseCode;
54     }
55     public String getKeyAuth() {
56         return keyAuth;
57     }
58     public void setKeyAuth(String keyAuth) {
59         this.keyAuth = keyAuth;
60     }
61     public String getCurrentPassword() {
62         return currentPassword;
63     }
64     public void setCurrentPassword(String currentPassword) {
65         this.currentPassword = currentPassword;
66     }
```

### 1.1.1.2. Capa de dominio

En esta capa es la intermediaria entre las otras dos capas que se tienen, es donde se encuentran los casos de uso también conocidos como interactors como se muestra en la figura 1.3 en ellos la lógica del negocio es ejecutada es por esto que es el núcleo de la aplicación.

Es importante mencionar que esta capa, al ser la encargada del negocio es donde se hacen validaciones en la información y dicha información se adapta para que sea trabajada en la capa de presentación o en la de datos

Además de contener los casos de uso en esta capa se encuentran las entidades y se hace uso de los repositorios para acceder a la información proporcionada por la capa de datos.



**Figura 1.3:** Capa de dominio [2]

Para tener un control sobre posibles errores en la capa de presentación o en la capa de datos se utilizan códigos de resultados al igual que una clase que contiene el resultado que se puede presentar, así como la información que se le regresa a la capa de presentación. Se hace uso de genéricos para poder reutilizar esta clase en toda la aplicación y no duplicar código. La clase es la siguiente.

```

1 public class BusinessResult<T> {
2     private Integer code = ResultCodes.ERROR;
3     private T result;
4     private List<T> results;
5
6     public Integer getCode() {

```

```

7         return code;
8     }
9
10    public void setCode(Integer code) {
11        this.code = code;
12    }
13
14    public T getResult() {
15        return result;
16    }
17
18    public void setResult(T result) {
19        this.result = result;
20    }
21
22    public List<T> getResults() {
23        return results;
24    }
25
26    public void setResults(List<T> results) {
27        this.results = results;
28    }
29}

```

La forma en la que se utiliza esta clase en un caso de uso se presenta en el siguiente código que permite iniciar sesión.

```

1 public class UserInteractorImpl implements UserInteractor {
2     public static final String TAG = UserInteractorImpl.class.
3         getCanonicalName();
4
5     private UserRepository repository;
6
7     public UserInteractorImpl() {
8         repository = new UserRepositoryImpl();
9     }
10
11    @Override

```

```

10    public MutableLiveData<BusinessResult<UsuarioModel>> logIn(UsuarioModel
11        usuarioModel) {
12            BusinessResult<UsuarioModel> resultado = new BusinessResult<>();
13            MutableLiveData<BusinessResult<UsuarioModel>> mutableLiveData = new
14            MutableLiveData<>();
15            usuarioModel.setValidPassword(RN002.isValidPassword(usuarioModel.
16                getPassword()));
17            usuarioModel.setValidEmail(RN002.isValidEmail(usuarioModel.getEmail()));
18            if (usuarioModel.getValidEmail() && usuarioModel.getValidPassword())
19            {
20                UsuarioData usuarioData = new UsuarioData();
21                usuarioData.setEmail(usuarioModel.getEmail());
22                usuarioData.setPassword(usuarioModel.getPassword());
23                mutableLiveData = repository.login(usuarioData);
24            } else {
25                resultado.setCode(ResultCodes.RN002);
26                resultado.setResult(usuarioModel);
27                mutableLiveData.setValue(resultado);
28            }
29        }
30
31        return mutableLiveData;
32    }
33}

```

A su vez el caso de uso utiliza sus propios clases de java para presentar información al usuario en la capa de presentación así como controlar posibles errores en la información que ingrese el usuario los campos de los formularios, un ejemplo de este tipo de clases es el siguiente.

```

1 public class UsuarioModel {
2     private Integer id;
3     private String email;
4     private String password;
5     private String keyAuth;
6     private String name;
7     private String secondPassword;
8     private String lastname;

```

```
9     private String currentPassword;
10    private Boolean validLastName = false;
11    private Boolean validName = false;
12    private Boolean validSecondPassword = false;
13    private Boolean validEmail = false;
14    private Boolean validPassword = false;
15    private Boolean validCurrentPassword = false;
16
17    public String getEmail() {
18        return email;
19    }
20
21    public void setEmail(String email) {
22        this.email = email;
23    }
24
25    public String getPassword() {
26        return password;
27    }
28
29    public void setPassword(String password) {
30        this.password = password;
31    }
32
33    public Boolean getValidEmail() {
34        return validEmail;
35    }
36
37    public void setValidEmail(Boolean validEmail) {
38        this.validEmail = validEmail;
39    }
40
41    public Boolean getValidPassword() {
42        return validPassword;
43    }
```

```
44
45     public void setValidPassword(Boolean validPassword) {
46         this.validPassword = validPassword;
47     }
48
49     public Integer getId() {
50         return id;
51     }
52
53     public void setId(Integer id) {
54         this.id = id;
55     }
56
57     public String getKeyAuth() {
58         return keyAuth;
59     }
60
61     public void setKeyAuth(String keyAuth) {
62         this.keyAuth = keyAuth;
63     }
64
65     public String getName() {
66         return name;
67     }
68
69     public void setName(String name) {
70         this.name = name;
71     }
72
73     public String getSecondPassword() {
74         return secondPassword;
75     }
76
77     public void setSecondPassword(String secondPassword) {
78         this.secondPassword = secondPassword;
```

```
79     }
80
81     public String getLastname() {
82         return lastname;
83     }
84
85     public void setLastname(String lastname) {
86         this.lastname = lastname;
87     }
88
89     public Boolean getValidLastName() {
90         return validLastName;
91     }
92
93     public void setValidLastName(Boolean validLastName) {
94         this.validLastName = validLastName;
95     }
96
97     public Boolean getValidName() {
98         return validName;
99     }
100
101    public void setValidName(Boolean validName) {
102        this.validName = validName;
103    }
104
105    public Boolean getValidSecondPassword() {
106        return validSecondPassword;
107    }
108
109    public void setValidSecondPassword(Boolean validSecondPassword) {
110        this.validSecondPassword = validSecondPassword;
111    }
112
113    public String getCurrentPassword() {
```

```

114     return currentPassword;
115 }
116
117 public void setCurrentPassword(String currentPassword) {
118     this.currentPassword = currentPassword;
119 }
120
121 public Boolean getValidCurrentPassword() {
122     return validCurrentPassword;
123 }
124
125 public void setValidCurrentPassword(Boolean validCurrentPassword) {
126     this.validCurrentPassword = validCurrentPassword;
127 }
128 }
```

### 1.1.1.3. Capa de presentación

En esta capa como se muestra en la figura 1.4 se trabaja con la lógica relacionada a las interfaces que se tienen en la aplicación, es decir a actividades, fragmentos y archivos XML.

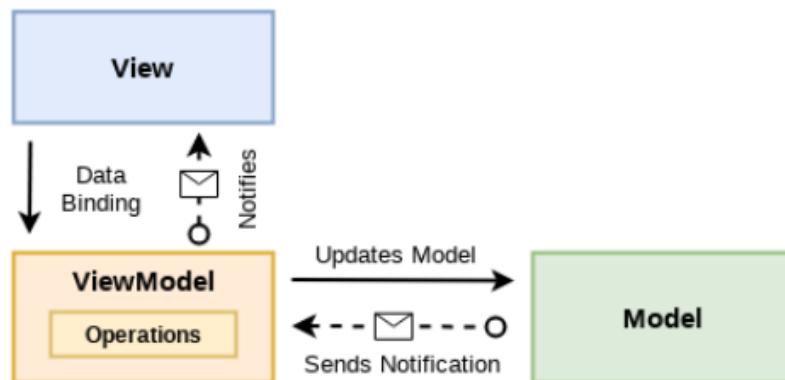


Figura 1.4: Capa de presentación [3]

En esta capa se pueden trabajar con patrones como MVC y MVP pero en este caso se utiliza el patrón MVVM cada uno con una función en particular. [3]

- **Modelo** Se encarga de representar la información que sera presentada en la vista.

- **Vista** Compuesta en este caso por las actividades y fragmentos de la aplicación, su tarea es mostrar la información, hacen uso de los viewmodels para poder realizar cambios en la interfaz.
- **ViewModel** El ViewModel sera el encargado de ejecutar los casos de uso o interactors con el objetivo de actualizar la vista de acuerdo a la información que presente el modelo.

## 1.2. Web

Esta sección tiene como objetivo presentar las principales características en el desarrollo de la aplicación Web.

### 1.2.1. Arquitectura de la aplicación

Para el desarrollo de la aplicación se implemento el patrón de diseño Modelo Vista Template (MTV), que como se menciono previamente, es el patrón que Django utiliza.

#### 1.2.1.1. Modelo

En esta capa se maneja todo el acceso a los datos de la aplicación. Django provee de un ORM que permite controlar una base de datos (PostgreSQL para este proyecto). De este modo, todas las tablas que componen la base de datos, están declaradas en esta capa.

#### 1.2.1.2. Vista

En esta capa se maneja la lógica del negocio. Se implementan las validaciones necesarias y se decide que datos deben de ser mostrados al usuario sin indicar como deben de ser presentados a diferencia del tradicional MVC. Esta capa debe de ser vista como un puente entre el Modelo y los Templates.

#### 1.2.1.3. Template

Esta es la capa de presentación. En ella se maneja la forma en la que serán mostrados los datos de la aplicación.

## 1.3. Conjuntos de entrenamiento

### 1.3.1. CROHME

El conjunto de datos de entrenamiento para el desarrollo de la red es el denominado **CROHME** por sus siglas en inglés: **Competition on Recognition of Online Handwritten Mathematical Expressions** publicado por los organizadores de la competencia internacional CROHME; para el caso del conjunto de entrenamiento fue posible reunir 7169 elementos que de acuerdo a investigación previa [4], son relativamente pocos para esperar una buena precisión, los elementos en el conjunto son archivos de tipo INKML.

#### 1.3.1.1. Formato del conjunto de datos

Como ya se mencionó, los elementos del conjunto son archivos de tipo INKML (Ink Markup Language) y que principalmente se compone de tres partes:

- Ink: Un conjunto de trazos definidos por puntos.
- Nivel de símbolo Ground-Truth: La segmentación e información de etiqueta de cada símbolo de la expresión.
- Ground-truth: La estructura MATHML de la expresión.

La información de *Ground-Truth* tanto de nivel de símbolo como de la expresión matemática fueron ingresadas manualmente por los colaboradores de la generación del conjunto, además, alguna información general es agregada en el archivo:

- Los canales (en este caso X, Y)
- La información del escritor (Identificación, entrega, edad, Mano dominante, género, etc ), si está disponible.
- Ground-Truth en L<sup>A</sup>T<sub>E</sub>X para fácilmente renderizarlo.

A continuación se muestra un ejemplo de un archivo del dataset que representa la expresión  $2^{-1}$ :

```

1 <ink xmlns="http://www.w3.org/2003/InkML">
2 <traceFormat>
3 <channel name="X" type="decimal"/>
4 <channel name="Y" type="decimal"/>
5 </traceFormat>
```

```

6 <annotation type="truth">$2^{-1}$</annotation>
7 <annotation type="UI">2011_IVC_CIEL_F696_E6</annotation>
8 <annotation type="copyright">LUNAM/IRCCyN</annotation>
9 <annotation type="writer">CIEL696</annotation>
10 <annotationXML type="truth" encoding="Content-MathML">
11   <math xmlns='http://www.w3.org/1998/Math/MathML'>
12     <msup>
13       <mn xml:id="2_1">2</mn>
14     <mrow>
15       <mo xml:id="-_1">-</mo>
16       <mn xml:id="1_1">1</mn>
17     </mrow>
18   </msup>
19 </math>
20 </annotationXML>
21 <trace id="0">
22 3849 2989, 3849 2989, 3847 2979, 3853 2964, 3868 2949, 3875 2949, 3887 2962,
23   3887 2974, 3880 2991, 3862 3011, 3822 3042, 3827 3052, 3839 3061, 3898
24   3049
25 </trace>
26 ...
27 <trace id="2">
28 4009 2905, 4009 2905, 4027 2900, 4049 2887, 4040 2947, 4040 2958
29 ...
30 </traceGroup>
31 </ink>
```

Sin embargo, para el propósito del trabajo terminal, el conjunto de datos no es útil en este formato, por lo que se tenía que transformar la información de los trazos en imágenes.

### 1.3.1.2. Conversión a imagen

Con la información que contiene el archivo INKML es posible generar una imagen con los puntos de los trazos en negro y fondo blanco. El primer reto fue identificar las etiquetas que contenían a los trazos, para ello se utilizó la biblioteca de Python `xml.etree` y una función de

terceros para poder utilizar dichos trazos posteriormente:

```

1 traces_data = []
2
3     tree = ET.parse(inkml_file_abs_path)
4     root = tree.getroot()
5     doc_namespace = "{http://www.w3.org/2003/InkML}"
6
7     'Stores traces_all with their corresponding id'
8     traces_all = [{ 'id': trace_tag.get('id'),
9                     'coords': [[round(float(axis_coord)) if float(axis_coord).
10                         is_integer() else round(float(axis_coord) * 10000) \
11                             for axis_coord in coord[1:].split(' ')] if coord.
12                             startswith(' ') \
13                             else [round(float(axis_coord)) if float(axis_coord).
14                                 is_integer() else round(float(axis_coord) * 10000) \
15                                     for axis_coord in coord.split(' ')]] \
16                             for coord in (trace_tag.text).replace('\n', '').split(',') ]
17             } \
18             for trace_tag in root.findall(doc_namespace + 'trace')]
```

Una vez obtenidos los trazos y con ayuda de matplotlib fueron separados como puntos  $x,y$  y utilizados en la función plot de matplotlib para posteriormente guardarla como imagen.

```

1 def inkml2img(input_path, output_path):
2     traces = get_traces_data(input_path)
3     plt.gca().invert_yaxis()
4     plt.gca().set_aspect('equal', adjustable='box')
5     plt.axes().get_xaxis().set_visible(False)
6     plt.axes().get_yaxis().set_visible(False)
7     plt.axes().spines['top'].set_visible(False)
8     plt.axes().spines['right'].set_visible(False)
9     plt.axes().spines['bottom'].set_visible(False)
10    plt.axes().spines['left'].set_visible(False)
11    for elem in traces:
12        ls = elem['trace_group']
13        for subls in ls:
```

```

14         data = np.array(subls)
15         x,y=zip(*data)
16         plt.plot(x,y,linewidth=2,c='black')
17         plt.savefig(output_path, bbox_inches='tight', dpi=100)
18         plt.gcf().clear()

```

Este procedimiento tenía que realizarse por cada uno de los elementos del conjunto de entrenamiento, además de también extraer la expresión matemática encerrada entre las etiquetas <annotation></annotation> con el atributo **type**, para ello nuevamente se utilizó la biblioteca de python xml.etree para acceder a los nodos del árbol directamente:

```

1 def inkml2tag(self, inkml_path):
2     tree = ET.parse(inkml_path)
3     root = tree.getroot()
4     prefix = "{http://www.w3.org/2003/InkML}"
5     GT_tag = [GT for GT in root.findall(prefix + 'annotation') if GT.attrib
6 == {'type': 'truth'}]
7     if GT_tag is None or len(GT_tag) == 0:
8         return ""
9     if GT_tag[0] is None or GT_tag[0].text is None:
10        return ""
11     return GT_tag[0].text

```

Con estos subscripts fue posible desarrollar una biblioteca que permite acceder a la carpeta con el conjunto de entrenamiento en formato inkml y guardarlos como imagen en otra carpeta junto con un archivo CSV contenido la ruta relativa de la imagen y la expresión en L<sup>A</sup>T<sub>E</sub>X correspondiente separados por coma.

```

1 converted_expressions/200924-1312-148.i.png, \left( \{ a + 7 \} \right)
2 converted_expressions/200926-1550-27.i.png, \mbox{ G }
3 converted_expressions/TrainData2_0_sub_71.i.png, \frac{\sin B + \sin C}{\cos B
+ \cos C}
4 converted_expressions/200923-1251-164.i.png, \zeta^{\phi}
5 converted_expressions/200923-1553-186.i.png, \sqrt{ \{ \mbox{ N } - \mbox{ P } \} }
6 converted_expressions/formulaire012-equation003.i.png, $4 \pi d^2$
7 converted_expressions/formulaire012-equation036.i.png, $a_{ns-3} + a_{n-1}s_2 +
a_{n-2}s_1 + 3a_{n-3} = 0$
8 converted_expressions/79_carlos.i.png, $p = \sqrt{a^2 + b^2 - 2ab \cos A}$

```

```

9 converted_expressions/200923-1553-313.i.png, \left( \sum{ \mbox{S} } \right) \
    right )
10 converted_expressions/formulaire010-equation073.i.png, $x^7-x^6-x^4-x^2-1$
```

### 1.3.1.3. Generador de secuencia de tokens

El archivo CSV generado con lo descrito anteriormente no es suficiente para cargarlo en TensorFlow, la expresión matemática en L<sup>A</sup>T<sub>E</sub>X debía ser expresada como una secuencia numérica, por lo que se necesitaba identificar a cada símbolo con un número único y conformar a la secuencia, de modo que la estructura del archivo CSV pasaría de tener la forma **RUTA\_IMAGEN, EXPRESION\_LATEX** a tener la forma **RUTA\_IMAGEN, SECUENCIA\_NUMÉRICA**, teniendo así una nueva representación del conjunto de entrenamiento conformada por una carpeta con las imágenes y un archivo CSV previamente descrito para poder cargarse en TensorFlow.

Para esto se desarrolló gracias a lex en Python un script para especificar los tokens tomando como base los símbolos especificados en la sección ??.

```

1 tokens = [
2     #'corta',
3     #'larga',
4     'alpha',
5     'pi',
6     'beta',
7     'gamma',
8     'lambda',
9     'sigma',
10    'mu',
```

---

```

1 def tokenizeDataset(self, file):
2
3     fileTokenized = open("tokenized_test.csv", "w")
4
5     myMap = dict()
6     for line in file:
7
8         self.lexer.input(",".join(line.split(",")[1:]))
9         #print(line.split(",")[1])
```

```

10     #print(repr(line.split(",") [1]))
11     token = self.lexer.token()
12     arr = []
13     while token is not None:
14         print(token)
15         myMap[token.type]= token.value #fileMap.write(token.type+","+token.
16         value+"\n")
17         arr.append(Rules.tokens.index(token.type) + 1 ) #+1 means shift one
due to 0 is reserved
18         token = self.lexer.token()
19         fileTokenized.write(line.split(",") [0] + "," + toString(arr) + "\n")
fileTokenized.close()

1 converted_expressions/200924-1312-148.i.png,64 33 41 74 47 58 42 65 34
2 converted_expressions/200926-1550-27.i.png,41 115 42
3 converted_expressions/TrainData2_0_sub_71.i.png,12 41 14 100 47 14 99 42 41
   16 100 47 16 99 42
4 converted_expressions/200923-1251-164.i.png,41 41 78 42 42 62 41 9 42
5 converted_expressions/200923-1553-186.i.png,13 41 41 107 42 46 41 109 42
   42 42
6 converted_expressions/formulaire012-equation003.i.png,55 2 79 62 53
7 converted_expressions/formulaire012-equation036.i.png,74 61 75 95 61 54 47 74
   61 41 75 46 52 42 95 61 53 47 74 61 41 75 46 53 42 95 61 52 47 54 74 61
   41 75 46 54 42 48 51
8 converted_expressions/79_carlos.i.png,86 48 13 41 74 62 53 47 77 62 53 46 53
   74 77 16 102 42
9 converted_expressions/200923-1553-313.i.png,64 33 18 41 41 108 42 42 65 34
10 converted_expressions/formulaire010-equation073.i.png,73 62 58 46 73 62 57 46
   73 62 55 46 73 62 53 46 52

```

Es importante destacar que se debe también guardar este mapeo único y no alterar el orden, por lo que de requerir agregar nuevos símbolos al conjunto es necesario hacerlo al final de los ya existentes, ya que la red entrenada devuelve secuencias numéricas que deben mapearse para tener la correspondiente expresión en L<sup>A</sup>T<sub>E</sub>X.

### 1.3.2. Harvard 100k

El segundo conjunto de entrenamiento utilizado fue el provisto por [5]. Los investigadores que realizaron el paper liberaron el conjunto de entrenamiento que recabaron, el cual cuenta con alrededor de cien mil expresiones matemáticas escritas a computadora.

Este conjunto de entrenamiento puede descargarse libremente en la página provista por el artículo. Consiste en un conjunto de imágenes png sin preprocessar y sus respectivos resultados esperados (Ground-Truth). Así mismo, los autores del artículo proveen scripts de normalización y de preprocessado para el tratamiento de las imágenes. Estos scripts se pueden encontrar en el github del artículo.

Se procedió a preprocessar las imágenes con los scripts provistos por los investigadores de Harvard y se obtuvo un conjunto de entrenamiento con 78000 imágenes. Debido a que en el presente Trabajo Terminal las estructuras matriciales no están contempladas, se descartaron todas las imágenes que contuvieran matrices, arreglos o listas, así como aquellas que produjeron errores tras ser procesadas. El conjunto final contiene 73000 imágenes de distintos tamaños los cuales son: (200,50), (240,40), (280,40), (360,60), (160,40), (360,50), (120,50), (320,50), (400,50), (360,10), (360,40), (200,40), (320,40), (280,50) y (240,50). En la Figura 1.5 se puede ver un ejemplo de una imagen del conjunto de entrenamiento de Harvard.

$$d(a.b) = (da).b + a.(db)$$

**Figura 1.5:** Ejemplo de una imagen del conjunto de entrenamiento Harvard 100k

### 1.3.3. Normalización

Entre los scripts provistos por los investigadores de Harvard, se encuentra un normalizador. Este código, realiza una serie de transformaciones seguras con el fin de estandarizar ciertas inconsistencias con las expresiones en LATEX. Un ejemplo podría ser las distintas formas que se tienen para expresar un exponente:  $a\hat{b}$  y  $a \{ b \}$  producirían el mismo resultado. Para mejorar el rendimiento de la red, ambos conjuntos de entrenamiento fueron preprocessados con el script de normalizado.

Cabe mencionar que ambos conjuntos de entrenamiento fueron unificados mediante la combinación de su Ground-Truth, por lo que es posible utilizarlos indistintamente. Para unificarlos se proceso el conjunto CROHME para convertir sus tokens en tokens del conjunto de Harvard. El código usado se muestra a continuación.

```

1 # get the tokens
2 tokens_file = open('latex_vocab.txt', 'r')
3 tokens = dict()
4 count = 0

```

```
5
6 for line in tokens_file:
7     count += 1
8     tokens[line.strip()] = count
9
10 tokens_file.close()
11
12 expressions_file = open('expressions.txt', 'r')
13 expressions_norm_file = open('expressions.norm.lst', 'r')
14 expressions_norm = expressions_norm_file.read().split('\n')
15 tokenized = open('tokenized.csv', 'w')
16 MAXLENGTH = 103
17 i = 0
18
19 for line in expressions_file:
20
21     if 'Object' in expressions_norm[i] or '\\div' in expressions_norm[i]:
22         i += 1
23         continue
24
25     expressions_split = line.split('$')
26     expression = expressions_norm[i].split(' ')
27     sequence = '1000'
28
29     for e in expression:
30         sequence += ' ' + str(tokens[e])
31
32     sequence += ' 1001'
33
34     for x in range(0, MAXLENGTH - len(expression)):
35         sequence += ' 0'
36
37     tokenized.write(expressions_split[0] + ',' + sequence + '\n')
38     i += 1
```

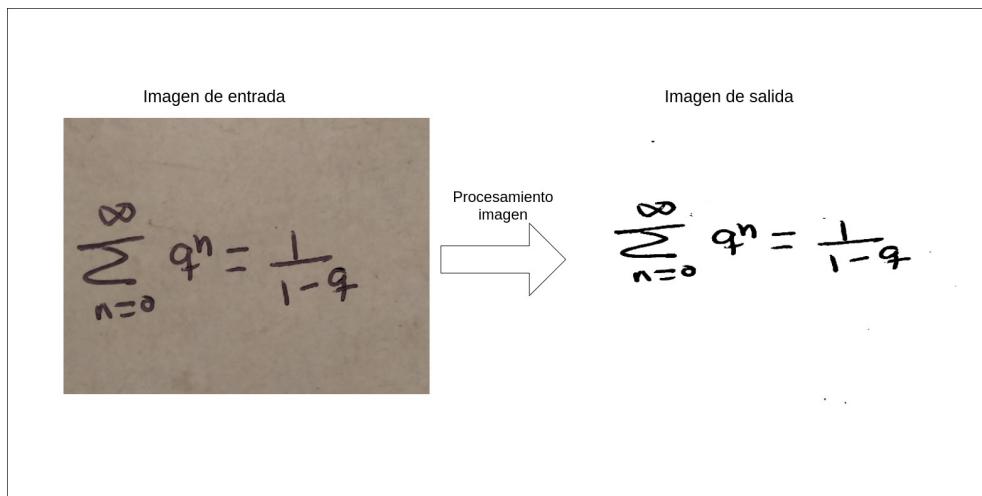
## 1.4. Desarrollo módulo de análisis de imágenes

Las imágenes generadas para el conjunto de entrenamiento se formaron insertando los puntos en fondo blanco, teniendo como resultado una imagen binaria (contenido en negro y fondo blanco) por lo que al considerar que las fotografías tomadas por un dispositivo móvil no iba a generar por sí mismo un formato semejante fue necesario desarrollar un módulo para esta tarea.

$$\int e^{x^2} x^3 dx$$

**Figura 1.6:** Imagen de ejemplo del conjunto de entrenamiento generado

El proceso consiste en convertir el área de la expresión matemática en negro y el resto en fondo blanco, para ello se consideraron las técnicas mencionadas en y específicamente se probaron los algoritmos de Otsu y Sauvola [6].



**Figura 1.7:** Fotografía procesada con algoritmo de Sauvola

De acuerdo a lo mencionado en el marco teórico existen técnicas para binarizar una imagen que calculan el umbral de manera automática, dos de estos algoritmos son el algoritmo de Otsu y el algoritmo de Sauvola, a continuación se describen brevemente y se dan detalles de la implementación o uso según el caso.

### 1.4.1. Algoritmo de Otsu

El método de Otsu evita tener que elegir un valor de umbral y lo determina automáticamente.

Consideré una imagen con solos dos valores distintos (imagen bimodal), donde el histograma consistiría solo de dos picos. Un buen valor de umbral sería a la mitad de estos dos valores. Similarmente el método de Otsu determina un valor de umbral óptimo global del histograma de la imagen.

Al tratarse de una imagen bimodal, el algoritmo intenta encontrar un valor de umbral ( $t$ ) que minimice la varianza ponderada dentro de la clase, dada por la relación:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \quad (1.1)$$

donde:

$$\begin{aligned} q_1(t) &= \sum_{i=1}^t P(i) \wedge q_2(t) = \sum_{i=t+1}^I P(i) \\ \mu_1(t) &= \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \wedge \mu_2(t) = \sum_{i=t+1}^I \frac{iP(i)}{q_2(t)} \\ \sigma_1^2(t) &= \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \wedge \sigma_2^2(t) = \sum_{i=t+1}^I [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)} \end{aligned} \quad (1.2)$$

```

1 # Filtro Gaussiano con ventana (5,5)
2 blur = cv.GaussianBlur(img,(5,5),0)
3 # Busca el histograma y su funcion de distribucion acumulada
4 hist = cv.calcHist([blur],[0],None,[256],[0,256])
5 hist_norm = hist.ravel()/hist.sum()
6 Q = hist_norm.cumsum()
7 bins = np.arange(256)
8 fn_min = np.inf
9 thresh = -1
10 for i in xrange(1,256):
11     p1,p2 = np.hsplit(hist_norm,[i]) # probabilidades
12     q1,q2 = Q[i],Q[255]-Q[i] # suma de clases
13     if q1 < 1.e-6 or q2 < 1.e-6:
14         continue
15     b1,b2 = np.hsplit(bins,[i]) # pesos
16     # encuentra medias y varianzas
17     m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
18     v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
19     # calcula la funcion de minimizacion

```

```
20     fn = v1*q1 + v2*q2
21     if fn < fn_min:
22         fn_min = fn
23         thresh = i
```

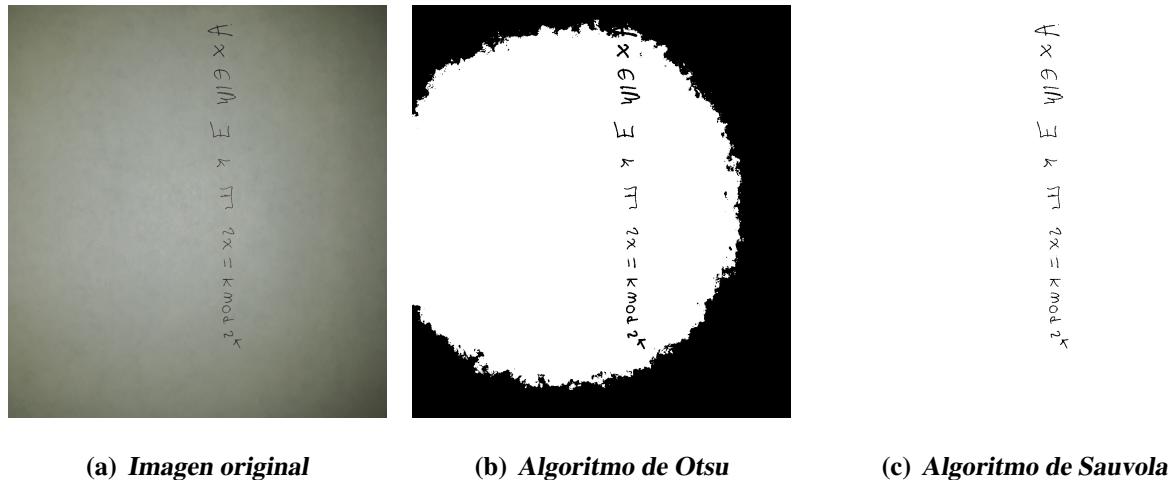
### 1.4.2. Algoritmo de Sauvola

El algoritmo de Sauvola tiene una implementación más extensa por lo que se dispuso de la utilización de la implementación contenida en **skimage** dentro del paquete filters.

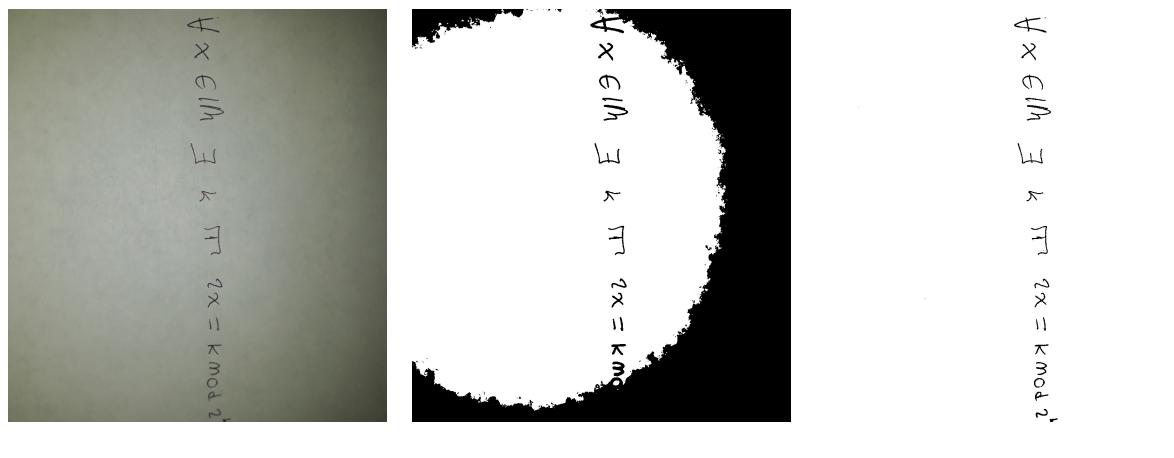
```
1 from skimage import morphology
2 import numpy as np
3 import skimage
4 import os, sys

1 def processBinarization(self, algorithm = ImageAlgorithm.SAUVOLA):
2     if algorithm == ImageAlgorithm.SAUVOLA:
3         image = skimage.io.imread(fname=self.pathImg, as_gray=True)
4         thresh_sauvola = threshold_sauvola(image, window_size=51)
5         self.binary_sauvola = image > thresh_sauvola
```

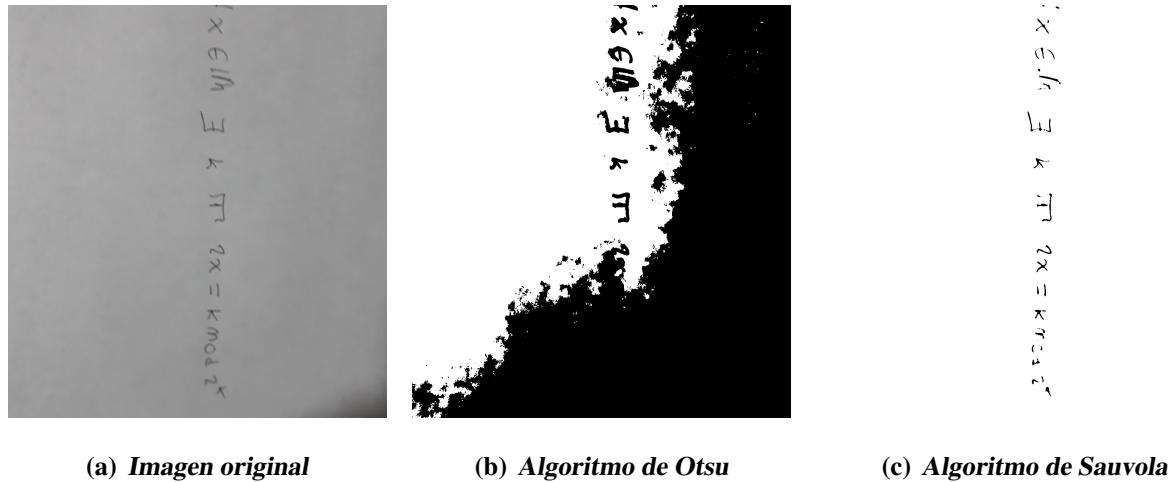
### 1.4.3. Resultados



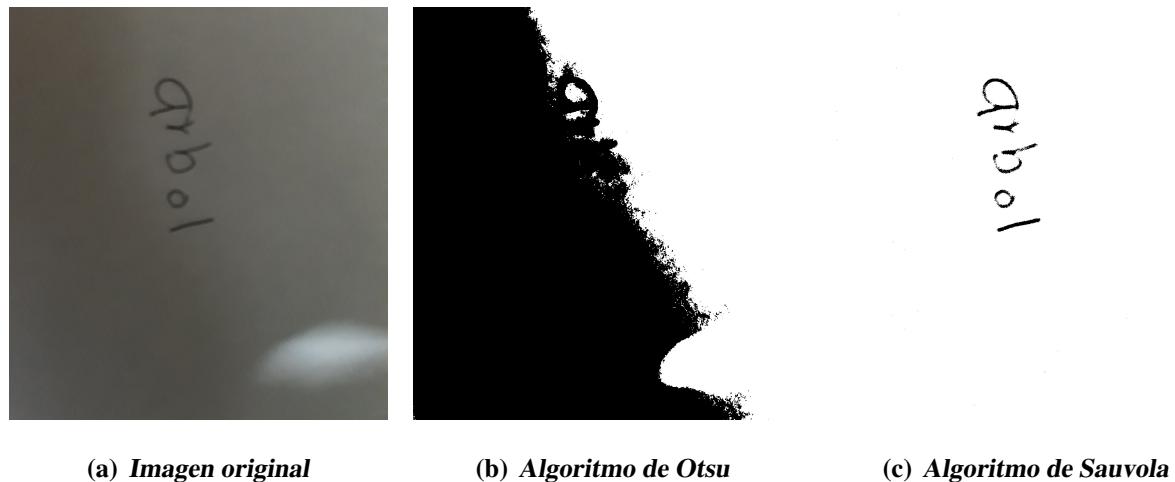
**Figura 1.8:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



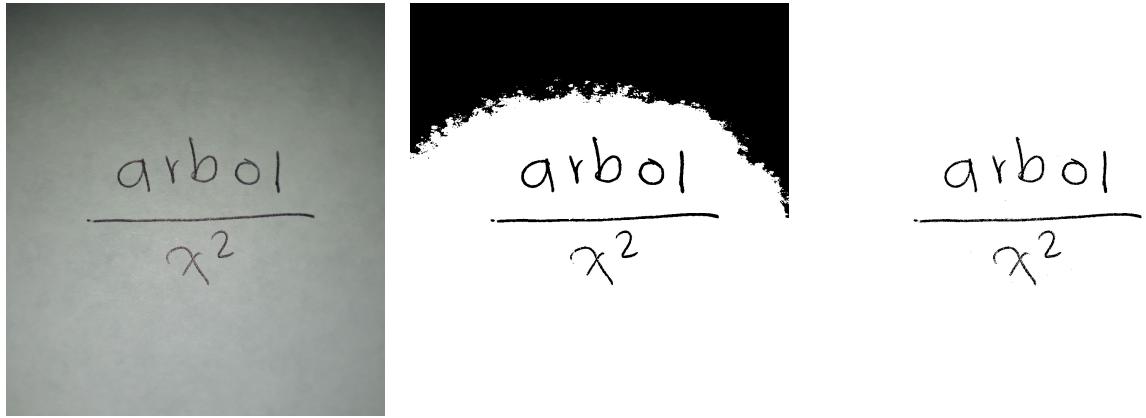
**Figura 1.9:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



**Figura 1.10:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



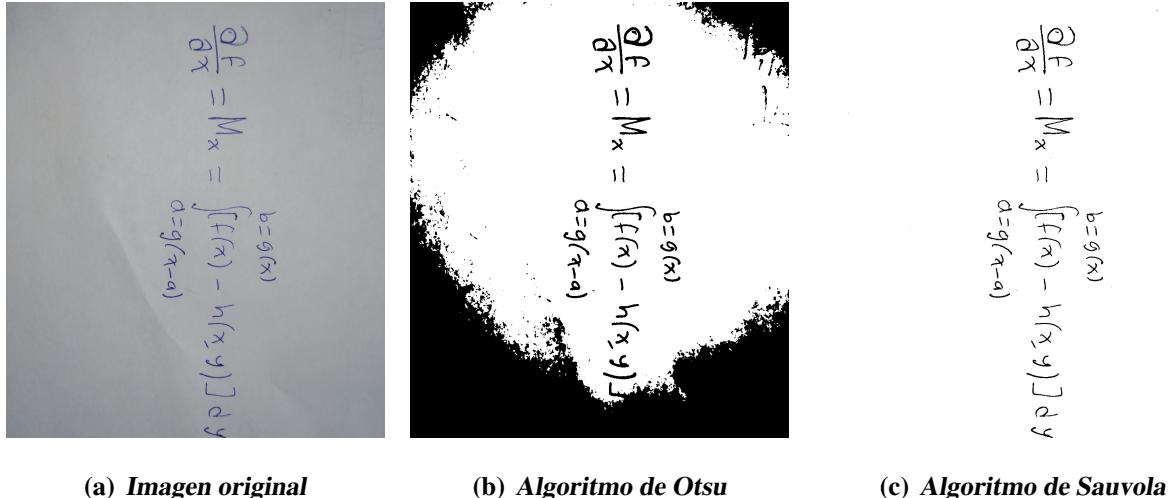
**Figura 1.11:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



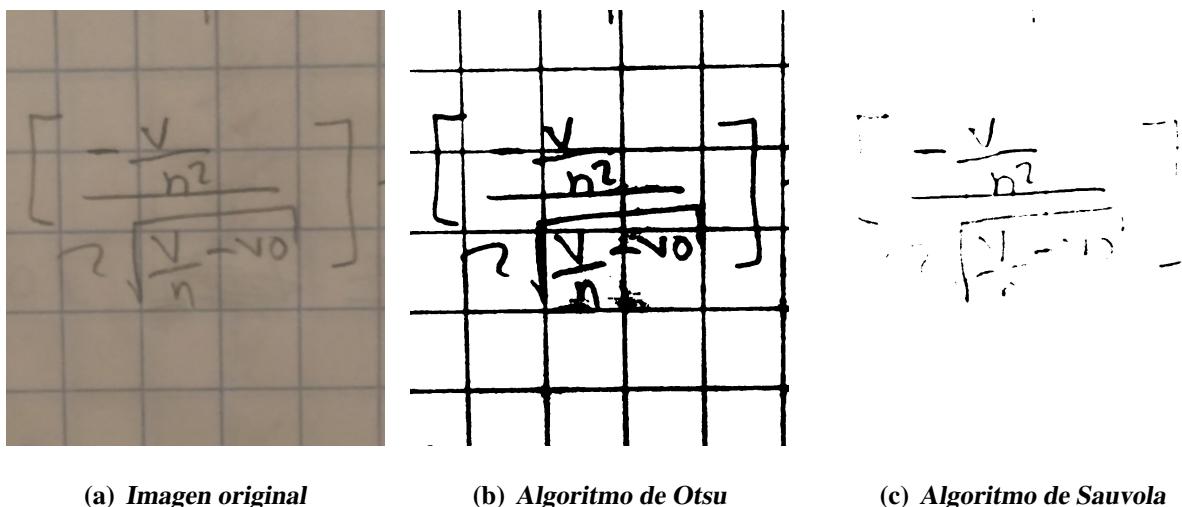
**Figura 1.12:** Resultados de la aplicación del algoritmo de Otsu y Sauvola

$$\frac{\partial f}{\partial x} = M_x = \frac{x}{b-a}$$

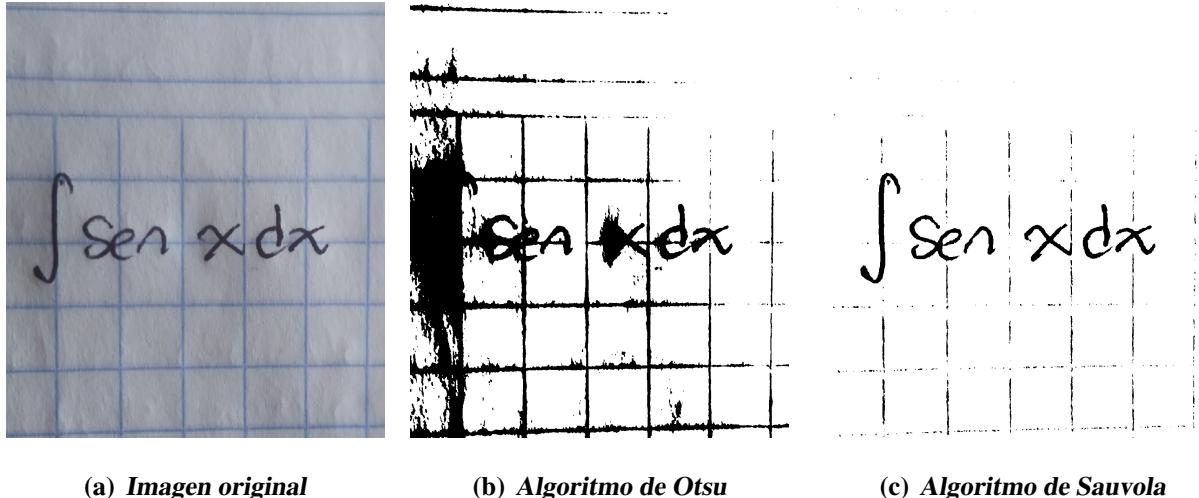
**Figura 1.13:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



**Figura 1.14:** Resultados de la aplicación del algoritmo de Otsu y Sauvola

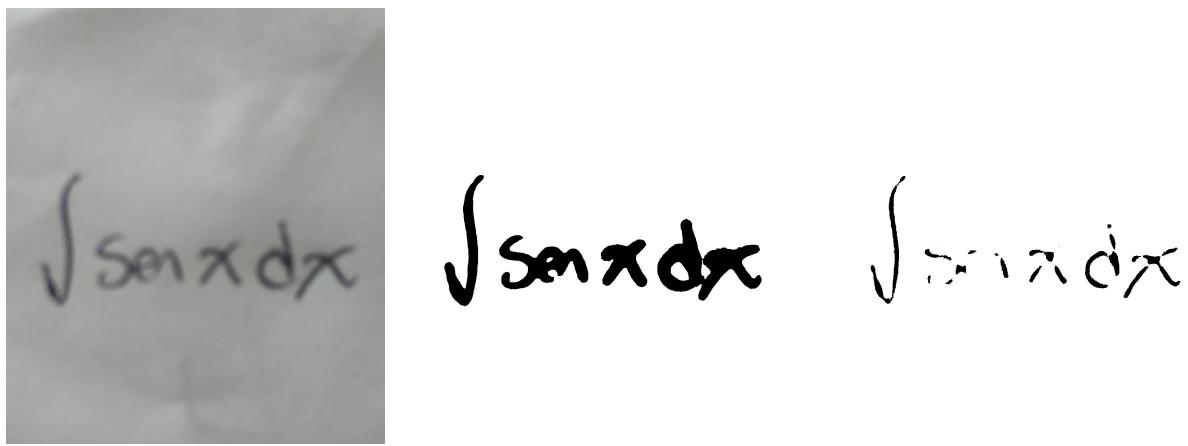


**Figura 1.15:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



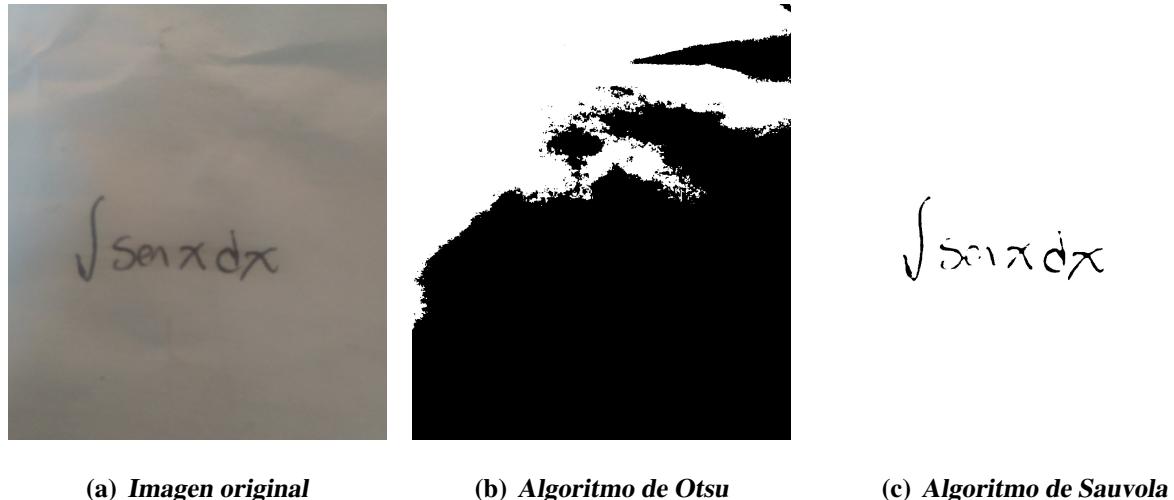
(a) *Imagen original*      (b) *Algoritmo de Otsu*      (c) *Algoritmo de Sauvola*

**Figura 1.16:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



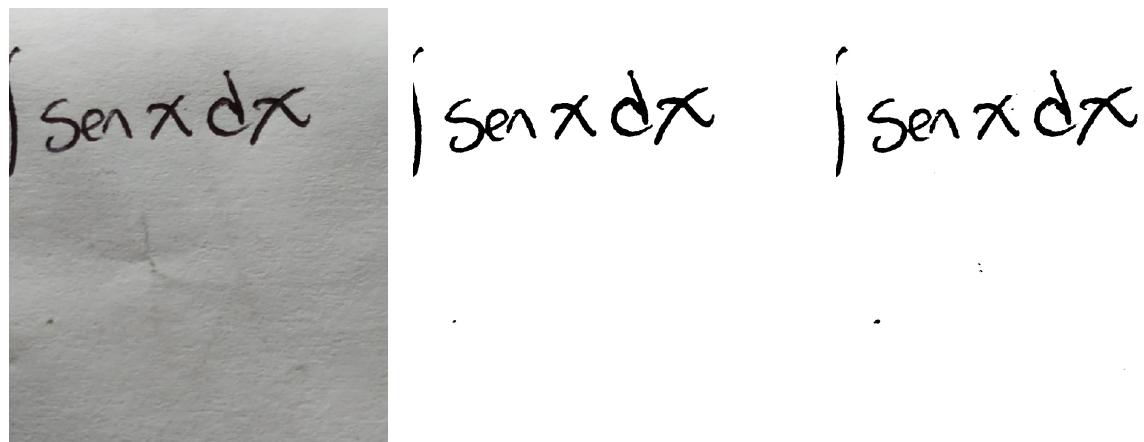
(a) *Imagen original*      (b) *Algoritmo de Otsu*      (c) *Algoritmo de Sauvola*

**Figura 1.17:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



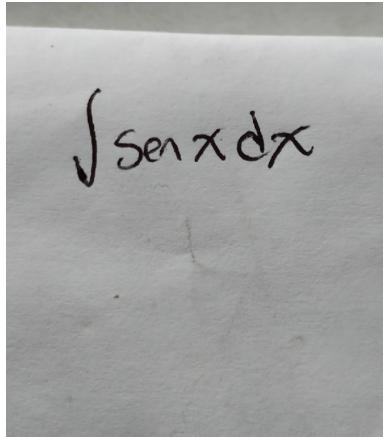
(a) *Imagen original*      (b) *Algoritmo de Otsu*      (c) *Algoritmo de Sauvola*

**Figura 1.18:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



(a) *Imagen original*      (b) *Algoritmo de Otsu*      (c) *Algoritmo de Sauvola*

**Figura 1.19:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



(a) *Imagen original*

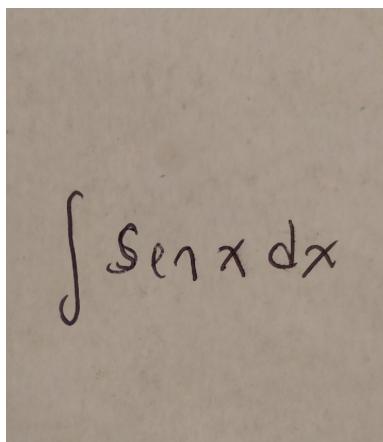
$$\int \sin x dx$$

(b) *Algoritmo de Otsu*

$$\int \sin x dx$$

(c) *Algoritmo de Sauvola*

**Figura 1.20:** Resultados de la aplicación del algoritmo de Otsu y Sauvola



(a) *Imagen original*

$$\int \sin x dx$$

(b) *Algoritmo de Otsu*

$$\int \sin x dx$$

(c) *Algoritmo de Sauvola*

**Figura 1.21:** Resultados de la aplicación del algoritmo de Otsu y Sauvola

## 1.5. Reconocimiento de expresiones matemáticas

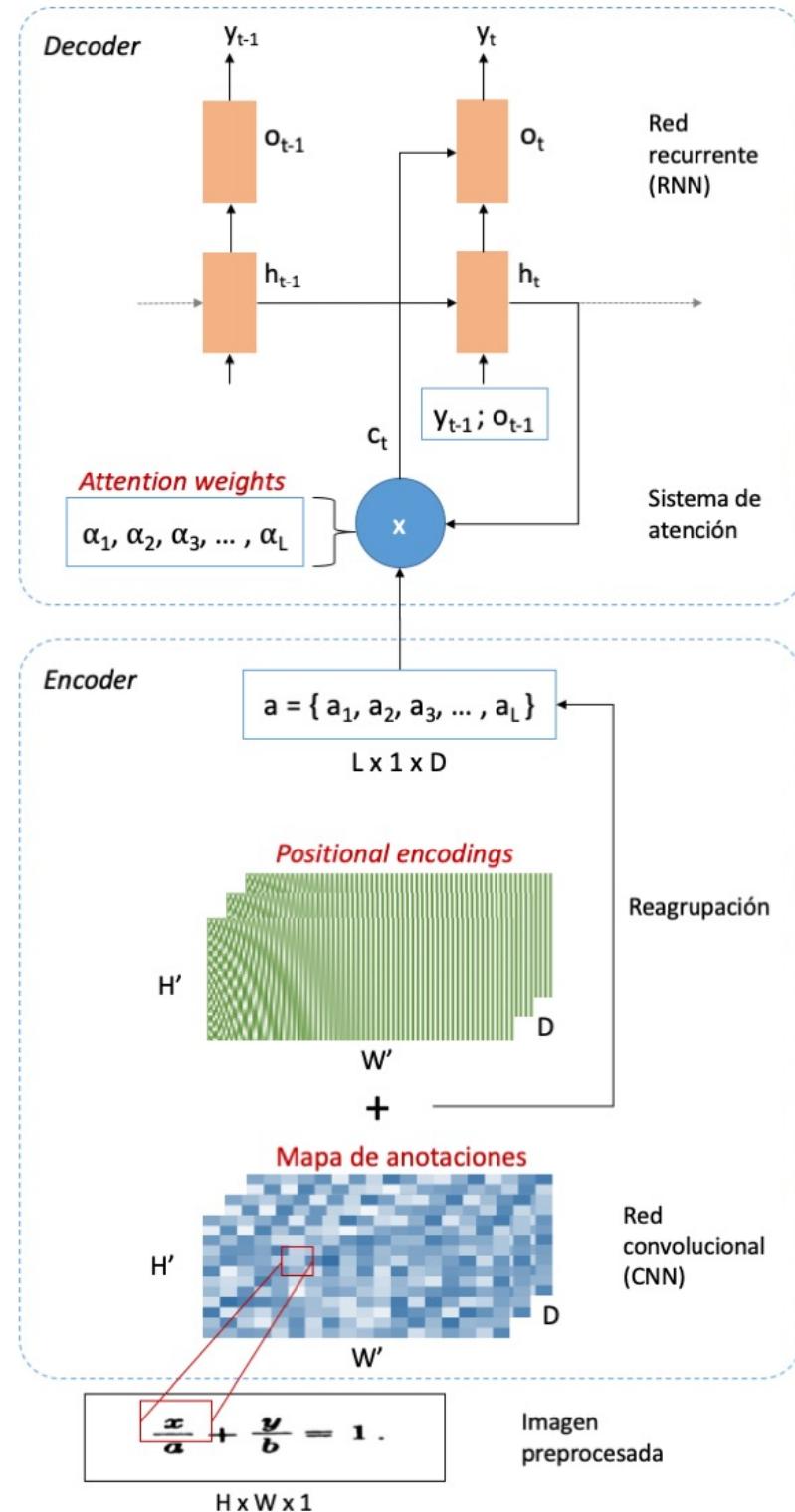
El problema de reconocer expresiones matemáticas en imágenes conciste en encontrar una función capaz de convertir una imagen preprocesada en una secuencia de caracteres que describa en su totalidad la expresión incluida en la imagen. La imagen de entrada  $x$  es preprocesada para obtener una imagen en escala de grises de tal forma que  $x \in \mathbb{R}^{H \times W \times 1}$ , siendo  $H$  y  $W$  la altura y el ancho de la imagen respectivamente. La secuencia de caracteres objetivo  $y$  es de la forma  $y_1, y_2, \dots, y_k$  siendo  $k$  la longitud de la secuencia y  $y_i$  es un carácter válido de `LATEX` para el presente Trabajo Terminal.

Para resolver el problema de reconocer expresiones matemáticas, existen aproximaciones secuenciales o globales. De acuerdo con [7] ambos métodos implementados de una forma convencional presentan las siguientes limitaciones: 1) La difícil segmentación de símbolos, 2) el análisis estructural es comúnmente basado en gramáticas libres de contexto, lo cual requiere un conocimiento previo de las expresiones a reconocer para diseñar una gramática, 3) la complejidad de los algoritmos de parseo se incrementa con el tamaño de la gramática usada.

Recientemente el modelo de *Encoder-Decoder* con un sistema de atención comenzó a cobrar relevancia por presentar excelentes resultados en campos del machine learning como el procesamiento del lenguaje natural, segmentación de imágenes e *Image Captioning*. Esta aproximación es completamente opuesta a las soluciones convencionales debido a que es una solución entrenable *end-to-end*, es decir, el modelo puede entrenarse como un todo y no por separado; su capacidad de predicción depende completamente del conjunto de entrenamiento por lo que para mejorar el modelo solo se necesita incrementar la cantidad de datos disponibles sin hacer cambios a la arquitectura de la red; la segmentación de símbolos puede hacerse automáticamente a través de la atención. Por las razones expuestas, se decidió utilizar este modelo de deep learning en el presente Trabajo Terminal.

### 1.5.1. Modelo

Como se mencionó anteriormente la arquitectura de la red neuronal es del tipo *Encoder-Decoder*, en la Figura 1.22 se muestra un esquema de la red neuronal implementada. La traducción de imágenes se realiza de la siguiente manera: Primero se extraen las características de la imagen en forma de grilla utilizando una red convolucional (CNN), luego se procedera a utilizar una versión de dos dimensiones de los *positional encodings* mostrados en el marco teórico y finalmente se procede a reagrupar las filas de la grilla en una sola, esta es la fase de codificación.

Figura 1.22: Modelo *Encoder-Decoder* de la red neuronal.

Número de capa	Capa Convolutinal				Capa de Max Pooling			
	Kernel	Strides	Padding	BN	Size	Strides	Padding	
1	64-(3,3)	(1,1)	(1,1)	-	(2,2)	(2,2)	(2,2)	
2	128-(3,3)	(1,1)	(1,1)	-	(2,2)	(2,2)	(0,0)	
3	256-(3,3)	(1,1)	(1,1)	si			-	
4	256-(3,3)	(1,1)	(1,1)	-	(2,1)	(2,1)	(0,0)	
5	512-(3,3)	(1,1)	(1,1)	si	(1,2)	(1,2)	(0,0)	
6	512-(3,3)	(1,1)	(0,0)	si			-	

**Tabla 1.1:** Especificación de la CNN. El 'Kernel' esta denotado como *número de filtros-(dimensiones del filtro)*.

Para la decodificación, las características codificadas serán usadas por una red neuronal recurrent RNN con un sistema de atención. El *Decoder* implementa un modelo de lenguaje condicional sobre un vocabulario  $\lambda$ , el cual esta dado por el conjunto de entrenamiento y que será expuesto más adelante.

### 1.5.1.1. Encoder

La extracción de características de la imagen de entrada  $x$  se realiza con una CNN mult capa con *max pooling* y *batch normalization* la cual es ahora un estandar y está completamente inspirada en [5]. La Tabla 1.1 sumariza la arquitectura de la red utilizada. Las características extraídas están almacenadas en vector  $v \in \mathbb{R}^{H' \times W' \times D}$ , siendo  $H'$  y  $W'$  las nuevas altura y anchura respectivamente y  $D$  el tamaño del último filtro de la red,  $D = 512$  en este caso.

Se procede a calcular los *positional encodings*. Existen dos formas de hacer esto, la primera es con parámetros aprendidos y la segunda es con valores fijos. El artículo [5] utiliza una RNN bidireccional con el vector  $v$  como entrada para aprender estos positional encodings. No obstante, acorde con los inventores de esta técnica [8], los positional encodings fijos producen resultados muy similares a los aprendidos. Así, se optó por utilizar positional encodings fijos y se utilizó la generalización a dos dimensiones propuesta por [9]. El vector de positional encodings  $pe \in \mathbb{R}^{H' \times W' \times D}$  se obtiene con las siguientes ecuaciones:

$$pe(w, h, 2i) = \sin\left(\frac{w}{10000^{\frac{4i}{D}}}\right) \quad (1.3)$$

$$pe(w, h, 2i+1) = \cos\left(\frac{w}{10000^{\frac{4i}{D}}}\right) \quad (1.4)$$

$$pe(w, h, 2j+D/2) = \sin\left(\frac{h}{10000^{\frac{4j}{D}}}\right) \quad (1.5)$$

$$pe(w, h, 2j+1+D/2) = \cos\left(\frac{h}{10000^{\frac{4j}{D}}}\right) \quad (1.6)$$

Con esto, se puede obtener el vector de anotaciones  $a$  como sigue:

$$a = v + pe \quad (1.7)$$

Por último, se reagrupa el vector de anotaciones  $a$  de tal forma que  $a \in \mathbb{R}^{LxD}$ , donde  $L = H'xW'$ , así el vector final de salida del *Encoder* es  $a = \{a_1, a_2, \dots, a_L\}$  con  $a_i \in \mathbb{R}^D$ .

### 1.5.1.2. Decoder

Las secuencias  $y_t$  se generan a partir de un modelo de lenguaje condicional que utiliza las anotaciones  $a$  obtenidas en el Encoder. La salida del decoder es la probabilidad de obtener el siguiente token dada la historia de tokens previamente generados y el vector de contexto obtenido mediante la atención.

El modelo de lenguaje condicional es como sigue:

$$p(y_t | y_1, \dots, y_{t-1}, a) = \text{softmax}(W^{out} o_t) \quad (1.8)$$

con

$$o_t = \tanh(W^c[h_t; c_t]) \quad (1.9)$$

$$h_t = LSTM(h_{t-1}, [y'_{t-1}; o_{t-1}]) \quad (1.10)$$

$$y'_t = E y_{t-1} \quad (1.11)$$

donde  $W^{out}$ ,  $W^c$  y  $E$  son parámetros que serán aprendidos. La matriz  $E$ , es también conocida en la literatura como la matriz de *embeddings*. El vector  $h_t$  es utilizado para sumarizar toda la historia de decodificación y el vector de contexto  $c_t$  es usado para capturar la información de contexto de la grilla de características de la imagen  $x$ . Los corchetes  $[;]$ , indican una concatenación entre los vectores dentro de ellos.

Para los estados iniciales  $h_0$  y  $o_0$  se utilizó un perceptrón multicapa (MLP) que aprendiera los mejores valores iniciales. Las ecuaciones que describen  $h_0$  y  $o_0$  respectivamente son:

$$h_0 = \tanh(W_{h_0}a + b_{h_0}) \quad (1.12)$$

$$o_0 = \tanh(W_{o_0}a + b_{o_0}) \quad (1.13)$$

En cada instante  $t$ , el vector de contexto es calculado. Dado que la mayoría de anotaciones  $a$  no contienen información útil para decidir el mejor candidato  $y_t$ , el modelo debe de saber cuáles anotaciones atender. Esta tarea, es delegada a un sistema de atención como el propuesto por [10]:

$$e_t = u(h_t, a) \quad (1.14)$$

$$\alpha_t = \text{softmax}(e_t) \quad (1.15)$$

$$c_t = \phi(\alpha_t, a) \quad (1.16)$$

$\phi$  y  $u$  son funciones que pueden ser escogidas libremente, en este caso se utilizaron las mismas que en [5]:

$$e_{it} = \beta^T \tanh(W_h h_{i-1} + W_v a) \quad (1.17)$$

$$c_t = \sum_i^L \alpha_{it} a_i \quad (1.18)$$

donde  $W_h$  y  $W_v$  son parámetros que serán aprendidos. Con esto el modelo de atención puede saber que parte de las anotaciones es importante en el instante  $t$ . Una consecuencia de esto, es que la segmentación de símbolos es delegada también a este sistema.

Finalmente, se decidió modificar la atención para implementar la técnica del *Coverage Vector* como en [4]. El coverage vector  $F$  se incorporó para dar más información al sistema de atención sobre el historial de la misma atención, de este modo, es posible mejorar la cobertura global del sistema lo cual ayuda a reducir los errores de símbolos no parseados. El vector de coverage se calcula como sigue:

$$\eta_t = \sum_l^{t-1} \alpha_l \quad (1.19)$$

$$F = Q * \eta_t \quad (1.20)$$

Donde  $\eta_t$  es la suma de todos los pesos de las atenciones pasadas y  $Q$  es una matriz de pesos que serán aprendidos. De este modo, la Ecuación 1.17 se modifica por:

$$e_{it} = \beta^T \tanh(W_h h_{i-1} + W_v a + W_f F) \quad (1.21)$$

de donde  $W_f$  es una matriz de pesos que serán aprendidos.

### 1.5.1.3. Implementación

El lenguaje de programación utilizado para desarrollar la red fue Python 3 junto con el framework para deep learning TensorFlow 2. Las unidades usadas en el estado  $h_t$  de la LSTM son 512 y 80 para la dimensión de *embedding* para los tokens 80. El algoritmo de optimización utilizado fue Adam con la Crosentropia Categórica como función de perdida.

El código que implementa la red es el siguiente:

```

1 def get_positional_encoding_2d(height, width, d_model):
2
3     positional_encodings = np.zeros((height, width, d_model))
4     d_model = int(d_model / 2)
5     h_vector = np.arange(height)
6     w_vector = np.expand_dims(np.arange(width), axis=1)
7
8     div_term = np.arange(d_model) // 2
9     div_term = np.exp((-2 * div_term / d_model) * np.log(10000.0))
10
11    positional_encodings[:, :, 0:d_model:2] = np.sin((positional_encodings[:, :, 0:d_model:2] + div_term[0::2]) * w_vector)
12    positional_encodings[:, :, 1:d_model:2] = np.cos((positional_encodings[:, :, 1:d_model:2] + div_term[1::2]) * w_vector)
13    positional_encodings[:, :, d_model::2] = np.sin((positional_encodings[:, :, d_model::2] + div_term[0::2]) * h_vector[:, np.newaxis, np.newaxis])
14    positional_encodings[:, :, d_model+1::2] = np.cos((positional_encodings[:, :, d_model+1::2] + div_term[0::2]) * h_vector[:, np.newaxis, np.newaxis])
15
16    return positional_encodings
17
18
19 class Encoder(tf.keras.Model):
20
21     def __init__(self):
22         super(Encoder, self).__init__()
23
24         self.pooling1 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides
```

```
25         =(2,2), padding='same')
26             self.pooling2 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides
27             =(2,2), padding='same')
28             self.pooling3 = tf.keras.layers.MaxPool2D(pool_size=(2,1), strides
29             =(2,1), padding='same')
30             self.pooling4 = tf.keras.layers.MaxPool2D(pool_size=(1,2), strides
31             =(1,2), padding='same')
32
33         self.conv1 = tf.keras.layers.Conv2D(64, (3,3), (1,1), 'same',
34             activation='relu')
35         self.conv2 = tf.keras.layers.Conv2D(128, (3,3), (1,1), 'same',
36             activation='relu')
37         self.conv3 = tf.keras.layers.Conv2D(256, (3,3), (1,1), 'same',
38             activation='relu')
39         self.conv4 = tf.keras.layers.Conv2D(256, (3,3), (1,1), 'same',
40             activation='relu')
41         self.conv5 = tf.keras.layers.Conv2D(512, (3,3), (1,1), 'same',
42             activation='relu')
43         self.conv6 = tf.keras.layers.Conv2D(512, (3,3), (1,1), 'valid',
44             activation='relu')
45
46         self.batch_normalization1 = tf.keras.layers.BatchNormalization()
47         self.batch_normalization2 = tf.keras.layers.BatchNormalization()
48         self.batch_normalization3 = tf.keras.layers.BatchNormalization()
49
50     def call(self, x, training=False):
51
52         y = self.conv1(x)
53         y = self.pooling1(y)
54         y = self.conv2(y)
55         y = self.pooling2(y)
56         y = self.conv3(y)
57         y = self.batch_normalization1(y, training)
58         y = self.conv4(y)
59         y = self.pooling3(y)
```

```

50     y = self.conv5(y)
51     y = self.batch_normalization2(y, training)
52     y = self.pooling4(y)
53     y = self.conv6(y)
54     y = self.batch_normalization3(y, training)
55
56     y += get_positional_encoding_2d(y.shape[1], y.shape[2], y.shape[3])
57
58     y = tf.reshape(y, (y.shape[0], -1, y.shape[3]))
59
60     return y

```

```

1 class InitialHidden(tf.keras.Model):
2
3     def __init__(self, units):
4         super(InitialHidden, self).__init__()
5
6         self.fc = tf.keras.layers.Dense(units, activation='tanh')
7
8     def __call__(self, x):
9
10        x = tf.math.reduce_mean(x, axis=1)
11
12        return self.fc(x)
13
14
15 class BahdanauAttention(tf.keras.layers.Layer):
16     def __init__(self, units):
17         super(BahdanauAttention, self).__init__()
18
19         self.W1 = tf.keras.layers.Dense(units)
20         self.W2 = tf.keras.layers.Dense(units)
21         self.V = tf.keras.layers.Dense(1)
22
23         self.Q = tf.keras.layers.Conv1D(filters=16, kernel_size=64, padding="same",
24                                       use_bias=False)

```

```

23     self.Uf = tf.keras.layers.Dense(units , use_bias=False)
24
25
26 def call(self , features , hidden , B):
27     # features (Encoder output) shape == (batch_size , L, 512)
28
29     # hidden shape == (batch_size , hidden_size)
30     # hidden_with_time_axis shape == (batch_size , 1, hidden_size)
31     hidden_with_time_axis = tf.expand_dims(hidden , 1)
32
33     F = self.Q(B)
34
35     # score shape == (batch_size , L, hidden_size)
36     score = tf.nn.tanh(self.W1(features) + self.W2(hidden_with_time_axis) +
37                         self.Uf(F))
38
39     # attention_weights shape == (batch_size , L, 1)
40     # you get 1 at the last axis because you are applying score to self.V
41     attention_weights = tf.nn.softmax(self.V(score) , axis=1)
42
43     # context_vector shape after sum == (batch_size , hidden_size)
44     context_vector = attention_weights * features
45     context_vector = tf.reduce_sum(context_vector , axis=1)
46
47
48
49 class Decoder(tf.keras.Model):
50
51     def __init__(self , units , embedding_dim , vocab_size):
52         super(Decoder , self).__init__()
53
54         self.Wout = tf.keras.layers.Dense(vocab_size , use_bias=False)
55         self.o_t = tf.keras.layers.Dense(units , use_bias=False , activation='tanh')
56
57

```

```

56     self.lstm = tf.keras.layers.LSTM(units, return_state=True)
57     self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
58
59     self.attention = BahdanauAttention(units)
60
61
62 def __call__(self, features, hidden, previous_y, previous_out):
63
64     # embedding_previous_y shape == (batch_size, embedding_dim)
65     embedding_previous_y = self.embedding(previous_y)
66
67     # previous_out shape == (batch_size, units)
68     # lstm_input shape == (batch_size, embedding_dim + units)
69     lstm_input = tf.concat([embedding_previous_y, previous_out], axis=1)
70     lstm_input = tf.expand_dims(lstm_input, axis=1)
71
72     # hidden is a list of hidden state and carry state respectively
73     # each element has a shape == (batch_size, units)
74     # actually output and state are the same tensors
75     output, state, carry = self.lstm(lstm_input, initial_state=hidden)
76
77     # context_vector shape == (batch_size, hidden_size)
78     # attention_weights == (batch_size, L, 1)
79     context_vector, attention_weights = self.attention(features, state)
80
81     # calculating the output
82     out = self.o_t(tf.concat([output, context_vector], axis=1))
83     output = self.Wout(out)
84
85     return output, out, attention_weights, [state, carry]

```

#### 1.5.1.4. Entrenamiento

La red fue entrinada en el entorno Colaboratory de Google utilizando una GPU, se utilizaron dos conjuntos de entrenamiento. El primero fue CROHME únicamente, mientras que el segun-

do, fue una combinación entre CROHME y Harvard 100k, esto con la intención de provar cual sería el resultado del modelo si se incrementara el tamaño del conjunto de entrenamiento. Dado que el modelo esta basado solamente en los datos, no fue necesario hacer ningún cambio a la arquitectura.

Para el primer conjunto de entrenamiento, se necesitaron 16 epochas tomando un total de 10 horas. Con respecto al segundo, se entreno la red por 6 epochas tomando un total de 30 horas.

Al utilizar Tensorflow 2, el cálculo de los gradientes de la función de perdida se puede automatizar, así como la actualización de los pesos, no obstante, dada la arquitectura de la red, no es posible utilizar los métodos *compile* y *fit* implementados en Keras para Tensorflow 2. Por lo tanto, el ciclo de entrenamiento tuvo que ser diseñado acorde con las necesidades de la arquitectura propuesta.

El código del ciclo de entrenamiento es el siguiente:

```

1 @tf.function
2 def train_step(img_tensor, target):
3     loss = 0
4     loss_counter = 0
5     loss_before = tf.constant(0.0)
6
7     with tf.GradientTape() as tape:
8         features = encoder(img_tensor, True)
9         tf.debugging.assert_all_finite(features, 'the features have exploded')
10
11     hidden = initial_state(features)
12     carry = initial_carry(features)
13     out = initial_out(features)
14     dec_input = tf.constant([BEGIN] * target.shape[0])
15
16     hidden_and_carry = [hidden, carry]
17
18     B = tf.zeros((features.shape[0], features.shape[1], 1))
19
20     for i in range(1, target.shape[1]):
21         # passing the features through the decoder
22         predictions, out, attention_weights, hidden_and_carry = decoder(
23             features, hidden_and_carry, dec_input, out, B)
24         B += attention_weights

```

```
25     real = get_num_token_end(target[:, i])
26
27     loss += loss_function(real, predictions)
28     tf.debugging.assert_all_finite(loss, 'the loss has exploded')
29
30     # is_done = tf.where(tf.less(real, end_tensor), is_done, end_tensor
31     )
32
33     if loss != loss_before:
34         loss_counter += 1
35         loss_before = loss
36
37
38     # using teacher forcing
39     dec_input = real
40
41
42     total_loss = (loss / int(target.shape[1]))
43
44     trainable_variables = encoder.trainable_variables + initial_state.
45     trainable_variables + initial_carry.trainable_variables + initial_out.
46     trainable_variables + decoder.trainable_variables
47
48     gradients = tape.gradient(loss, trainable_variables)
49
50     for g in gradients:
51         tf.debugging.assert_all_finite(g, 'the gradients have exploded')
52
53     optimizer.apply_gradients(zip(gradients, trainable_variables))
54
55     train_loss(loss)
56     # train_accuracy(y_train, predictions)
57
58     return loss, total_loss, loss_counter
59
60
61
62 start = time.time()
```

```

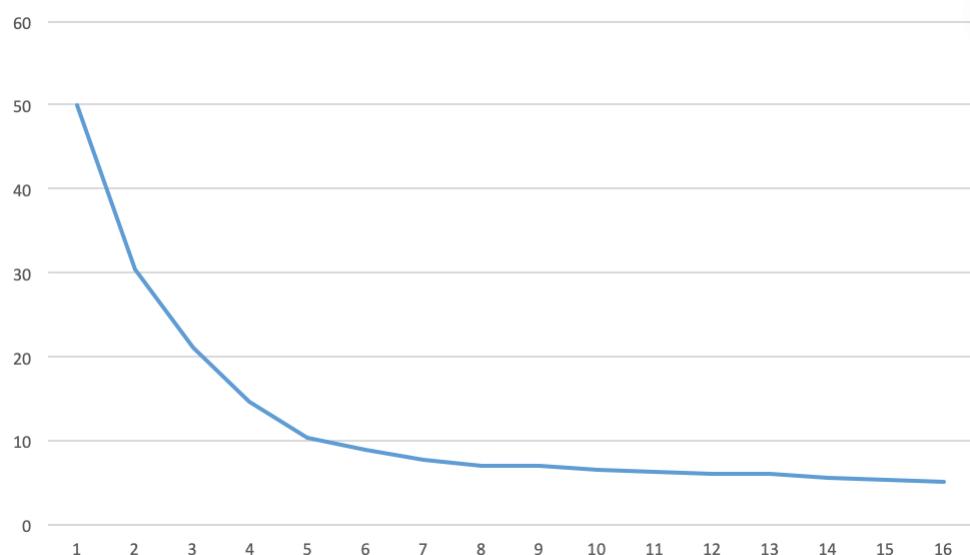
57 total_loss = 0
58 total_val_loss = 0
59
60 for (batch, (img_tensor, target)) in enumerate(dataset):
61     # img_tensor = noise(img_tensor, training=True)
62     batch_loss, t_loss, loss_counter = train_step(img_tensor, target)
63     total_loss += t_loss
64
65     print ('Epoch {} Batch {} Loss {:.6f}'.format(
66         epoch, batch, batch_loss.numpy() / loss_counter))

```

### 1.5.1.5. Resultados

Al utilizar dos conjuntos de entrenamiento distintos se obtuvieron dos resultados completamente diferentes.

Para el caso de la red entrenada con CROHME, se puede notar que el sistema de atención aún no funciona completamente, esto es debido posiblemente a que la red aún no ha sido capaz de abstraer todas las características importantes del conjunto de entrenamiento. En la Figura 1.23, se puede apreciar que a pesar del entrenamiento, la función de pérdida sigue siendo mayor a 5.

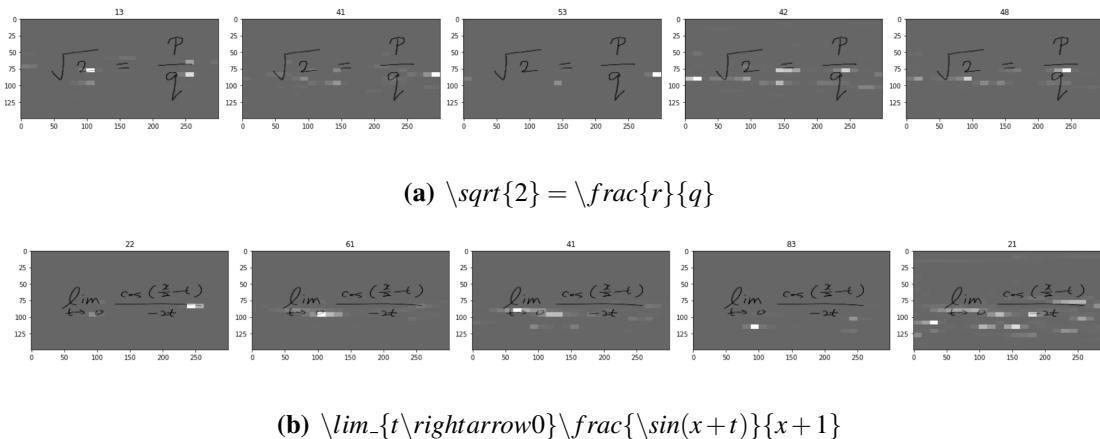


**Figura 1.23:** Función de pérdida del modelo entrenado con CROHME.

La razón de esto, es el reducido tamaño de CROHME, pues 7170 imágenes no han sido suficientes para obtener un buen resultado en la predicción de expresiones matemáticas escritas

a mano. Incrementar la capacidad de la red mediante el aumento del número de parámetros no es una opción viable, pues al ser un conjunto pequeño, el riesgo de sobreentrenamiento aumenta considerablemente.

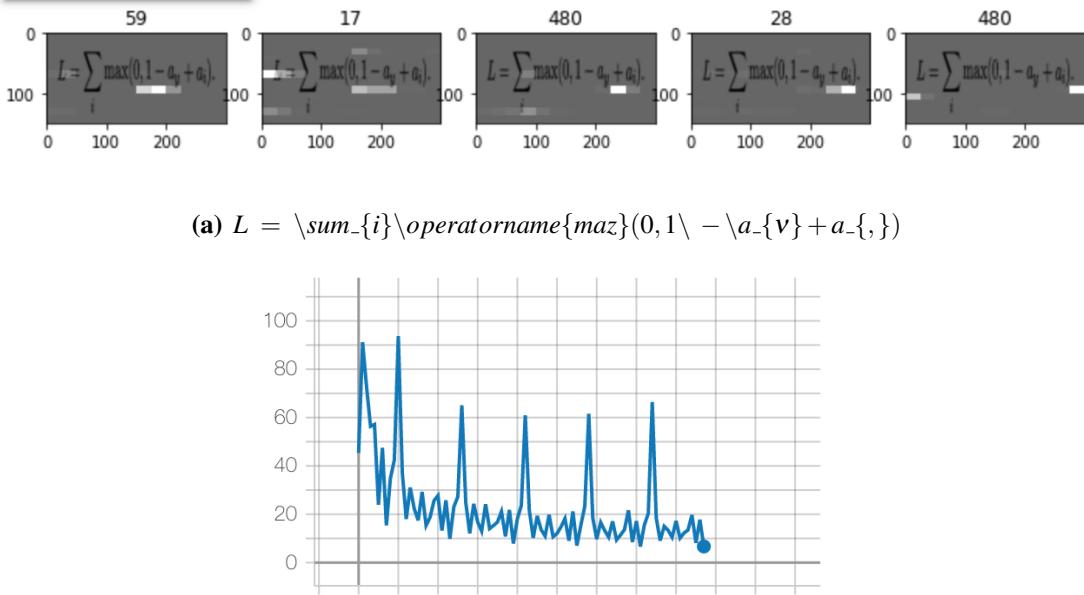
No obstante, se puede observar que la red si ha aprendido a segmentar símbolos y lo que es mucho más sobresaliente, el modelo es capaz de representar las relaciones espaciales y de jerarquía que existen en una expresión matemática, sin embargo, solo es capaz de hacer esto para secuencias cortas. La Figura 1.24 muestra la comparación de la atención entre dos imágenes, la primera (a), es una secuencia corta mientras que la segunda (b) es una secuencia más larga.



**Figura 1.24:** (a) Atención de una secuencia corta junto con la secuencia predicha por el modelo. (b) Atención en una secuencia larga junto con la secuencia predicha por el modelo.

Con respecto al conjunto entrenado con la combinación entre CROHME y Harvard 100k se obtuvieron resultados satisfactorios al momento de reconocer expresiones matemáticas renderizadas por computadora. La Figura 1.25 muestra los resultados de una predicción de la red. No obstante, la red no muestra buenos resultados para predecir imágenes con expresiones escritas a mano. Esto es natural, pues el conjunto se compone principalmente de imágenes renderizadas por computadora.

Lo que se puede concluir, es que el modelo propuesto funciona, es capaz de aprender a segmentar símbolos y entender las relaciones bidimensionales que componen a las expresiones matemáticas, sin embargo su efectividad depende completamente del conjunto de entrenamiento, pues este debe de tener un tamaño suficientemente largo y debe de representar adecuadamente las expresiones que se buscan predecir.



(b) **Función de perdida del modelo entrenado con Hardvard 100k y CROHME**

**Figura 1.25:** (a) Atención de una secuencia larga renderizada por computadora junto con la secuencia predicha por el modelo. (b) Función de perdida del modelo.

### 1.5.1.6. Experimentos Previos

Antes de encontrar la configuración adecuada para la red neuronal, se llevaron a cabo varios experimentos. A continuación, se presentan los más relevantes.

#### Variante con GRU

Se utilizó una GRU como en [11] en lugar de una LSTM en el *Decoder*. No existe un consenso respecto a cual tipo de RNN basadas en compuertas es mejor, por lo que lo más recomendable es experimentar con ambas. En este caso particular, la GRU no logró abstraer las características del problema. El código fuente del *Decoder* con GRU se muestra a continuación:

```

1 class RNN_Decoder(tf.keras.Model):
2     def __init__(self, embedding_dim, units, vocab_size):
3         super(RNN_Decoder, self).__init__()
4         self.units = units
5
6         self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
7

```

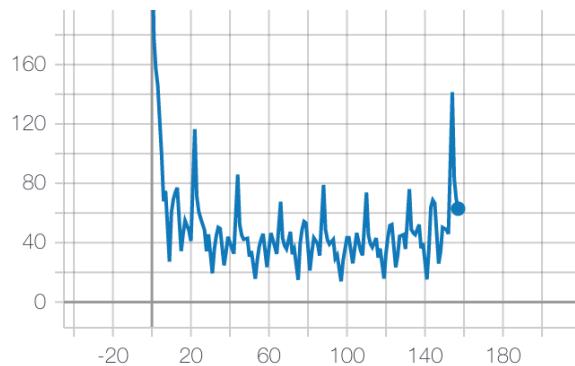
```
8     self.gru = tf.keras.layers.GRU(self.units,
9                                     return_sequences=True,
10                                    return_state=True,
11                                    recurrent_initializer='glorot_uniform')
12
13     self.fc1 = tf.keras.layers.Dense(self.units, kernel_regularizer=tf.keras.
14                                     regularizers.l1_l2(l1=0.01, l2=0.01))
15     self.fc2 = tf.keras.layers.Dense(vocab_size, kernel_regularizer=tf.keras.
16                                     regularizers.l1_l2(l1=0.01, l2=0.01))
17
18 def call(self, x, features, hidden):
19     # defining attention as a separate model
20     context_vector, attention_weights = self.attention(features, hidden)
21
22     # x shape after passing through embedding == (batch_size, 1,
23     # embedding_dim)
24     x = self.embedding(x)
25
26     # x shape after concatenation == (batch_size, 1, embedding_dim +
27     # hidden_size)
28     x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
29
30     # passing the concatenated vector to the GRU
31     output, state = self.gru(x, initial_state=hidden)
32
33     # shape == (batch_size, max_length, hidden_size)
34     x = self.fc1(output)
35
36     # x shape == (batch_size * max_length, hidden_size)
37     x = tf.reshape(x, (-1, x.shape[2]))
38
39     # output shape == (batch_size * max_length, vocab)
40     x = self.fc2(x)
```

```

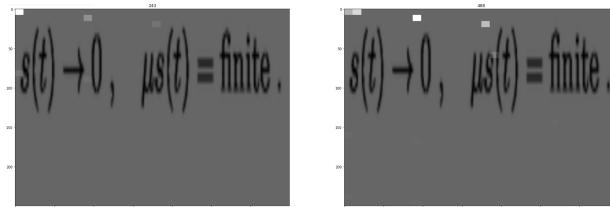
39
40     return x, state, attention_weights
41
42 def reset_state(self, batch_size):
43     return tf.zeros((batch_size, self.units))

```

Una muestra del resultado del entrenamiento se muestra en la Figura 1.26, en (a) vemos que la función de perdida se estanca, demostrando que no es capaz de modelar el problema en su totalidad mientras que en (b), la atención no mostró aprendizaje alguno.



(a) **Función de perdida más característica del modelo.**



(b) **Atención del sistema.**

**Figura 1.26:** (a) Función de perdida del sistema, el modelo se entrena en varias ocasiones, siempre obteniendo una gráfica parecida. (b) Se muestra la atención obtenida por este modelo, se observa un aprendizaje nulo.

### Variante con alto Dropout

Acorde con [12], el *dropout* es una técnica utilizada para regularizar redes neuronales, es decir, para que la probabilidad de que un modelo se sobreentrene disminuya. Consiste en aplicar un muestreo a las salidas de alguna capa en un modelo con la finalidad de desactivar un porcentaje de neuronas. Suponiendo a  $\mu$  como el vector de muestreo y a  $p(\mu)$  como la distribución de

probabilidad que modela si la neurona se desactiva o no, podemos aplicar dropout a una capa de alguna red neuronal que modele una distribución  $p(y|x)$  de la siguiente manera:

$$P' = p(\mu)p(y|x, \mu) \quad (1.22)$$

De este modo, algunas neuronas [5] son "desconectadas" haciendo que cada neurona aprenda sobre la información proveída en el entrenamiento de manera independiente. Esto produce también, un efecto de regularización.

La técnica del dropout es recomendada en la mayoría de los modelos de deep learning, usualmente con una probabilidad de desconexión de 0.5 %, sin embargo, este radio no provoca ningún efecto en la red. Esto es debido posiblemente a que al ser implementado en las capas convolucionales, un dropout muy alto para una red no tan profunda provoca una perdida significativa de la información. En la Figura 1.27 se pueden ver los resultados de generales de este modelo.

```

1 class CNN_Layer(tf.keras.layers.Layer):
2
3     def __init__(self, num_filters, kernel_size, padding,
4                  apply_batch_normalization=False, apply_dropout=False, dropout_rate=0.5):
5         super(CNN_Layer, self).__init__()
6
7         self.conv = tf.keras.layers.Conv2D(filters=num_filters, kernel_size=
8                                         kernel_size,
9                                         strides=(1,1), padding=padding,
10                                        use_bias=False)
11
12         self.batch_normalization = tf.keras.layers.BatchNormalization()
13         self.dropout = tf.keras.layers.Dropout(rate=dropout_rate)
14         self.apply_batch_normalization = apply_batch_normalization
15         self.apply_dropout = apply_dropout
16
17     def __call__(self, x, training=False):
18
19         y = self.conv(x)
20         if self.apply_batch_normalization:
21             y = self.batch_normalization(y, training)
22             y = tf.nn.relu(y)
23         if self.apply_dropout:
24             y = self.dropout(y, training)

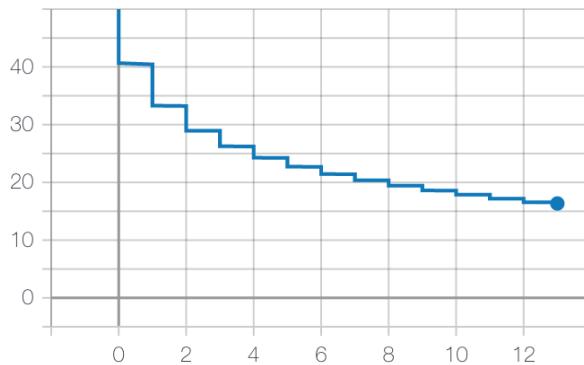
```

```
22
23     return y
24
25
26 class CNN_Encoder(tf.keras.Model):
27
28     def __init__(self, embedding_dim):
29         super(CNN_Encoder, self).__init__()
30
31         self.pooling1 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides
32             =(2,2))
33         self.pooling2 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides
34             =(2,2), padding='same')
35         self.pooling3 = tf.keras.layers.MaxPool2D(pool_size=(2,1), strides
36             =(2,1), padding='same')
37         self.pooling4 = tf.keras.layers.MaxPool2D(pool_size=(1,2), strides
38             =(1,2), padding='same')
39
40         self.conv1 = CNN_Layer(64, (3,3), 'valid', apply_dropout=True,
41             dropout_rate=0.2)
42         self.conv2 = CNN_Layer(128, (3,3), 'valid')
43         self.conv3 = CNN_Layer(256, (3,3), 'valid', apply_batch_normalization
44             =True)
45         self.conv4 = CNN_Layer(256, (3,3), 'valid', apply_dropout=True)
46         self.conv5 = CNN_Layer(512, (3,3), 'valid', apply_batch_normalization
47             =True, apply_dropout=True)
48         self.conv6 = CNN_Layer(512, (3,3), 'same', apply_batch_normalization=
49             True)
```

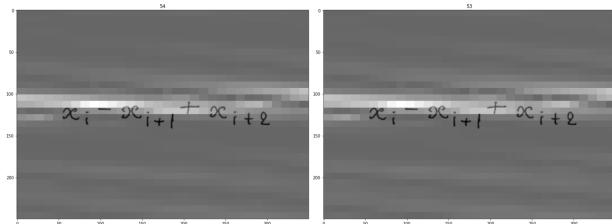
```

49     y = self.conv3(y, training)
50     y = self.conv4(y, training)
51     y = self.pooling3(y)
52     y = self.conv5(y, training)
53     y = self.pooling4(y)
54     y = self.conv6(y, training)
55
56     y += get_positional_encoding_2d(y.shape[1], y.shape[2], y.shape[3])
57     y = tf.reshape(y, (y.shape[0], -1, y.shape[3]))
58
59     return y

```



(a) Función de perdida más característica del modelo.



(b) Atención del sistema.

**Figura 1.27:** (a) Función de perdida del sistema, el modelo se entreno en varias ocasiones, siempre obteniendo una gráfica parecida. (b) Se muestra la atención obtenida por este modelo, se observa que el modelo aprendió que a indentificar donde estaba la ecuación, no obstante no aprendio a segmentar los símbolos.

# Bibliografía

---

- [1] M. S. de Lorenzo, “Clean Architecture Guide (with tested examples): Data Flow != Dependency Rule.” <https://proandroiddev.com/clean-architecture-data-flow-dependency-rule-615ffdd79e29>, 2018. [Consultado: 2019-09-21]. III, 2
- [2] F. Cejas, “Architecting Android...The clean way?” <https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/>, 2014. [Consultado: 2019-09-21]. III, 4, 9
- [3] F. Cejas, “Architecting Android...Reloaded.” <https://fernandocejas.com/2018/05/07/architecting-android-reloaded/>, 2018. [Consultado: 2019-09-21]. III, 15
- [4] J. Zhang, J. Dua, S. Zhang, D. Liub, Y. Hub, J. Hub, S. Weib, and L. Daia, “Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition,” *Pattern Recognition*, p. 196–206, 2017. 17, 39
- [5] Y. Deng, A. Kanervisto, and A. M. Rush, “What you get is what you see: A visual markup decompiler,” *CoRR*, vol. abs/1609.04938, 2016. 23, 37, 39, 52
- [6] Z. Hadjadj, A. Meziane, Y. Cherfa, M. Cheriet, and I. Setitra, “Isauvola: Improved sauvola’s algorithm for document image binarization,” vol. 9730, pp. 737–745, 07 2016. 25
- [7] J. Zhang, J. Du, and L. Dai, “Multi-scale attention with dense encoder for handwritten mathematical expression recognition,” *CoRR*, vol. abs/1801.03530, 2018. 35
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017. 37
- [9] Z. Wang and J.-C. Liu, “Translating math formula images to latex sequences using deep neural networks with sequence-level training,” vol. abs/1908.11415, 2019. 37
- [10] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2014. 39

- [11] K. Cho, B. van Merriënboer, Ç. Gülcöhre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *CoRR*, vol. abs/1406.1078, 2014. 49
- [12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. 51