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 Tonatihu Barrera Pérez
Juan Carlos Garcia Medina
Ian Mendoza Jaimes

DIRECTORES:
Dr. Jorge Cortés Galicia

Ciudad de México, 25 de mayo de 2020

Índice general

Índice de figuras IV

Índice de tablas VI

1. Marco teórico 1
- 1.1. Análisis de Imágenes . . . 1
  - 1.1.1. Preprocesamiento . . . 1
  - 1.1.2. Realce de la imagen . . . 2
    - 1.1.2.1. Transformaciones a nivel de grises básicas . . . 3
    - 1.1.2.2. Negativos de Imagen . . . 4
    - 1.1.2.3. Transformaciones Log . . . 5
    - 1.1.2.4. Transformaciones Power-Law . . . 5
  - 1.1.3. Segmentación de la imagen . . . 5
- 1.2. Deep Learning . . . 7
  - 1.2.1. Gráfos computacionales . . . 7
  - 1.2.2. Batch Normalization . . . 7
  - 1.2.3. Redes Neuronales Convolucionales . . . 9
    - 1.2.3.1. Pooling . . . 10
- 1.3. Redes Neuronales Recurrentes . . . 11
  - 1.3.1. Arquitectura Encoder-Decoder . . . 12
  - 1.3.2. Long Short-Term Memory . . . 13
  - 1.3.3. Modelos de Lenguaje Condicional . . . 14
  - 1.3.4. Sistema de Atención . . . 14
  - 1.3.5. Positional Embeddings . . . 14
  - 1.3.6. Aprendizaje profundo como servicio . . . 15
- 1.4. Métodos de reconocimiento de expresiones matemáticas . . . 16
  - 1.4.1. Análisis sintáctico dirigido . . . 16
  - 1.4.2. Análisis estructural . . . 17
  - 1.4.3. Image Captioning . . . 17
- 1.5. Aplicación web . . . 17
  - 1.5.1. Django . . . 18
    - 1.5.1.1. Modelo-Vista-Template . . . 18
  - 1.5.2. API REST . . . 19

1.5.2.1. Application Programming Interface . . . 19
1.5.2.2. Represenational State Transfer . . . 20
1.5.2.3. Autentificación . . . 20
1.6. Aplicación Android . . . 21
1.6.1. Arquitectura Clean . . . 21
1.6.2. Componentes . . . 22
1.6.2.1. Entidades . . . 22
1.6.2.2. Casos de uso . . . 22
1.6.2.3. Adaptadores . . . 22
1.6.2.4. Frameworks y controladores . . . 22
1.6.3. SOLID . . . 22
1.7. Estado del arte . . . 23

2. Desarrollo del sistema **25**
2.1. Android . . . 25
2.1.1. Arquitectura de la aplicación . . . 25
2.1.1.1. Capa de datos . . . 26
2.1.1.2. Capa de dominio . . . 33
2.1.1.3. Capa de presentación . . . 39
2.2. Web . . . 40
2.2.1. Arquitectura de la aplicación . . . 40
2.2.1.1. Modelo . . . 40
2.2.1.2. Vista . . . 40
2.2.1.3. Template . . . 40
2.3. Conjunto de entrenamiento: CROHME . . . 41
2.3.1. Formato del conjunto de datos . . . 41
2.3.2. Conversión a imagen . . . 42
2.3.3. Generador de secuencia de tokens . . . 45
2.4. Reconocimiento de expresiones matemáticas . . . 48
2.4.1. Modelo . . . 48
2.4.1.1. Encoder . . . 50
2.4.1.2. Decoder . . . 51
2.4.1.3. Implementación . . . 52
2.4.1.4. Resultados . . . 57
2.4.1.5. Experimentos Previos . . . 57

3. Pruebas del sistema **58**
3.1. Pruebas unitarias . . . 58
3.1.1. Aplicación android . . . 58
3.1.1.1. Pruebas sobre la clase DateFormater . . . 58
3.1.1.2. Pruebas sobre la clase RNN002 . . . 60
3.2. Pruebas de integración . . . 64
3.3. Pruebas de requerimientos funcionales . . . 64

Bibliografía **65**

Índice de figuras

1.1. Transformación de nivel de grises para realce de contraste. 3
1.2. Algunas transformaciones básicas a nivel de grises usadas para realce de imagen. 4
1.3. Imagen con bloques (Izquierda) y conjunto de segmentos de linea extraídos (Derecha). 6
1.4. Ejemplo de un perceptrón de una sola capa representado mediante un gráfo computacional, siendo x la entrada, w^1 la matriz de pesos, b^1 la matriz de bias, $u^{(1)}$ y $u^{(2)}$ nodos intermedios en el gráfo y y^1 la salida de la red. La ecuación modelada es $y^1 = f(W^1x + b^1)$. 8
1.5. Matriz de pesos conocida como kernel o filtro aplicada a una imagen. 10
1.6. Representación gráfica de la operación Max Pooling. 11
1.7. Ejemplo de una RNN que modela la ecuación $s(t) = f(s^{(t-1)}, \theta)$, donde θ representa los parámetros W y V, f la función no lineal aplicada a $Wx^{(t)}$ y V la matriz de pesos aplicada a $s^{(t-1)}$. 12
1.8. Una compuerta LSTM típica. 13
1.9. Ejemplo de sistema con módulo de Machine Learning 15
1.10. **a)** La primera etapa del método de reconocimiento, se anotan las coordenadas de de cada caracter. **b)** Se realiza un análisis sintáctico con las gramáticas libres de contexto definidas. 16
1.11. Utilización de un árbol de recubrimiento mínimo para el reconocimiento de expresiones matemáticas. 17
1.12. Arquitectura de image captioning para reconocer expresiones matemáticas en imagenes. 18
1.13. Representación del patrón de diseño Modelo-Vista-Template. 19
1.14. Consumidor y proveedor de servicios comunicandose mediante solicitudes y respuestas REST. 20
1.15. Arquitectura Clean [1] 21

2.1. Tres capas que se tienen al utilizar la arquitectura Clean [2] 26
2.2. Capa de datos [3] 28
2.3. Capa de dominio [3] 33
2.4. Capa de presentación [4] 39
2.5. Modelo *Encoder-Decoder* de la red neuronal. 49

3.1. Resultados de las pruebas de la clase DateFormater 60
3.2. Resultados de las pruebas de la clase RNN002 63

Índice de tablas

1.1. Resumen de productos similares 24

2.1. Especificación de la CNN. El 'Kernel' esta denotado como *número de filtros*-(*dimensiones del filtro*). 50

Capítulo 1

Marco teórico

Con el desarrollo del sistema producto del presente trabajo terminal se involucran ciertos conceptos provenientes en su mayoría de ramas de ciencias de la computación y en general en alusión a la Inteligencia Artificial, por lo que es conveniente dar contexto sobre los elementos necesarios para el desarrollo del trabajo terminal.

1.1. Análisis de Imágenes

Una imagen puede definirse como una función bidimensional. $f(x,y)$, donde x y y son coordenadas espaciales (plano) y la amplitud de f en cualquier par de coordenadas (x,y) es llamada *intensity* o *gray level* (nivel de grises) de la imagen en ese punto.
Cuando x,y y los valores de amplitud de f son todas cantidades discretas finitas, la imagen se denomina imagen digital. El campo de análisis de imagen se refiere a procesar imagenes por medio de una computadora digital. Una imagen digital está compuesta por número de elementos finitos, de los cuales cada uno tiene una posición particular y valor. Estos elementos son referidos como **picture elements, image elements, pels** y **pixels**. Pixel es el término más ampliamente usado para denotar los elementos de una imagen digital.

El análisis de imágenes comprende un conjunto de operaciones sobre una o varias imágenes con el propósito de obtener una imagen con mayor realce o para extraer características útiles, es un tipo de dispensación de señales en el que la entrada es una imagen y la salida puede ser otra imagen o características asociadas a la imagen, algunos de los pasos generales se describen a continuación:

1.1.1. Preprocesamiento

Preprocesamiento es un nombre común para operaciones con imágenes al más bajo nivel de abstracción. Tanto entrada como salida son imágenes de intensidad. Estas imágenes tienen el mismo tipo de datos que la original, con una imagen de intensidad usualmente representada por una

matriz de valores de función de imagen (Brillo) El objetivo de preprocesar es la mejora de los datos de la imagen que borre distorciones o realce características importantes para procesamiento posterior, incluso las transformaciones geométricas de las imagenes e.g (rotación, escalamiento y traslación) son también clasificadas como métodos de preprocesamiento, ya que técnicas similares son utilizadas [5].

1.1.2. Realce de la imagen

El objetivo principal de realce de imagen es también procesar una imagen dada tal que el resultado sea mas ajustable que la imagen original para aplicaciones específicas. Por ejemplo para la remoción de ruido.

Acentúa o afina características de la imagen como ejes, límites o contraste para hacer un despliegue gráfico mas útil para el análisis.

El realce no incrementa o decrementa el contenido de la información inherente de los datos pero sí incrementa el rango dinámico de las características elegidas de tal modo que puedan ser detectadas fácilmente.

Proveé ḿejoréntrada para otras técnicas avanzadas de procesamiento automatizadas de imágenes.

Los enfoques de realce de imagen se dividen en categorías amplias: métodos de dominio espacial y métodos de dominio de frecuencia. El término *dominio espacial* se refiere al plano mismo de la imagen y las aproximaciones en esta categoría son basadas en manipulación directa en una imagen. Técnicas de procesamiento de *dominio de frecuencia* están basadas en modificar la transformada de Fourier de una imagen.

El término *dominio espacial* se refiere al agregado de pixeles que componen una imagen. Los métodos de dominio espacial son procesos que operan directamente en estos pixeles. Los procesos de dominio espacial serán denotados por la expresión

$$g(x,y) = T[f(x,y)] \tag{1.1}$$

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen procesada y T es un operador sobre f, definido sobre alguna vecindad de (x,y). Además, T puede operar sobre un conjunto de imágenes de entrada, como llevar a cabo la suma `pixel-by-pixel` de K imágenes para reducción de ruido.

La principal aproximación para definir una vecindad de un punto (x,y) es usar una área de una sub-imagen cuadrada o rectangular centrada en (x,y), como se muestra en

El centro de la subimagen es movido de pixel a pixel iniciando por ejemplo en la esquina superior izquierda. El operador T es aplicado en cada posición (x,y) para producir la salida g en esa posición. El proceso utiliza solo los pixeles en el área de la imagen expandida por la vecindad. A pesar de que otras vecindades le dan forma, como aproximaciones a un círculo son usadas

en ocasiones, arreglos cuadrados y rectangulares son por mucho los más predominantes por la facilidad de implementación.

La forma mas simple de T es cuando la vecindad es de tamaño $1x1$ (un solo pixel). En este caso, g depende solo del valor de f en (x,y) y T se convierte en una *función de transformación a nivel de grises* (también llamada *intensity o mapeo*) de la forma

$$s = T(r) \tag{1.2}$$

donde por simplicidad de notación, r y s son variables denotando respectivamente el nivel de gris de $f(x,y)$ y $g(x,y)$ en cualquier punto (x,y). Por ejemplo, si $T(r)$ tiene la forma mostrada en 1.1(a), el efecto de esta transformación sería producir una imagen de mas alto contraste que la original al oscurecer los niveles debajo de m y darle brillo a los niveles arriba de m en la imagen original. En esta técnica conocida como contrast stretching, los valores de r por debajo de m son comprimidos por la función de transformación en un rango estrecho de s hacia el negro. El efecto opuesto da lugar para valores de r por encima de m. En el caso limitante mostrado en 1.1(b), $T(r)$ produce una imagen two-level (binaria). Un mapeo de esta forma es llamado una función de *thresholding*.

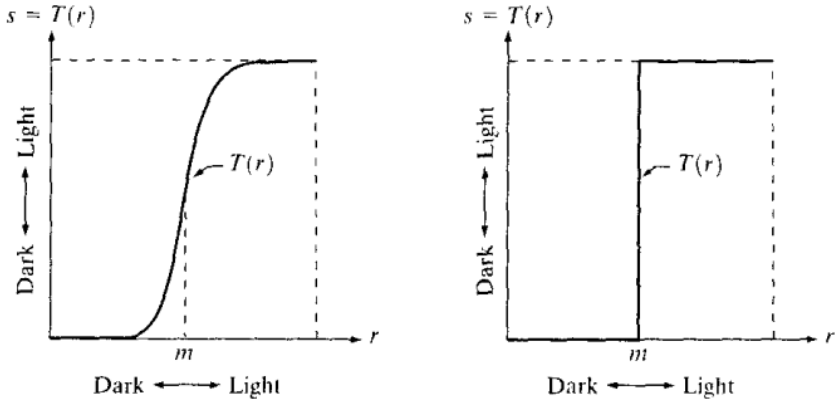

Figura 1.1: Transformación de nivel de grises para realce de contraste.

1.1.2.1. Transformaciones a nivel de grises básicas

Las transformaciones a nivel de grises son las más básicas en el proceso de realce de imágenes. Los valores de los pixeles antes y después de ser procesados se denotan por r y s respectivamente. Como se mencionó ateriormente, estos valores son relacionados por 1.2, como se trabaja con cantidades digitales, los valores de las funciones de transformación típicamente son almacenadas en un arreglo unidimensional y los mapeos de r y s son implementados via por búsquedas de tabla (lookup tables). Para un entorno de 8 bits, una tabla lookup que contiene los valores de T tendrá 256 entradas.

Como introducción a las transformaciones a nivel de grises, considera la figura 1.2 la cual muestra tres tipos básicos de funciones usadas frecuentemente para realce de imágenes: lineal (transformaciones negative e identity), logarítmicas (transformaciones log e inverse-log) y power-law (transformaciones de potencia *n*th y raíz *n*th). La función identidad es el caso trivial en el cual las intensidades de salida son idénticas a la intensidades de entrada.

1.1.2.2. Negativos de Imagen

El negativo de una imagen con niveles de gris en el rango $[0, L-1]$ es obtenido al usar la transformación negativa mostrada en 1.2, la cual es dado por la expresión

$$s = L - 1 - r \tag{1.3}$$

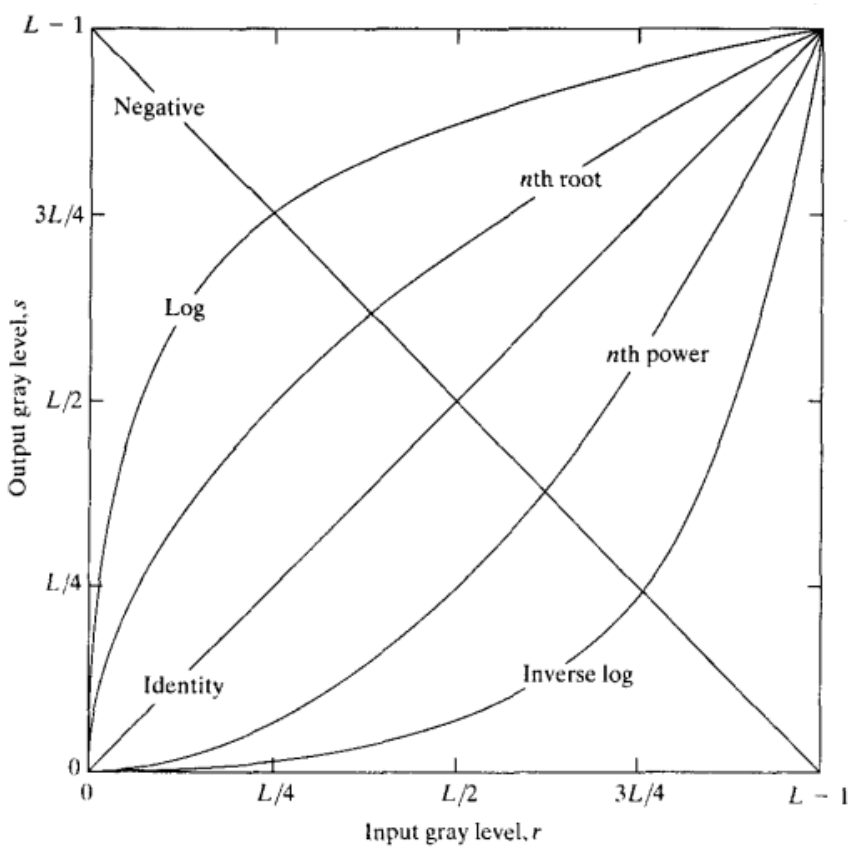

Figura 1.2: Algunas transformaciones básicas a nivel de grises usadas para realce de imagen.

Invertir los niveles de intensidad de una imagen en esta formar produce el equivalente del negativo de fotografía. Este tipo de procesamiento es particularmente ajustado para el realce de detalles de blanco y negro embebidos en regiones oscuras de una imagen, especialmente cuando las áreas negras dominan en tamaño.

1.1.2.3. Transformaciones Log

La forma general de la transformación log que se muestra en la figura 1.2 es

$$s = clog(1+r) \tag{1.4}$$

donde c es una constante y se asume que $r \geq 0$. La forma de la curva log en 1.2 muestra que esta transformación mapea un rango estrecho de valores con bajos niveles de gris en la imagen de entrada a un rango más amplio de niveles de salida. Lo opuesto es verdadero de valores más altos de niveles de entrada. Este tipo de transformaciones se usa para expandir los valores de pixeles oscuros en una imagen mientras se comprimen los valores de nivel más altos. Lo opuesto es verdadero de la transformación inversa de log.

Cualquier curva que tiene la forma general de funciones log mostradas en 1.2 completaría esta expansión/compresión de niveles de gris de una imagen. Las transformaciones log tienen la importante característica de que comprime el rango dinámico de imágenes con grandes variaciones en valores de pixeles. Una clásica ilustración de una aplicación en la cual los valores de pixel tienen un gran rango dinámico es el espectro de Fourier, no es inusual encontrar valores de espectro entre 0 y 10^6 o más grandes.

1.1.2.4. Transformaciones Power-Law

Las transformaciones power-law tienen la forma básica

$$s = cr^{\gamma} \tag{1.5}$$

donde c y γ son constantes positivas. En ocasiones 1.5 se escribe como $c(r+\varepsilon)^{\gamma}$ para considerar una compensación (esto es, una salida medible cuando la entrada es cero). Sin embargo, las compensaciones típicamente son un problema de calibración y como resultado se ignoran normalmente en la ecuación 1.5. Las gráficas de r vs s para distintos valores de γ se muestran en

1.1.3. Segmentación de la imagen

El término segmentación utilizada en el contexto de análisis de imágenes se refiere a la partición de una imagen en un conjunto de regiones que la cubren. El objetivo en muchas de las tareas es que para las regiones se representen áreas significativas de la imagen, como áreas urbanas, fronteras o bosques de una imagen satelital. En otras tareas de análisis, las regiones pueden ser conjuntos de bordes de pixeles agrupados en estructuras como segmentos de líneas y segmentos de arcos circulares en imágenes de objetos industriales en 3D. Las regiones pueden también estar definidas como grupos de pixeles teniendo ambos un borde y una forma particular como un circulo o una elipse or polígono. Cuando las regiones de interés no cubren la imagen completa, aún se requiere el proceso de segmentación en regiones de y de fondo para ignorarse. [5]

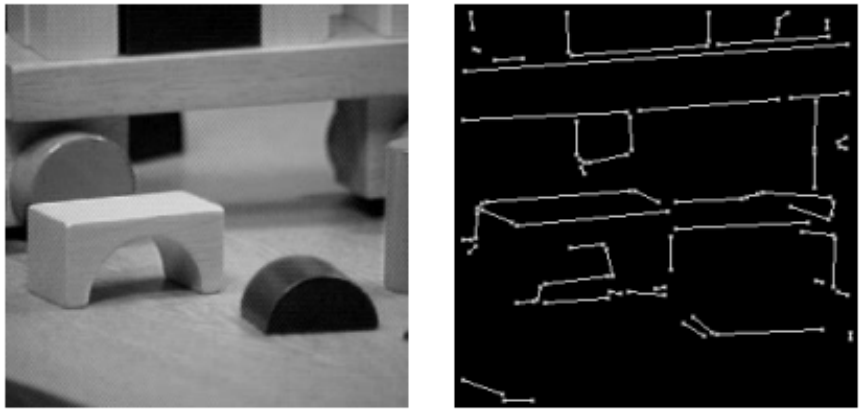

Figura 1.3: Imagen con bloques (Izquierda) y conjunto de segmentos de linea extraídos (Derecha).

1.2. Deep Learning

El aprendizaje profundo o *deep learning* es una rama del aprendizaje automático (*machine learning* en inglés) que intenta modelar abstracciones de alto nivel a través de complejas arquitecturas computacionales que admiten transformaciones no lineales. El deep learning es la evolución de las ya conocidas redes neuronales las cuales experimentaban problemas de desvanecimiento del gradiente si se usaban demasiadas capas. Con las nuevas técnicas propuestas en el deep learning se logro evitar este problema y así, poder entrenar arquitecturas de millones de parámetros.

El perceptrón multicapa (MLP) es el modelo más básico y aún así uno de los más útiles en el campo de las redes neuronales. Su principal objetivo es tratar de aproximar una función f^*. En el caso del *aprendizaje supervisado*, la función f^* toma la forma $y^* = f^*(x)$ de donde x es un parámetro de entrada que puede ser desde un simple número hasta un tensor y y^* es la salida de la función. Entonces, una red neuronal que quiera aproximar f^* definirá una función de mapeo de la forma $y = f(x, \theta)$ y aprenderá los parámetros θ (usualmente conocidos como W y b) que dan como resultado la mejor aproximación de tal forma que $y \approx y^*$.

Las redes neuronales profundas son llamadas redes porque pueden representarse mediante la composición de varias funciones de mapeo. Es decir, un MLP que quiera aproximar una función puede expresarse como $y^n = f^n(y^{n-1}, \theta^n)$ con $y^1 = f^1(x, \theta^1)$ y n siendo la profundidad de la red.

1.2.1. Gráfos computacionales

Para describir con formalidad a las redes neuronales es preciso utilizar una notación que pueda ser expresada a través de gráfos. Según [6] se puede indicar cada nodo del gráfo como una variable que puede ser un tensor para no perder generalidad.

Para terminar con la definición de estos gráfos, es necesario introducir el concepto de operación, la cual simplemente es una función entre dos o mas nodos del gráfo. Sin perder generalidad, se dice que una operación retorna únicamente una variable, es decir, un solo nodo.

En la Figura 1.4 se puede ver un ejemplo de un perceptrón de una sola capa que es representado mediante un gráfo computacional. Gracias a esta definición, se puede extender facilmente el gráfo mostrado en 1.4 para modelar a un perceptrón de n capas.

1.2.2. Batch Normalization

Cada capa de una red neuronal tiene entradas con su correspondiente distribución la cual es afectada durante el entrenamiento por la aleatoriedad en la inicialización de parámetros y en la aleatoriedad de los datos de entrada. Los efectos de esta aleatoriedad en la distribución de las entradas de la red hacia las capas internas en el momento de entrenamiento es descrita como un cambio en la covarianza, el cual provoca inestabilidad en la red.

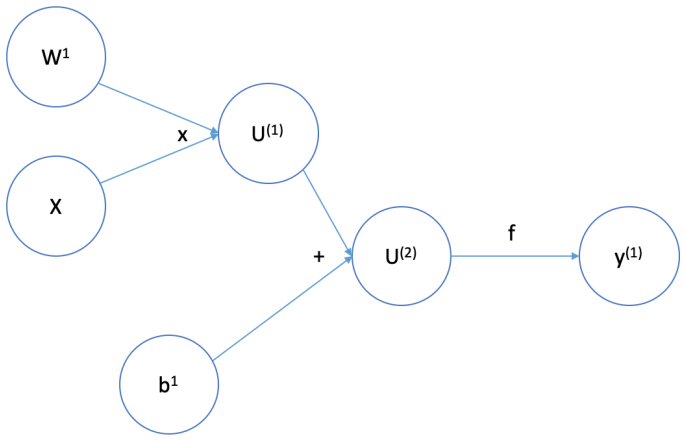

Figura 1.4: Ejemplo de un perceptrón de una sola capa representado mediante un gráfo computacional, siendo x la entrada, w^1 la matriz de pesos, b^1 la matriz de bias, $u^{(1)}$ y $u^{(2)}$ nodos intermedios en el gráfo y y^1 la salida de la red. La ecuación modelada es $y^1 = f(W^1 x + b^1)$.

La técnica de *Batch Normalization* es utilizada para incrementar la estabilidad, la velocidad y el desempeño de una red neuronal. Esta técnica fue introducida en el 2005 por [7].

En una red neuronal, la técnica de Batch Normalization consiste en una etapa de normalización de las medias y varianzas de las entradas de una capa de la red. Esta operación esta definida de la siguiente manera:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1.6}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{1.7}$$

done B denota algún mini batch de tamaño m. Para una capa con una entrada $x \in \mathbb{R}^n$ entonces:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \varepsilon}} \tag{1.8}$$

done $k \in [1,n]$, $i \in [1,m]$, $\mu_B^{(k)}$ y $\sigma_B^{(k)^2}$ son la media y varianza por cada dimensión respectivamente; ε es añadido como una constante arbitraria y pequeña para asegurar la estabilidad numérica del denominador.

Finalmente, la salida de la capa es:

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)} \tag{1.9}$$

de donde los parámetros $\gamma^{(k)}$ y $\beta^{(k)}$ serán aprendidos durante el entrenamiento.

1.2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales también llamadas convolutivas (CNN), son una arquitectura especial de redes neuronales que permite el procesamiento de datos que tengan una estructura de grilla. Algunos ejemplos de esta estructura son: una serie de tiempo es una grilla 1D o una imágene, la cual puede ser vista como una grilla 2D de pixeles. Se puede decir, que una red convolutiva no es más que una red neuronal que utiliza la operación de convolución en lugar de una multiplicación tensorial normal [6].

La operación de convolución esta definida de la siguiente manera:

$$y(t) = x(t) * w(t) \tag{1.10}$$

$$y(t) = \int_{-\infty}^{\infty} x(\tau) w(t-\tau) d\tau \tag{1.11}$$

En terminolgía de las CNN x es la entrada a la red, w es llamado el **kernel** o **filtro** y y el mapa de características.

Propiamente en los sitemas computacionales no se pueden tener funciones continuas, estas tienen que ser discretas, por lo tanto la operación de convolución que una CNN utiliza es:

$$y(t) = \sum_{\tau=-\infty}^{\infty} x(\tau) w(t-\tau) \tag{1.12}$$

Naturalmente, si lo que se esta trabajando es una imagen, tenemos que aplicar la operación de convolución en dos ejes, es decir:

$$y(i,j) = \sum_{m}\sum_{n} x(i,j) w(i-m, j-n) \tag{1.13}$$

Si se toma el gráfo computacional del ejemplo de la Figura 1.4 el cual modela a un perceptrón de una sola capa y se cambia su operación de multiplicación matricial por la operación de convolución, tenemos una capa convolucional:

$$y^1 = f(W^1 * x + b^1) \tag{1.14}$$

La manera en la que se entrenan estas variantes de MLP es con el mismo algoritmo de optimización que un perceptrón.

En la Figura 1.5 se puede ver un ejemplo de cómo una imagen es procesada por la etapa de convolución en una CNN.

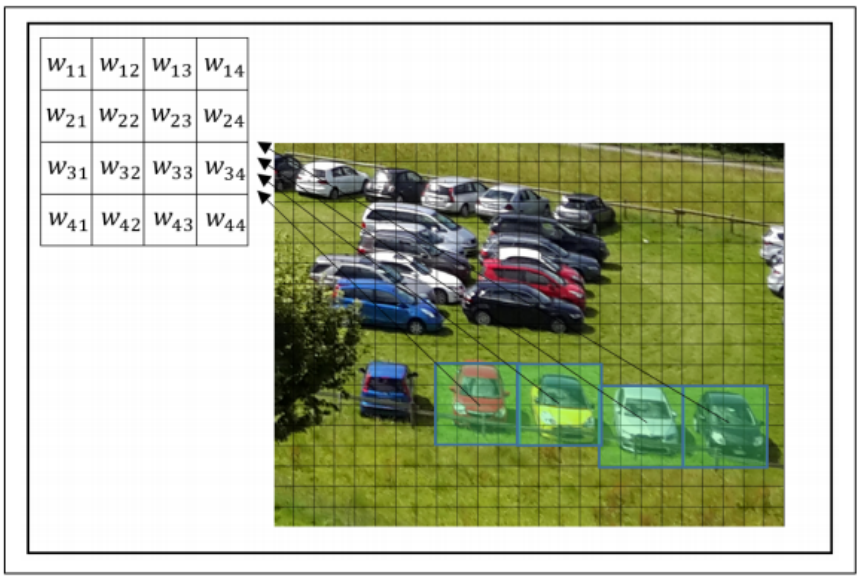

Figura 1.5: Matriz de pesos conocida como kernel o filtro aplicada a una imagen.

1.2.3.1. Pooling

Es una de las operaciones adicionales que se implementan en las CNN. En general se puede establecer que una típica capa convolucional tiene tres partes:

1. Operación de convolución: Es donde utiliza una entrada y la convoluciona con un filtro w.
2. Función de activación: A veces es llamada fase de detección. Consiste en aplicar una función f al resultado de la primera etapa para modificar la salida y volverla no lineal.
3. Etapa de Pooling: Es una manera de modificar aún más la salida de modo que se incremente la significancia estadística de los datos extraidos.

La operación de pooling permite que la red sea tolerante a traslaciones y rotaciones leves de la imagen. Por ejemplo, se puede aplicar la función *max pooling* a cada uno de los valores de $f(W * h)$ para obtener una generalización de los datos más importantes observados por la etapa de detección.

Una manera efectiva de reducir el tamaño de los datos que la red tiene que procesar a través de sus capas, es utilizar la operación pooling para remuestrear. Esto se logra definiendo una ventana a la que la que se le aplicará la operación. Esta técnica también permite que algunas CNN puedan procesar imágenes de longitud variable.

Existen diversas operaciones de pooling, las más comúnes son *Max Pooling* y *Average Pooling*. En la Figura 1.6 se puede apreciar una representación gráfica de la operación de Max Pooling sobre alguna matríz.

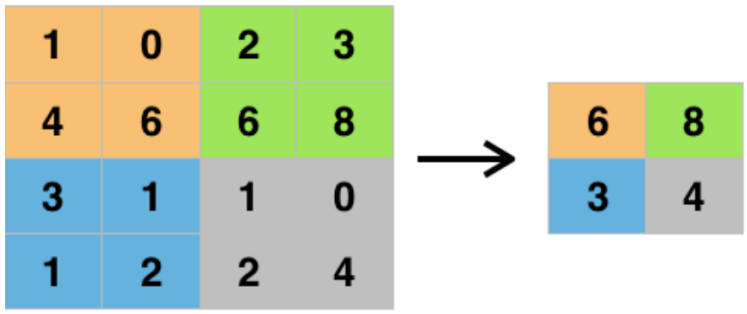

Figura 1.6: Representación gráfica de la operación Max Pooling.

1.3. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNNs) son una familia de redes neuronales especializadas en procesar secuencias de datos. Es decir, pueden procesar entradas de la forma: $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$. Esta particular arquietectura, permite modelar secuencias de datos de longitud variable que serían imprácticas para cualquier otra arquitectura.

Las RNNs modelan sistemas dinámicos, es decir, que cada instante t tenemos un sistema $s(t)$ con un estado diferente. Para que los siguientes estados del sistema tengan información de los estados pasados, las RNNs tienen al menos una conexión que es recurrente, es decir, que depende de un estado anterior o en algunos casos de estados futuros.

Una de sus principales ventajas es la compartición de parámetros. Esto quiere decir que para diferentes estados del sistema, podemos aplicar los mismos parámetros θ, permitiendo así, entrenar solo un conjunto de parámetros para todo el sistema en vez de un conjunto particular para cada estado.

En la Figura 1.7 se puede apreciar un ejemplo de una RNN que cuya ecuación es: $s(t) = f(s^{(t-1)}, \theta)$. Como se han definido las redes neuronales en términos de gráfos, es posible utilizar el algoritmo de back-propagation para calcular los gradientes de cada estado. Normalmente cuando se aplica back-propagation a una RNN se le conoce como back-propagation through time (BPTT).

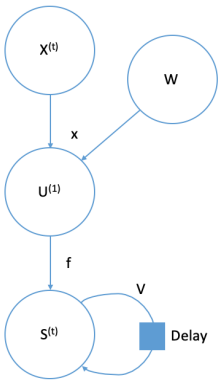

Figura 1.7: Ejemplo de una RNN que modela la ecuación $s(t) = f(s^{(t-1)}, \theta)$, donde θ representa los parámetros W y V, f la función no lineal aplicada a $Wx^{(t)}$ y V la matriz de pesos aplicada a $s^{(t-1)}$.

La manera que tiene una RNN para poder procesar una secuencia de longitud variable y brindar información respecto de ella, es a través de los estados $s(t)$ ya que estos si tienen una longitud fija. De este modo se puede decir que funcionan muy parecido a una CNN, ya que sumarizan las características de la secuencia hasta el momento t. Para realizar alguna predicción final, normalmente se utiliza algúna otra capa que efectue una función f, justo como lo hacen las CNNs.

1.3.1. Arquitectura Encoder-Decoder

Esta arquitectura de RNN se ha vuelto muy popular pues permite a una red procesar secuencias de longitud variable y obtener como resultado secuencias de longitud variable. Este tipo de problemas son conocidos como problemas sequence-to-sequence. Algunos ejemplos de problemas de este estilo son el reconocimiento del habla en tiempo real o el reconocimiento de la escritura.

La arquitectura se compone de dos redes recurrentes, la primera es denominada el *encoder*, el cual es una red que 'codifica' la entrada $x^{(t)}$. Si lo que se quiere es codificar una serie de tiempo, normalmente se usará una RNN y como salida será la señal $s(t)$. En este tipo de redes a la señal $s(t)$ se le conoce como el contexto C.

La segunda red que compone a la arquitectura es el *decoder*, el cual toma como entrada el contexto C generado previamente por el encoder y lo utiliza para generar la secuencia de salida. El decoder es en general una RNN.

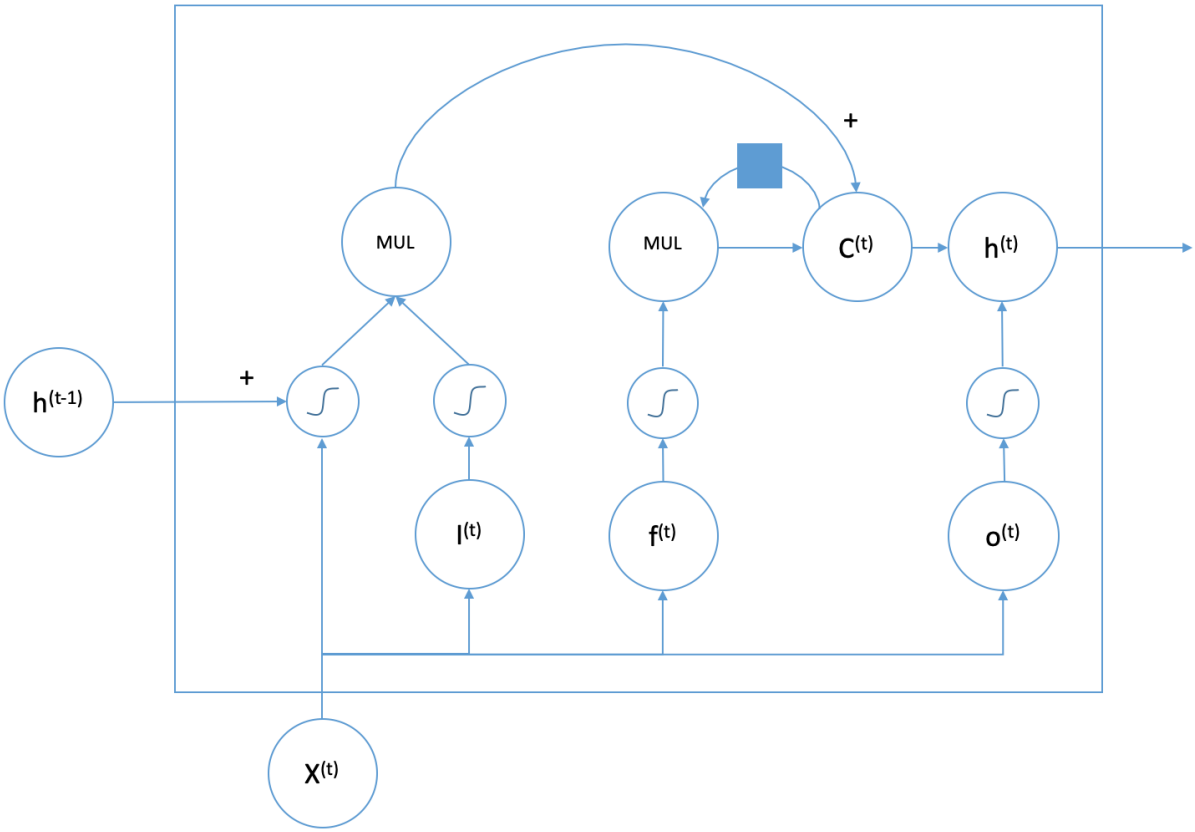

Figura 1.8: Una compuerta LSTM típica.

1.3.2. Long Short-Term Memory

Hasta ahora, las arquitecturas más exitosas para procesar secuencias son las llamadas Gated RNN. Estas redes permiten aprender secuencias muy largas o que requieran memorizar mucha información. El problema que existía, era que el gradiente tendía desaparecer conforme la red se volviera más y más profunda. Las Gated RNN son la solución más exitosa a este problema.

La Gated RNN más popular es la Long Short-Term Memory (LSTM). La idea de esta red, es crear ciclos entre los mismos nodos pero que esten condicionados con el contexto, de modo que la red pueda decidir cuando olvidar la información previa.

La LSTM es puede verse como una compuerta lógica que puede ser implementada en otra RNN para crear una arquitectura más robusta. En la Figura 1.8 se muestra un ejemplo de la compuerta LSTM. La LSTM implementa diferentes puertas las cuales son: **input gate**, **forget gate** y la **output gate**. Todas estas puertas ayudan a la LSTM a controlar en que momento es oportuno olvidar la información previamente encontrada en los anteriores estados.

1.3.3. Modelos de Lenguaje Condicional

Un modelo de lenguaje define la distribución de probabilidad sobre una serie de tokens en un lenguaje natural. Dependiendo de como el modelo es diseñado, un token puede representar una letra, una palabra o incluso un solo bit. Cuando se utilizan modelos de deep learning para resolver este problema, usualmente este modelo es llamado modelo de lenguaje neural.

Los modelos de lenguaje neurales tienen ventajas tales como:

- Resuelven el problema de dimensionalidad presentado por otros modelos.
- Son capaces de reconocer la similitud entre dos palabras sin perder la generalidad de codificarlas de una forma distinta.
- Alcanzan una buena comprensión estadística entre una palabra y su contexto.

En estos modelos, las palabras tienden a ser representadas por un identificador, el cual usualmente es un número, no obstante, una práctica muy común y que ha demostrado ser muy útil son los *word embeddings*. Estos concisten en codificar las palabras no a través de un solo número, sino a través de un conjunto de números decimales.

Un modelo de lenguaje condicional, es entonce, un modelo de lenguaje que toma en cuenta las palabras que han sido generadas previamente. Es decir:

$$P(w_1,\ldots,w_m) = \prod_{i=1}^{m} P(w_i|w_1,\ldots,w_{i-1}) \tag{1.15}$$

1.3.4. Sistema de Atención

El sistema de atención fue introducido por [8] y es en escencia un promedio con pesos. El resultado de un sistema de atención es un vector de contexto c_t el cual es obtenido al realizar el promedio entre las anotaciones a_t y multiplicarlas por unos pesos α_t, los cuales usualemente estan en el rango $[0,1]$. El modelo de atención es capaz de aprender en que parte del texto enfocarse en cada instante de tiempo t por lo que ha supuesto un impresionante avance en los modelos de lenguaje natural.

1.3.5. Positional Embeddings

Debido a que en un modelo de lenguaje una palabra puede representar cosas diferentes dependiendo de la posición que ocupe en la secuencia, es en ocasiones necesario brindar a la red de más información sobre la ubicación de cada token. Los *Positional Embeddings* fueron originalmente propuestos por [9]. La técnica conciste en sumar un vector *pe* a los *words embeddings* de cada token. Los autores del artículo proponen al vector *pe* de tal forma que:

$$pe_t^{(i)} = f(t)^{(i)} = \begin{cases} sin(\omega_k t), & \text{si } i = 2k \\ cos(\omega_k t), & \text{si } i = 2k+1 \end{cases} \tag{1.16}$$

con

$$\omega_k = \frac{1}{10000^{2k/d}} \tag{1.17}$$

donde d es la dimensión de los *word embeddings*.

1.3.6. Aprendizaje profundo como servicio

El entrenamiento de redes neuronales profundas, conocidas como aprendizaje profundo es en la actualidad áltamente complejo computacionalmente. Requiere un sistema con la combinación correcta de software, drivers, memoria, red y recursos de almacenamiento, por factores como estos los proveedores de cloud computing como Amazon Web Services, Microsoft Azure o Google Cloud ofrecen actualmente servicios de Machine Learning proveyendo API's como pueden ser de entrenamiento de modelos o de visualización de datos y generación de estadísticas, permitiendo que desarrolladores y científicos de datos se enfoquen mas en tareas como el entrenamiento de modelos, análisis de datos, etc.

Los modelos de aprendizaje profundo requieren de un proceso experimental e iterativo, requiriendo cientos e incluso miles de ejecuciones que requieren de un amplio poder de cómputo para encontrar la combinación correcta de las configuraciones e hiper-parámetros de la red neuronal. Esto puede tomar semanas o incluso meses y este tipo de servicios garantiza una disponibilidad en todo momento y en conjunto permite que un sistema completo esté dividido en módulos consiguiendo que el mantenimiento y detección de fallos sean tareas más sencillas.

En la figura 1.9 se puede observar la arquitectura de un sistema divido en módulos y que hace uso de herramientas de Machine Learning para dar respuesta a otros módulos del sistema.

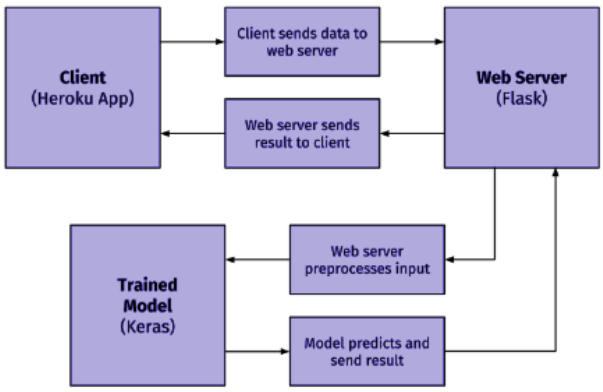

Figura 1.9: Ejemplo de sistema con módulo de Machine Learning

1.4. Métodos de reconocimiento de expresiones matemáticas

1.4.1. Análisis sintáctico dirigido

Los lenguajes libres de contexto, son aquellos que tienen una notación recursiva natural llamada *gramática libre de contexto*. Por ende, un lenguaje libre de contexto es todo aquel que puede ser representado por una gramática libre de contexto.

Una gramática libre de contexto G, queda definida de la siguiente manera:

$$G = (V, T, P, S) \tag{1.18}$$

de donde, V es el conjunto de variables, T el conjunto de símbolos terminales, P el conjunto de producciones y S el punto de inicio [10].

El lenguaje matemático, es decir, las expresiones matemáticas son un lenguaje libre de contexto. Esto quiere decir que es posible definir una gramática libre de contexto para definir a las expresiones matemáticas. Esta es una práctica muy común en la teoría de compiladores.

Sabiendo esto, una aproximación a la resolución del problema de reconocer expresiones matemáticas en imágenes es la que describe [11]. El método consiste en dos etapas:

- Reconocimiento de caracteres: Utilizar algún método de reconocimiento de patrones para localizar los caracteres y anotar sus coordenadas.
- Análisis sintáctico dirigido: Utilizar la etapa previa para determinar la jerarquía, basándose en un conjunto de reglas definidas (las gramáticas libres de contexto).

En la Figura 1.10, se puede ver un ejemplo del proceso de reconocimiento propuesto por este método.

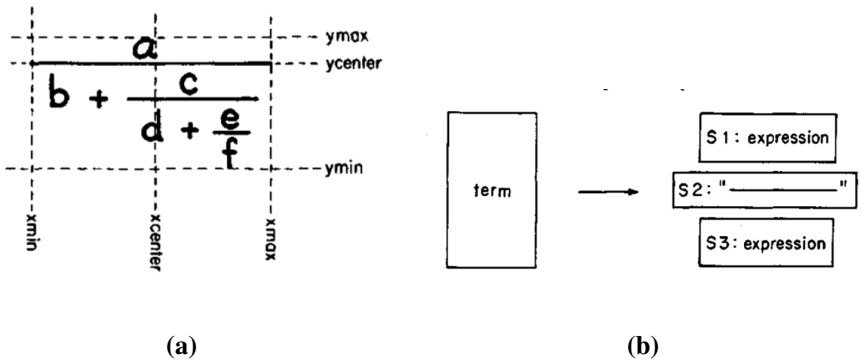

Figura 1.10: a) La primera etapa del método de reconocimiento, se anotan las coordenadas de de cada caracter. **b)** Se realiza un análisis sintáctico con las gramáticas libres de contexto definidas.

1.4.2. Análisis estructural

Este método coincide con el primero en que debe de utilizar una técnica de reconocimiento de patrones para etiquetar primero los símbolos. Las etiquetas que utiliza tienen que ver son su posición en la imagen, así como su tamaño.

La diferencia en este método, radica en su segunda etapa. No se realizará un análisis con gramáticas, en su lugar se intentara deducir la estructura jerárquica con algún otro método. El artículo [12] propone utilizar el algoritmo de Kruskal para obtener un árbol de recubrimiento mínimo por niveles, de este modo se puede saber la estructura de la expresión matemática recorriendo el gráfo resultante.La Figura 1.11 muestra un ejemplo de como funciona este método.

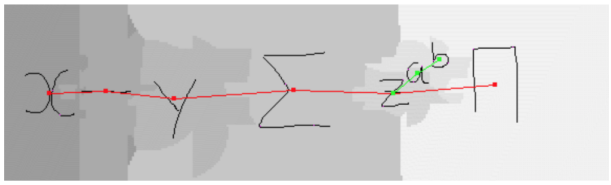

Figura 1.11: Utilización de un árbol de recubrimiento mínimo para el reconocimiento de expresiones matemáticas.

1.4.3. Image Captioning

Es una rama emergente del *deep learning* que ha ido ganando atención en los últimos años. Este campo es un punto intermedio entre la vision por computadora y el procesamiento del lenguaje natural. El actual estado del arte en image captioning tiene una aproximación similar a los modelos sequence-to-sequence, los cuales utilizan una arquitecura Encoder-Decoder [13].

Si se trata al problema de reconocer expresiones matemáticas en imagenes como un problema sequence-to-sequence de image captioning, se puede emplear una arquitectura de Encoding-Decoding para hacer la conversión de la imagen a LaTeX de manera directa. Esto permite a la red manejar imagenes de longitud variable y reconocer los símbolos a la vez que va reconociendo las expresiones.

La arquitectura que se está utilizando actualmente para solucionar este problema se muestra en la Figura 1.12 [13][14][15].

1.5. Aplicación web

En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador. En otras palabras, es un programa que se codifica en un lenguaje interpretable por los navegadores web en la que se confía la ejecución al navegador [16].

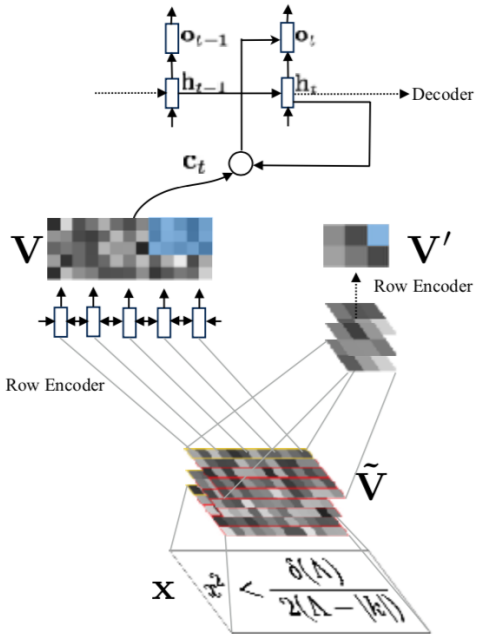

Figura 1.12: Arquitectura de image captioning para reconocer expresiones matemáticas en imagenes.

1.5.1. Django

Django es un framework de desarrollo web completamente desarrollado en Python. Permite de una manera rápida poder implementar una aplicación web en el lenguaje Python. Las siguientes, son de las principales características de Django:

- **Rápido**: Tiene como filosofía ayudar a los desarrolladores a crear aplicaciones en el menor tiempo posible.
- **Completo**: Incluye cientos de librerías que permiten ahorrar tiempo y automatizar tareas.
- **Seguro**: Es una de las principales características de Django ya que incluye soluciones a los principales ataques que puede sufrir una aplicación web.
- **Escalable**: Con el patrón de diseño de Django es posible incrementar o decrementar la capacidad de un sitio.
- **Versátil**: Es utilizado por muchas empresas y organizaciones a lo largo del mundo para crear diferentes tipos de proyectos.

1.5.1.1. Modelo-Vista-Template

El Modelo-Vista-Template (MTV) es el patrón de diseño que Django implementa. Este patrón es una modificación al conocido Modelo-Vista-Controlador (MVC). La diferencia radica en que

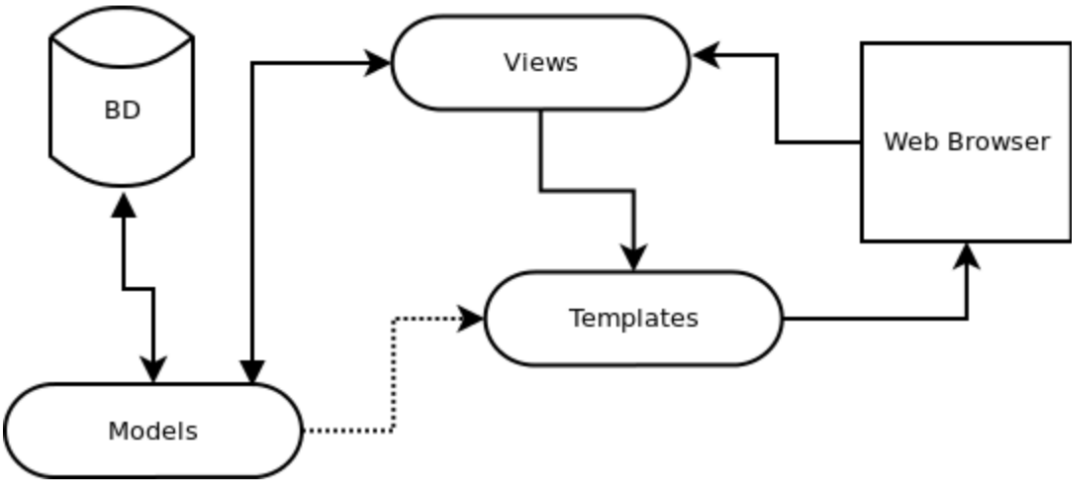

Figura 1.13: Representación del patrón de diseño Modelo-Vista-Template.

Django se encarga de hacer la parte del controlador, por ende el desarrollador solamente tiene que preocuparse por implementar la lógica de negocio y de como mostrará los datos. En la Figura 1.13, se puede ver una representación del patrón MTV. En el patrón de diseño MTV [17]:

- **Modelo**: La capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- **Template**: La capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento.
- **Vista**: La capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelos y las plantillas.

1.5.2. API REST

1.5.2.1. Application Programming Interface

Una Interfaz de Programación de Aplicaciones o API es un conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones.

Las API permiten que sus servicios se comuniquen con otros, sin necesidad de saber como están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero [18].

1.5.2.2. Represenational State Transfer

Representational State Transfer (REST) es un estilo de arquitectura basado en un conjunto de principios que describen como recursos interconectados son definidos y direccionados. Estos principios fueron descritos en el año 2000 por Roy Fielding como parte de obtención de su doctorado.
Es importante hacer énfasis en que REST es un **estilo de software de arquitectura** y no un conjunto de estandares. Como resultado, dichas aplicaciones o arquitecturas son en ocasiones referidas como aplicaciones RESTful o REST-style [19].
Una aplicación o arquitectura considerada RESTful o REST-style es caracterizada por:

- La funcionalidad y estado son dividiad en recursos distribuidos.
- Cada recurso es únicamente direccionable usando un conjunto uniforme y mínimo de comandos (típicamente usando comandos HTTP de GET, POST, PUT o DELETE).
- El protocolo es cliente/servidor, stateless, por capas y soporta almacenamiento cache.

La figura 1.14 ilustra el uso de REST para servicios Web.

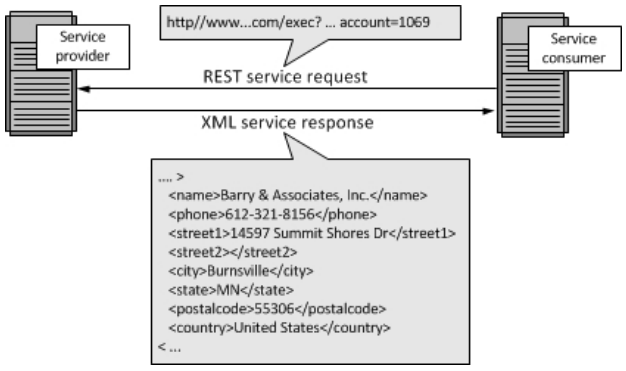

Figura 1.14: Consumidor y proveedor de servicios comunicandose mediante solicitudes y respuestas REST.

1.5.2.3. Autentificación

Para hablar de autentificación es necesario primero entender la diferencia entre identificación y autentificación, por un lado la identificación es la capacidad de identificar de forma exclusiva a un usuario de un sistema o una aplicación que se está ejecutando, mientras que la autentificación es la capacidad de demostrar que un usuario o una aplicación es quien dicha persona o aplicación asegura ser [20].

Para el caso de una REST API es en muchas ocasiones necesario que se lleve a cabo la autentificación para permitir o denegar el acceso a recursos de un servidor por ejemplo de acuerdo

a los permisos concedidos en función de las credenciales de autentificación para así asegurar que los datos sean visibles únicamente a aquellos que proporcionen las credenciales adecuadas y dispongan de los permisos necesarios.

1.6. Aplicación Android

1.6.1. Arquitectura Clean

El escribir código de buena calidad puede resultar difícil por lo cual es necesario seguir ciertas reglas para lograr esto, es por eso que se utiliza la arquitectura Clean en android.

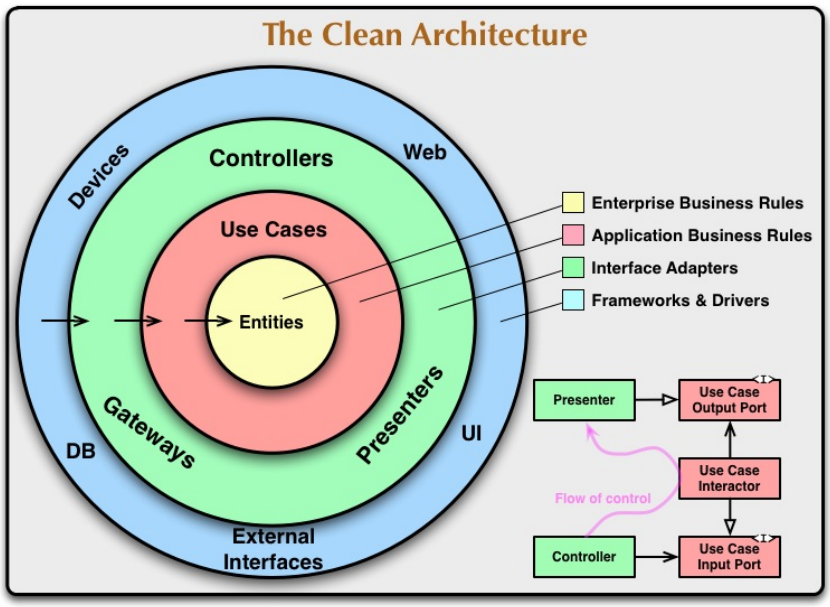

Figura 1.15: Arquitectura Clean [1]

La arquitectura Clean como se muestra en la figura 1.15 fue descrita por Robert C. Martin en 2012 y al utilizarla se puede llegar a tener las siguientes características en los sistemas [1]:

- Independientes de frameworks. El sistema no depende de las librerías o frameworks que se utilicen por lo que estas son fácilmente reemplazables.
- Que se puedan probar. Las reglas de negocio se pueden probar sin base de datos, interfaz de usuario, servidor web u otro elemento externo.
- Independientes de la interfaz de usuario. La interfaz de usuario puede cambiar fácilmente.

- Independientes de la base de datos. Las reglas de negocio no están ligadas a la fuente de datos por lo cual esta puede cambiar.
- Independencia de cualquier agente externo. Las reglas de negocio no dependen de otras capas, por lo cual se vuelve la parte más importante de la arquitectura.

La forma en la que se estructura la arquitectura permite seguir la regla de dependencia en la cual capas internas no deben conocer nada sobre las capas externas.

1.6.2. Componentes

1.6.2.1. Entidades

Son objetos de negocio de la aplicación, por lo cual en esta capa se tienen las reglas de negocio y también se tienen DTOs.

1.6.2.2. Casos de uso

En esta capa se ejecutan las reglas de negocio del sistema, se implementan todos los casos de uso que se tienen. Un caso de uso tiene que recibir datos estructurados y devolver datos estructurados por lo cual esta capa no se debe de ver afectada por capas superiores.

1.6.2.3. Adaptadores

La información que viene de los casos de uso y de las entidades se transforma en esta capa a algo que pueda ser entendido por la capa externa la cual puede ser una conexión a base de datos, una interfaz de usuario o un servidor web.

1.6.2.4. Frameworks y controladores

Esta capa externa está compuesta por framworks y herramientas tales como la base de datos, una interfaz de usuario, un cliente HTTP o un framework web.

1.6.3. SOLID

SOLID es un acrónimo que hace referencia a los cinco principios de la programación orientada a objetos, este acrónimo fue descrito por Robert C. Martin el seguir estos conceptos permite tener software que sea escalable y fácil de mantener, estos principios están ligados con la alta cohesión y bajo acoplamiento en el software. [21]

S-Responsabilidad simple Cada clase debe de tener una sola responsabilidad. De no seguir esto puede generar el problema de que alguna clase tenga comportamiento que nada tiene que ver con ella debido a que dicho comportamiento no se aisló en otra clase diferente. [22]

O-Abierto/Cerrado Las clases, métodos y módulos deben de ser abiertos a la extensión pero cerrados en su modificación. Esto implica no reescribir código que ya se tiene si no crear nuevo código que haga uso de lo que ya se tiene desarrollado, una forma de hacer esto es hacer uso de la herencia o utilizar interfaces.[22]

L-Sustitución Liskov Las subclases nunca deben de romper la definición de la clase padre. Cuando se utiliza una clase y existen clases que heredan de dicha clase debe de ser posible el utilizar una de estas clases en lugar de la clase padre.[22]

I-Segregación de la interfaz Si algún método de una interfaz no es utilizado no se debe de obligar a tener que implementarlo. Eso indica que las interfaces deben de ser bastante especificas para no tener métodos innecesarios por lo cual es preferible tener muchas interfaces con pocos métodos a pocas interfaces con pocos métodos que no se utilicen.[22]

D-Inversión de dependencias Los módulos de alto nivel no deben de depender de los de bajo nivel. Ambos deben depender de abstracciones, a su vez, las abstracciones no dependen de los detalles sino al contrario. Con esto se logra que las clases no estén totalmente acopladas debido a que en caso contrario es difícil de mantener. [22]

1.7. Estado del arte

Algunos sistemas similares que se han desarrollado son:

- Mathphix [23]
- MyScript Nebo [24]
- SESHAT [25]
- IDEAL Math Writer [26]

Los cuales se describen en la tabla 1.1:

SOFTWARE	CARACTERISTICAS	PRECIO EN EL MERCADO

Mathpix	Es una aplicación de escritorio en la que puedes usar un comando para tomar una captura de pantalla y convertir el texto capturado a LaTeX. También cuenta con un API de pago	Este producto cuenta con diferentes planes de pago según su uso o el tiempo que decidas pagarlo. Tiene distinto trato para empresas. Un ejemplo de suscripción mensual es $99 dólares el mes.
MyScript Nebo	Es una aplicación Android que transforma el texto escrito en el dispositivo en texto digital. Incluye soporte para ecuaciones. Es necesario el uso de una pluma digital.	$189.00 en Google Play Store
Mathpix	Es una aplicación de escritorio en la que puedes usar un comando para tomar una captura de pantalla y convertir el texto capturado a LaTeX. También cuenta con un API de pago	Este producto cuenta con diferentes planes de pago según su uso o el tiempo que decidas pagarlo. Tiene distinto trato para empresas. Un ejemplo de suscripción mensual es $99 dólares el mes.
SESHAT	Es un proyecto de doctorado de la Universidad Politécnica de Valencia open source. Convierte el texto de imágenes en texto digital y en formato LaTeX. Soporta ecuaciones. Necesita ser instalado mediante terminal en Linux.	No es una aplicación comercial

Tabla 1.1: Resumen de productos similares

Capítulo 2

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.

2.1. Android

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

2.1.1. 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 mencionado 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 2.1 con lo cual cada una de ellas tiene su propósito definido.

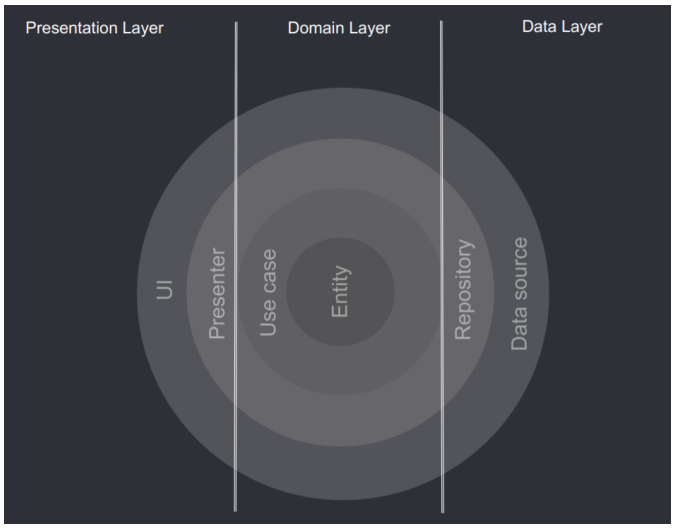

Figura 2.1: Tres capas que se tienen al utilizar la arquitectura Clean [2]

2.1.1.1. 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 2.2. 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 findAllProyecttosByUser se recupera la información necesaria para mandarla a las capas superiores.

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

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

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

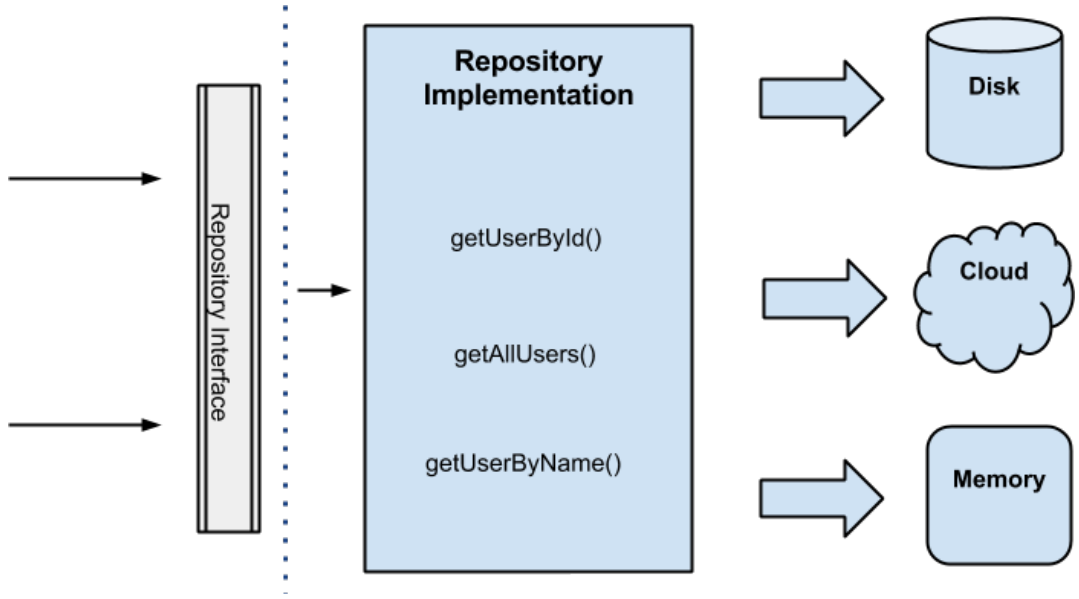

Figura 2.2: Capa de datos [3]

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 pude 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 realizara la petición con alguno de los métodos que tiene HTTP, se tienen los parámetro que se envían y cada método nos regresa una llamada asincrona 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,
      @Header("Authorization") String token);
```

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

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

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 utilizara para poder observar las peticiones que se realizan y brindar una retroalimentación a la hora de hacer pruebas y por ultimo el parser que se utilizara para trabajar y pasar de clases a datos que el API REST entienda y pueda utilizar, en este caso se utilizo 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(
        BASE_URL)
4               .addConverterFactory(GsonConverterFactory.create());
5
6       private static Retrofit retrofit = builder.build();
7       private static OkHttpClient.Builder httpClient = new OkHttpClient.Builder
        ();
8       private static HttpLoggingInterceptor loggingInterceptor = new
        HttpLoggingInterceptor();
9
10      public static <S> S createService(Class<S> serviceClass) {
11          if (!httpClient.interceptors().contains(loggingInterceptor)) {
12              loggingInterceptor.level(HttpLoggingInterceptor.Level.BODY);
13              httpClient.addInterceptor(loggingInterceptor);
14              builder.client(httpClient.build());
15              retrofit = builder.build();
16          }
17          return retrofit.create(serviceClass);
18      }
19  }
```

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      }
```

```
67 }
```

2.1.1.2. 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 2.3 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.

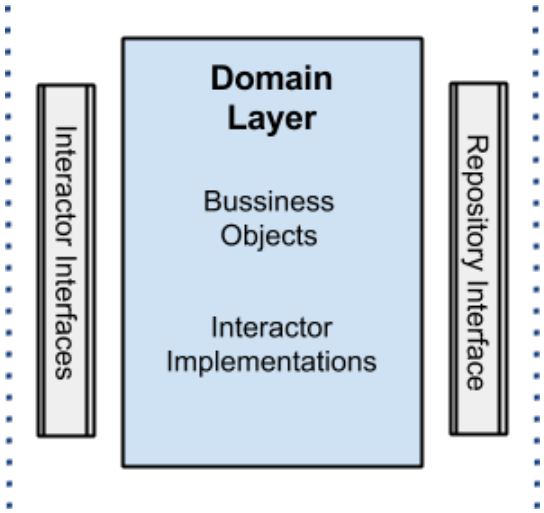

Figura 2.3: Capa de dominio [3]

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() {
```

```
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public T getResult() {
        return result;
    }

    public void setResult(T result) {
        this.result = result;
    }

    public List<T> getResults() {
        return results;
    }

    public void setResults(List<T> results) {
        this.results = results;
    }
}
```

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

```
public class UserInteractorImpl implements UserInteractor {
    public static final String TAG = UserInteractorImpl.class.
    getCanonicalName();
    private UserRepository repository;

    public UserInteractorImpl() {
        repository = new UserRepositoryImpl();
    }

    @Override
```

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

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 }
```

2.1.1.3. Capa de presentación

En esta capa como se muestra en la figura 2.4 se trabaja con la lógica relacionada a las interfaces que se tienen en la aplicación, es decir a actividades, fragmentos y archivos XML.

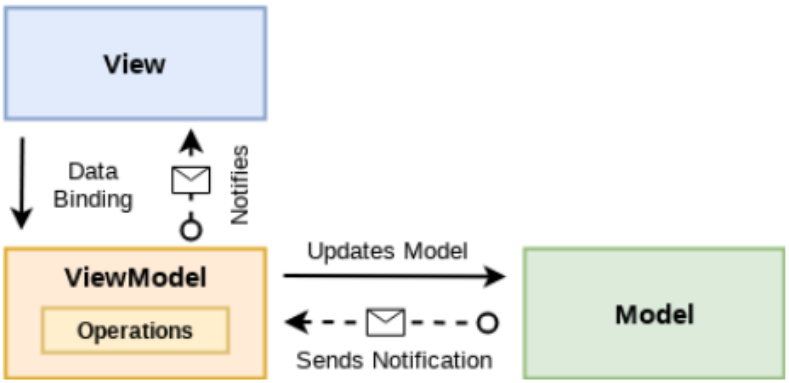

Figura 2.4: Capa de presentación [4]

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. [4]

- **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.

2.2. Web

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

2.2.1. 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.

2.2.1.1. 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.

2.2.1.2. 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.

2.2.1.3. Template

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

2.3. Conjunto de entrenamiento: 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 [15], son relativamente pocos para esperar una buena precisión, los elementos en en el conjunto son archivos de tipo INKML.

2.3.1. 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 LaTeX para fácilmente renderizarlo.

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

```
<ink xmlns="http://www.w3.org/2003/InkML">
<traceFormat>
<channel name="X" type="decimal"/>
<channel name="Y" type="decimal"/>
</traceFormat>
<annotation type="truth">$2^{-1}$</annotation>
```

```
<annotation type="UI">2011_IVC_CIEL_F696_E6</annotation>
<annotation type="copyright">LUNAM/IRCCyN</annotation>
<annotation type="writer">CIEL696</annotation>
<annotationXML type="truth" encoding="Content-MathML">
  <math xmlns='http://www.w3.org/1998/Math/MathML'>
    <msup>
      <mn xml:id="2_1">2</mn>
      <mrow>
        <mo xml:id="-_1">-</mo>
        <mn xml:id="1_1">1</mn>
      </mrow>
    </msup>
  </math>
</annotationXML>
<trace id="0">
3849 2989, 3849 2989, 3847 2979, 3853 2964, 3868 2949, 3875 2949, 3887 2962,
    3887 2974, 3880 2991, 3862 3011, 3822 3042, 3827 3052, 3839 3061, 3898
    3049
</trace>
...
<trace id="2">
4009 2905, 4009 2905, 4027 2900, 4049 2887, 4040 2947, 4040 2958
</trace>
<traceGroup xml:id="3">
  ...
</traceGroup>
</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.

2.3.2. Conversión a imagen

Con la información de los trazos 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:

```
traces_data = []


    tree = ET.parse(inkml_file_abs_path)
    root = tree.getroot()
    doc_namespace = "{http://www.w3.org/2003/InkML}"

    'Stores traces_all with their corresponding id'
    traces_all = [{'id': trace_tag.get('id'),
                   'coords': [[round(float(axis_coord)) if float(axis_coord).is_integer() else round(float(axis_coord) * 10000) \
                               for axis_coord in coord[1:].split(' ')] if coord.startswith(' ') \
                              else [round(float(axis_coord)) if float(axis_coord).is_integer() else round(float(axis_coord) * 10000) \
                               for axis_coord in coord.split(' ')] \
                             for coord in (trace_tag.text).replace('\n', '').split(',')]} \
                             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 guardarlo como imagen.

```
def inkml2img(input_path, output_path):
    traces = get_traces_data(input_path)
    plt.gca().invert_yaxis()
    plt.gca().set_aspect('equal', adjustable='box')
    plt.axes().get_xaxis().set_visible(False)
    plt.axes().get_yaxis().set_visible(False)
    plt.axes().spines['top'].set_visible(False)
    plt.axes().spines['right'].set_visible(False)
    plt.axes().spines['bottom'].set_visible(False)
    plt.axes().spines['left'].set_visible(False)
    for elem in traces:
        ls = elem['trace_group']
        for subls in ls:
            data = np.array(subls)
            x,y=zip(*data)
```

```
            plt.plot(x,y,linewidth=2,c='black')
    plt.savefig(output_path, bbox_inches='tight', dpi=100)
    plt.gcf().clear()
```

Esto 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:

```
def inkml2tag(self, inkml_path):
    tree = ET.parse(inkml_path)
    root = tree.getroot()
    prefix = "{http://www.w3.org/2003/InkML}"
    GT_tag = [GT for GT in root.findall(prefix + 'annotation') if GT.attrib
   == {'type': 'truth'}]
    if GT_tag is None or len(GT_tag) == 0:
      return ""
    if GT_tag[0] is None or GT_tag[0].text is None:
      return ""
    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 conteniendo la ruta relativa de la imagen y la expresión en LaTeX correspondiente separados por coma.

```
converted_expressions/200924-1312-148.i.png, \left ( { a + 7 } \right )
converted_expressions/200926-1550-27.i.png, \mbox { G }
converted_expressions/TrainData2_0_sub_71.i.png,\frac{\sin B + \sin C}{\cos B
    + \cos C}
converted_expressions/200923-1251-164.i.png, { \mbox { z } } ^ { \phi }
converted_expressions/200923-1553-186.i.png, \sqrt { { \mbox { N } - \mbox {
   P } } }
converted_expressions/formulaire012-equation003.i.png,$4 \pi d^2$
converted_expressions/formulaire012-equation036.i.png,$a_ns_3 + a_{n-1}s_2 +
   a_{n-2}s_1 + 3a_{n-3} = 0$
converted_expressions/79_carlos.i.png,$p = \sqrt{a^2 + b^2 - 2ab \cos A}$
converted_expressions/200923-1553-313.i.png, \left ( \sum { \mbox { S } } \
   right )
```

```
converted_expressions/formulaire010-equation073.i.png,$x^7-x^6-x^4-x^2-1$
```

2.3.3. Generador de secuencia de tokens

El archivo CSV generado con lo descrito anteriormente no es suficiente para cargarlo en TensorFlow, la expresión matemática en LaTeX 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 **??**.

```
tokens = [
    #'corta',
    #'larga',
    'alpha',
    'pi',
    'beta',
    'gamma',
    'lambda',
    'sigma',
    'mu',
```

```
  def tokenizeDataset(self, file):

    fileTokenized = open("tokenized_test.csv", "w")

    myMap = dict()
    for line in file:

      self.lexer.input(",".join(line.split(",")[1:]))
      #print(line.split(",")[1])
      #print(repr(line.split(",")[1]))
      token = self.lexer.token()
      arr = []
      while token is not None:
        print(token)
        myMap[token.type]= token.value  #fileMap.write(token.type+","+token.
    value+"\n")
        arr.append(Rules.tokens.index(token.type) + 1 ) #+1 means shift one
    due to 0 is reserved
        token = self.lexer.token()
      fileTokenized.write(line.split(",")[0] + "," + toString(arr) + "\n")
    fileTokenized.close()
```

```
converted_expressions/200924-1312-148.i.png,64 33 41 74 47 58 42 65 34
converted_expressions/200926-1550-27.i.png,41 115 42
converted_expressions/TrainData2_0_sub_71.i.png,12 41 14 100 47 14 99 42 41
    16 100 47 16 99 42
converted_expressions/200923-1251-164.i.png,41 41 78 42 42 62 41 9 42
converted_expressions/200923-1553-186.i.png,13 41 41 41 107 42 46 41 109 42
    42 42
converted_expressions/formulaire012-equation003.i.png,55 2 79 62 53
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
converted_expressions/79_carlos.i.png,86 48 13 41 74 62 53 47 77 62 53 46 53
    74 77 16 102 42
converted_expressions/200923-1553-313.i.png,64 33 18 41 41 108 42 42 65 34
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 LaTeX.

2.4. 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 $x \in \mathbb{R}^{HxWx1}$, siendo H y W la altura y el ancho de la imagen respectivamente. La secuencia de caracteres objetivo y es de la forma $y_1, y_2, \ldots, y_k$ siendo k la longitud de la secuencia y y_i es un caracter válido de LaTeX para el presente Trabajo Terminal.

Para resolver el problema de reconocer expresiones matemáticas, existen aproximaciones secuenciales o globales. De acuerdo con [27] 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.

2.4.1. Modelo

Como se mencionó anteriormente la arquitectura de la red neuronal es del tipo *Encoder-Decoder*, en la Figura 2.5 se muestra un esquema de la red neuronal implementada. La traducción de imagenes se realiza de la siguiente manera: Primero se extraen las características de la imágen 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.

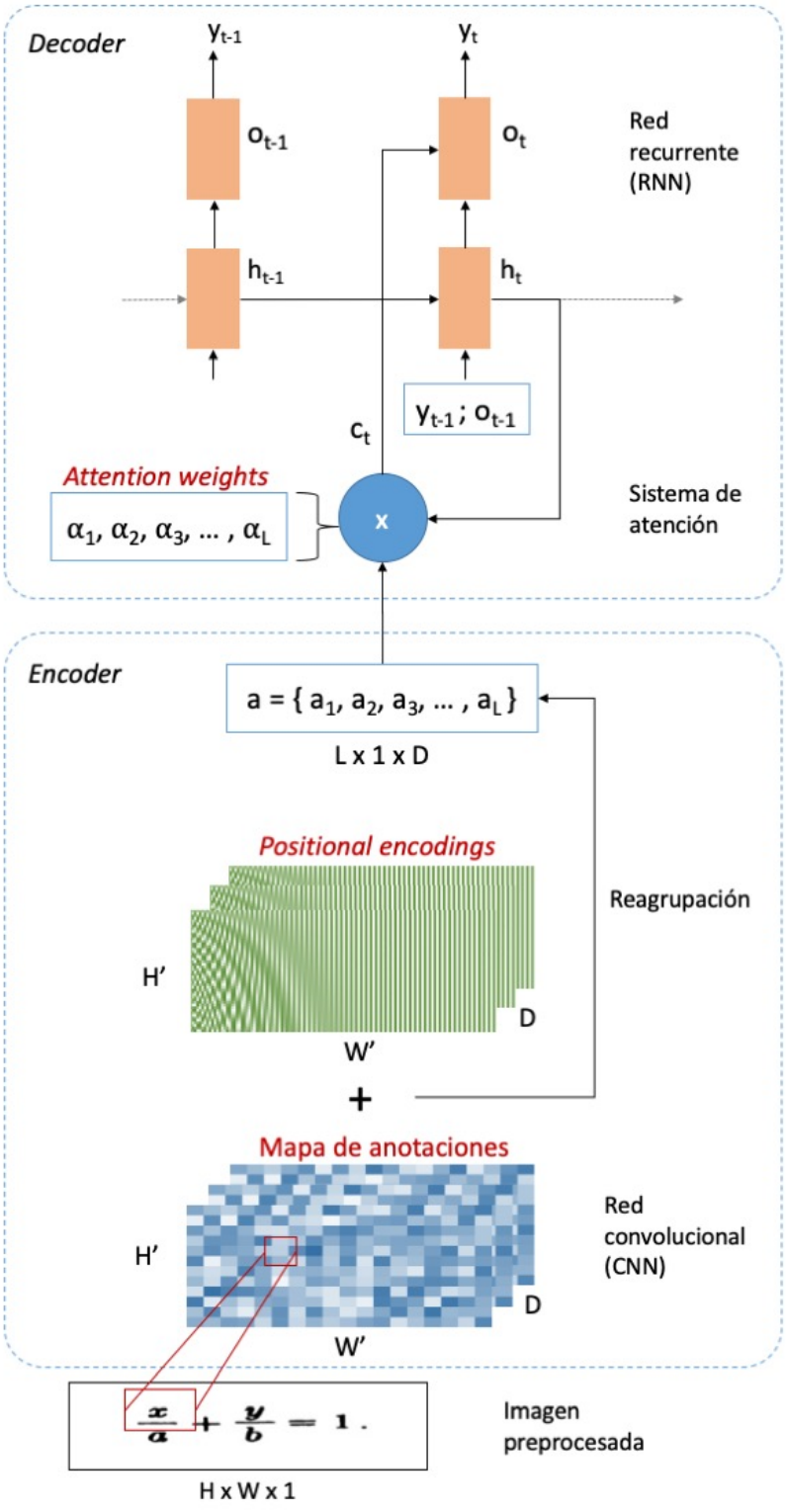

Figura 2.5: Modelo *Encoder-Decoder* de la red neuronal.

Número de capa	Capa Convolucional				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 2.1: Especificación de la CNN. El 'Kernel' esta denotado como *número de filtros*-(*dimensiones del filtro*).

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

2.4.1.1. Encoder

La extracción de características de la imagen de entrada x se realiza con una CNN multicapa con *max pooling* y *batch normalization* la cual es ahora un estandar y está completamente inspirada en [28]. La Tabla 2.1 sumariza la arquitectura de la red utilizada. Las características extraidas están almacenadas en vector $v \in \mathbb{R}^{H'xW'xD}$, 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 [28] utiliza una RNN bidireccional con el vector v como entrada para aprender estos positional encodins. No obstante, acorde con los inventores de esta técnica [9], los positional encodings fijos producen resultados muy similares a los aprendidos. Así, se optó por utilizar positional encodings fijos y se utlizó la generalización a dos dimensiones propuesta por [29]. El vector de positional encodings $pe \in \mathbb{R}^{H'xW'xD}$ se obtiene con las siguientes ecuaciones:

$$pe(w,h,2i) = sin\left(\frac{w}{10000^{\frac{4i}{D}}}\right) \tag{2.1}$$

$$pe(w,h,2i+1) = cos\left(\frac{w}{10000^{\frac{4i}{D}}}\right) \tag{2.2}$$

$$pe(w,h,2j+D/2) = sin\left(\frac{h}{10000^{\frac{4j}{D}}}\right) \tag{2.3}$$

$$pe(w,h,2j+1+D/2) = cos\left(\frac{h}{10000^{\frac{4j}{D}}}\right) \tag{2.4}$$

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

$$a = v + pe \tag{2.5}$$

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 $a = \{a_1, a_2, \ldots, a_L\}$ con $a_i \in \mathbb{R}^D$.

2.4.1.2. 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:

$$p(y_t|y_1, \ldots, y_{t-1}, a) = softmax(W^{out} o_t) \tag{2.6}$$

con

$$o_t = tanh(W^c[h_t; c_t]) \tag{2.7}$$

$$h_t = LSTM(h_{t-1}, [y'_{t-1}; o_{t-1}]) \tag{2.8}$$

$$y'_t = Ey_{t-1} \tag{2.9}$$

donde W^{out}, W^c y E son parámetros que serán aprendidos. La matríz E, es también conocida en la literatura como la matríz 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:

$$h_0 = tanh(W_{h_0} a + b_{h_0}) \tag{2.10}$$

$$o_0 = tanh(W_{o_0}a + b_{o_0}) \tag{2.11}$$

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 [8]:

$$e_t = u(h_t, a) \tag{2.12}$$

$$\alpha_t = softmax(e_t) \tag{2.13}$$

$$c_t = \phi(\alpha_t, a) \tag{2.14}$$

ϕ y u son funciones que pueden ser escogidas libremente, en este caso se utilzarón las mismas que en [28]:

$$e_{it} = \beta^T tanh(W_h h_{i-1} + W_v a) \tag{2.15}$$

$$c_t = \sum_{i}^{L} \alpha_{it} a_i \tag{2.16}$$

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.

2.4.1.3. 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 .

El código que implementa la red es el siguiente:

```
def get_positional_encoding_2d(height, width, d_model):

    positional_encodings = np.zeros((height, width, d_model))
    d_model = int(d_model / 2)
    h_vector = np.arange(height)
    w_vector = np.expand_dims(np.arange(width), axis=1)

    div_term = np.arange(d_model) // 2
    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
       =(2,2), padding='same')
25          self.pooling2 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides
       =(2,2), padding='same')
26          self.pooling3 = tf.keras.layers.MaxPool2D(pool_size=(2,1), strides
       =(2,1), padding='same')
27          self.pooling4 = tf.keras.layers.MaxPool2D(pool_size=(1,2), strides
       =(1,2), padding='same')
28
29          self.conv1 = tf.keras.layers.Conv2D(64,  (3,3), (1,1), 'same',
       activation='relu')
30          self.conv2 = tf.keras.layers.Conv2D(128, (3,3), (1,1), 'same',
       activation='relu')
31          self.conv3 = tf.keras.layers.Conv2D(256, (3,3), (1,1), 'same',
       activation='relu')
32          self.conv4 = tf.keras.layers.Conv2D(256, (3,3), (1,1), 'same',
```

```
    activation='relu')
33          self.conv5 = tf.keras.layers.Conv2D(512, (3,3), (1,1), 'same',
    activation='relu')
34          self.conv6 = tf.keras.layers.Conv2D(512, (3,3), (1,1), 'valid',
    activation='relu')
35
36          self.batch_normalization1 = tf.keras.layers.BatchNormalization()
37          self.batch_normalization2 = tf.keras.layers.BatchNormalization()
38          self.batch_normalization3 = tf.keras.layers.BatchNormalization()
39
40      def call(self, x, training=False):
41
42          y = self.conv1(x)
43          y = self.pooling1(y)
44          y = self.conv2(y)
45          y = self.pooling2(y)
46          y = self.conv3(y)
47          y = self.batch_normalization1(y, training)
48          y = self.conv4(y)
49          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):
```

```
    super(InitialHidden, self).__init__()

    self.fc = tf.keras.layers.Dense(units, activation='tanh')

  def __call__(self, x):

    x = tf.math.reduce_mean(x, axis=1)
    return self.fc(x)


class BahdanauAttention(tf.keras.layers.Layer):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()

    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(Encoder output) shape == (batch_size, L, 512)

    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # score shape == (batch_size, L, hidden_size)
    score = tf.nn.tanh(self.W1(features) + self.W2(hidden_with_time_axis))

    # attention_weights shape == (batch_size, L, 1)
    # you get 1 at the last axis because you are applying score to self.V
    attention_weights = tf.nn.softmax(self.V(score), axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)
```

```
39
40      return context_vector, attention_weights
41
42
43  class Decoder(tf.keras.Model):
44
45    def __init__(self, units, embedding_dim, vocab_size):
46      super(Decoder, self).__init__()
47
48      self.Wout = tf.keras.layers.Dense(vocab_size, use_bias=False)
49      self.o_t = tf.keras.layers.Dense(units, use_bias=False, activation='tanh'
        )
50      self.lstm = tf.keras.layers.LSTM(units, return_state=True)
51      self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
52
53      self.attention = BahdanauAttention(units)
54
55
56    def __call__(self, features, hidden, previous_y, previous_out):
57
58      # embedding_previous_y shape == (batch_size, embedding_dim)
59      embedding_previous_y = self.embedding(previous_y)
60
61      # previous_out shape == (batch_size, units)
62      # lstm_input shape == (batch_sise, embedding_dim + units)
63      lstm_input = tf.concat([embedding_previous_y, previous_out], axis=1)
64      lstm_input = tf.expand_dims(lstm_input, axis=1)
65
66      # hidden is a list of hidden state and carry state respectively
67      # each element has a shape == (batch_size, units)
68      # actually output and state are the same tensors
69      output, state, carry = self.lstm(lstm_input, initial_state=hidden)
70
71      # context_vector shape == (batch_size, hidden_size)
72      # attention_weights == (batch_size, L, 1)
```

```
73      context_vector, attention_weights = self.attention(features, state)
74
75      # calculating the output
76      out = self.o_t(tf.concat([output, context_vector], axis=1))
77      output = self.Wout(out)
78
79      return output, out, attention_weights, [state, carry]
```

2.4.1.4. Resultados

2.4.1.5. Experimentos Previos

Capítulo 3

Pruebas del sistema

En este capítulo se detallan las diversas pruebas realizadas a los módulos desarrollados con el fin de verificar y validar su correcto funcionamiento.

3.1. Pruebas unitarias

3.1.1. Aplicación android

Para realizar las pruebas en android se utilizo la biblioteca JUnit la cual es utilizada para realizar pruebas unitarias en Java.

JUnit se utilizo sobre aquellas clases que se utilizaron para trabajar partes en especifico de la lógica del negocio que no involucraban utilitarias de android, es decir, que solamente utilizan Java.

3.1.1.1. Pruebas sobre la clase DateFormater

La clase DateFormater se utiliza en la aplicación para obtener fechas y presentarlas en un formato correcto en pantalla, así como pasar de una cadena de texto a un objeto de tipo Date, por lo cual las as pruebas realizadas sobre esta clase fueron tres.

1. En la primera se valida que la conversión de una cadena de texto a un objeto de tipo Date se haga correctamente siempre y cuando se trate de una cadena valida.

2. En la segunda prueba se valida que se pueda pasar de un tipo date a una cadena de texto y que el valor que se obtenga sea el mismo.

3. En la ultima prueba se valida que se lance una excepción si el valor a convertir a una cadena de texto no es valido.

El código utilizado en cada una de las pruebas es el siguiente.

```
1  package com.equipo.superttapp;
2
3  import com.equipo.superttapp.util.DateFormater;
4
5  import org.junit.Assert;
6  import org.junit.Test;
7
8  import java.util.Calendar;
9  import java.util.Date;
10
11 public class DateFormaterTest {
12     @Test
13     public void convertStringToDate_validString_shouldParseToDate() {
14         Calendar cal = Calendar.getInstance();
15         cal.set(2020, 0, 1, 0, 0, 0);
16         cal.setTimeInMillis(1577858400000L);
17         Date date = cal.getTime();
18         Assert.assertEquals("Probando que se obtenga un objeto date con la
       fecha en string", date.getTime(), DateFormater.convertStringToDate("
       01/01/2020").getTime());
19     }
20
21     @Test
22     public void convertDateToString_validDate_shouldParseToString() {
23         Calendar cal = Calendar.getInstance();
24         cal.set(2020, 0, 1);
25         Date date = cal.getTime();
26         Assert.assertEquals("Probando que no se obtenga null despues de
       realizar un cast", "01/01/2020", DateFormater.convertDateToString(date));
27     }
28
29     @Test(expected = NullPointerException.class)
30     public void convertDateToString_NoValidDate_shouldThrowException() {
31         DateFormater.convertDateToString(null);
32     }
```

```
33 }
```

En el código anterior, cada uno de los métodos anotados con @Test es una prueba a ejecutar, la forma en la que se valida que la ejecución sea correcta es con la linea de código correspondiente a Assert.assertEquals, la cual compara dos valores y los compara, en caso de ser iguales la prueba es exitosa en caso contrario la prueba ha fallado.

En la ultima prueba la anotación @Test tiene como parámetro una excepción a esperar, si la excepción se dispara la prueba es correcta, por otro lado, si no se da el caso la prueba ha fallado.

La correcta ejecución de las pruebas se puede apreciar en la figura **??**

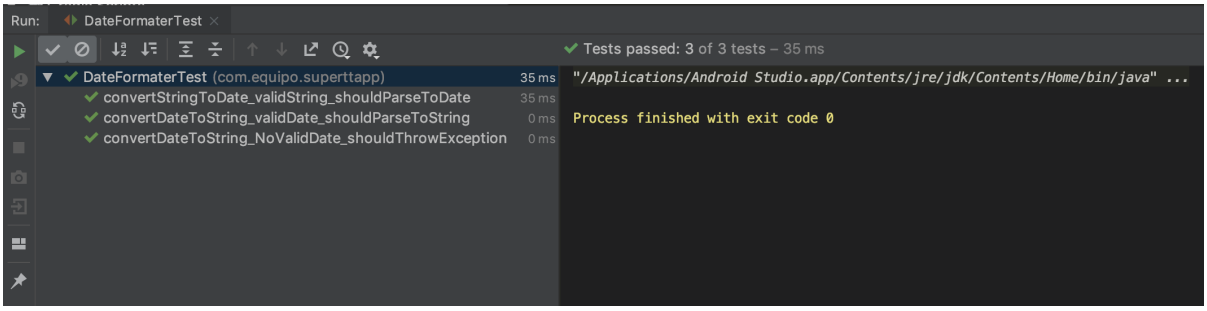

Figura 3.1: Resultados de las pruebas de la clase DateFormater

3.1.1.2. Pruebas sobre la clase RNN002

La clase RNN002 se utiliza en la aplicación para validar el formato de diversos campos que se tienen en los diferentes formularios que incluye la aplicación, es por esto que las pruebas que se realizaron en este caso fueron las siguientes.

1. En la primera prueba se verifica que para un email valido, el resultado sea true.
2. En la siguiente prueba, para un email no valido, el resultado debe de ser false.
3. En la siguiente prueba, para una contraseña valida, el resultado debe de ser true.
4. En la siguiente prueba, para una contraseña no valida, el resultado debe de ser false.
5. En la siguiente prueba, para dos contraseñas validas, el resultado debe de ser true.
6. En la siguiente prueba, para dos contraseñas, una de ellas no valida, el resultado debe de ser false.
7. En la siguiente prueba, para un nombre valido, el resultado debe de ser true.
8. En la siguiente prueba, para un nombre no valido, el resultado debe de ser false.
9. En la siguiente prueba, para un apellido valido, el resultado debe de ser true.

10. En la siguiente prueba, para un apellido no valido, el resultado debe de ser false.
11. En la siguiente prueba, para un nombre de proyecto valido, el resultado debe de ser true.
12. En la ultima prueba, para un nombre de proyecto no valido, el resultado debe de ser false.

El código utilizado en cada una de las pruebas es el siguiente.

```
1  package com.equipo.superttapp;
2
3  import com.equipo.superttapp.util.RN002;
4
5  import org.junit.Assert;
6  import org.junit.Test;
7
8  public class RN002Test {
9      @Test
10     public void isEmailValid_validEmail_shouldReturnTrue() {
11         Assert.assertTrue("Prrobando que se regrese true en una email valido"
       , RN002.isEmailValid("carlostonatihu@gmail.com"));
12     }
13
14     @Test
15     public void isEmailValid_invalidEmail_shouldReturnFalse() {
16         Assert.assertFalse("Prrobando que se regrese false en una email no
       valido", RN002.isEmailValid("carlostonatihumail.com"));
17     }
18
19     @Test
20     public void isPassordValid_validPassword_shouldReturnTrue() {
21         Assert.assertTrue("Debe retornar true ya que la contrasena es valida"
       , RN002.isPasswordValid("madremiawilly"));
22     }
23
24     @Test
25     public void isPassordValid_invalidPassword_shouldReturnFalse() {
26         Assert.assertFalse("Debe retornar false ya que la contrasena es
       invalida", RN002.isPasswordValid(""));
```

```
27      }
28
29      @Test
30      public void isSecondPassordValid_validPassword_shouldReturnTrue() {
31          Assert.assertTrue("Debe retornar true ya que las dos contrasenas son
        validas", RN002.isSecondPasswordValid("madremiawilly", "madremiawilly"));
32      }
33
34      @Test
35      public void isSecondPassordValid_invalidPassword_shouldReturnFalse() {
36          Assert.assertFalse("Debe retornar true ya que las dos contrasenas son
         validas", RN002.isSecondPasswordValid("madremiawilly", null));
37      }
38
39      @Test
40      public void isNameValid_validName_shouldReturnTrue() {
41          Assert.assertTrue("Debe retornar true ya que el nombre es valido",
        RN002.isNameValid("Carlos Tonatihu"));
42      }
43
44      @Test
45      public void isNameValid_invalidName_shouldReturnFalse() {
46          Assert.assertFalse("Debe retornar false ya que el nombre no es valido
        ", RN002.isNameValid(" "));
47      }
48
49      @Test
50      public void isLastNameValid_validLastName_shouldReturnTrue() {
51          Assert.assertTrue("Debe retornar true ya que el apellido es valido",
        RN002.isLastnameValid("Barrera"));
52      }
53
54      @Test
55      public void isLastNameValid_invalidLastName_shouldReturnFalse() {
56          Assert.assertFalse("Debe retornar false ya que el apellido no es
```

```
    valido", RN002.isNameValid(null));
57   }
58
59   @Test
60   public void isProyectoNombreValid_validProyectoNombre_shouldReturnTrue()
    {
61       Assert.assertTrue("Debe retornar true ya que el nombre del proyecto
    es valido", RN002.isProyectoNombreValid("Proyecto 1"));
62   }
63
64   @Test
65   public void isProyectoNombreValid_invalidProyectoNombre_shouldReturnFalse
    () {
66       Assert.assertTrue("Debe retornar false ya que el nombre del proyecto
    no es valido", RN002.isProyectoNombreValid("1"));
67   }
68 }
```

En el caso de estas pruebas presentadas en el código anterior, se utilizaron los métodos Assert.assertTrue y Assert.assertFalse para verificar que las pruebas fueran correctas. En el caso del primero de estos métodos la prueba es exitosa si el valor que retorna el método a probar es falso, mientras que para el segundo método si el valor de retorno es falso la prueba es correcta.

La ejecución de las pruebas se muestra en la figura 3.2.

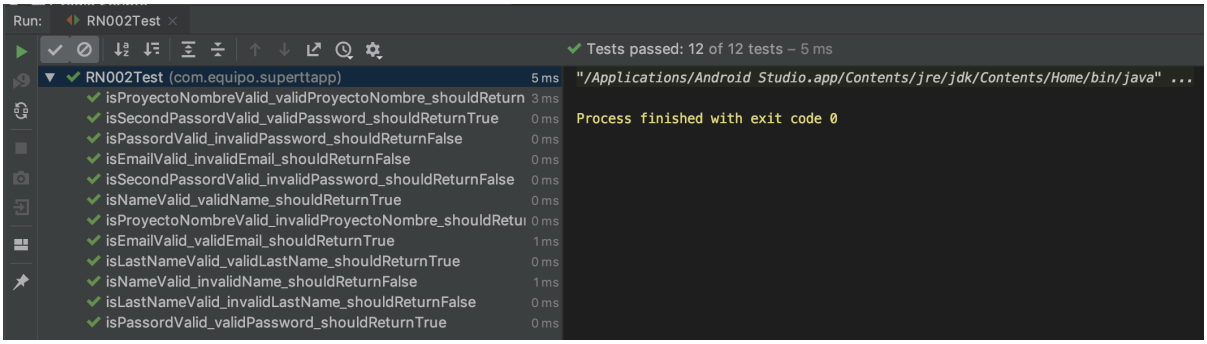

Figura 3.2: Resultados de las pruebas de la clase RNN002

3.2. Pruebas de integración

3.3. Pruebas de requerimientos funcionales

Bibliografía

[1] R. C. Martin, "The Clean Architecture." https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html, 2012. [Consultado: 2019-09-21]. IV, 21

[2] 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]. IV, 26

[3] F. Cejas, "Architecting Android...The clean way?." https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/, 2014. [Consultado: 2019-09-21]. IV, IV, 28, 33

[4] F. Cejas, "Architecting Android...Reloaded." https://fernandocejas.com/2018/05/07/architecting-android-reloaded/, 2018. [Consultado: 2019-09-21]. IV, 39

[5] A. Seitz, S, "Computer vision." https://courses.cs.washington.edu/courses/cse576/, 2000. [CSE576: Computer Vision]. 2, 5

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 7, 9

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. 8

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014. 14, 52

[9] 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. 14, 50

[10] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. 2001. 16

[11] R. H. Anderson, "Syntax directed recognition of hand-printed two-dimensional mathematics," *Harvard University*. 16

[12] E. Tapia and R. Rojas, "Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance," *Freie Universit at Berlin, Institut fur Informatik*. 17

[13] G. Genthial and R. Sauvestre, "Image to latex," *Stanford University*. 17

[14] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush, "Image-to-markup generation with coarse-to-fine attention," *34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017*, 2017. 17

[15] J. Zhanga, J. Dua, S. Zhanga, 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, 41

[16] S. Luján-Mora, "Programación en internet: clientes web," 2001-10-08. 17

[17] A. Holovaty and J. Kaplan-Moss, "El libro de Django 1.0." `https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv`. 19

[18] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: What do software developers want?," *Journal of Technical Writing and Communication*, vol. 48, p. 295–330, 07 2018. 19

[19] E. Sundvall, M. Nyström, D. Karlsson, M. Eneling, R. Chen, and H. Orman, "Applying representational state transfer (rest) architecture to archetype-based electronic health record systems," *BMC medical informatics and decision making*, vol. 13, p. 57, 05 2013. 20

[20] "Identificación y autentificación." `https://www.ibm.com/support/knowledgecenter/es/SSFKSJ_7.5.0/com.ibm.mq.sec.doc/q009740_.htm`. 20

[21] C. Salazar, "Los principios de SOLID de la programación orientada a objetos." `https://www.codesolt.com/tutoriales/fundamentos/solid/`, 2018. [Consultado: 2019-09-21]. 22

[22] A. Murthy, "Exploring S.O.L.I.D Principle in Android." `https://proandroiddev.com/exploring-s-o-l-i-d-principle-in-android-a90947f57cf0`, 2018. [Consultado: 2019-09-21]. 23

[23] "Mahphix." `https://mathpix.com/about`. 23

[24] "MyScript Nebo." `https://www.nebo.app/es/`. 23

[25] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "An integrated grammar-based approach for mathematical expression recognition," *Pattern Recognition*, vol. 51, pp. 135 – 147, 2016. 23

[26] "IDEAL Math Writer." `https://ideal-math-writer.soft112.com/`. 23

[27] J. Zhang, J. Du, and L. Dai, "Multi-scale attention with dense encoder for handwritten mathematical expression recognition," *CoRR*, vol. abs/1801.03530, 2018. 48

[28] 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. 50, 52

[29] 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. 50