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, 5 de noviembre de 2019

Índice general

Índice de figuras

Índice de tablas

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

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 [2].

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.

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. [2]

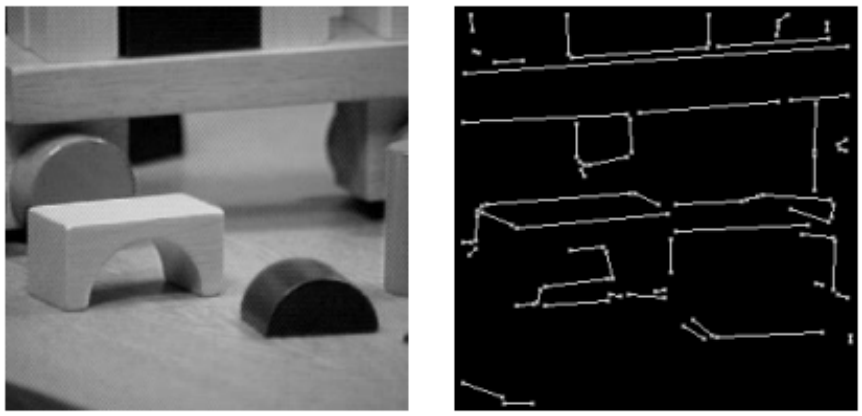

Figura 1.1: Imagen con bloques (Izquierda) y conjunto de segmentos de linea extraídos (Derecha).

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 [3] 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 1.2 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 1.2 para modelar a un perceptrón de n capas.

1.2.2. 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

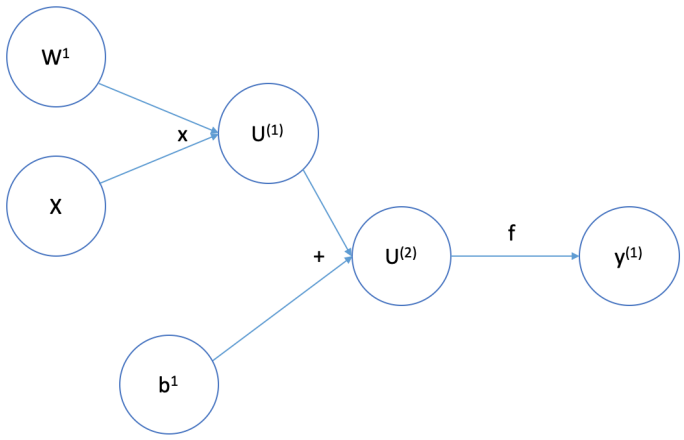

Figura 1.2: Ejemplo de un perceptrón de una sola capa representado mediante un gráfo computacional, siendo x la entrada, w^1 la matriz de pesos, b^1 la matriz de bias, $u^{(1)}$ y $u^{(2)}$ nodos intermedios en el gráfo y y^1 la salida de la red.

una multiplicación tensorial normal [3].

La operación de convolución que se utiliza en matemáticas e ingeniería esta definida de la siguiente manera:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)w(t-\tau)d\tau \tag{1.1}$$

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:

$$y(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau) \tag{1.2}$$

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

$$y(i,j) = \sum_{m}\sum_{n} x(i,j)w(i-m,j-n) \tag{1.3}$$

En la Figura 1.3 se puede ver un ejemplo de cómo una imagen es procesada por la etapa de convolución en una CNN.

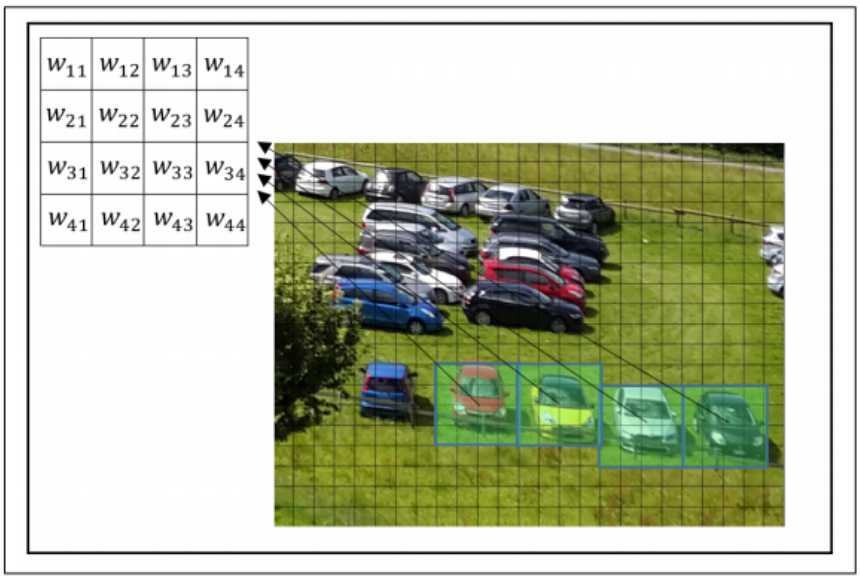

Figura 1.3: Matriz de pesos conocida como kernel o filtro aplicada a una imagen.

1.2.2.1. 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.

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 1.4 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).

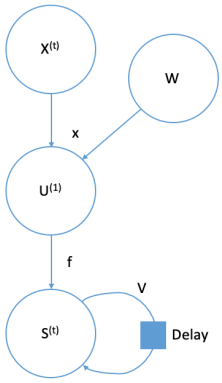

Figura 1.4: Ejemplo de una RNN que modela la ecuación $s(t) = f(s^{(t-1)}, \theta)$, donde θ representa los parámetros W y V, f la función no lineal aplicada a $Wx^{(t)}$ y V la matriz de pesos aplicada a $s^{(t-1)}$.

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.

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 1.5 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.

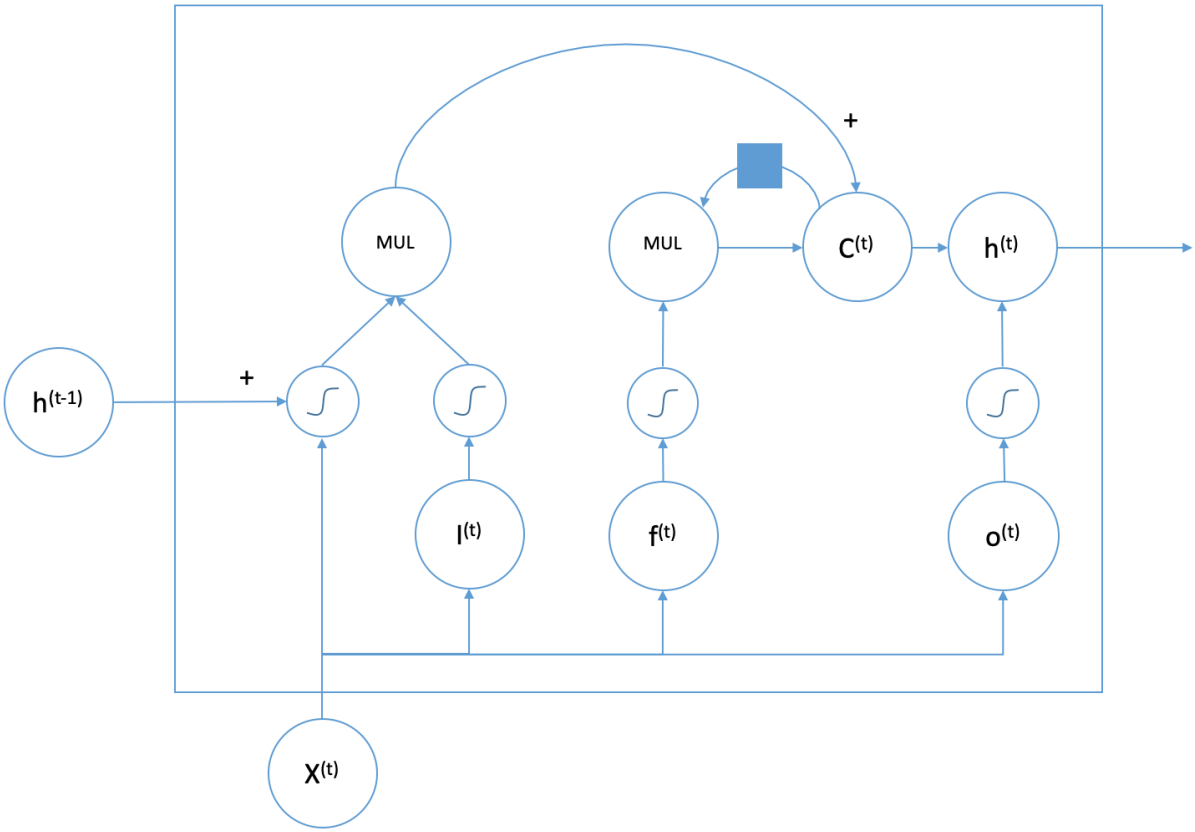

Figura 1.5: Una compuerta LSTM típica.

1.3.3. 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 1.6 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.

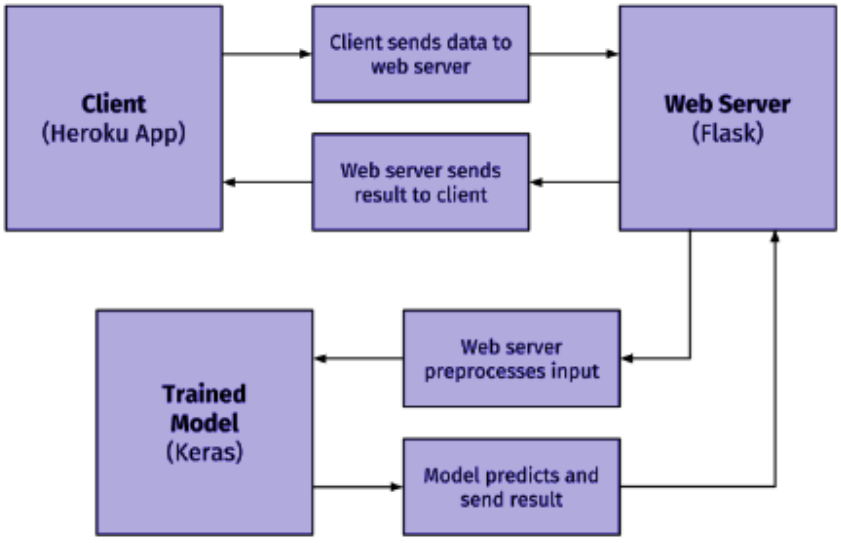

Figura 1.6: Ejemplo de sistema con módulo de Machine Learning

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:

$$G = (V, T, P, S) \tag{1.4}$$

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 [8].

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 [9]. 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 1.7, se puede ver un ejemplo del proceso de reconocimiento propuesto por este método.

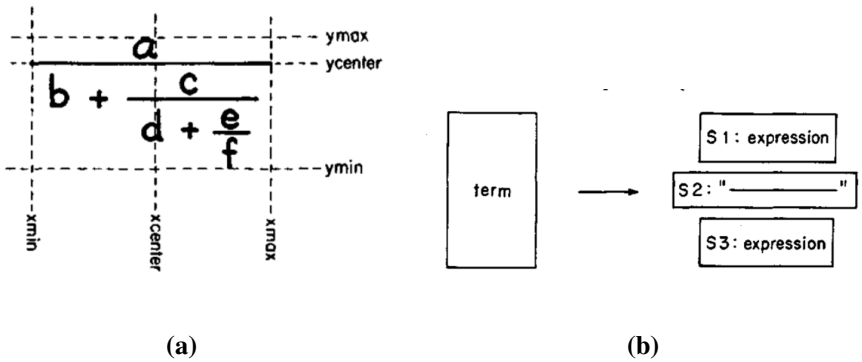

Figura 1.7: a) La primera etapa del método de reconocimiento, se anotan las coordenadas de de cada caracter. **b)** Se realiza un análisis sintáctico con las gramáticas libres de contexto definidas.

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 [10] 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 1.8 muestra un ejemplo de como funciona este método.

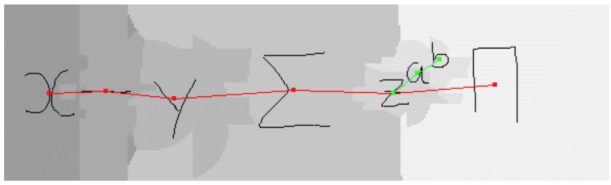

Figura 1.8: Utilización de un árbol de recubrimiento mínimo para el reconocimiento de expresiones matemáticas.

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 [5].

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 esta utilizando actualmente para solucionar este problema se muestra en la Figura 1.9 [5][6][7].

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 [11].

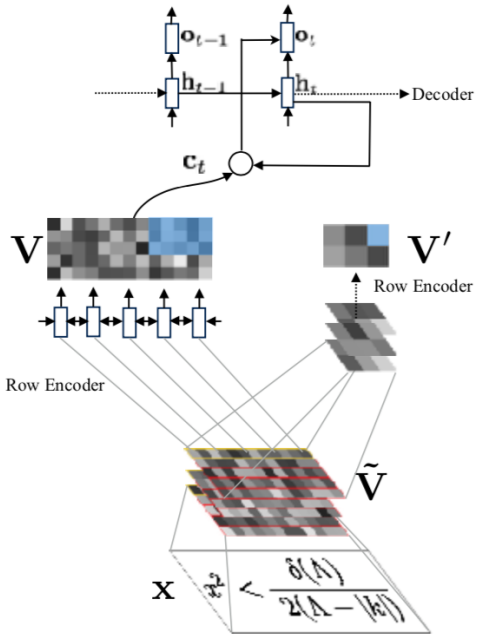

Figura 1.9: Arquitectura de image captioning para reconocer expresiones matemáticas en imagenes.

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

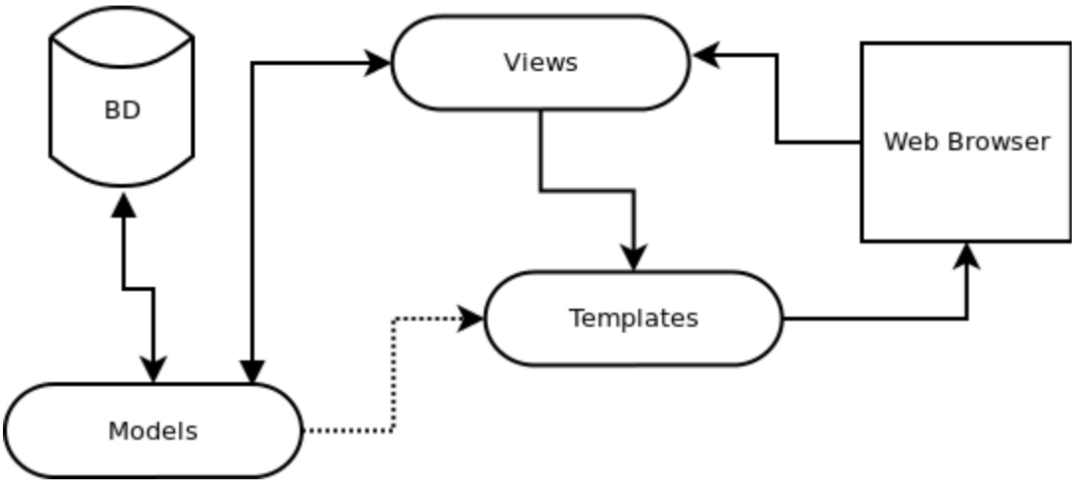

Figura 1.10: Representación del patrón de diseño Modelo-Vista-Template.

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 1.10, se puede ver una representación del patrón MTV. En el patrón de diseño MTV [12]:

- **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 [13].

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 [14].
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 1.11 ilustra el uso de REST para servicios Web.

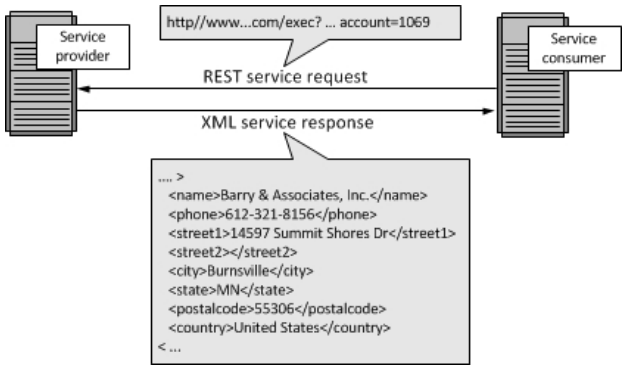

Figura 1.11: Consumidor y proveedor de servicios comunicandose mediante solicitudes y respuestas REST.

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 [15].

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.

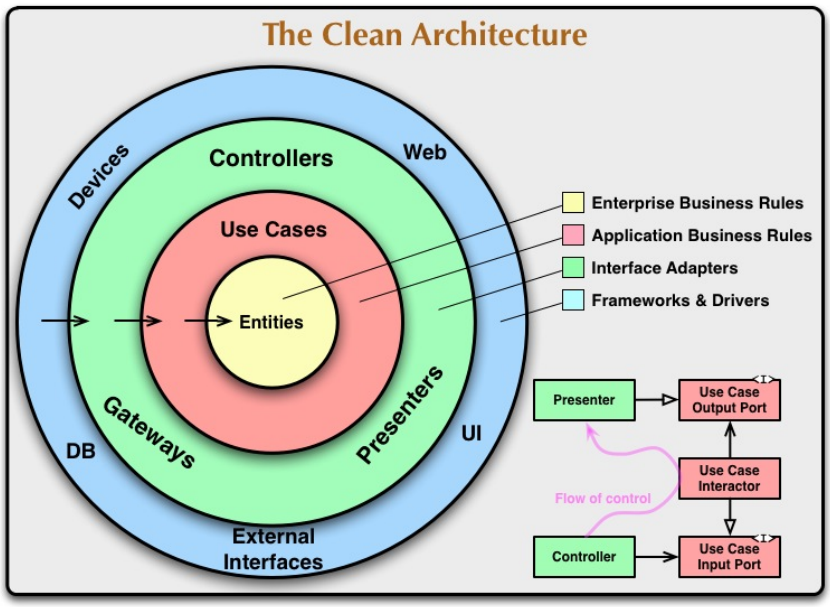

Figura 1.12: Arquitectura Clean [1]

La arquitectura Clean como se muestra en la figura 1.12 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. [16]

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. [17]

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.[17]

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.[17]

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.[17]

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. [17]

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]. III, 15

[2] A. Seitz, S, "Computer vision." https://courses.cs.washington.edu/courses/cse576/, 2000. [CSE576: Computer Vision]. 1, 2

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 3, 4

[4] J. Brownlee, "Encoder-Decoder Long Short-Term Memory Networks." https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/.

[5] G. Genthial and R. Sauvestre, "Image to latex," *Stanford University*. 11

[6] 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. 11

[7] 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. 11

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

[9] R. H. Anderson, "Syntax directed recognition of hand-printed two-dimensional mathematics," *Harvard University*. 10

[10] 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*. 11

[11] S. Luján-Mora, "Programación en internet: clientes web," 2001-10-08. 11

[12] 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. 13

[13] 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. 13

[14] 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. 14

[15] "Identificación y autentificación." https://www.ibm.com/support/knowledgecenter/es/SSFKSJ_7.5.0/com.ibm.mq.sec.doc/q009740_.htm. 14

[16] C. Salazar, "Los principios de SOLID de la programación orientada a objetos." https://www.codesolt.com/tutoriales/fundamentos/solid/, 2018. [Consultado: 2019-09-21]. 16

[17] 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]. 17