Consumo de servicios web de transferencia de estado representacional con el robot humanoide Fibonacho y sus aplicaciones

Reporte Técnico 201804

Sebastián Bejos[1], Ivan Feliciano[1],

Laboratorio de Algoritmos para la Robótica, Centro de Desarrollo Tecnológico,
Facultad de Estudios Superiores Acatlán, UNAM

Resumen En este documento se describe la integración de servicios web con el robot Fibonacho para añadir algunas funcionalidades que permiten que el robot realice tareas como el procesamiento de imágenes para reconocer personas, navegar o traducir texto extraído de una fotografía; además de interactuar con un usuario mediante un discurso oral. A través de las API REST con las que cuentan los servicios web y por una desarrollada en el laboratorio, el robot puede ejecutar modelos de aprendizaje profundo de manera remota trabajando con peticiones y respuestas del protocolo HTTP.

Keywords: NAO, Servicios Web, Aprendizaje Profundo, Cómputo en la nube, REST.

1. Introducción

Actualmente existen diferentes plataformas que ofrecen servicios web para solucionar problemas de visión computacional [1] o procesamiento de lenguaje natural [2] usando modelos de aprendizaje profundo. Éstas también incluyen servicios para entrenar y lanzar modelos creados por los mismos usuarios [3]. Todos estos recursos cuentan con interfaces de programación de aplicaciones con una arquitectura de transferencia de estado representacional o REST (por sus siglas en inglés, representational state transfer) la cual permite que cualesquiera dispositivos que manejen el protocolo HTTP puedan acceder a éstos. El aprendizaje profundo es un tipo de aprendizaje automático donde las computadoras construyen conceptos complejos a partir de conceptos más simples [4].

Fibonacho es un robot humanoide programable y autónomo que a pesar de contar con un poder de procesamiento suficiente para realizar tareas como jugar un partido de fútbol con otros robots [5], las aplicaciones modernas exigen que los dispositivos se adapten constantemente a nuevas necesidades de los usuarios, como el acceso a los datos del robot a cualquier hora y desde cualquier lugar,

el uso del aprendizaje profundo para resolver problemas como la clasificación de imágenes en múltiples categorías, encontrar la posición de objetos en una imagen, reconocer personas y el procesar el lenguaje natural de un discurso [6].

En este trabajo se integraron servicios web con el robot Fibonacho. Los recursos son brindados por las plataformas Google Cloud, Wit.ai, Kairos y por el mismo Laboratorio de Algoritmos para la Robótica. Estos servicios permiten al robot realizar tareas como el procesamiento de lenguaje natural de un discurso oral, el procesamiento de imágenes para reconocimiento de rostros y objetos y la clasificación de imágenes en escenarios.

2. Antecedentes

2.1. Robot NAO

NAO es un robot humanoide autónomo y programable desarrollado por la empresa de Aldebaran Robotics.

El robot NAO mide 57.3 cm de altura, 27.3 cm de ancho, y pesa menos de 4.3 kg. El cuerpo está construido de un material de plástico y tiene una batería de ion de litio que lo abastece para un uso normal de aproximadamente 90 minutos o 60 en un modo activo.

Las versiones *V5* y *V4* tiene una CPU Atom de 1.6 GHz, 1 GB de RAM, una memoria flash de 2 GB y una micro SDHC de 8GB.

Se comunica remotamente con otros dispositivos mediante WiFi o por medio de un cable Ethernet. También cuenta con un puerto USB cuyo principal uso es para añadir un dispositivo como un sensor 3D o un Arduino.

Para la parte multimedia, el robot está equipado con un sistema de transmisión estéreo compuesto por dos bocinas en las orejas. Dos micrófonos en la cabeza con un paso de banda eléctrico entre 300Hz y 8kHz. Dos cámaras idénticas están localizadas en la parte anterior de la cabeza. Éstas proveen imágenes con una resolución de hasta 1280px $\times$ 960px y 30 cuadros por segundo.

Cuenta con diodos emisores de luz distribuidos entre la cabeza, orejas, ojos, pecho y pies. Cada uno de estos últimos cuenta con resistencias sensibles a la fuerza, que son sensores encargados de medir la resistencia al cambio de acuerdo a una presión aplicada. Trabajan en un rango de 0 N a 25 N.

En el torso está localizada una unidad de medición inercial, compuesta por un girómetro y un acelerómetro, la cual permite una estimación de la velocidad y postura del torso.

El robot está equipado con dos sensores ultrasónicos, que sirven para estimar la distancia a obstáculos. Dependiendo de la versión del robot, el rango de detección varía de 0.20 cm a 0.80 cm en la última versión y 0.25 cm a 2.55 cm en versiones

anteriores. Cuando la distancia es menor al límite inferior de cada versión, el robot únicamente sabe que hay un obstáculo presente. Tiene un cono efectivo de 60 grados.

Entre los sensores restantes están los de posición de las articulaciones, los de contacto y los táctiles. Los dos últimos están en la cabeza, manos, pecho y pies.

Tiene en total 25 articulaciones repartidas entre la cabeza, brazos, piernas y pelvis. Todas las articulaciones cuentan con controladores de posición. Dada una articulación que enlaza dos partes del cuerpo del robot, la parte del cuerpo que está más cerca del tronco se considera fija y la parte que está más lejos es la que rota alrededor de los ejes de la articulación. Para realizar una rotación de las partes del cuerpo, definimos un sistema de referencia en cada articulación. En el sistema de referencia se tienen tres ángulos de rotación: *roll* (dirección), *pitch* (elevación) y *roll* (alabeo). Las rotaciones *roll*, son alrededor del eje X, las *pitch* sobre Y y *yaw* con respecto a Z. La figura 1 muestra el sistema de referencia.

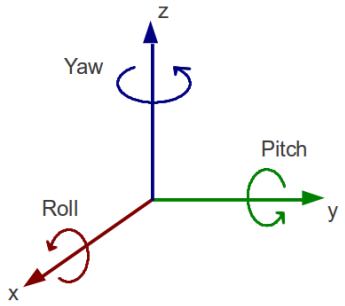

Figura 1: Sistema de referencia de los ángulos de rotación

En la figura 2 se muestran algunos de los componentes del robot NAO.

2.2. NAOqi

NAOqi es el nombre del software principal que corre sobre el robot y lo controla.El marco de trabajo de NAOqi es el marco de trabajo de programación usado para programar robots de Aldebaran. Brinda soluciones a necesidades básicas de la robótica como el paralelismo, el manejo de recursos, la sincronización y los eventos.

NAOqi cuenta con bastantes API divididas en grupos de acuerdo a las funcionalidades que ofrecen.

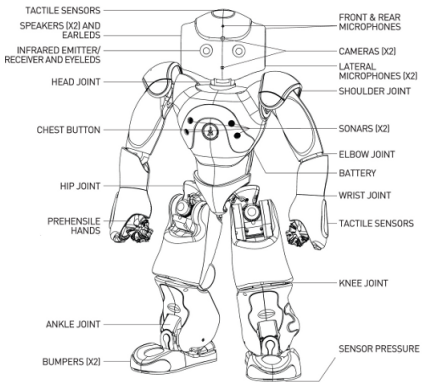

Figura 2: Componentes del robot NAO

NAOqi contiene un grupo de módulos enfocados a los movimientos del robot, ya sea para navegar de manera segura, cambiar entre posturas predefinidas, realizar movimientos de forma autónoma y hasta generar movimientos personalizados.

ALMotion. Es la principal herramienta para permitir que el robot se mueva.

Contiene cuatro principales grupos de métodos para controlar:

- la rigidez de articulaciones, básicamente si un motor está prendido o apagado.
- la posición de articulaciones, para la planeación de trayectorias (interpolación) y cambios en los valores de los motores como respuesta a datos en sensores (control reactivo).
- el caminado, control de distancia y velocidad, posición en un ambiente, etc.
- efector del robot en el espacio cartesiano, determinar el movimiento de una cadena de articulaciones para lograr que un actuador se ubique en una posición concreta (cinemática inversa).

El eje **X** es positivo con respecto al frente del robot, el eje **Y** de derecha a izquierda y el **Z** es vertical. La figura 3 muestra la definición de los ejes con respecto al robot. El módulo de `ALMotion` usa el Sistema Internacional de Unidades (metros, segundos, radianes, etc).

NAOqi cuenta con componentes de software para el audio; para manejar la salida o entrada a través de sus bocinas y micrófonos, para la detección y localización de sonidos, y para el manejo del lenguaje.

ALTextToSpeech. Este módulo permite al robot hablar. Envía órdenes a un componente que convierte texto a un discurso hablado, y autoriza la personalización de la voz. El resultado de la síntesis es enviado a las bocinas del robot.

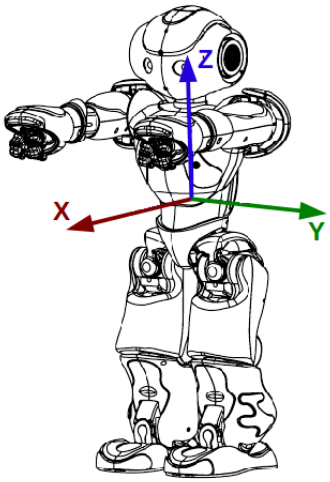

Figura 3: Sistema de ejes sobre los que el robot ejecuta movimientos

ALAnimatedSpeech. El módulo `ALAnimatedSpeech` brinda la posibilidad de hacer que el robot hable de una manera expresiva. El funcionamiento de este módulo es como sigue:

1. `ALAnimatedSpeech` recibe texto que puede ser *anotado* con *instrucciones*.
2. Divide el texto en subcadenas de menor tamaño a la original.
3. Analiza el texto y anota las cosas que reconoce para realizar movimientos de acuerdo al contexto.
4. Cualquier parte del texto que no esté anotado con animaciones se llena con *modos de lenguaje corporal*.
5. El módulo prepara al robot para ejecutar cada instrucción para que estas sean llamadas tan pronto como se necesiten. Esto permite que el discurso y las instrucciones estén sincronizados.
6. `ALAnimatedSpeech` hace que el robot diga el texto y se mantenga al tanto del discurso para lanzar las instrucciones en el momento correcto.

ALAudioRecorder. `ALAudioRecorder` provee servicios de grabación en archivos «WAV» u «OGG» a partir de las señales recibidas de los micrófonos del robot.

Este módulo depende la biblioteca de Linux SDNFile para codificar de manera eficiente entradas de audio en tiempo real.

Las capacidades de grabación están limitadas a los siguientes formatos:

- cuatro canales 48000 Hz para OGG y WAV

- un canal (anterior, posterior, izquierda o derecha) para OGG y WAV.

Los módulos de visión provistos por NAOqi se encargan del manejo de video e imágenes, detección de objectos predefinidos en video, y cuenta con herramientas para crear una memoria visual donde aprenda y reconozca ciertos patrones, y herramientas para navegación, usar una imagen como brújula, entre otras.

ALVideoDevice. Este módulo es responsable de proveer, de una manera eficiente, imágenes de la cámara del robot a todos los módulos que las procesan, como `ALFaceDetection` o `ALVisionRecognition`.

Para empezar a utilizar este módulo es necesario seguir los siguientes pasos:

1. Hacer que tu módulo de visión se suscriba al proxy de `ALVideoDevice`, llamando el método `subscribeCamera()` y pasando como parámetros la resolución, espacio de color y tasa de fotogramas.
2. En el bucle del proceso principal, obtener una imagen llamando a los métodos `getImageLocal()` o `getImageRemote()` (dependiendo si el módulo es local o remoto).
3. Liberar la imagen llamando `releaseImage()`.
4. Cuando se detiene el módulo, se llama `unsubscribe()` después de salir de bucle principal.

3. Servicios Web REST

3.1. Transferencia de Estado Representacional (REST)

La Transferencia de Estado Representacional es un estilo de arquitectura de software. Este estilo es una abstracción de elementos arquitectónicos dentro de un sistema de hipermedia distribuido como es la Web. REST ignora los detalles de la implementación de componentes y sintaxis de protocolos de manera que pueda enfocarse en los papeles de los componentes, las restricciones sobre su interacción con otros componentes, y la interpretación de elementos de datos significativos. Abarca las limitaciones fundamentales sobre los componentes, conectores y datos que definen las bases de la arquitectura web y, por lo tanto, la esencia de su comportamiento como una aplicación basada en red. REST no es un estándar, sin embargo sí un conjunto de restricciones. No está atado al protocolo HTTP, pero a menudo se asocia con éste.

API REST

Un *servicio web* es un sistema de software diseñado para admitir la interacción interoperable de una máquina a otra máquina a través de una red. Programas

cliente usan *interfaces de programación de aplicaciones* (API por sus siglas en inglés) para comunicarse con servicios web. Una API expone un conjunto de datos y funciones para facilitar interacciones entre programas de computadora y permitiendo que intercambien información.

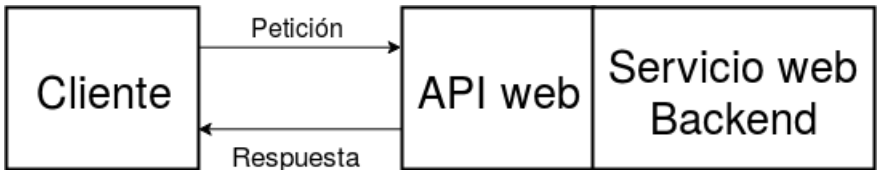

Figura 4: Una API web es el frente de un servicio web, escuchando y respondiendo las peticiones de los cliente directamente.

El estilo arquitectónico REST se aplica comúnmente al diseño de API para servicios web modernos. Una API web que sigue el estilo REST es una API REST. Tener una API REST hace a un servicio web RESTful. Una API REST está formada de recursos entrelazados.

REST es una arquitectura basada en recursos. Se accede a un recurso a través de una interfaz común basada en los métodos estándar de HTTP. REST solicita a los desarrolladores usar métodos HTTP explícitamente y de una forma que sea consistente con la definición del protocolo. Cada recurso se identifica con un URL. Todos los recursos deben soportar las operaciones HTTP más comunes, además REST permite que ese recurso tenga diferentes representaciones, por ejemplo, texto, XML, JSON, etc. El cliente REST puede solicitar una representación en específico por medio del protocolo HTTP. El cuadro 1 describe los elementos usados en REST.

4. Servidor REST

Se creó un servidor el cual es cliente de los servicios web de terceros y a la vez ofrece los mantenidos por el LAR. El resultado de la unión cliente-servidor se entrega por medio de una API REST. Todos los servicios brindados tienen en común que están basados en modelos de aprendizaje automático, específicamente de aprendizaje profundo. Estos dan solución a problemas de visión computacional como la detección de rostros o el reconocimiento óptico de caracteres, el reconocimiento de voz, el procesamiento de lenguaje natural y la traducción automática neuronal. A continuación se enlistan los servicios de terceros que el servidor consume.

Cuadro 1: Elementos de REST

Elemento	Descripción
Recurso	Objetivo conceptual de una referencia de hipertexto. Por ejemplo: podcast.
Identificador de recurso	Un URL que identifica un recurso en específico. Por ejemplo: `http://convoynetwork.com/podcast/123`
Metadatos del recurso	Información que describe al recurso. Por ejemplo: autor, etiqueta, etc.
Representación	El contenido del recurso. Por ejemplo: un JSON, un HTML o una imagen JPEG.
Metadatos de la representación	Información que describe como procesar la representación. Por ejemplo: tipo de medio, fecha, etc.
Datos de control	Información que describe cómo optimizar el procesamiento de respuesta. Por ejemplo: if-modified-since, cache-control-expiry.

- Google Cloud Vision, permite comprender el contenido de una imagen. Son dos las funcionalidades ocupadas; el etiquetado de imágenes en miles de categorías y el reconocimiento óptico de caracteres (OCR).
- Google Cloud Translation, se encarga de traducir una cadena arbitraria en cualquier idioma admitido.
- Kairos, es una API de reconocimiento facial. Esta cuenta con varios métodos, de los cuales se utilizaron *enroll* y *recognize*. El primero para añadir a un nuevo rostro a una base de datos junto con un identificador, el segundo para encontrar a un usuario cuya cara ha sido almacenada.
- Wit.ai, un servicio para procesamiento de lenguaje natural capaz de convertir oraciones en información estructurada.

Los servicios mantenidos por el LAR son dos modelos de TensorFlow. El primero permite la detección de objetos en imágenes y es parte de la API de detección de objetos de TensorFlow. El segundo servicio es un clasificador de imágenes en cuatro categorías.

El servidor mantiene la ejecución de un contenedor de Docker en el que se encuentra corriendo una aplicación web desarrollada en Python. Dicha aplicación es la API REST que se comunica con el robot y dispositivos que soliciten los recursos. La figura 5 muestra de manera general la estructura del servidor.

5. API REST

Esta interfaz es el producto de la integración de modelos de aprendizaje automático enfocados a casos de uso relacionados con la robótica.

La API REST permite integrar dentro de una aplicación un conjunto de herramientas para el análisis de imágenes. Todas éstas basadas en modelos de

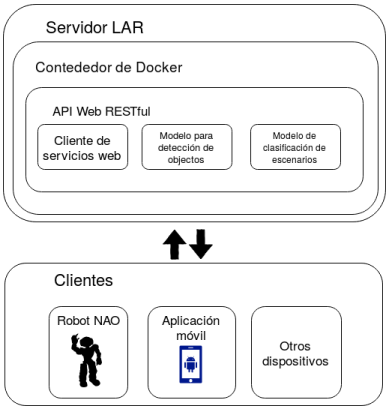

Figura 5: Diagrama del servidor

aprendizaje automático. El URL base al que todos los recursos son relativos es `http://132.248.180.17/`.

La API tiene tres recursos:

- `/register`: para que un nuevo usuario se registre y pueda ocupar los otros recursos. Envía una dirección de correo y una contraseña.
- `/refreshtoken`: para que el usuario pueda obtener un nuevo token de acceso al enviar los datos con los que se registró.
- `/vision`: el recurso más importante. Se le solicita que haga procesamiento de imágenes.

5.1. Recurso /vision

Este recurso es el encargado de la detección de características en una imagen. Estas características son: la **detección de objetos**, el **reconocimiento de rostros**, de personas previamente guardadas o de nuevos sujetos para su almacenamiento, la **clasificación en cuatro escenarios**, lugares dentro del laboratorio de algoritmos para la robótica, la **detección de etiquetas o categorías** y la **traducción de texto encontrado en una imagen**.

En este recurso la única forma de persistencia de datos es al guardar el rostro de una nueva persona para su posterior reconocimiento.

POST /vision

El único método definido para el recurso `/vision` es POST. Se envía a la API

una imagen ya sea codificada en base 64 o mediante su URL y las características que se desean obtener. Las características disponibles son las siguientes:

- `FACE_ENROLL`: Detecta un rostro en la imagen y con un identificador enviado en el cuerpo de la petición se almacena en una galería de Kairos. Se emplea `/enroll` de Kairos.
- `FACE_RECOGNITION`: Encuentra rostros dentro de una imagen y los relaciona con los rostros similares previamente guardados en una galería de Kairos. Usa `/recognize` de la API de Kairos.
- `OBJECT_DETECTION`: Busca objetos en una la imagen, usa la API de detecctión de objetos de TensorFlow. Regresa las coordenadas de un cuadro delimitador para cada objeto detectado.
- `LABELS_DETECTION`: Clasifica una imagen en distintas categorías. Se vale de la API de Google Cloud Vision
- `OCR_TRANSLATION`: Lleva a cabo el reconocimiento de texto en una imagen, para posteriormente traducirlo. Utiliza la API de Google Cloud Vision para el reconocimiento de caracteres y la de Google Cloud Translation para la segunda parte del proceso.
- `CLASSIFY_INDOOR_SCENES`: Clasifica una imagen en cuatro categorías. Cada categoría es un lugar dentro del área del LAR.

Mensaje de solicitud

Headers. En los encabezados de la petición debe de ir el tipo de contenido que se envía y un token de acceso único para cada usuario registrado. Esto último simplemente es para evitar que cualquiera pueda hacer peticiones a la API.

```
Content-Type: application/json
Authorization: ACCESS_TOKEN
```

Cuerpo. En el cuerpo del mensaje se envía un JSON con la siguiente estructura. En la tabla 2 se describen los atributos del JSON.

```
{
  "imageContent": "Hello, world!",
  "imageSource": "Hello, world!",
  "features": [
    {
      "type": "Hello, world!",
      "subjectID": "Hello, world!"
    }
  ]
}
```

Cuadro 2: Descripción de los elementos del JSON del cuerpo de la solicitud.

Propiedades	
`imageContent`	
Tipo de dato	string
Descripción	Imagen codificada en base 64.
`imageSource`	
Tipo de dato	string
Descripción	URL público de la imagen.
`features`	
Tipo de dato	array
Descripción	Una arreglo de las características que se desean detectar en la imagen. Se debe solicitar al menos una de las seis disponibles. Por ejemplo FACE_ENROLL, FACE_RECOGNITION, CLASSIFY_INDOOR_SCENES, etc.
Campos obligatorios	
imageContent imageSource features	

Mensaje de respuesta

Cuerpo. En el cuerpo del mensaje de respuesta, si se obtiene un código 200, va un JSON con las características encontradas. En éste también se incluyen mensajes de error que no pertenecen al estándar del protocolo HTTP. En la tabla 3 se describen los atributos del JSON.

```
{
  "features": {
    "faceRecognition": [
      {
        "topLeftX": 1,
        "topLeftY": 1,
        "width": 1,
        "height": 1,
        "subjectId": "Hello, world!",
        "confidence": 1
      }
    ],
    "objectDetection": [
      {
        "category": "Hello, world!",
        "confidence": 1,
        "topLeftX": 1,
        "topLeftY": 1,
        "width": 1,
        "height": 1
```

```
    }
  ],
  "labelsDetection": [
    {
      "name": "Hello, world!",
      "confidence": 1
    }
  ],
  "ocrTranslation": {
    "sourceText": "Hello, world!",
    "targetText": "Hello, world!",
    "sourceLanguage": "Hello, world!"
  },
  "faceEnroll": {
    "topLeftX": 1,
    "topLeftY": 1,
    "width": 1,
    "height": 1,
    "confidence": 1,
    "gender": "Hello, world!"
  }
  "indoorScenesClassifier" : {
    "indoorScene": "Hello, world!"
  }
}}
```


Cuadro 3: Atributos que componen el JSON del cuerpo de la respuesta.

`features`	
Tipo de dato	objeto
Propiedades	
`faceRecognition`	
Tipo de dato	array
Descripción	Un arreglo de objetos que contienen a los rostros reconocidos.
`objectDetection`	
Tipo de dato	array
Descripción	Contiene un arreglo con todos los objetos detectados.
`labelsDetection`	
Tipo de dato	array
Descripción	Contiene un arreglo con las etiquetas de la imagen.
`ocrTranslation`	
Tipo de dato	objeto
Propiedades	
`sourceText`	
Tipo de dato	string

Descripción	El texto en formato UTF-8.
`targetText`	
Tipo de dato	string
Descripción	El texto traducido.
`sourceLanguage`	
Tipo de dato	string
Descripción	El idioma original.
Descripción	Un objecto con el texto encontrado en la imagen.
`faceEnroll`	
Tipo de dato	objeto
Propiedades	
`topLeftX`	
Tipo de dato	number
Descripción	Coordenada sobre el eje x.
`topLeftY`	
Tipo de dato	number
Descripción	Coordenada
`width`	
Tipo de dato	number
Descripción	Ancho del recuadro que delimita la imagen.
`height`	
Tipo de dato	number
Descripción	Altura del recuadro que delimita la imagen.
`confidence`	
Tipo de dato	number
Descripción	Valor de 0-1 que representa una probabilidad.
`gender`	
Tipo de dato	string
Descripción	Sexo de la persona con ese rostro (M o F)
Descripción	Características del rostro detectado.
`indoorScenesClassifier`	
Tipo de dato	objeto
Propiedades	
`indoorScene`	
Tipo de dato	string
Descripción	La escena detectada, puede ser cualquiera de las cuatro posibles (exit, soccer_court, desks, office)
Descripción	Escenario reconocido.
Descripción	Lista con las respuestas de acuerdo a las características que se solicitaron.

Ejemplo de una petición a la API con cURL

cURL es una herramienta en la línea de comandos para transferir datos usando diferentes protocolos. Por facilidad, se muestra el uso de la API a través de esta herramienta. Para solicitar el recurso `vision`, para procesamiento de imágenes, se necesita un token de acceso. El token se obtiene creando un usuario haciendo una petición al recurso `register`. El fragmento de código para solicitar este servicio con `cURL` es el siguiente:

```
curl -X POST \
http://132.248.180.17/register \
-H 'content-type: application/json'\
-d '{
    "username" : "mock_user@gmailcom",
    "password" : "p45sw0rd"
    }'
```

La respuesta de la API es un JSON como el que sigue:

```
{
    "message": "User created successfully.",
    "token": "T6v23Tnw95S8BX2yNR8Os8RB6L4HAnvhTTrfjmxB5UyMaef"
}
```

Con el token se pueden hacer las peticiones que se deseen al recurso `visión`. Supongamos que queremos analizar una imagen para encontrar rostros, objetos, traducir texto y clasificar de escenarios. Se hace como sigue:

```
curl -X POST \
http://132.248.180.17/vision \
-H 'authorization: T6v23Tnw95S8BX2yNR8Os8RB6L4HAnvhTTrfjmxB5UyMaef' \
-H 'content-type: application/json' \
-d '{
"imageSource" : "https://www.robotshop.com/blog/en/files/NAO-Hanover.jpg
↪",
   "features" : [
      {
        "type": "FACE_RECOGNITION"
      },
      {
        "type" : "OBJECT_DETECTION"
      },
      {
        "type" : "OCR_TRANSLATION"
      },
      {
        "type" : "CLASSIFY_INDOOR_SCENES"
      }]}'
```

La API de CloudNAO envía el el cuerpo de su respuesta el siguiente JSON:


```
{
    "features": {
        "objectDetection": [
            {
                "confidence": 0.9916030764579773,
                "category": "person",
                "topLeftY": 42,
                "height": 158,
                "topLeftX": 430,
                "width": 50
            },
            {
                "confidence": 0.9870478510856628,
                "category": "person",
                "topLeftY": 79,
                "height": 115,
                "topLeftX": 253,
                "width": 49
            },
            {
                "confidence": 0.9252263307571411,
                "category": "person",
                "topLeftY": 11,
                "height": 148,
                "topLeftX": 2,
                "width": 53
            },
            {
                "confidence": 0.9083054661750793,
                "category": "sports ball",
                "topLeftY": 248,
                "height": 50,
                "topLeftX": 219,
                "width": 49
            },
        ],
        "indoorScenesClassify": {
            "indoor_scene": "soccer_court"
        }
    },
    "errors": {
        "faceRecognition": {
            "message": "invalid url was sent"
        },"ocrTranslation": {
            "message": "Text not found"
        }
    }}
```

6. Aplicaciones

6.1. Reconocimiento de rostros

Un aplicación fue el uso de los servicios de detección de rostros y reconocimiento de sujetos de la API de Kairos en una aplicación móvil. Ésta permite que los usuarios ejecuten un potente recurso en la nube para reconocimiento de caras enviando sólo una imagen capturada por el robot.

El robot Fibonacho, a través de la API de NAOqi `ALFaceDetection`, cuenta con métodos que permiten la detección de rostros y el reconocimiento de caras previamente almacenadas. La detección de rostros se ejecuta lo suficientemente rápido como para poner en funcionamiento un rastreador o seguidor de rostros. El reconocimiento de personas lleva casi el mismo tiempo de procesamiento y funciona a pesar de algunas variaciones en las caras, por ejemplo, si el sujeto lleva una gorra o lentes.

Aunque la API de NAOqi funciona muy bien, tiene algunas desventajas como: guardar en el disco del robot la foto de un rostro para su futuro reconocimiento, el proceso de aprendizaje de nuevas personas no es simple para usuarios inexpertos y no aprovecha más información ofrecida en la imagen.

Hay varias plataformas para detección de rostros, pero se eligió Kairos por todas las características que ofrece gratis a los desarrolladores. A través de la API de Kairos tenemos acceso a funcionalidades que son parte de la solución a los problemas que no abarca `ALFaceDection`. Se guardan en la nube las imágenes de nuevos sujetos detectados y añadir una nueva cara se hace con una simple petición al recurso `enroll`. Kairos también permite analizar la imagen para detectar el género, edad o raza de las personas.

En la API REST se utilizan los servicios de detección de rostros y reconocimiento de sujetos de la API de Kairos. Dentro de la aplicación móvil, la API REST es el intermediario que se comunica con la API de Kairos, esto es por facilidad y por no tener que manejar muchas API y SDK por cada servicio que se desee usar.

Cuando el usuario selecciona en el menú de navegación de la aplicación la tarea de **Reconocimiento de personas** se muestra una pantalla con tres botones, uno para obtener una fotografía del robot, otro para detectar rostros de nuevas personas, y el tercero para reconocer sujetos ya almacenados. Si el usuario desea añadir a una nueva persona, se abre un formulario emergente con un campo para agregar el nombre de la persona y si no hay errores y la detección se realizó correctamente, la cara de la persona queda almacenada y se pueden realizar futuros reconocimientos. Si se desea realizar el reconocimiento de personas cuyo rostro fue previamente guardado, se presiona el botón con la etiqueta *Reconoce personas* y si todo sale bien, se muestra una lista de las personas en la fotografía. Al dar clic sobre el nombre de la persona en la lista, el robot ejecuta un movimiento de saludo diciendo una frase simple con el nombre de la persona.

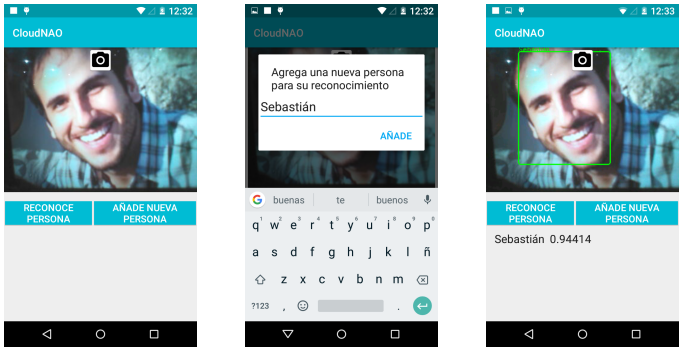

Figura 6: Ejemplo de cómo se registrar un nuevo sujeto para su posterior reconocimiento.

La velocidad de la ejecución de la tarea depende mucho de la conexión a internet. El ejemplo en el que se ocupa el reconocimiento de rostros es bastante sencillo, el robot sólo hace un gesto de saludo. No obstante podemos añadir más funcionalidades donde aprovechemos el análisis completo que ofrece Kairos, la detección del género, la edad y la raza.

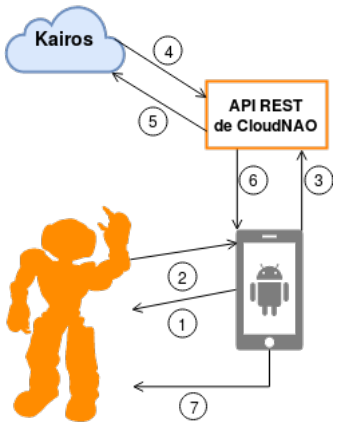

Figura 7: El flujo de el reconocimiento de personas. 1) El usuario presiona el botón para capturar una imagen desde el robot. 2) El robot envía la imagen a la aplicación. 3) El usuario presiona el botón Reconoce personas y envía la imagen a la API REST. 4) La API solicita el recurso `recognize` de Kairos. 5) Kairos envía un JSON como respuesta. 6) La API envía un JSON a la aplicación móvil. 7) La aplicación ejecuta remotamente el módulo de NAOqi para realizar el discurso animado.

6.2. Reconocimiento óptico de caracteres y traducción de texto

Una aplicación interesante es la detección de texto en una imagen y su traducción. En la API REST se combinan dos servicios para añadir al robot Fibonacho la funcionalidad de extracción y traducción de texto en imágenes.

El reconocimiento óptico de caracteres (OCR por sus siglas en inglés) permite detectar y extraer texto de imágenes para luego almacenarlo en un formato que una máquina pueda entender.

A diferencia del caso de estudio anterior, el robot Fibonacho no cuenta con una API para detección de texto en imágenes, ni con una herramienta para traducir sentencias. Por las características del robot, la interacción con humanos es amigable, por lo que una funcionalidad como la de asistir a los usuarios en la traducción de texto a través de imágenes tiene una amplia aplicación.

La API de Vision de Google Cloud ofrece el servicio de OCR que además incluye la detección del idioma en que se encuentra escrito el texto encontrado. Google Cloud cuenta también con una API para traducción de texto. En la API REST se combinan estos dos servicios para añadir al robot Fibonacho la funcionalidad de extracción y traducción de texto en imágenes.

El usuario simplemente adquiere una fotografía capturada por la cámara del robot y se hace la petición a la API REST que utiliza los servicios de Google Cloud Vision y Translation para obtener el texto y hacer la traducción, respectivamente. Si existe texto en la imagen y fue correctamente procesado por la API de Vision, éste se traduce, el resultado se muestra en la aplicación y el robot repite oralmente el texto traducido.

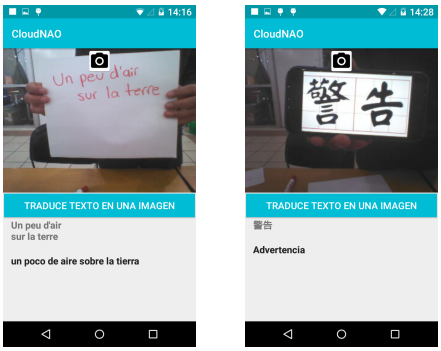

Figura 8: Ejemplo de la traducción de dos sentencias, la primera en francés y la segunda en chino.

Gracias a los servicios utilizados, la traducción del texto en una imagen se resuelve fácilmente. La aplicación ofrece una interfaz fácil para cualquier usuario.

El tiempo de ejecución de la tarea es de unos cuantos segundos (entre 5 aproximadamente), aunque es una variable dependiente de la velocidad de la conexión a internet.

6.3. Reconocimiento de voz

Este caso de estudio se aplica directamente sobre el robot Fibonacho, conectándolo a la API de Wit.ai para procesamiento de un discurso oral y poder interactuar con él a través de comandos de voz.

Una aplicación interesante de tecnologías que realizan el procesamiento de lenguaje natural es la creación de bots conversacionales o asistentes por voz. El robot Fibonacho es una plataforma con las características para servir como asistente por voz, que a pesar de contar con un módulo de reconocimiento de voz (`ALSpeechRecognition`), éste es muy básico y limitado.

Una aplicación de reconocimiento de voz utilizando únicamente la API de NAOqi es estática. El módulo `ALSpeechRecognition` sólo reconoce palabras predefinidas no la intención del discurso del usuario. Por ejemplo, si deseamos que el robot salude cuando un usuario lo hace, se deben definir todas las formas posibles en las que la persona puede decir "Hola".

Wit.ai provee una API para construir aplicaciones con las que los usuarios se puedan comunicar a través de voz o texto. Las aplicaciones desarrollados con Wit.ai aprenden conforme reciban más información, se vuelven más inteligentes con cada interacción de los usuarios.

Se creó una aplicación sencilla para interactuar con el robot usando el servicio `speech` de Wit.ai. Dentro de la API REST existe un módulo para hacer las peticiones a la API de Wit.ai. Éste también puede usarse directamente sobre el robot, evitando un intermediario. Se solicita el recurso `speech` de la API de Wit.ai enviando un archivo de audio generado por el robot. La API envía como respuesta un JSON con la transcripción del audio en texto, las entidades e intenciones encontradas. El robot es quien se encarga de manejar el flujo de acciones dependiendo de las intenciones.

Las intenciones y entidades de una aplicación se definen en la plataforma web de Wit.ai. Ésta cuenta con una herramienta para crear estos dos elementos a partir de oraciones que el usuario posiblemente enviará en un mensaje. Por ejemplo, cuando el usuario inicie una conversación posiblemente sea con un "Hola robot", por lo que podemos definir una intención con valor `saludo`. Se pueden añadir más sentencias que correspondan a la intención `saludo` como "Buenas tardes" o "Buena día Fibonacho". La entidades permite realizar específicamente cierta acción. Por ejemplo, si le decimos: "Cambia a la posición de descanso", la intención es `cambiar posición` y la entidad es a qué posición debe cambiar, a la de descanso en nuestro ejemplo. A partir de esto podemos definir acciones muy simples para el robot guiadas por las intenciones y entidades. En la tabla

4 se describen las intenciones y entidades que se definieron para esta aplicación y un ejemplo de lo que puede decir el usuario para ejecutar la acción asociada con la intención.

Cuadro 4: El valor de la entidad en la sentencia del usuario se encuentra subrayado.

Intención	Entidades	El usuario puede decir ...
CAMBIAR POSTURA	postura	Ve a la posición de descanso
CAMINAR	dirección	Camina hacia la derecha
INICIO	nombre del usuario	Hola NAO, me llamo Ivan
FIN		Adiós robot
PROCESAMIENTO DE IMÁGENES	tipo de procesamiento	Lee el texto
GUARDA IMÁGENES	número de fotografías	Toma diez fotos

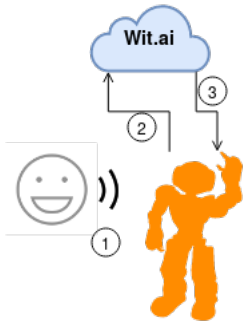

Figura 9: El flujo de la interacción con el robot a través de un discurso oral. 1) El usuario envía un mensaje que el robot graba en un archivo de audio de 3 segundos de duración. 2) El robot envía el archivo binario de audio a la API de Wit.ai. 3) Wit.ai envía un JSON con las entidades encontradas.

Pese a ser una aplicación muy simple se puede volver tan compleja como se desee, el añadir más entidades hace más interesante interactuar con el robot. Podemos agregar otros servicios para que cada vez se parezca más a un asistente por voz, como los calendarios y contactos de Google, la búsqueda de lugares y zonas de interés con servicios como Foursquare o los Mapas de Google, uso de la la gráfica de conocimiento de Google o de Wolfram Alpha, servicios de noticias,

etc. El robot no pone las limitaciones de la aplicación, ya que éste solamente es una interfaz entre los servicios en la nube y los usuarios.

6.4. Clasificación de imágenes para localización

Una aplicación de uno de los servicios que brinda la API REST, y que además fue desarrollado por el LAR, es la clasificación de imágenes en cuatro clases, donde cada clase representa un escenario dentro dentro del área del laboratorio. Este servicio de procesamiento de imágenes permite que el robot conozca en qué espacio se encuentra al navegar en esta zona.

El problema de clasificación de imágenes es la tarea de asignarle a una imagen de entrada una etiqueta a partir de un conjunto de categorías. Este es uno de los principales problemas dentro del campo de la visión computacional, que a pesar de su simplicidad tiene bastantes aplicaciones prácticas. Entre esas aplicaciones, muchas interesan al campo de la robótica móvil, por ejemplo, para la navegación de un robot de manera autónoma, nos gustaría que supiera en que lugar está simplemente tomando una fotografía, así podría saber si ha llegado al lugar de su objetivo, o a partir de la zona donde se ubica planear una trayectoria.

Se implementó en TensorFlow un modelo para clasificar imágenes de algunos lugares sobre los que podría navegar el robot Fibonacho.

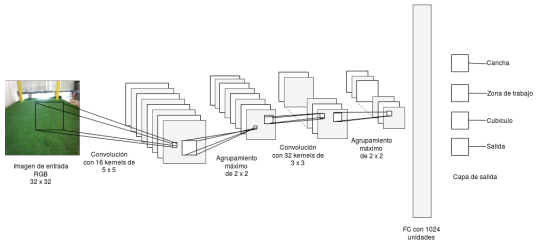

Figura 10: Topología de la red neuronal convolucional utilizada.

El modelo es una red neuronal convolucional, que recibe como entrada un arreglo con los pixeles de una imagen tomada por el robot, y la salida es la categoría a la que pertenece esa imagen. Las clases en las que se desea clasificar las imágenes son lugares alrededor del Laboratorio de Algoritmos para la Robótica, que se ubica en el cubículo 15 del Centro de Desarrollo Tecnológico de la FES Acatlán. Se eligieron las siguientes cuatro zonas:

- El cubículo.
- La salida de emergencia.
- La cancha de entrenamiento de fútbol para el robot NAO.

- Zona de trabajo del Laboratorio.

Los modelos elaborados en TensorFlow contienen la gráfica de cómputo y los valores de los parámetros que se han entrenado. La gráfica se guarda en un archivo con extensión `.meta` y éste contiene la información requerida para continuar el entrenamiento, realizar evaluaciones o ejecutar inferencias sobre una gráfica entrenada previamente.

La API REST se encarga de cargar la gráfica que está en el archivo `image_classifier.ckpt.meta` para que al recibir una imagen enviada por el robot Fibonacho o cualquier otro dispositivo cliente se envíe en el cuerpo de la respuesta un JSON con la cadena que identifica a la categoría inferida.

Se hicieron 40 solicitudes a la API REST desde el robot, 10 por cada categoría. Esto es, se enviaron 10 imágenes de la cancha, 10 de la salida, 10 del cubículo y 10 del área de trabajo. En los cuadros 5, 6, 7 y 8, se resumen los resultados obtenidos. En la figura 11 se muestran algunas imágenes capturadas en las pruebas.

Se puede ver que existen más aciertos con imágenes de la cancha de fútbol, 8 de 10 inferencias correctas. Clasificando imágenes de la salida de emergencia y del cubículo se tienen 7 de 10 predicciones correctas. Para el área de trabajo tuvo su peor desempeño con 6 de 10 aciertos. Si bien no se obtiene la precisión reportada durante el entrenamiento del modelo, el tiempo de ejecución y los resultados son buenos para la funcionalidad de auxiliar a la navegación del robot. El tiempo de solicitud y repuesta varía entre peticiones. Factores que influyen al tiempo son el tamaño de la imagen enviada y la velocidad de la conexión entre cliente y del servidor.

Cuadro 5: Resultados de las imágenes de la cancha de fútbol.

Clase	Predicción	Tiempo	Correcta
cancha	z. trabajo	3.0337650776	F
cancha	cancha	3.0559039116	T
cancha	z. trabajo	2.09687018394	F
cancha	cancha	3.0169699192	T
cancha	z. trabajo	2.32713413239	F
cancha	cancha	3.0284428596	T
cancha	cancha	3.0252449512	T
cancha	cancha	3.1204240322	T
cancha	cancha	3.013076067	T
cancha	cancha	3.2163701057	T

(a) Clase = cancha, predicción = cancha. Tiempo = 3.05

(b) Clase = salida, predicción = salida. Tiempo = 3.29

(c) Clase = zona de trabajo, predicción = cubículo. Tiempo = 5.21

(d) Clase = cubículo, predicción = cubículo. Tiempo = 3.53

Figura 11: Predicciones hechas enviando imágenes del robot a la API REST. El tiempo son los segundos que se tardó en enviar la solicitud y en recibir la respuesta.

Cuadro 6: Resultados para fotografías de la salida de emergencia.

Clase	Predicción	Tiempo	Correcta
salida	salida	3.2262349129	T
salida	salida	3.0402858257	T
salida	salida	3.298979044	T
salida	cubículo	3.6488580704	F
salida	salida	3.4767799377	T
salida	salida	3.6147930622	T
salida	salida	3.7430388927	T
salida	cubículo	3.7496609688	F
salida	salida	3.8713350296	T
salida	cubículo	4.0420210361	F

Cuadro 8: Resultados de obtenidos con imágenes del cubículo.

Clase	Predicción	Tiempo	Correcta
cubículo	cubículo	3.5342819691	T
cubículo	cubículo	3.9999611378	T
cubículo	z. trabajo	4.6536300182	F
cubículo	z. trabajo	3.8637499809	F
cubículo	z. trabajo	4.887745142	F
cubículo	cubículo	4.4888358116	T
cubículo	cubículo	4.6198430061	T
cubículo	cubículo	4.4448840618	T
cubículo	cubículo	4.5735599995	T
cubículo	cubículo	5.2998209	T

Cuadro 7: Resultados para la fotografías del área de trabajo.

Clase	Predicción	Tiempo	Correcta
z. trabajo	z. trabajo	5.3504459858	T
z. trabajo	cubículo	5.2893049717	F
z. trabajo	cubículo	5.2149860859	F
z. trabajo	z. trabajo	5.2179970741	T
z. trabajo	cubículo	5.6882929802	F
z. trabajo	cubículo	5.6848390102	F
z. trabajo	z. trabajo	5.9792921543	T
z. trabajo	z. trabajo	5.8859920502	T
z. trabajo	z. trabajo	5.722206831	T
z. trabajo	z. trabajo	6.7581658363	T

7. Conclusiones

Los servicios web utilizados en este trabajo permiten que una plataforma con capacidades de procesamiento limitado, como el robot Fibonacho, adquieran nuevas funcionalidades que permiten introducir al robot a otros campos de estudio.

A través de la API REST creada, el robot o cualquier dispositivo cliente tiene acceso a los recursos mediante simples peticiones HTTP. Sobre la API REST se pueden integrar nuevos modelos de aprendizaje o servicios que solucionen problemas específicamente para el robot Fibonacho, como el clasificador de escenarios, o cualquier dispositivo.

Entre los trabajos a futuro están: crear modelos que sustituyan a los consumidos por el servidor para no depender de terceros, lanzar la aplicación que ejecuta la API REST sobre una Plataforma como Servicio (PaaS) y crear nuevos servicios para diferentes dispositivos que se utilicen en el campo de la robótica móvil.

Referencias

1. Google Developers. Google cloud vision. `https://cloud.google.com/vision/overview/docs/`, 2018.
2. Inc. Wit.ai. Wit.ai. `https://wit.ai/docs`, 2018.
3. Google Developers. Cloud ml engine for tensorflow. `https://cloud.google.com/ml-engine/docs/tensorflow/getting-started-training-prediction`, 2018.
4. Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. The MIT Press, 2017.
5. SPL Committee. Robocup standard platform league. `http://spl.robocup.org/`, 2018.
6. Pieter Abbeel Ken Goldberg Ben Kehoe, Sachin Patil. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 2014.

7. Slaven Marusica Marimuthu Palaniswamia Jayavardhana Gubbia, Rajkumar Buyyab. Internet of things(iot):a vision, architectural elements, and future directions. *Future Generation Computer Systems*, 2013.
8. Google Developers. Android developers. `https://developer.android.com/docs/`, 2018.
9. Google Developers. Firebase documentation. `https://firebase.google.com/docs/`, 2018.
10. Jake Wharton. Butter knife. `http://jakewharton.github.io/butterknife/`, 2013.
11. Google Developers. Volley overview. `https://developer.android.com/training/volley/`, 2018.
12. Facebook Open Source. React getting stared. `https://reactjs.org/docs/getting-started.html`, 2018.
13. Inc. Kairos AR. Developing with kairos. `https://www.kairos.com/docs/api/`, 2018.
14. Google Developers. Cloud translation. `https://cloud.google.com/translate/overview/docs/`, 2018.
15. Kristin Marsicano Bill Phillips, Chris Stewart. *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch, LLC., 2017.
16. Laurence Moroney. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. Apress, 2017.
17. Armando Escalante Borko Furht. *Handbook of Cloud Computing*. Springer, 2010.
18. Francesco Giannini, Vincenzo Laveglia, Alessandro Rossi, Dario Zanca, and Andrea Zugarini. Neural networks for beginners. A fast implementation in matlab, torch, tensorflow. *CoRR*, 2017.
19. Sarah Guido Andreas C. Müller. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016.
20. Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc., 2017.
21. Nikhil Buduma. *Fundamentals of Deep Learning*. O'Reilly Media, Inc., 2017.
22. Aldebaran. Naoqi documentation. `http://doc.aldebaran.com/2-1/index.html`, 2016.
23. Miguel Grinberg. *Flask Web Development*. O'Reilly Media, Inc., 2014.
24. Brian Totty David Gourley. *HTTP The Definitive Guide*. O'Reilly Media, Inc., 2002.
25. Mike Amundsen Leonard Richardson. *RESTful Web APIs*. O'Reilly Media, Inc., 2013.
26. Sam Ruby Leonard Richardson. *RESTful Web Services*. O'Reilly Media, Inc., 2007.
27. Dan Sloughter. *The Calculus of Functions of Several Variables*. Orange Grove Texts Plus, 2001.
28. Martín Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software disponible en tensorflow.org.
29. Balmohan V. Limaye Sudhir R. Ghorpade. *A Course in Multivariable Calculus and Analysis*. Springer, 2010.
30. J.R. Parker. *Algorithms for Image Processing and Computer Vision*. Wiley Publishing, Inc., 2011.
31. Baoxin Li Ragav Venkatesan. *Convolutional Neural Networks in Visual Computing*. CRC Press, 2018.
32. Yefeng Zheng Lin Yang Le Lu, Gustavo Carneiro. *Deep Learning and Convolutional Neural Networks for Medical Image Computing*. Springer, 2017.

33. Ameet Talwalkar Mehryar Mohri, Afshin Rostamizadeh. *Foundations of Machine Learning*. The MIT Press, 2012.
34. Elnaz Jahani Heravi Hamed Habibi Aghdam. *Guide to Convolutional Neural Networks*. Springer, 2017.
35. Simon Haykin. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., 2007.
36. Raúl Rojas. *Neural Networks A Systematic Introduction*. Springer, 1991.
37. Mark Massé. *REST API Design Rulebook*. O'Reilly Media, Inc., 2012.
38. Syed Afaq Ali Shah Mohammed Bennamoun Salman Khan, Hossein Rahmani. *A Guide to Convolutional Neural Networks for Computer Vision*. Morgan & Claypool Publishers, 2018.
39. Richard G. Palmer John A. Hertz, Anders S. Krogh. *Introduction To The Theory Of Neural Computation*. Westview Press, 1991.
40. Robert J. Marks Russell Reed. *Neural Smithing*. The MIT Press, 1999.
41. Yoshua Bengio Patrick Haffner Yann Lecun, Léon Bottou. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.
42. Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.
43. Greg Soltis. Announcing streaming for the firebase rest api. `https://firebase.googleblog.com/2014/03/announcing-streaming-for-firebase-rest.html`.