

# Universidad Central del Ecuador

---

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

INGENIERIA EN SISTEMAS DE LA INFORMACIÓN

Minería de Datos

**Tema:** Manual Técnico proyecto integrador  
(NamiTeach)

## **Integrantes:**

- Angelica Acaro
- Cristian Morales
- Leonardo Orbe
- Joel Taco





## Contenido

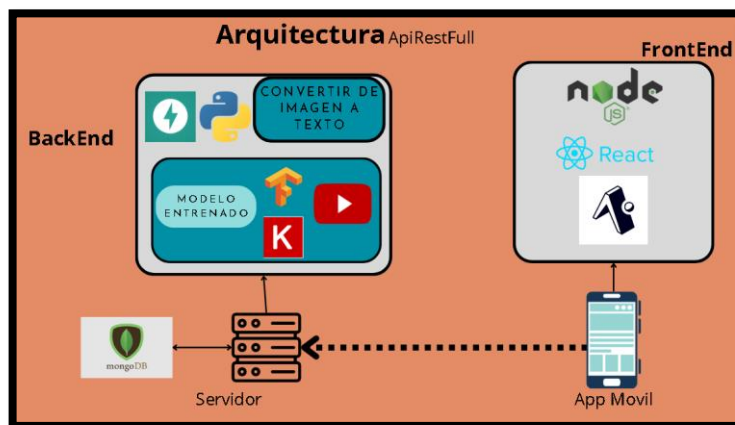
Universidad Central del Ecuador .....	0
Integrantes:.....	0
Introducción: .....	2
Arquitectura de la Aplicación: .....	2
Descripción de las tecnologías Utilizadas .....	2
Tecnologías Usadas en FrontEnd:.....	2
Tecnologías utilizadas en BackEnd: .....	3
Estructura del Proyecto .....	3
BackEnd: .....	3
FrontEnd:.....	4
Documentación .....	5
BackEnd:.....	5
/convert/text .....	5
/basicknowledge.....	6
Despliegue.....	7
Requisitos indispensables para el despliegue .....	7
BackEnd .....	7
Dependencias .....	7
levantar API .....	8
Publicar Api para ser consumida desde la app-movil .....	10
FronEnd: .....	10
Red Neuronal.....	12
Como se fabricó la red de entrenamiento: .....	12

## Introducción:

NamiTeach se trata de una aplicación que busca solventar los problemas de aprendizaje de niños de grado escolar, por medio de una aplicación móvil la cual con ayuda de la cámara de un dispositivo móvil, es capaz de capturar preguntas, instrucciones o indicaciones para obtener el conocimiento base que el niño requiere para resolver dicho problema y entregarle una serie de videos (capsulas informativas) para que logre entenderlo, en esta etapa de la aplicación se busca ofrecer los recursos necesarios para que el cliente entienda como resolver su problema, tiene un enfoque proactivo y busca ayudar de manera real, en lugar de solo realizar la tarea el niño y quitarle la responsabilidad. A futuro se podría incluir nuevas funcionalidades como ser capaz de analizar una tarea por completo, inclusive entregar el conocimiento no solo con videos, si no generar la respuesta al problema, pero con un enfoque educativo explicando el cómo y porque se llega a esa respuesta.

## Arquitectura de la Aplicación:

El sistema cuenta con 2 grandes módulos, el BackEnd que está compuesta por un API Rest la cual responde a peticiones realizadas por un FrontEnd, en este caso particular una App Movil.



## Descripción de las tecnologías Utilizadas

### Tecnologías Usadas en FrontEnd:

- **React Native:** Es un marco de desarrollo de aplicaciones móviles que permite crear aplicaciones móviles multiplataforma utilizando JavaScript y React.
- **JavaScript:** Es un lenguaje de programación ampliamente utilizado para el desarrollo web y móvil.
- **Axios:** Es una biblioteca de JavaScript utilizada para realizar solicitudes HTTP desde el cliente (en este caso, solicitudes a una API).

- **Expo:** Es una plataforma y conjunto de herramientas para construir aplicaciones React Native de forma rápida y sencilla, proporcionando acceso a una variedad de API y servicios integrados.

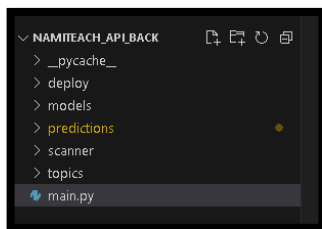
## Tecnologías utilizadas en BackEnd:

- Python
- FastApi: Framework de desarrollo web rápido para construir APIs RESTful con Python.
- Tensor Flow: Biblioteca de código abierto para machine learning y deep learning desarrollada por Google.
- Keras: Interfaz de alto nivel para construir y entrenar modelos de deep learning.
- BERT: Bidirectional Encoder Representations from Transformers (BERT) es un modelo de lenguaje preentrenado basado en la arquitectura de transformers. Utilizado para tareas de procesamiento del lenguaje natural (NLP), como la clasificación de texto y la extracción de información.
- Tesseract-OCR: OCR (Optical Character Recognition) de código abierto desarrollado por Google. Utilizado para reconocer texto en imágenes y convertirlo en texto digital.

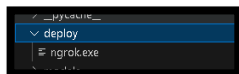
## Estructura del Proyecto

### BackEnd:

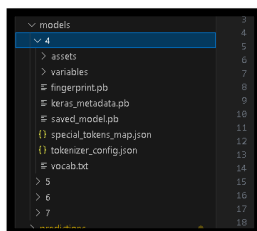
El proyecto donde se ubica la sección back end se denomina `namiteach_api_back` que es la carpeta Raiz



1. Main.py: Este archivo contiene la lógica del api, el objeto iniciador y los endpoints .
2. Deploy: Esta carpeta contiene el ejecutable de Ngrok para la publicación del api

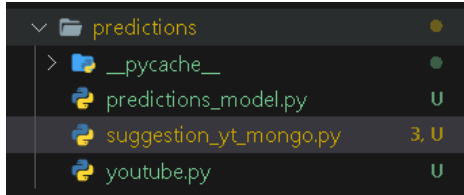


3. Models: Esta carpeta contiene todos los modelos entrenados previamente para realizar las predicciones



Cabe mencionar que existe un modelo por cada grado escolar debido a la complejidad de precisión al unificar todas las etiquetas, dentro de cada una de estas carpetas encontraremos la estructura del modelo con sus pesos.

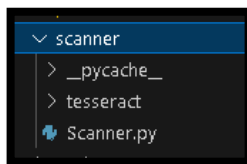
4. Predictions: Esta carpeta contiene las clases encargadas de la predicción de nuevos problemas y su solución extraída de youtube:



Predictions\_model.py: contiene toda la lógica para realizar el etiquetado de nuevos problemas, se encarga de cargar los modelos, tokenizar los problemas y obtener su etiqueta según la predicción  
 Youtube.py: contiene toda la lógica que se encarga de obtener el contenido videográfico desde YouTube para el tema base obtenido después de la predicción

Suggestion\_yt\_mongo: Contiene toda la lógica de programación referente al algoritmo de recomendaciones, la conexión y operaciones CRUD con la Base de datos Mongo

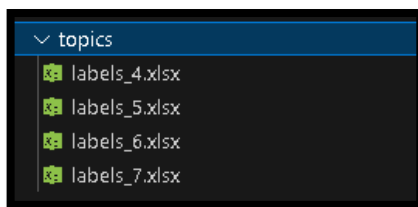
5. Scanner: esta carpeta es el módulo que se encarga de extraer el texto desde las imágenes



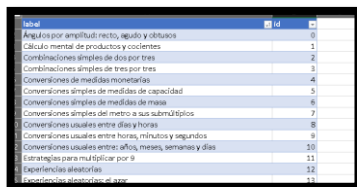
Tesseract: es la carpeta que contiene el módulo de tesseract, con sus respectivos archivos y ejecutable.

Scanner.py: es el archivo que contiene la lógica para convertir una imagen en texto.

6. Topics: carpeta que contiene la información sobre los temas con los que fueron entrenadas las redes



Todos son archivos Excel los cuales contienen de forma organizadas las etiquetas de los conocimientos base.



tema	id
Ángulos por amplitud recto, agudo y obtuso	0
Cálculo mental de productos y cocientes	1
Combinaciones simples de dos por tres	2
Combinaciones simples de tres por tres	3
Conversiones de medidas monetarias	4
Conversiones simples de medidas de capacidad	5
Conversiones simples de medidas de masa	6
Conversiones simples del metro a sus submúltiplos	7
Conversiones usuales entre días y horas	8
Conversiones usuales entre horas, minutos y segundos	9
Conversiones usuales entre: años, meses, semanas y días	10
Estrategias para multiplicar por 9	11
Exercicios aleatorios	12
Exercicios aleatorios de suma	13

## FrontEnd:

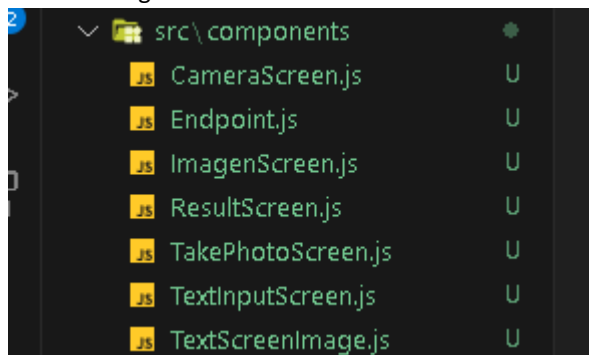
La carpeta donde se almacena el desarrollo del front end

MINERIA-COURSE

- La clase App.js(main) define las pantallas que estarán disponibles para el usuario, junto con los títulos correspondientes para cada pantalla.



- En la capeta src/components e se encuentran 7 componentes: CameraScreen es la funcionalidad para tomar fotos, ImagenScreen tiene la funcionalidad para obtener la imágenes desde la galería, TakePhotoScreen contiene la pantalla principal que alberga tres botones (Tomar Foto, Subir foto, escribir la pregunta), TextInputScreen alberga la pantalla para la funcionalidad de escribir la pregunta y TextScreen Image alberga el texto extraído de las fotos tomadas con las cámaras o las fotos de las galerías.



- La carpeta node\_modules es un directorio en proyectos de Node.js que contiene todas las dependencias instaladas para ese proyecto.



## Documentación

### BackEnd:

Por el lado de BackEnd contamos únicamente con 2 métodos(endpoints) para ser utilizados

#### */convert/text*

Tipo: POST

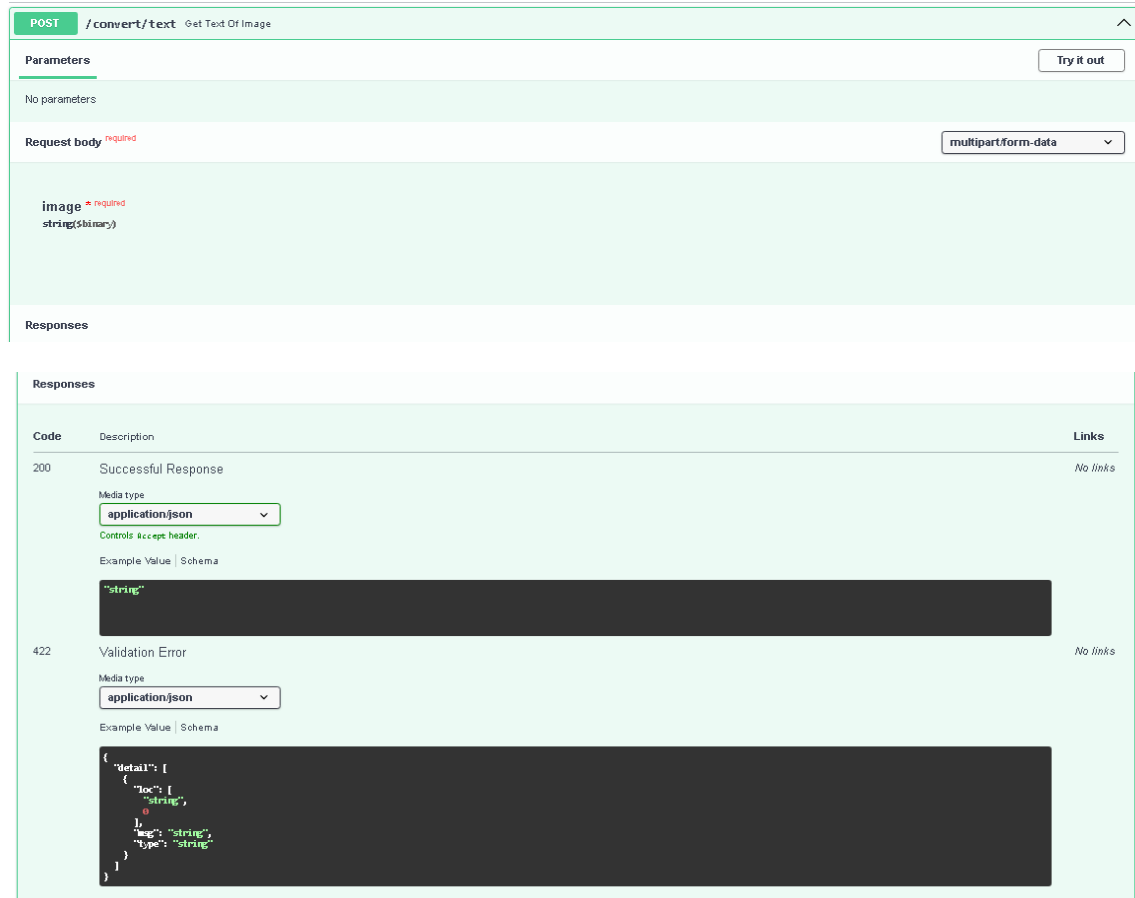
Descripción: Recibe una imagen en el cuerpo de la petición y retorna el contenido de la misma, solo si tiene contenido alfabético, caso contrario, retorna una cadena vacía.

Request Body (obligatorio): dato de tipo imagen (PNG, JPG, ...)

Responses:

- Código 200: Respuesta Exitosa  
Resultado: String con el texto de la imagen

- Código 422: Error de Validación  
Resultado: {detail": [{loc": ["string",0], "msg": "string", "type": "string"}]}



The image shows a Swagger UI for the endpoint `POST /convert/text`. The request body is required and uses `multipart/form-data`. It has a field `image` of type `string` with a required flag. The responses section shows two responses: a 200 'Successful Response' with media type `application/json` and an example value `"string"`, and a 422 'Validation Error' with media type `application/json` and an example value `{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`.

## /basicknowledge

Tipo: POST

Descripción: Recibe una cadena de texto (un problema, aunque no está restringido) y un entero referente al nivel académico que pertenece el texto, como resultado se obtiene una lista de videos los cuales están relacionados al conocimiento necesario para aprender a resolver el problema

Request Body (obligatorio): {"instruction": "string", "level": 0}

Instruction es un String, level es un entero del 4 al 7

Responses:

- Código 200: Respuesta Exitosa  
Resultado: lista de enlaces de youtube
- Código 422: Error de Validación  
Resultado: {detail": [{loc": ["string",0], "msg": "string", "type": "string"}]}

POST

/basicknowledge

Get Basic Content

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "instruction": "string",
  "level": 0
}
```

Responses

Code	Description	Links
200	Successful Response	No links
	<div>Media type</div> <div>application/json</div> <div>Controls accept header.</div> <div>Example Value   Schema</div> <div> <pre>"string"</pre> </div>	
422	Validation Error	No links
	<div>Media type</div> <div>application/json</div> <div>Example Value   Schema</div> <div> <pre>{   "detail": [     {       "loc": [         "string",         0       ],       "msg": "string",       "type": "string"     }   ] }</pre> </div>	

## Despliegue

### Requisitos indispensables para el despliegue

- Python instalado al menos la versión 3.5.11
- Visual Studio Code
- Tener instalado node.js (<https://nodejs.org/en>)
- Tener Motor de Base de datos Mongo instalado en la instancia local y puerto 27017

### BackEnd

Para desplegar el back end vamos a trabajar con la carpeta **namiteach\_api\_back**, que se encuentra dentro de la carpeta general del proyecto

### Dependencias

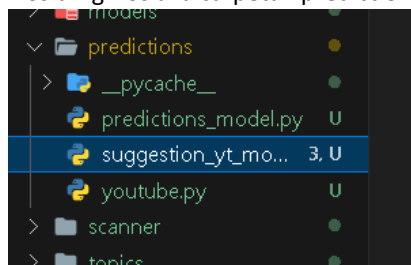
- fastapi: Instala el framework FastAPI, que se utiliza para crear APIs REST en Python.
- uvicorn: Instala Uvicorn, un servidor ASGI para ejecutar aplicaciones FastAPI.
- pytesseract: Proporciona una interfaz para el OCR Tesseract.
- Pillow: Biblioteca de procesamiento de imágenes que se utiliza en conjunto con pytesseract.
- opencv-python: Proporciona herramientas y funciones para el procesamiento de imágenes y visión por computadora.



- python-multipart: Ayuda en el manejo de solicitudes con datos multipart/form-data, útil para la carga de archivos en una API.
- google-api-python-client: Biblioteca cliente para interactuar con diversas API de Google.
- transformers: Biblioteca de Hugging Face para trabajar con modelos de transformers en procesamiento del lenguaje natural (NLP).
- bert-for-tf2: Implementación específica para TensorFlow 2.x del modelo BERT.
- tensorflow: Biblioteca de código abierto para machine learning y deep learning.
- pandas: Librería de análisis de datos que proporciona estructuras de datos y herramientas para manipulación y análisis de datos.
- openpyxl: Librería para trabajar con archivos Excel (.xlsx) en Python.
- ✓ pip install fastapi
- ✓ pip install uvicorn
- ✓ pip install pytesseract
- ✓ pip install Pillow
- ✓ pip install opencv-python
- ✓ pip install python-multipart
- ✓ pip install google-api-python-client
- ✓ pip install transformers bert-for-tf2
- ✓ pip install tensorflow
- ✓ pip install pandas
- ✓ pip install openpyxl
- ✓ pip install spacy
- ✓ python -m spacy download en\_core\_web\_sm
- ✓ pip install fastapi[all] motor

### levantar API

1. Primero es necesario levantar una instancia del motor de base de datos Mongo, no importa la dirección ni el puerto en la que esta sea levantada
  2. Creamos una base de datos llamada namiteach\_db o si se desea algún otro nombre solo es necesario modificar el siguiente archivo del proyecto con la información de la instancia existente
- Nos dirigimos a la carpeta : predictions



Y accedemos al archivo suggestion\_yt\_mongo.py

Al entrar al archivo de inicio se encontrará la conexión con la instancia y la información de la base de datos y la información de la colección de donde se obtendrán los registros:

```
from motor.motor_asyncio import AsyncIOMotorClient

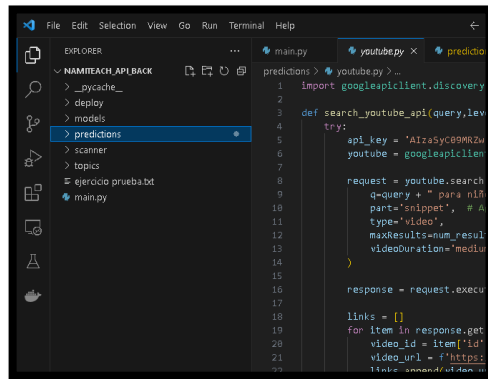
MONGO_URI = "mongodb://localhost:27017"
DATABASE_NAME = "namiteach_db"
COLLECTION_NAME = "suggestions_yt"

client = AsyncIOMotorClient(MONGO_URI)
db = client[DATABASE_NAME]

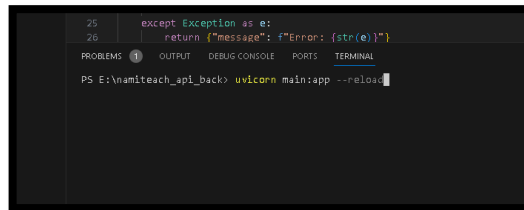
def connect_to_mongo():
    return db
```

Como se mencionó estos parámetros son totalmente modificables. Y Al realizar la operaciones CRUD la estructura de los registros se mapeará automáticamente.

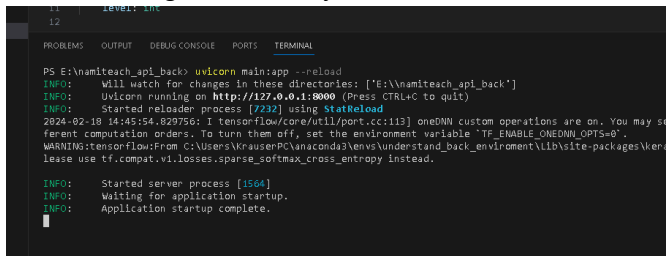
3. Abrimos el proyecto en visual Studio Code llamado namiteach\_api\_back:



4. Ahora nos que abrir una nueva terminal en la opción Terminal de la barra de herramientas, en la parte superior
5. Y ejecutamos el siguiente comando



Dándonos el siguiente mensaje en la terminal:



Esto significa que el backEnd está activo en el puerto 8000.

### *Publicar Api para ser consumida desde la app-movil*

Para que la app móvil pueda consumir nuestro servicio, es necesario publicarlo, por temas de prueba hemos utilizado el servicio de Ngrok, el cual es bastante sencillo y corto, que se encuentra detallado en el siguiente tutorial: <https://www.youtube.com/watch?v=L8iBLXXeFnQ> (5 minutos de video)

Cabe mencionar que en la carpeta deploy dentro del mismo proyecto ya se encontrará el programa de Ngrok para la publicación

### FronEnd:

Para desplegar el front end vamos a trabajar con la carpeta **minería-course**, que se encuentra dentro de la carpeta general del proyecto

#### Requisitos:

1. Verificar que todo se ha instalado correctamente mediante una consola:  
`node --version`  
`code --version`  
`npm --version`
2. descargar la aplicación de Expo Go para realiza la prueba:



Es para poder probar la aplicación.

3. Luego abrir la carpeta llamada minería-course del proyecto visual Studio Code
4. Abrimos una nueva terminal en VSC
5. Ejecutamos el siguiente comando: `npm i`

```
PS E:\minería-course\minería-course> npm i
```

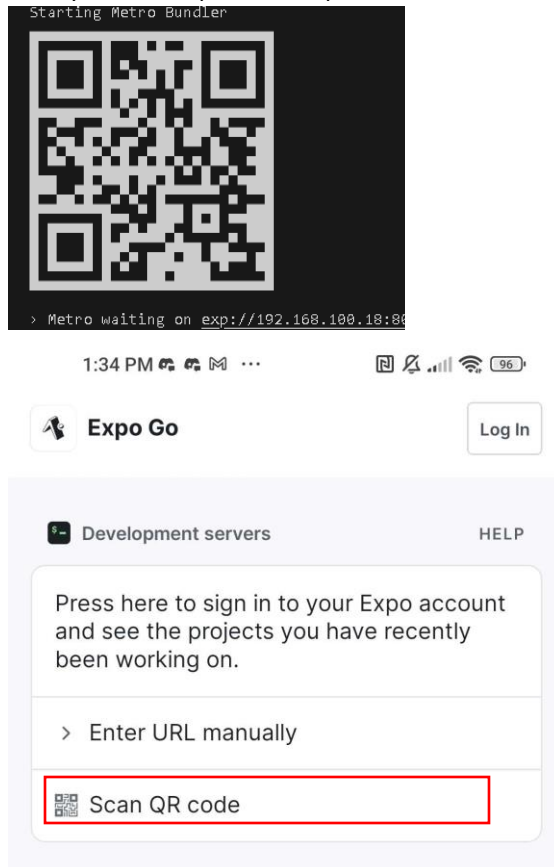
6. Ejecutamos el siguiente comando: `npm start`

```
16 vulnerabilities (5 moderate, 11 high)

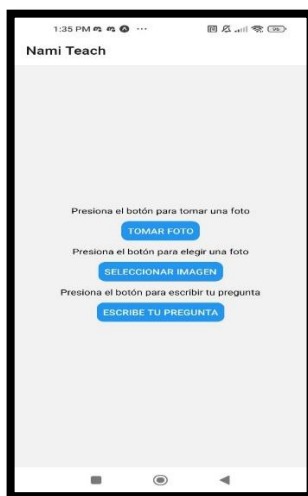
To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
PS E:\mineria-course\mineria-course>
PS E:\mineria-course\mineria-course> npm start
```

7. Con ayuda de la aplicación Expo escaneamos el código



8. Finalmente se despliega la app móvil



## Red Neuronal

### Como se fabricó la red de entrenamiento:

Se utilizo Colab con el poder de su GPU T4:

Para revisar su estructura completa, se encuentra en el archivo **entrenamiento\_red\_neuronal**, que se encuentra dentro de la carpeta del proyecto

- Instalar dependencias e importar librerías:

```
[ ] !pip install transformers

Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.2)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.10/dist-packages (from transformers) (0.16.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from transformers) (1.24.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.12.1)
Requirement already satisfied: tokenizers in /usr/local/lib/python3.10/dist-packages (from transformers) (0.15.1)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.10.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from transformers) (4.5.0)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.10/dist-packages (from requests) (3.3.2)
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-packages (from requests) (3.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.7)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from requests) (2023.7.22)

[ ] from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup, TrainedModel
import torch
import numpy as np
import pandas as pd
import os
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
```

- Cargar los datos de entrenamiento, validación y etiquetas:

```
[ ] # Drive
TRAIN_DATA_PATH = '/content/drive/MyDrive/ProjectoMineria/DatosEntrenamiento/train_com_adjusted_4.xlsx'
TEST_DATA_PATH = '/content/drive/MyDrive/ProjectoMineria/DatosEntrenamiento/test_com_adjusted_4.xlsx'
LABELS_DATA_PATH = '/content/drive/MyDrive/ProjectoMineria/DatosEntrenamiento/label_adjusted_4.xlsx'

# SE CARGAN LOS CSV EN DATA FRAMES
df_labels = pd.read_excel(LABELS_DATA_PATH)
df_train = pd.read_excel(TRAIN_DATA_PATH)
df_test = pd.read_excel(TEST_DATA_PATH)
```

- Limpiar la información para ser procesada

```
# Función para cambiar la clase por un identificador
def change_label_to_number(label_df):
    row = df_labels[df_labels['label'] == label_df]
    if not row.empty:
        return (row['id'].values[0]).astype(int)
    else:
        # Manejar el caso en el que no se encuentra la etiqueta
        return None # o cualquier valor predeterminado que desees

# Función para aplicar a los 2 conjuntos de datos
def adjust_dataset(df):
    # Aplica la función al DataFrame
    df['label'] = df['basic_knowledge_1'].apply(lambda x: change_label_to_number(x))
    df.drop(['basic_knowledge_1', 'type'], axis=1, inplace=True)
    return df

df_train_adjusted = adjust_dataset(df_train)
df_train_adjusted['question'] = df_train_adjusted['question'].astype(str)
df_train_adjusted.head()

df_test_adjusted = adjust_dataset(df_test)
df_test_adjusted['question'] = df_test_adjusted['question'].astype(str)
df_test_adjusted.head()
```

- Tokenizar los datos:

```
# TOKENIZACIÓN
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
PRE_TRAINED_MODEL_NAME = 'dcuchile/bert-base-spanish-wwm-cased'
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
# lo que hace es convertir los textos a tokens para que la red de bert pueda entender las preguntas

[ ] # Tokenizar los textos para poder entrenarlos con bert
max_len = 500 # 500 son los caracteres para escribir preguntas extensas
train_encodings = tokenizer(list(df_train_adjusted['question']), max_length=max_len, truncation=True, padding='max_length', return_tensors='tf')
test_encodings = tokenizer(list(df_test_adjusted['question']), max_length=max_len, truncation=True, padding='max_length', return_tensors='tf')
```

- Modelo de Bert a utilizar (3er modelo generado y escogido como definitivo)

```
class BERTQuestionsClassifier(tf.keras.Model):
    def __init__(self, n_classes, hidden_units=512, dropout_rate=0.3):
        super(BERTQuestionsClassifier, self).__init__()
        self.bert = TFBertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
        self.dropout1 = layers.Dropout(dropout_rate)
        self.hidden_layer1 = layers.Dense(hidden_units, activation='relu')
        self.dropout2 = layers.Dropout(dropout_rate)
        self.hidden_layer2 = layers.Dense(hidden_units//2, activation='relu') # Nueva capa oculta
        self.dropout3 = layers.Dropout(dropout_rate)
        self.hidden_layer3 = layers.Dense(hidden_units//2, activation='relu') # Otra nueva capa oculta
        self.dropout4 = layers.Dropout(dropout_rate)
        self.linear = layers.Dense(n_classes, activation='softmax')

    def call(self, inputs):
        input_ids = inputs['input_ids']
        attention_mask = inputs['attention_mask']

        outputs = self.bert(input_ids, attention_mask=attention_mask)
        pooler_output = outputs.pooler_output

        # Aplicar dropout y capas ocultas adicionales
        drop_output1 = self.dropout1(pooler_output)
        hidden_output1 = self.hidden_layer1(drop_output1)
        drop_output2 = self.dropout2(hidden_output1)

        # Nuevas capas ocultas
        hidden_output2 = self.hidden_layer2(drop_output2)
        drop_output3 = self.dropout3(hidden_output2)
        hidden_output3 = self.hidden_layer3(drop_output3)
        drop_output4 = self.dropout4(hidden_output3)

        # Capa de salida lineal
        output = self.linear(drop_output4)

        return output
```

- Crear instancia del modelo a usar:

```
[ ] # Crear instancia del modelo con el número de clases que debe clasificar
N_CLASSES = len(df_labels['label'])
print(N_CLASSES)
model = BERTQuestionsClassifier(N_CLASSES)
```

- Compilación del modelo y entrenamiento:

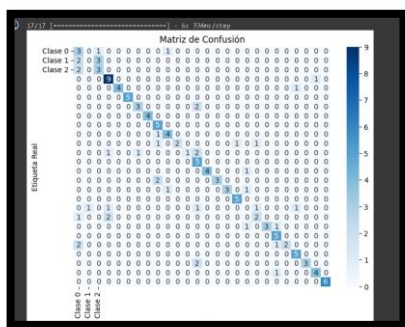
```
[ ] # Compilar el modelo
model.compile(optimizer=Adam(learning_rate=2e-5, weight_decay=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

[ ] # Configuración del conjunto de entrenamiento y prueba con tensorflow
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), df_train_adjusted['label']))
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings), df_test_adjusted['label']))

Etapa de entrenamiento y pruebas del modelo

[ ] EPOCHS = 20
history = model.fit(train_dataset.batch(8), epochs=EPOCHS, validation_data=test_dataset.batch(8))
```

- Algunas pruebas de validación:



- Finalmente, almacenamiento del modelo para transportarlo al api back end:

```
import os

output_dir = "/content/drive/MyDrive/ProjectoMineria/red_neuronal/ito"

os.makedirs(output_dir, exist_ok=True)

model.save(output_dir)

tokenizer.save_pretrained(output_dir)

("content/drive/MyDrive/ProjectoMineria/red_neuronal/ito/tokenizer_config.json",
 "content/drive/MyDrive/ProjectoMineria/red_neuronal/ito/special_tokens_map.json",
 "content/drive/MyDrive/ProjectoMineria/red_neuronal/ito/vocab.txt",
 "content/drive/MyDrive/ProjectoMineria/red_neuronal/ito/model_tokens.json")
```