Lingua Vertex: Language modeling with vertex ai

Abdul Vahed Shaik

San Jose State University, Department of Computer Software Engineering, San Jose, CA, USA

Abstract— ***The daily use of data for a variety of objectives, including marketing, business, sentiment analysis, election prediction, and communication has increased rapidly. In this paper I have implemented different language modeling tasks like text classification, sentiment analysis, and text summarization. I achieved this by fine-tuning the pretrained BERT model and leveraging GCP and Vertex Al to automate every stage of the ML system, including data collecting, model deployment, and serving.***

Keywords— ***Text Classification, Summarization, MLOps, Sentiment Analysis, Question Answering, VertexAI, BERT, HuggingFace, Streamlit, Docker, TensorBoard, Pytorch.***

I. INTRODUCTION

Text classification, Sentiment analysis, and summarization are crucial language modeling tasks. Language modeling is a tool for word prediction that analyses the structure of human language. The fundamental element of natural language processing (NLP), which can handle more text and language, is BERT. I have made use of this capability to carry out the language modeling duties by optimizing the pre-trained BERT model and automating all of the machine learning processes, such as collecting data, model deployment, and serving, through the use of Vertex AI and GCP.

I have performed the following language modeling related tasks using pre-trained models:

1.Text Classification

2.Text Summarization

II. RELATED WORK

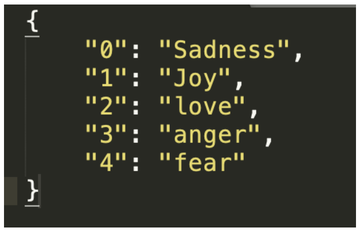
Text is a widespread medium used by people to exchange ideas and have conversations. Many text papers are produced along with the endless interactions that take place on online sites. As a result, numerous academics have developed various Bert model variations in recent years that are simple to adjust for language modeling applications. Previous research on question answering was conducted by Talmar and Berant (2019)[11] who thoroughly examined question answering models using ten datasets, and by Yogatama et al. (2019)[10] who assessed a model across four datasets. DistilBert, which is quick, inexpensive, and trained by distilling Bert base, and the Stanford Question Answering dataset are used in this model.

Typical Deep Learning models for summarization use Seq2Seq architectures with RNNs (Nallapati et al.). Attention mechanisms frequently work in tandem with transformers (Vaswani et al., 2017; Lewis et al., 2020; Raffel et al., 2020a). To address the issue of these models not functioning effectively with financial information, HuggingFace developed the Pegasus summarizer model. With the help of this model and various user decoding techniques, users will be able to comprehend key terms in the text and produce a summary.DATASET

The datasets I utilized for the project, one for each of the language modeling tasks, are listed below:

1.Emotion Classification

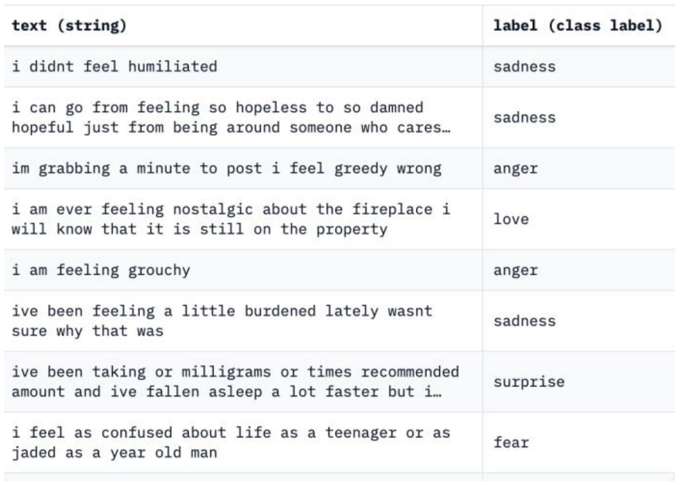
Emotion from Hugging Face Datasets: Using this dataset, I finetuned the pretrained BERT model to better classify emotions. There are labels in the collection that belong to different classifications. Anger, fear, pleasure, love, sadness, and surprise are the six fundamental emotions represented in this collection of Twitter posts.



```
{
    "0": "Sadness",
    "1": "Joy",
    "2": "love",
    "3": "anger",
    "4": "fear"
}
```

Fig 1: Multiclass labels of the emotion classification

Below screenshot shows a preview of the dataset:



text (string)	label (class label)
i didnt feel humiliated	sadness
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares…	sadness
im grabbing a minute to post i feel greedy wrong	anger
i am ever feeling nostalgic about the fireplace i will know that it is still on the property	love
i am feeling grouchy	anger
ive been feeling a little burdened lately wasnt sure why that was	sadness
ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i…	surprise
i feel as confused about life as a teenager or as jaded as a year old man	fear

Fig 2: Preview of the emotion dataset

III. METHODOLOGIES

A. Modeling

We The pre-trained BERT model has been refined and used to the emotion categorization job on Vertex AI. A large corpus of unlabeled text, comprising the whole Wikipedia (2,500 million words) and Book Corpus (800 million words), is used to pre-train the BERT model. Using BERT has the benefit of requiring little adjustment for the emotion classification task. Additionally, this leads to faster development and makes use of a smaller dataset overall.

B. MLOps

All of the necessary phases of an ML system, from data collection to model deployment and servicing, may be conducted and automated with the help of an ML pipeline.

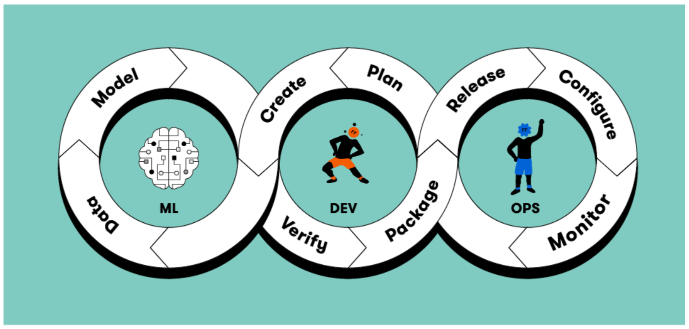


Fig 3: A typical MLOps architecture

The goal of MLOps, an ML engineering culture and practice, is to bring together ML system operation (Ops) and ML system development (Dev). By using MLOps, you provide automation and monitoring during the whole development of an ML system, from infrastructure management to integration, testing, releasing, and deployment. Given pertinent training data for their use case, we can put into practice and train an ML model with predictive performance on an offline holdout dataset. Nevertheless, creating an integrated machine learning system and keeping it running in production continually presents a greater difficulty than simply creating an ML model.

C. GCP and Vertex AI

I'm using the Vertex AI pipeline to custom train a pre-trained BERT model, adjust its hyperparameters, then use it to classify emotions.

Understanding Vertex AI:

Vertex AI is a unified MLOps platform designed to facilitate faster deployment, more experimentation, and confident model management for data scientists and ML engineers. With pre-trained and bespoke tooling within a single AI platform, it facilitates the faster development, deployment, and scaling of ML models.

It is an AI platform that helps in building a unified ML application by integrating Google cloud services. The key features of VertexAI includes:

- A unified UI for the entire ML workflow
- End-to-end integration for data and AI
- Pre-trained APIs for vision, video, natural language, and others
- Support for all opensource frameworks

Emotion Classification using Vertex AI:

Using the bert-base-cased model that was Pre-trained, I have developed an end-to-end training pipeline for the Vertex AI emotion categorization challenge. Emotion Classification is the dataset that is utilized in this task.

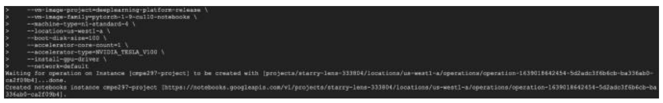
The primary goal of this work is to create, train, optimize, and implement a PyTorch model based on BERT on Vertex AI, with a focus on providing assistance for PyTorch model deployment and training on Vertex AI.

We start by creating notebook instance on the GCP console by running the following command on the cloud console:

```
gcloud notebooks instances create cmpe297-project --vm-image-project=deeplearning-platform-release --vm-image-
family=pytorch-1-9-cu110-notebooks --machine-type=n1-standard-4 --location=us-central1-a --boot-disk-size=100 --
accelerator-core-count=1 --accelerator-type=NVIDIA_TESLA_V100 --install-gpu-driver --network=default
```

Fig 4: Command to create a workbench instance on GCP

Upon running this command, a notebook instance gets created in the workbench section of GCP's vertex AI. The result can be visualized as follows:



```
>     --vm-image-project=deeplearning-platform-release \
>     --vm-image-family=pytorch-1-9-cu110-notebooks \
>     --machine-type=n1-standard-4 \
>     --location=us-west1-a \
>     --boot-disk-size=100 \
>     --accelerator-core-count=1 \
>     --accelerator-type=NVIDIA_TESLA_V100 \
>     --install-gpu-driver \
>     --network=default
Waiting for operation on Instance [cmpe297-project] to be created with [projects/starry-lens-333804/locations/us-west1-a/operations/operation-1639018642454-5d2adc3f6b6cb-ba336ab0-
ca2f09b4]...done.
Created notebooks instance cmpe297-project [https://notebooks.googleapis.com/v1/projects/starry-lens-333804/locations/us-west1-a/operations/operation-1639018642454-5d2adc3f6b6cb-ba
336ab0-ca2f09b4].
```

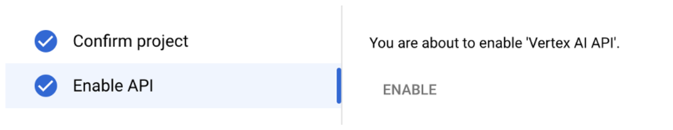
Fig 5: Result of gcloud command execution for notebook creation

In order to create this notebook instance, we have upgraded the required quotas from 0 to 1 since running our code requires GPU.

Google Cloud notebooks cover all the necessary requirements to fulfill the training, fine tuning and deploying of our model for emotion classification. The requirements include:

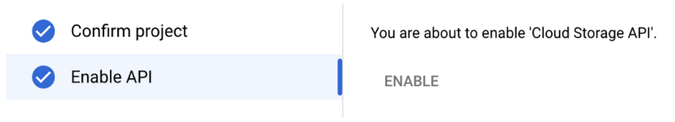
- The Google Cloud SDK
- Git
- Python 3
- Virtualenv which is a tool to create an isolated Python environment.
- Jupyter notebook with Python 3
- Transformers
- Datasets
- Hypertune
- I have used 'aiplatform' library that simplifies ML workflows by utilizing wrapper classes and opinionated defaults.

The following APIs are required to be enabled as a requirement to run the project on GCP:



You are about to enable 'Vertex AI API'.

Fig 8: Enabling Vertex AI API



You are about to enable 'Cloud Storage API'.

Fig 6: Enabling Cloud Storage API

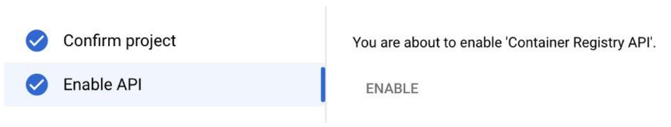


Fig 7: Enabling Container Registry API

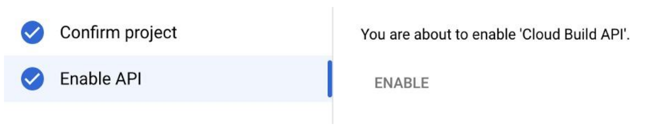
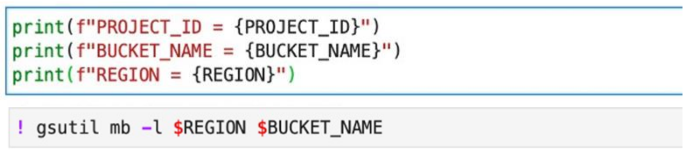


Fig 8: Enabling Cloud Build API

Setting up the cloud storage bucket:

The model artifacts and the outcomes of the hyperparameter tuning are stored in the cloud storage bucket. This is used in the process of developing and implementing the model on Vertex AI. The following commands were run on the notebook instance in order to build a cloud bucket.



```
print(f"PROJECT_ID = {PROJECT_ID}")
print(f"BUCKET_NAME = {BUCKET_NAME}")
print(f"REGION = {REGION}")
```

```
! gsutil mb -l $REGION $BUCKET_NAME
```

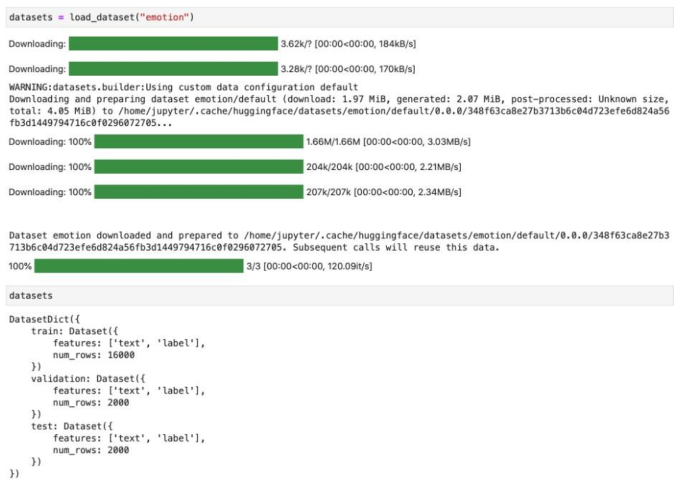
Fig 9: Created a cloud bucket

Model Training

In this project, we use a pre-trained BERT model from Hugging Face Transformers to train the PyTorch emotion classification model locally first, and then we use the Vertex AI training service.

Training locally:

Using the Hugging Face Datasets library, I first downloaded the emotion dataset, which we then loaded using the 'load_dataset' function.



```
datasets = load_dataset("emotion")
```

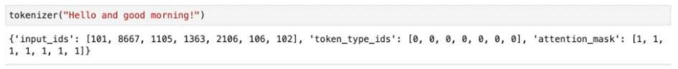
```
Downloading: 3.62k/? [00:00<00:00, 184kB/s]

Downloading: 3.28k/? [00:00<00:00, 170kB/s]
WARNING:datasets.builder:Using custom data configuration default
Downloading and preparing dataset emotion/default (download: 1.97 MiB, generated: 2.07 MiB, post-processed: Unknown size,
total: 4.05 MiB) to /home/jupyter/.cache/huggingface/datasets/emotion/default/0.0.0/348f63ca8e27b3713b6c04d723efe6d824a56
fb3d1449794716c0f0296072705...
Downloading: 100% 1.66M/1.66M [00:00<00:00, 3.03MB/s]

Downloading: 100% 204k/204k [00:00<00:00, 2.21MB/s]

Downloading: 100% 207k/207k [00:00<00:00, 2.34MB/s]


Dataset emotion downloaded and prepared to /home/jupyter/.cache/huggingface/datasets/emotion/default/0.0.0/348f63ca8e27b3
713b6c04d723efe6d824a56fb3d1449794716c0f0296072705. Subsequent calls will reuse this data.
100% 3/3 [00:00<00:00, 120.09it/s]
```

```
datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

Fig 10: Loading the dataset on the notebook instance

In order to tokenize and preprocess the input data into the format needed by the model, I implemented the Hugging Face Transformers "Tokenizer" class. A screenshot of some of the tokens created for text input is shown below:



```
tokenizer("Hello and good morning!")
```

```
{'input_ids': [101, 8667, 1105, 1363, 2106, 106, 102], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1,
1, 1, 1, 1, 1]}
```

Fig 11: Results of the tokenizer on input text

Model Fine-tuning

Transfer learning is a method that allows a deep learning model that was trained on a big dataset to function similarly on a smaller dataset. In order to accomplish this, the middle levels are frozen, and the input and output layers are changed to better fit our needs.

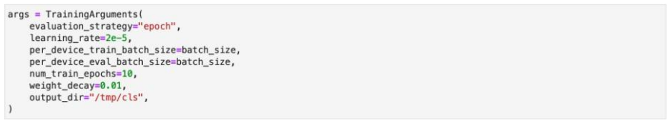
In this work, we are finetuning the pre-trained BERT model for our emotion categorization problem.

Text Classification Using BERT
Input Text
BERT
Classifier
Sport
Politics
Entertainment
Business
Tech

Fig 12: Fine-tuning BERT

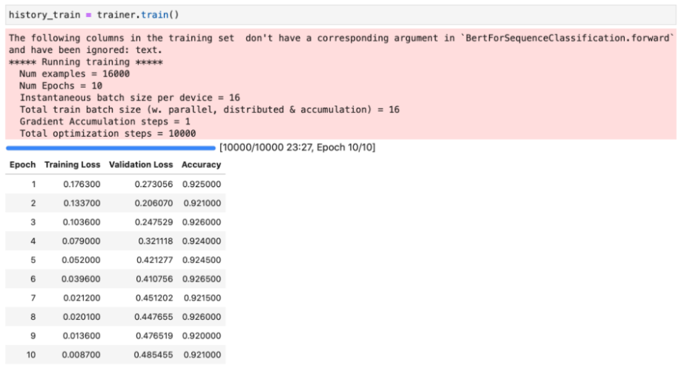
A new head for emotion classification has been utilized in place of the one that was used to pre-train the model on masked language modeling tasks. When we fine-tune the model, pre-trained weights will be assigned; otherwise, they won't have any at first.

Then constructed a trainer object with all the necessary training settings, such as the batch size, learning rate, number of epochs, and weight decay. The same is seen in the screenshots below:



```
args = TrainingArguments(
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=10,
    weight_decay=0.01,
    output_dir="/tmp/cls",
)
```

Fig 13: Creating a trainer object to fine-tune BERT



```
history_train = trainer.train()
```

```
The following columns in the training set  don't have a corresponding argument in `BertForSequenceClassification.forward` and have been ignored: text.
***** Running training *****
  Num examples = 16000
  Num Epochs = 10
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 10000
```

[10000/10000 23:27, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.176300	0.273056	0.925000
2	0.133700	0.206070	0.921000
3	0.103600	0.247529	0.926000
4	0.079000	0.321118	0.924000
5	0.052000	0.421277	0.924500
6	0.039600	0.410756	0.926500
7	0.021200	0.451202	0.921500
8	0.020100	0.447655	0.926000
9	0.013600	0.476519	0.920000
10	0.008700	0.485455	0.921000

Fig 14: Results of fine-tuning BERT for 10 epochs

I also recorded TensorBoard logs for model training and noticed that fine tuning the BERT model for only one epoch suffices as it achieves good accuracy. I trained for 10 epochs so as to visualize the logs on TensorBoard as part of our study.

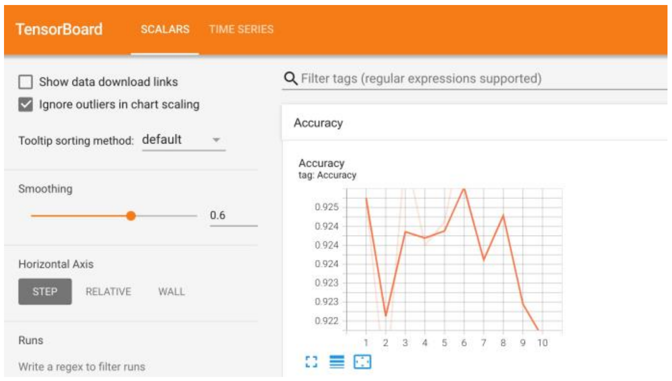


Fig 15: TensorBoard logs for tracking Accuracy

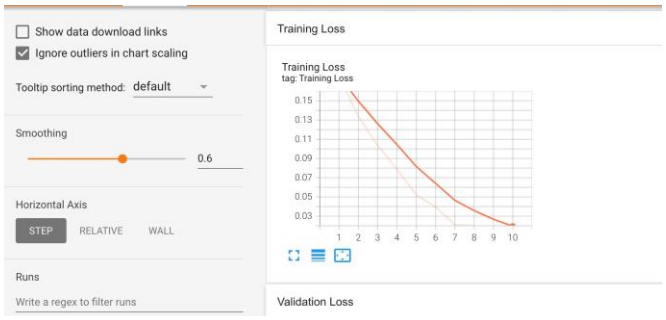


Fig 16: TensorBoard logs for tracking the training loss

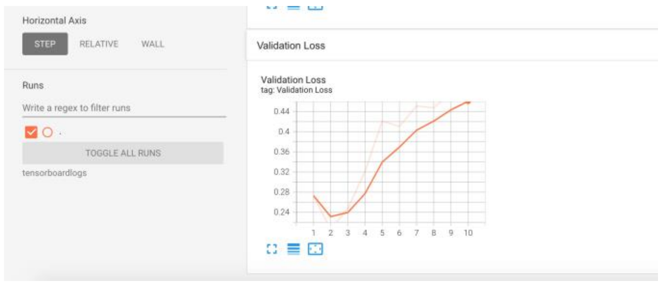
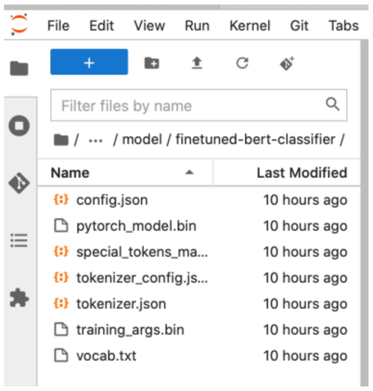


Fig 17: TensorBoard logs for tracking the validation loss

Additionally, we can observe that after training, the model artifacts are saved in the local folders:

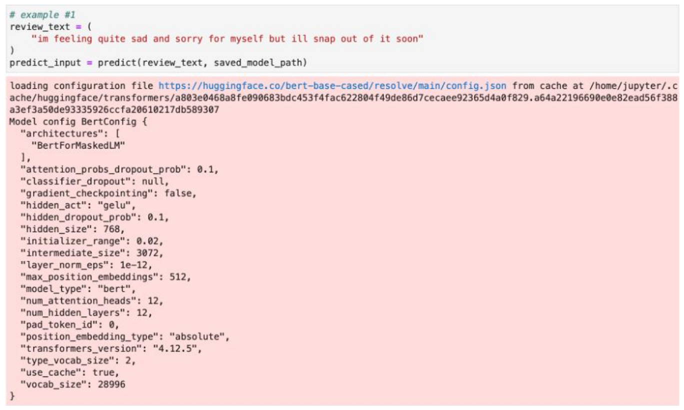


Name	Last Modified
config.json	10 hours ago
pytorch_model.bin	10 hours ago
special_tokens_ma...	10 hours ago
tokenizer_config.js...	10 hours ago
tokenizer.json	10 hours ago
training_args.bin	10 hours ago
vocab.txt	10 hours ago

Fig 18: Model artifacts locally saved

Running predictions locally

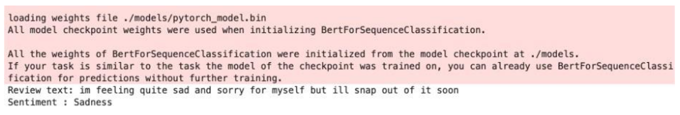
After preprocessing the input, we can now predict the emotion labels for the text input.



```
# example #1
review_text = (
    "im feeling quite sad and sorry for myself but ill snap out of it soon"
)
predict_input = predict(review_text, saved_model_path)
```

```
loading configuration file https://huggingface.co/bert-base-cased/resolve/main/config.json from cache at /home/jupyter/.c
ache/huggingface/transformers/a803e0468a8fe090683bdc453f4fac622804f49de86d7cecaee92365d4a0f829.a64a22196690e0e82ead56f388
a3ef3a50de93335926ccfa20610217db589307
Model config BertConfig {
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.12.5",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}
```

Fig 19: Local predictions the fine-tuned BERT - 1



```
loading weights file ./models/pytorch_model.bin
All model checkpoint weights were used when initializing BertForSequenceClassification.

All the weights of BertForSequenceClassification were initialized from the model checkpoint at ./models.
If your task is similar to the task the model of the checkpoint was trained on, you can already use BertForSequenceClassi
fication for predictions without further training.
Review text: im feeling quite sad and sorry for myself but ill snap out of it soon
Sentiment : Sadness
```

Fig 20: Local predictions the fine-tuned BERT - 2

Training on Vertex AI

Using Vertex AI to design a training pipeline is the most efficient way to handle larger datasets and models, like the ones we have. Vertex AI's training task is completed by packaging the code and setting up a training pipeline to coordinate training tasks. Our three steps have been as follows:

• Preparing the training code for distribution as a Python source code

• Using Vertex AI to finally deploy the optimal model to an endpoint;

• Hyperparameter training job.

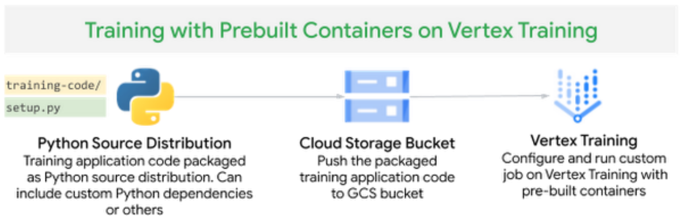

Training with Prebuilt Containers on Vertex Training
training-code/
setup.py
Python Source Distribution
Training application code packaged as Python source distribution. Can include custom Python dependencies or others
Cloud Storage Bucket
Push the packaged training application code to GCS bucket
Vertex Training
Configure and run custom job on Vertex Training with pre-built containers

Fig 21: Custom training on Vertex AI

Step 1: Packaging the training application

The first step involves packaging the training application code with all dependencies and uploading it to the Cloud Storage bucket or Container Registry (dependencies include transformers, datasets and tqdm – which are added in the setup.py file). We achieve this by creating a Python source with pre-built containers on Vertex AI.

We now run the custom training job on Vertex AI using Vertex SDK to create and submit the training job. The below screenshot shows the submission of the job from the notebook instance and also the training job successful on GCP's Vertex AI platform:

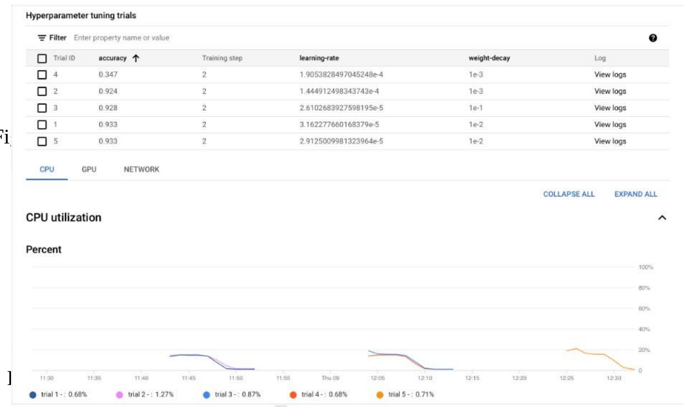
We can see that the model artifacts are successfully saved in the bucket after training.

Hyperparameter Tuning

As I fine-tune the BERT model, I'm experimenting with hyperparameters like learning rate and weight decay. These hyperparameter values can significantly affect the model's performance because they are closely correlated with the training algorithm's behavior. I have packaged the training code and associated dependencies in a Docker container and used the Vertex Training service to automate the hyperparameter tuning. The container was subsequently pushed to Google Container Registry.

Following are the steps involved in performing a hyperparameter tuning job on Vertex AI service:

1. We repeatedly try the training application with the learning rate and weight decay hyperparameters set to their respective values.
2. Every trial run's outcomes are recorded by Vertex AI. Our training application uses the cloudml-hypertune Python tool to send the metrics to Vertex AI.
3. After the job is finished, we receive a Pandas data frame with a summary of all the trial runs that have the optimal settings based on all the criteria we requested.


Hyperparameter tuning trials

Trial ID	accuracy ↑	Training step	learning-rate	weight-decay	Log
4	0.347	2	1.9053828497045248e-4	1e-3	View logs
2	0.924	2	1.444912498343743e-4	1e-3	View logs
3	0.928	2	2.6102683927598195e-5	1e-1	View logs
1	0.933	2	3.162277660168379e-5	1e-2	View logs
5	0.933	2	2.9125009981323964e-5	1e-2	View logs

We can visualize the logs for each of these training runs on Vertex AI. Below screenshot shows the training logs for Trial ID 2:

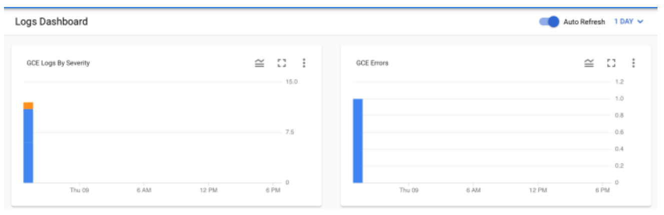


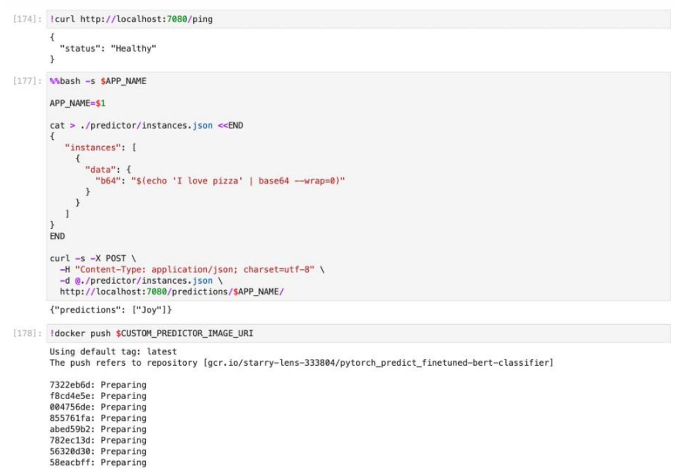
Fig 22: Hyperparameter tuning logs for one trial - 2

Deployment on VertexAI

A Docker container image running an HTTP server is used to deploy our optimized BERT model to VertexAI. The procedures for implementing the model for vertex predictions are as follows:

To service the model, I first constructed a customer container that was compatible with Vertex Predictions and bundled all the necessary model artifacts. At this point, I used the ping command to see how the container was doing. In order to confirm that the predictions are being served, I send a prediction request to the model on the container server upon success.

I then uploaded this model to Vertex as a model resource to serve predictions along with a customized container picture.



```
[174]: !curl http://localhost:7080/ping

{
  "status": "Healthy"
}

[177]: %%bash -s $APP_NAME

APP_NAME=$1

cat > ./predictor/instances.json <<END
{
   "instances": [
     {
       "data": {
         "b64": "$(echo 'I love pizza' | base64 --wrap=0)"
       }
     }
   ]
}
END

curl -s -X POST \
  -H "Content-Type: application/json; charset=utf-8" \
  -d @./predictor/instances.json \
  http://localhost:7080/predictions/$APP_NAME/

{"predictions": ["Joy"]}

[178]: !docker push $CUSTOM_PREDICTOR_IMAGE_URI

Using default tag: latest
The push refers to repository [gcr.io/starry-lens-333804/pytorch_predict_finetuned-bert-classifier]

7322eb6d: Preparing
f8cd4e5e: Preparing
004756de: Preparing
855761fa: Preparing
abed59b2: Preparing
782ec13d: Preparing
56320d30: Preparing
58eacbff: Preparing
```

Fig 23: Pushing the best performing model after hyperparameter tuning

Additionally, I developed a Vertex endpoint and implemented our model resource to provide emotion predictions predicated on the text input.

IV. Experiments

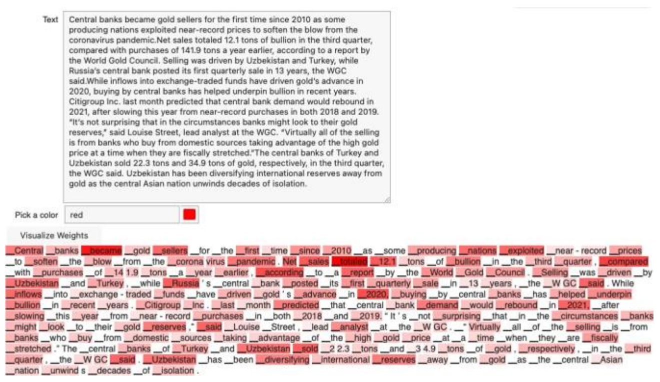
Text Summarization

As a language modeling task, summarization compresses or summarizes a text document by retaining only the most important details, hence assisting in preventing information overload.

I have utilized Hugging Face's Pegasus model, which is specifically designed for financial statement summaries, to execute summarization. Bloomberg market and financial news was used to train it.

Given the assumption that a term with a high word density is more significant for summarization, I highlight the words based on their self-attention weights: high-weight words are bolded, while low-weight words are highlighted in a lighter manner.

We The user can select from a variety of decoding algorithms to generate summaries, and based on the decoding strategy chosen, a control to adjust various hyperparameters is offered. We can select which of the several different summaries produced by the model to combine into a comprehensive overview.



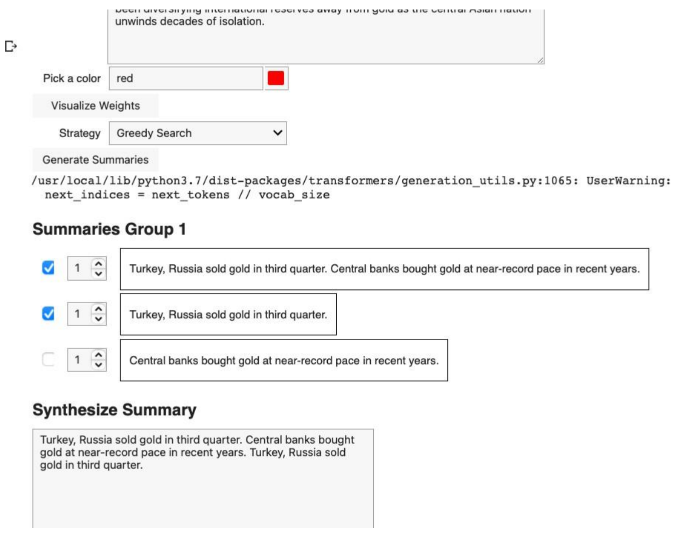
Text: Central banks became gold sellers for the first time since 2010 as some producing nations exploited near-record prices to soften the blow from the coronavirus pandemic.Net sales totaled 12.1 tons of bullion in the third quarter, compared with purchases of 141.9 tons a year earlier, according to a report by the World Gold Council. Selling was driven by Uzbekistan and Turkey, while Russia's central bank posted its first quarterly sale in 13 years, the WGC said.While inflows into exchange-traded funds have driven gold's advance in 2020, buying by central banks has helped underpin bullion in recent years. Citigroup Inc. last month predicted that central bank demand would rebound in 2021, after slowing this year from near-record purchases in both 2018 and 2019. "It's not surprising that in the circumstances banks might look to their gold reserves," said Louise Street, lead analyst at the WGC. "Virtually all of the selling is from banks who buy from domestic sources taking advantage of the high gold price at a time when they are fiscally stretched."The central banks of Turkey and Uzbekistan sold 22.3 tons and 34.9 tons of gold, respectively, in the third quarter, the WGC said. Uzbekistan has been diversifying international reserves away from gold as the central Asian nation unwinds decades of isolation.

Pick a color: red 🟥

```
_Central _banks _became _gold _sellers _for _the _first _time _since _2010 _as _some _producing _nations _exploited _near - record _prices
_to _soften _the _blow _from _the _corona virus _pandemic . Net _sales _totaled _12.1 _tons _of _bullion _in _the _third _quarter , _compared
_with _purchases _of _14 1.9 _tons _a _year _earlier , _according _to _a _report _by _the _World _Gold _Council . _Selling _was _driven _by
_Uzbekistan _and _Turkey , _while _Russia ' s _central _bank _posted _its _first _quarterly _sale _in _13 _years , _the _W GC _said . While
_inflows _into _exchange - traded _funds _have _driven _gold ' s _advance _in _2020, _buying _by _central _banks _has _helped _underpin
_bullion _in _recent _years . _Citigroup _Inc . _last _month _predicted _that _central _bank _demand _would _rebound _in _2021, _after
_slowing _this _year _from _near - record _purchases _in _both _2018 _and _2019. " It ' s _not _surprising _that _in _the _circumstances _banks
_might _look _to _their _gold _reserves ," _said _Louise _Street , _lead _analyst _at _the _W GC . _" Virtually _all _of _the _selling _is _from
_banks _who _buy _from _domestic _sources _taking _advantage _of _the _high _gold _price _at _a _time _when _they _are _fiscally
_stretched ." The _central _banks _of _Turkey _and _Uzbekistan _sold _2 2.3 _tons _and _3 4.9 _tons _of _gold , _respectively , _in _the _third
_quarter , _the _W GC _said . _Uzbekistan _has _been _diversifying _international _reserves _away _from _gold _as _the _central _Asian
_nation _unwind s _decades _of _isolation .
```

The different decoding strategies that can be selected are *Greedy Search, Beam Search, Diverse Beam Search, and Top k sampling.*

Greedy search selects the token with the highest probability at each time step.



unwinds decades of isolation.

Pick a color: red 🟥

Strategy: Greedy Search

```
/usr/local/lib/python3.7/dist-packages/transformers/generation_utils.py:1065: UserWarning:
  next_indices = next_tokens // vocab_size
```

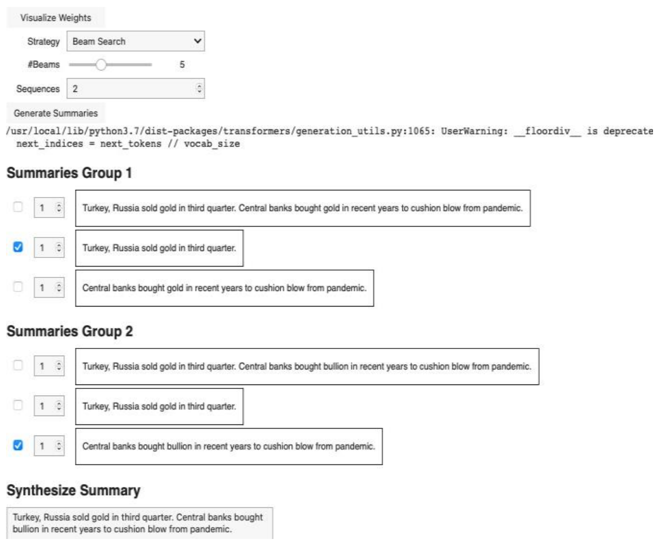
Summaries Group 1

✓	1	Turkey, Russia sold gold in third quarter. Central banks bought gold at near-record pace in recent years.
✓	1	Turkey, Russia sold gold in third quarter.
☐	1	Central banks bought gold at near-record pace in recent years.

Synthesize Summary

Turkey, Russia sold gold in third quarter. Central banks bought gold at near-record pace in recent years. Turkey, Russia sold gold in third quarter.

Beam search chooses a number of tokens with the highest probability based on the beam width in addition to the token with the highest probability at each time step. The beam width and the number of final generated beams are equal.



Visualize Weights

Strategy: Beam Search

#Beams: 5

Sequences: 2

Generate Summaries

```
/usr/local/lib/python3.7/dist-packages/transformers/generation_utils.py:1065: UserWarning: __floordiv__ is deprecate
  next_indices = next_tokens // vocab_size
```

Summaries Group 1

- [ ] 1 Turkey, Russia sold gold in third quarter. Central banks bought gold in recent years to cushion blow from pandemic.
- [x] 1 Turkey, Russia sold gold in third quarter.
- [ ] 1 Central banks bought gold in recent years to cushion blow from pandemic.

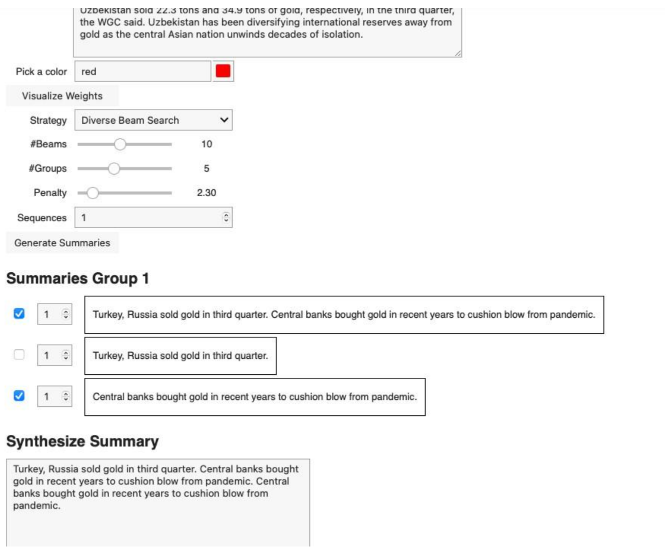
Summaries Group 2

- [ ] 1 Turkey, Russia sold gold in third quarter. Central banks bought bullion in recent years to cushion blow from pandemic.
- [ ] 1 Turkey, Russia sold gold in third quarter.
- [x] 1 Central banks bought bullion in recent years to cushion blow from pandemic.

Synthesize Summary

Turkey, Russia sold gold in third quarter. Central banks bought bullion in recent years to cushion blow from pandemic.

Diverse beam search follows almost the beam search algorithm, but also adds a diversity penalty in order to enhance the diversity between the top most probable generated beams.



Uzbekistan sold 22.3 tons and 34.9 tons of gold, respectively, in the third quarter, the WGC said. Uzbekistan has been diversifying international reserves away from gold as the central Asian nation unwinds decades of isolation.

Pick a color: red 🟥

Visualize Weights

Strategy: Diverse Beam Search

#Beams: 10

#Groups: 5

Penalty: 2.30

Sequences: 1

Generate Summaries

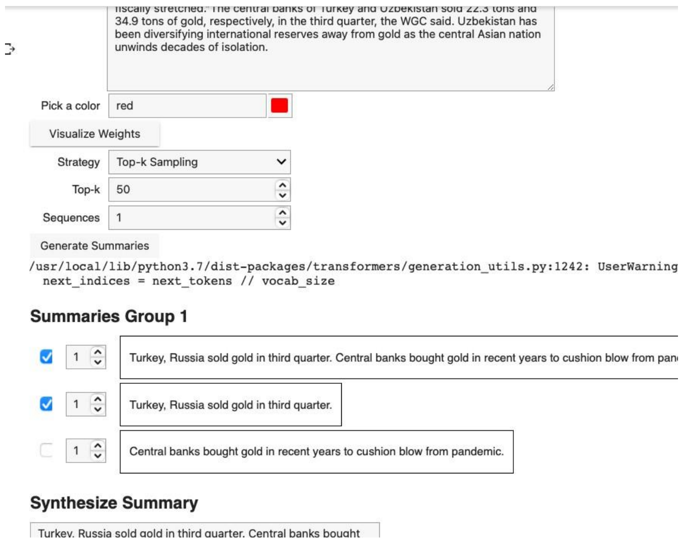
Summaries Group 1

- [x] 1 Turkey, Russia sold gold in third quarter. Central banks bought gold in recent years to cushion blow from pandemic.
- [ ] 1 Turkey, Russia sold gold in third quarter.
- [x] 1 Central banks bought gold in recent years to cushion blow from pandemic.

Synthesize Summary

Turkey, Russia sold gold in third quarter. Central banks bought gold in recent years to cushion blow from pandemic. Central banks bought gold in recent years to cushion blow from pandemic.

With **top-k sampling**, the range of potential future tokens is restricted to the top k higher-ranked tokens in the distribution.



fiscally stretched." The central banks of Turkey and Uzbekistan sold 22.3 tons and 34.9 tons of gold, respectively, in the third quarter, the WGC said. Uzbekistan has been diversifying international reserves away from gold as the central Asian nation unwinds decades of isolation.

Pick a color: red 🟥

Visualize Weights

Strategy: Top-k Sampling

Top-k: 50

Sequences: 1

Generate Summaries

```
/usr/local/lib/python3.7/dist-packages/transformers/generation_utils.py:1242: UserWarning
  next_indices = next_tokens // vocab_size
```

Summaries Group 1

- [x] 1 Turkey, Russia sold gold in third quarter. Central banks bought gold in recent years to cushion blow from pan
- [x] 1 Turkey, Russia sold gold in third quarter.
- [ ] 1 Central banks bought gold in recent years to cushion blow from pandemic.

Synthesize Summary

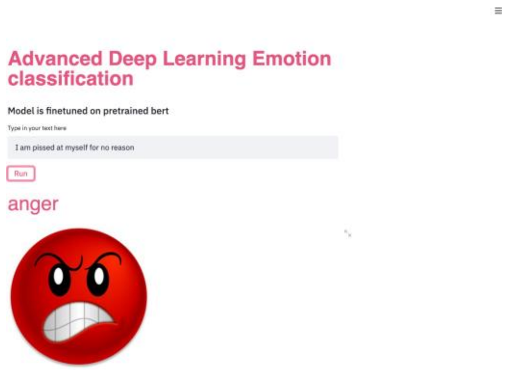
Turkey, Russia sold gold in third quarter. Central banks bought

V. Evaluation and Results

To display our model predictions on the user interface, I created a Streamlit application. By creating a service account key on the cloud console, authenticated the application to call the endpoint after deploying the model to the Vertex AI endpoint.

Subsequently used Google Cloud's "aiplatform" package to offer the fine tuned BERT model predictions to the Streamlit application..

Below are the screenshots of the model serving predictions for each of the class labels in the emotion dataset (sadness (0), joy (1), love (2), anger (3), fear (4)).



Advanced Deep Learning Emotion classification

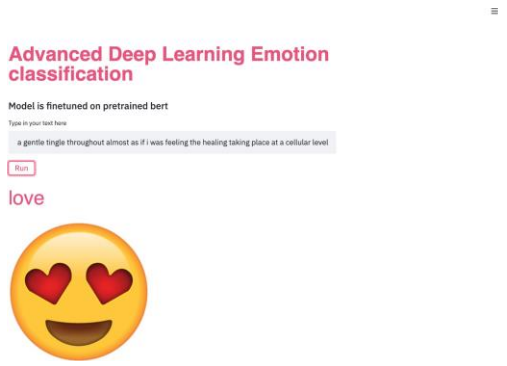
Model is finetuned on pretrained bert

Type in your text here

I am pissed at myself for no reason

Run

anger



Advanced Deep Learning Emotion classification

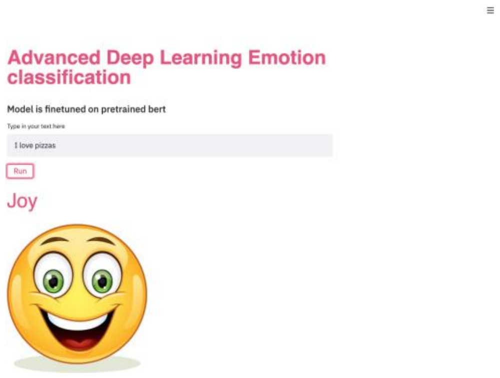
Model is finetuned on pretrained bert

Type in your text here

a gentle tingle throughout almost as if i was feeling the healing taking place at a cellular level

Run

love



Advanced Deep Learning Emotion classification

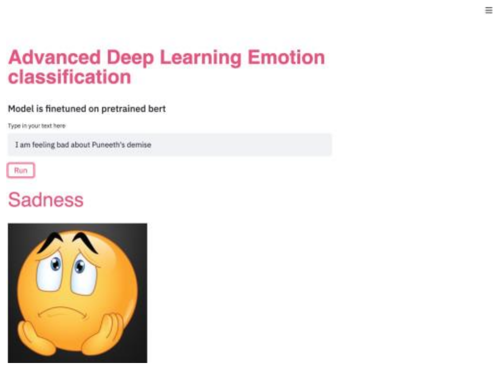
Model is finetuned on pretrained bert

Type in your text here

I love pizzas

Run

Joy



Advanced Deep Learning Emotion classification

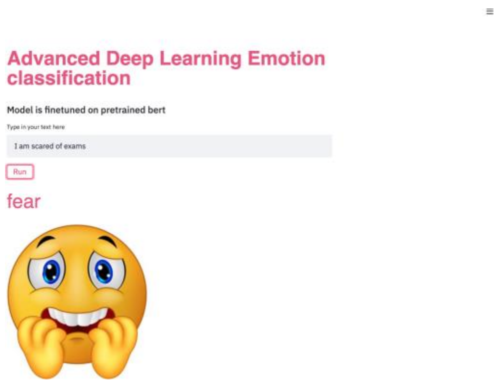
Model is finetuned on pretrained bert

Type in your text here

I am feeling bad about Puneeth's demise

Run

Sadness



Advanced Deep Learning Emotion classification

Model is finetuned on pretrained bert

Type in your text here

I am scared of exams

Run

fear

VI. Challenges

One Model Training: Since it is a Large language model, it was a quite time consuming to train and analyze the model.

Hyperparameter tuning: Finding the right set of hyperparameters was a bit challenging since I was required to run multiple trials of them.

VII. CONCLUSION

Because of the large size of our model (pre-trained BERT) and dataset (Huggingface's Emotion dataset), Vertex AI's distributed server design and processing capability made establishing a training pipeline the most efficient approach. I was able to develop a repeatable pipeline and adjust the hyperparameters to determine the optimal model thanks to Vertex AI. With Vertex AI, we were able to effectively custom train our BERT model and deploy it to the endpoint. By utilizing the Streamlit application's model API, we were also able to receive the predictions effectively.

Additionally, we used pre-trained BERT models to perform language modeling tasks like Text Summarization on several datasets, and we achieved noteworthy model results. Pre-trained models enable us to create more quickly, use less data, and produce better outcomes.

Through this project, I've deepened my understanding of language models, their capabilities, and limitations. I've also learned about training data, bias mitigation, and model interpretability.

References

[1] https://huggingface.co/datasets/emotion

[2] Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., ... & He, L. (2020). A survey on text classification: From shallow to deep learning. arXiv preprint arXiv:2008.00364.

[3] https://github.com/nlpunibo/Question-Answering-SQUAD

[4] https://huggingface.co/transformers/

[5]https://cloud.google.com/vertex-ai/docs/tutorials/image-recognition-automl

[6]Pegasus model: https://huggingface.co/human-centered-summarization/financial-summarization-pegasus