```
!pip install -q datasets
```

```
!pip install --upgrade sentence-transformers==4.1.0
```

Show hidden output

```
import transformers, sentence_transformers
print(f"transformers: {transformers.__version__}")
print(f"sentence-transformers: {sentence_transformers.__version__}")
```

```
transformers: 4.51.3
sentence-transformers: 4.1.0
```

Should be: transformers: 4.51.3 sentence-transformers: 4.1.0

```
from huggingface_hub import notebook_login
notebook_login()
```

```
from datasets import load_dataset

dataset = load_dataset("aegean-ai/ai-lectures-spring-24")
print(dataset)  # Inspect the dataset structure
```

```
DatasetDict({
    train: Dataset({
        features: ['mp4', 'info.json', 'en.vtt', 'json', '__key__', '__url__'],
        num_rows: 8
    })
})
```

Double-click (or enter) to edit

Encode embeddings

```
!pip install fuzzywuzzy
!pip install bertopic
```

Show hidden output

```
import cv2
import numpy as np
from PIL import Image
import tempfile
import os
import re
from collections import OrderedDict
import spacy
from spacy import displacy
from bertopic import BERTopic
```

```
from datasets import load_dataset
import json
import os
# Create a directory to save the JSON files if it doesn't exist
os.makedirs("json_files", exist_ok=True)

# Access the train split (since that's what your dataset shows)
train_dataset = dataset['train']

# Iterate through each example and save the JSON file
for i, example in enumerate(train_dataset):
    # Get the JSON data
    json_data = example['json']

    # Define the output filename (you can customize this)
    output_filename = f"json_files/lecture_{i}.json"

    # Save the JSON data to a file
    with open(output_filename, 'w') as f:
        json.dump(json_data, f, indent=2)  # indent for pretty printing

print(f"Saved {len(train_dataset)} JSON files to the 'json_files' directory.")
```

```
Saved 8 JSON files to the 'json_files' directory.
```

```
sample = dataset['train']
#subtitles = sample["en.vtt"]  # Subtitles are in VTT format
metadata = sample["json"]
```

```
def advanced_clean(text):
    """Specialized cleaning for lecture transcripts"""
    # Remove filler words and artifacts
    text = re.sub(r'\b(uh|um|ah|kind of|sort of|you know)\b', '', text, flags=re.IGNORECASE)
    # Remove XML-like tags and timestamps
    text = re.sub(r'<[^>]+>', '', text)
    # Remove repeated phrases
    text = re.sub(r'(\b\w+\b)(?:\s+\1)+', r'\1', text)
```

```
        # Remove very short sentences
        if len(text.split()) < 5:
            return ""
        return text.strip()

    def remove_repeated_phrases(text):
        # This regex finds repeated phrases (1 to ~15 words) and removes duplicates
        pattern = r'\b((\w+\s+){1,15})\1'
        return re.sub(pattern, r'\1', text, flags=re.IGNORECASE)

    def remove_substring_sentences(sentences):
        sentences = list(set(sentences))  # Remove exact duplicates first
        filtered = []

        for s in sentences:
            if not any((s != other and s in other) for other in sentences):
                filtered.append(s)

        return filtered


    def convert_to_seconds(timestamp):
        """Convert various timestamp formats to seconds (float)"""
        if isinstance(timestamp, (int, float)):
            return float(timestamp)
        elif isinstance(timestamp, str):
            if ':' in timestamp:  # Format like "00:01:30.500"
                parts = timestamp.split(':')
                if len(parts) == 3:  # HH:MM:SS.sss
                    return float(parts[0]) * 3600 + float(parts[1]) * 60 + float(parts[2])
                elif len(parts) == 2:  # MM:SS.sss
                    return float(parts[0]) * 60 + float(parts[1])
            return float(timestamp)  # Try direct conversion
        return 0.0  # Default fallback


    nlp = spacy.load("en_core_web_sm")

    def process_vtt_metadata(metadata):
        chunks = []  # Will contain lists of sentences for each video
        time_chunks = []  # Will contain timestamp information for each sentence
        all_caption_texts = []
        all_timestamps = []  # To store timestamp info for each caption

        # Loop through each video in the dataset
        for video in metadata:
            captions = video['captions']
            for caption in captions:
                all_caption_texts.append(caption['text'])
                # Store start and end times for each caption
                all_timestamps.append((caption['start'], caption['end']))

            single_line_text = " ".join(all_caption_texts)
            single_line_text = advanced_clean(single_line_text)
            single_line_text = remove_repeated_phrases(single_line_text)

            # Process with spaCy
            doc = nlp(single_line_text)

            # Get sentences and approximate their timestamps
            sentences = [sent.text for sent in doc.sents]
            sentence_timestamps = approximate_sentence_timestamps(sentences, all_timestamps, all_caption_texts)

            chunks.append(sentences)
            time_chunks.append(sentence_timestamps)

            # Reset for next video
            all_caption_texts = []
            all_timestamps = []

        return chunks, time_chunks

    from fuzzywuzzy import fuzz  # or: from thefuzz import fuzz

    def approximate_sentence_timestamps(sentences, caption_timestamps, caption_texts):
        """
        Approximate timestamps for sentences using fuzzy matching with original captions.
        All timestamps are converted to and handled in seconds.
        """
        sentence_times = []

        # Convert all caption timestamps to seconds upfront
        caption_timestamps_seconds = [
            (convert_to_seconds(start), convert_to_seconds(end))
            for start, end in caption_timestamps
        ]

        for sentence in sentences:
            # Find best matching caption for start of sentence
            start_match_idx, start_score = find_best_fuzzy_match(
                sentence, caption_texts, is_start=True
            )
            start_time = caption_timestamps_seconds[start_match_idx][0] if start_match_idx != -1 else None

            # Find best matching caption for end of sentence
            end_match_idx, end_score = find_best_fuzzy_match(
                sentence, caption_texts, is_start=False
            )
            end_time = caption_timestamps_seconds[end_match_idx][1] if end_match_idx != -1 else None
            # Fallback logic if no good matches found
            if start_time is None or end_time is None:
                if not sentence_times:  # First sentence
                    start_time = caption_timestamps_seconds[0][0]
                    end_time = caption_timestamps_seconds[0][1]
                else:
                    prev_end = sentence_times[-1][1]  # Already in seconds
                    avg_duration = max(3.0, len(sentence.split()) * 0.5)  # More dynamic duration estimate
                    start_time = prev_end
                    end_time = prev_end + avg_duration
            # Ensure start comes before end
            elif start_time >= end_time:
                end_time = start_time + max(1.0, len(sentence.split()) * 0.3)

            sentence_times.append((start_time, end_time))

        return sentence_times

    def find_best_fuzzy_match(sentence, captions, is_start=True, threshold=75):
        """
        Find the best fuzzy match between a sentence and captions.
        If is_start=True, looks at the beginning of the sentence.
        If is_start=False, looks at the end of the sentence.
        """
        best_idx = -1
        best_score = 0

        # Take first/last 10 words for matching
        words = sentence.split()
        sample_size = min(10, len(words))
        query = " ".join(words[:sample_size] if is_start else words[-sample_size:])

        for i, caption in enumerate(captions):
            # Use partial ratio since we're matching parts of texts
            score = fuzz.partial_ratio(query.lower(), caption.lower())

            if score > best_score:
                best_score = score
                best_idx = i

        # Only return if above threshold
        return (best_idx, best_score) if best_score >= threshold else (-1, best_score)
```

process Embeddings

```
a, b = process_vtt_metadata(metadata)
```

```
import pickle
# Save to file
def save_data(data, filename):
    with open(filename, 'wb') as f:  # 'wb' = write binary
        pickle.dump(data, f)

# Load from file
def load_data(filename):
    with open(filename, 'rb') as f:  # 'rb' = read binary
        return pickle.load(f)
```

```
save_data((a, b), 'caption_data.pkl')
```

```
# Load saved data
loaded_chunks, loaded_time_chunks = load_data('caption_data.pkl')
```

```
from sentence_transformers import SentenceTransformer
clip_model = 'clip-ViT-B-32'
# sentence_model = SentenceTransformer("all-MiniLM-L6-v2")
sentence_model = SentenceTransformer(clip_model)
```

```
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior
```

```
total_models = []
all_embeddings = []  # To store embeddings for each video

for video in loaded_chunks:
    # Get embeddings first
    video_embeddings = sentence_model.encode(video, show_progress_bar=True)
    all_embeddings.append(video_embeddings)  # Store embeddings

    # Then fit BERTopic
    topic_model = BERTopic(embedding_model=sentence_model)
    topics, probs = topic_model.fit_transform(video)
    total_models.append(topic_model)
    print(topic_model.get_topic_info())
```

```
Batches: 100%                                   1/1 [00:00<00:00,  1.17it/s]
   Topic  Count              Name  \
0     -1     12  -1_to_and_the_we

                                Representation  \
0  [to, and, the, we, of, that, is, you, are, in]

                              Representative_Docs
0  [in this video I would like to start the discu...
Batches: 100%                                   2/2 [00:00<00:00,  7.68it/s]
   Topic  Count               Name  \
0      0     38  0_this_so_that_of
1      1     17  1_the_we_this_and

                                  Representation  \
0  [this, so, that, of, and, in, the, okay, be, we]
1  [the, we, this, and, of, that, is, in, to, will]

                              Representative_Docs
0  [so that is we the zero and we'll be getting i...
1  [and maybe the   I mean if right  and maybe th...
Batches: 100%                                   5/5 [00:00<00:00, 13.07it/s]
   Topic  Count               Name  \
0      0     77     0_the_of_we_to
1      1     69  1_okay_so_the_one

                                   Representation  \
0     [the, of, we, to, that, and, is, this, in, be]
1  [okay, so, the, one, and, right, this, is, be,...

                              Representative_Docs
0  [so if I may  and as far as the output feature...
1                               [okay, okay, okay]
Batches: 100%                                   2/2 [00:00<00:00, 22.21it/s]
   Topic  Count             Name  \
0      0     30   0_so_and_we_to
1      1     18  1_the_we_of_and

                                   Representation  \
0  [so, and, we, to, this, have, definitely, our,...
1    [the, we, of, and, to, in, have, is, that, are]

                              Representative_Docs
0  [and so that's basically our head  set, and  I...
1  [so here is the point where the head so here i...
Batches: 100%                                   1/1 [00:00<00:00, 14.83it/s]
   Topic  Count              Name  \
0      0     11   0_the_we_and_of
1      1     11  1_let_say_64_the

                                   Representation  \
0   [the, we, and, of, are, that, this, to, see, so]
1  [let, say, 64, the, those, so, have, of, compo...

                              Representative_Docs
0  [in an earlier video we saw the structure of t...
1  [so this is what we have seen  we can this is ...
Batches: 100%                                   4/4 [00:00<00:00, 13.95it/s]
   Topic  Count              Name  \
0      0    100  0_the_so_of_this
1      1     24    1_the_we_of_to

                                Representation  \
0   [the, so, of, this, okay, to, is, y0, y1, we]
1  [the, we, of, to, and, that, in, have, so, is]

                              Representative_Docs
0  [so the second term is simply FS2 of y1 y 0 + ...
1  [I think it's worthwhile  providing some  guid...
Batches: 100%                                   6/6 [00:00<00:00, 14.23it/s]
   Topic  Count             Name  \
0      0     75   0_it_so_to_and
1      1     58  1_the_and_we_to
2      2     28   2_the_is_so_to

                                   Representation  \
0  [it, so, to, and, going, here, we, okay, that,...
1   [the, and, we, to, is, that, of, this, in, have]
2   [the, is, so, to, what, of, this, just, and, we]

                              Representative_Docs
0  [so we always going to have present here, so i...
1  [definitely this is  a very easy to evaluate t...
2  [so what we will of every predictor so what we...
Batches: 100%                                   2/2 [00:00<00:00, 16.92it/s]
   Topic  Count                 Name  \
0     -1      8  -1_is_of_the_linear
1      0     28     0_so_this_is_the
2      1     26       1_the_to_of_we

                                   Representation  \
0  [is, of, the, linear, to, so, and, discriminat...
1  [so, this, is, the, we, of, and, it, generaliz...
2  [the, to, of, we, is, and, this, probability, ...

                              Representative_Docs
0  [so this is going to be  effectively to be  ef...
1  [and this is the probability of X and this is ...
2  [x this is the form of U logistic regression s...
```

Final Embeddings are located in all_embeddings

The original sentences are located in "loaded_chunks" The time stamps for each sentence are in "loaded_time_chunks"

Insert into Qdrant

```
!pip install -q moviepy qdrant-client
from sentence_transformers import SentenceTransformer
from moviepy.editor import VideoFileClip
from google.colab import files
import os


from qdrant_client.models import VectorParams, Distance


from qdrant_client import QdrantClient, models


# --- Corrected Video Segment Upload Script ---

from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams, PointStruct
from sentence_transformers import SentenceTransformer
from uuid import uuid4
import numpy as np

# --- Start Qdrant client (in memory)
client = QdrantClient(":memory:")

collection_name = "video_captions"

# --- Load embedding model
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")  # 384 dimensions
print("Embedding model loaded.")

# --- Generate real embeddings
real_embeddings = []
for sentences in loaded_chunks:  # loaded_chunks = list of list of sentences
    sentence_embeddings = embedding_model.encode(sentences)
    real_embeddings.append(sentence_embeddings)

# --- Detect embedding dimension
embedding_dim = real_embeddings[0].shape[1]  # Correctly detect 384
print(f"Detected embedding dimension: {embedding_dim}")

# --- Recreate collection with correct size
try:
    client.get_collection(collection_name)
    print(f"Collection '{collection_name}' already exists.")
except:
    client.create_collection(
        collection_name=collection_name,
        vectors_config=VectorParams(
            size=embedding_dim,
            distance=Distance.COSINE,
        )
    )
    print(f"Created collection '{collection_name}' with {embedding_dim} dimensions.")

# --- Insert video segments into Qdrant
for video_idx, (sentences, embeddings, time_data) in enumerate(zip(loaded_chunks, real_embeddings, loaded_time_chunks)):
    points = []

    # Handle time_data correctly: it's a list of (start, end) pairs
    starts = [start for start, end in time_data]
    ends = [end for start, end in time_data]

    # Verify data alignment
    assert len(sentences) == len(embeddings) == len(starts) == len(ends)

    for sentence_idx, (sentence, embedding, start, end) in enumerate(zip(
        sentences,
        embeddings,
        starts,
        ends
    )):
        points.append(
            PointStruct(
                id=str(uuid4()),
                vector=embedding.tolist(),  # Convert numpy array to list
                payload={
                    "video_id": video_idx,
                    "text": sentence,
                    "position": sentence_idx,
                    "start_sec": float(start),
                    "end_sec": float(end),
                    "duration_sec": float(end) - float(start),
                }
            )
        )
    # Upload batch to Qdrant
    operation_info = client.upsert(
        collection_name=collection_name,
        points=points,
        wait=True
    )

    print(f"✅ Uploaded {len(points)} segments for Video {video_idx} "
          f"(Time range: {starts[0]:.1f}s to {ends[-1]:.1f}s)")
```

```
Embedding model loaded.
Detected embedding dimension: 384
Created collection 'video_captions' with 384 dimensions.
✅ Uploaded 12 segments for Video 0 (Time range: 3.0s to 286.7s)
✅ Uploaded 55 segments for Video 1 (Time range: 2.0s to 906.5s)
✅ Uploaded 146 segments for Video 2 (Time range: 3.3s to 2392.0s)
✅ Uploaded 48 segments for Video 3 (Time range: 666.5s to 666.6s)
✅ Uploaded 22 segments for Video 4 (Time range: 2.3s to 503.1s)
✅ Uploaded 124 segments for Video 5 (Time range: 281.2s to 281.2s)
✅ Uploaded 161 segments for Video 6 (Time range: 1.9s to 1.9s)
✅ Uploaded 62 segments for Video 7 (Time range: 5.6s to 1603.8s)
```

Retrieval Processing

```
# Install Python wrapper
!pip install imageio[ffmpeg]
!pip install ffmpeg-python

# Install system ffmpeg
!apt-get update
!apt-get install ffmpeg
```

```
Requirement already satisfied: imageio[ffmpeg] in /usr/local/lib/python3.11/dist-packages (2.37.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from imageio[ffmpeg]) (2.0.2)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.11/dist-packages (from imageio[ffmpeg]) (11.1.0)
Requirement already satisfied: imageio-ffmpeg in /usr/local/lib/python3.11/dist-packages (from imageio[ffmpeg]) (0.6.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from imageio[ffmpeg]) (5.9.5)
Requirement already satisfied: ffmpeg-python in /usr/local/lib/python3.11/dist-packages (0.2.0)
Requirement already satisfied: future in /usr/local/lib/python3.11/dist-packages (from ffmpeg-python) (1.0.0)
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Hit:4 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Get:10 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Fetched 384 kB in 2s (210 kB/s)
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem 
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ffmpeg is already the newest version (7:4.4.2-0ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
```

Gradio APP using FFMPEG

```
# utils/video_utils.py
import ffmpeg
import numpy as np
from io import BytesIO

def extract_segment(video_bytes, start_sec, end_sec):
    """Extract video segment using FFmpeg without saving to disk."""
    try:
        out, _ = (
            ffmpeg.input('pipe:0')
            .trim(start=start_sec, end=end_sec)
            .setpts('PTS-STARTPTS')  # Reset timestamps
            .output('pipe:', format='mp4', vcodec='libx264')
            .run(input=video_bytes, capture_stdout=True, capture_stderr=True)
        )
        return out
    except ffmpeg.Error as e:
        print(f"FFmpeg error: {e.stderr.decode()}")
        return None
```

```
!pip install gradio
```

Show hidden output

```
# --- HEALTH CHECK for Qdrant and Dataset ---

# Test 1: Check if Qdrant collection exists
try:
    collections = client.get_collections()
    print(f"✅ Qdrant collections found: {collections}")
except Exception as e:
    print(f"❌ Error connecting to Qdrant: {e}")

# Test 2: Try searching manually (fallback to .search())
```

```
try:
    dummy_vec = embedding_model.encode("test query").tolist()
    hits = client.search(
        collection_name=collection_name,
        query_vector=dummy_vec,
        limit=1
    )
    print(f"✅ Retrieved {len(hits)} hits from Qdrant.")
except Exception as e:
    print(f"❌ Error during Qdrant search: {e}")

# Test 3: Check if dataset has mp4 bytes
try:
    sample = dataset[0]
    if "mp4" in sample and isinstance(sample["mp4"], bytes):
        print("✅ Dataset has valid mp4 bytes.")
    else:
        print("❌ Dataset missing mp4 bytes.")
except Exception as e:
    print(f"❌ Error accessing dataset: {e}")
```

```
WARNING:py.warnings:<ipython-input-35-62d387b4f00b>:13: DeprecationWarning: `search` method is deprecated and will be removed in the future. Use
  hits = client.search(

✅ Qdrant collections found: collections=[CollectionDescription(name='video_captions')]
✅ Retrieved 1 hits from Qdrant.
❌ Error accessing dataset: "Invalid key: 0. Please first select a split. For example: `my_dataset_dictionary['train'][0]`. Available splits: ['
```

```
# --- Retrieve top subtitles given a user query ---

def retrieve_top_subtitles(question, top_k=3):
    try:
        # 1. Embed the user's question
        query_embedding = embedding_model.encode(question).tolist()

        # 2. Search in Qdrant
        hits = client.search(
            collection_name=collection_name,
            query_vector=query_embedding,
            limit=top_k
        )

        # 3. Collect the subtitles
        subtitles = []
        for hit in hits:
            if hasattr(hit, "payload") and "subtitles" in hit.payload:
                subtitle_text = hit.payload["subtitles"]
                subtitles.append(subtitle_text)

        if subtitles:
            print(f"✅ Retrieved {len(subtitles)} subtitles:")
            for i, subtitle in enumerate(subtitles, 1):
                print(f"Top {i}: {subtitle}")
        else:
            print("⚠️ No subtitles found in hits.")

        return subtitles

    except Exception as e:
        print(f"❌ Error during retrieval: {e}")
        return []


sample = dataset[0]

# Step 1: See available keys
print(f"Sample keys: {sample.keys()}")

# Step 2: See what type 'mp4' is
print(f"Type of sample['mp4']: {type(sample['mp4'])}")

# Step 3: Preview only first 100 characters/bytes
if isinstance(sample["mp4"], bytes):
    print(f"First 100 bytes: {sample['mp4'][:100]}")
elif isinstance(sample["mp4"], str):
    print(f"First 100 characters: {sample['mp4'][:100]}")
else:
    print("Unexpected format inside 'mp4' field.")
```

Show hidden output

Next steps: Explain error

```
  # --- Corrected Gradio App with Subtitles Summary ---

import gradio as gr
import tempfile
import os
from datasets import load_dataset  # Ensure you have datasets installed
# NOTE: Make sure you already initialized client, collection_name, embedding_model separately!

# --- Load your HuggingFace dataset ---
dataset = load_dataset("aegean-ai/ai-lectures-spring-24", split="train")

# --- Helper function ---
def extract_segment(video_bytes, start_sec, end_sec):
    temp_dir = tempfile.gettempdir()
```

```
        temp_path = os.path.join(temp_dir, f"clip_{start_sec:.2f}_{end_sec:.2f}.mp4")

        with open(temp_path, "wb") as f:
            f.write(video_bytes)

        return temp_path

    # --- Main retrieval + answering function ---
    def answer_with_video(question):
        # 1. Encode the question
        question_embedding = embedding_model.encode(question).tolist()

        # 2. Qdrant search (use .search because of old client version)
        hits = client.search(
            collection_name=collection_name,
            query_vector=question_embedding,
            limit=3
        )

        # --- Safety: No hits found ---
        if not hits:
            return "No relevant video segments found.", "", "", ""

        # --- Extract video clips and subtitles ---
        video_clip_paths = []
        subtitle_texts = []

        for i, hit in enumerate(hits):
            video_idx = hit.payload.get("video_id", 0)
            start = hit.payload.get("start_sec", 0)
            end = hit.payload.get("end_sec", 5)

            # Safety: check if video_idx is valid
            if video_idx >= len(dataset):
                return f"Error: Invalid video_idx {video_idx} returned.", "", "", ""

            sample = dataset[video_idx]

            # Safety: check mp4 field
            if "mp4" not in sample or not isinstance(sample["mp4"], bytes):
                return "Error: mp4 bytes not found in dataset.", "", "", ""

            video_bytes = sample["mp4"]

            if video_bytes is None:
                return "Error: No video bytes available.", "", "", ""

            # Save video bytes temporarily
            temp_dir = tempfile.gettempdir()
            clip_path = os.path.join(temp_dir, f"clip_{i}.mp4")

            with open(clip_path, "wb") as f:
                f.write(video_bytes)

            video_clip_paths.append(clip_path)

            # Collect subtitle if available
            subtitle = hit.payload.get("subtitles", "")
            if subtitle:
                subtitle_texts.append(subtitle)

        # --- Generate better answer text ---
        if subtitle_texts:
            answer_text = "Summary based on retrieved clips:\n\n" + "\n".join(
                [f"- {text}" for text in subtitle_texts]
            )
        else:
            answer_text = f"Found {len(video_clip_paths)} relevant segments."

        # --- Ensure 3 outputs ---
        while len(video_clip_paths) < 3:
            video_clip_paths.append("")

        return answer_text, video_clip_paths[0], video_clip_paths[1], video_clip_paths[2]

    # --- Gradio App ---
    with gr.Blocks() as demo:
        gr.Markdown("## 📚 Chat with Your Course Videos (RAG System)")

        with gr.Row():
            question = gr.Textbox(label="Ask a question about the course videos")
            submit_btn = gr.Button("Search")

        answer = gr.Textbox(label="Generated Answer")
        video1 = gr.Video(label="Relevant Clip 1")
        video2 = gr.Video(label="Relevant Clip 2")
        video3 = gr.Video(label="Relevant Clip 3")

        submit_btn.click(
            fn=answer_with_video,
            inputs=question,
            outputs=[answer, video1, video2, video3]
        )

    # Launch the app
    demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://f7de51f44494364414.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory t
```

📚 Chat with Your Course Videos (RAG System)

Ask a question about the course videos

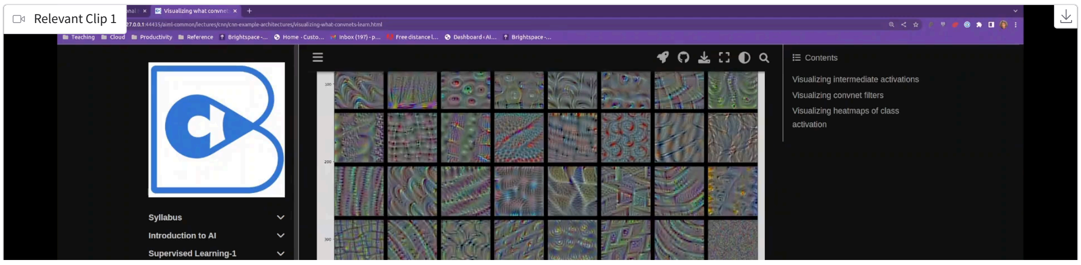
Using only the videos, explain how ResNets work.

Search

Generated Answer

Found 3 relevant segments.

Relevant Clip 1


Visualizing what convnet
Syllabus
Introduction to AI
Supervised Learning-1
Contents
Visualizing intermediate activations
Visualizing convnet filters
Visualizing heatmaps of class activation