UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

APLICAȚIE WEB PENTRU VIDEOCLIP-URI CU FUNCȚII DE ÎNȚELEGEREA VORBIRII ȘI REGĂSIRE PE BAZĂ DE TEXT

Absolvent
Trifan Robert-Gabriel

Coordonator științific
Prof. Dr. Ionescu Radu

București, iunie 2024

Rezumat

Popularitatea videoclipurilor a avut o creștere constantă în ultimii ani, reprezetand 82.5% din traficul web în 2023. Statistici recente arată că oamenii petrec în media 17 ore pe săptămâna vizionând videoclipuri online, acestea fiind cu 52% mai predispuse să fie distribuite pe rețelele de socializare decât alte tipuri de conținut. [3]

În acest context, lucrarea de față își propune să contribuie la creșterea accesibilității și personalizării conținutului video, prin dezvoltarea unei aplicații web care permite adăugarea de subtitrări, căutarea videoclipurilor pe bază de text și clasificarea acestora în funcție de conținutul lor. Aplicația oferă utilizatorilor posibilitatea de a vizualiza subtitrările în timp real și de a căuta cuvinte cheie în metadatele videoclipurilor precum titlu, descriere, topic și subtitrare.

Abstract

The popularity of videos has been steadily increasing in recent years, representing 82.5% of web traffic in 2023. Recent statistics show that people spend an average of 17 hours a week watching online videos, which are 52% more likely to be shared on social networks than other types of content. [3]

In this context, this work aims to contribute to increasing the accessibility and personalization of video content, by developing a web application that allows the addition of subtitles, searching for videos based on text and classifying them according to their content. The application offers users the possibility to view subtitles in real time and to search for keywords in the metadata of videos such as title, description, topic and subtitle.

Cuprins

Capitolul 1

Introducere

1.1 Motivație

Motivul pentru care am ales această temă îl reprezintă nevoia de subtitrări pentru videoclipuri, în special pentru filme, dar și pentru tutoriale sau alte tipuri de conținut video, a căror înțelegere este îngreunată de calitatea slabă a sunetului sau de faptul că sunt într-o limbă străină. Astfel, în această lucrare, am ales să abordez problema subtitrărilor prin intermediul unui model de recunoaștere a vorbirii, care extrage audio dintr-un videoclip și generează textul corespunzător.

De asemenea, am ales să abordez și problema căutării videoclipurilor, care constă în căutarea cuvintelor cheie în metadate precum titlu, descriere, topicuri sau chiar în conținutul videoclipului, folosind o bază de date special concepută pentru acest scop, Elasticsearch. [5]

1.2 Domenii abordate

Această lucrare abordează 5 domenii principale:

- **Procesarea semnalelor audio** - pentru extragerea audio din videoclipuri și recunoașterea vorbirii
- **Procesarea limbajului natural** - pentru clasificarea videoclipurilor în funcție de conținutul lor
- **Frontend** *(React.js)* pentru interfața cu utilizatorul și pentru a oferi acces la funcționalitățile sistemului
- **Backend** - pentru gestionarea cererilor prin intermediul unui API, cu ajutorul a 3 servicii principale:
  - **Server** *(Node.js, Express.js)* pentru gestionarea cererilor și a răspunsurilor

  - **Recunoașterea vorbirii** *(Flask)* pentru gestionarea cererilor de recunoaștere a vorbirii
  - **Clasificarea videoclipurilor** *(Flask)* pentru clasificarea videoclipurilor în funcție de conținutul lor

- **Baze de date**
  - **MongoDB** pentru stocarea metadatelor videoclipurilor
  - **Elasticsearch** pentru căutarea videoclipurilor în funcție de cuvintele cheie

1.3 Structura lucrării

Voi împărți această lucrare în 2 capitole principale, fiecare cu subcapitolele sale.

- **Inteligență artificială** - în care voi aborda *recunoașterea vorbirii* și *clasificarea videoclipurilor* și voi prezenta setul de date folosit, arhitectura modelului, antrenarea și evaluarea acestuia, precum și procesările ulterioare
- **Inginerie software** - în care voi aborda partea de frontend, backend și baze de date, precum și detaliile tehnice ale implementării

Capitolul 2

Inteligență artificială

2.1 Recunoașterea vorbirii

Pentru a putea recunoaște vorbirea dintr-un videoclip, am ales să folosesc arhitectura *wav2vec 2.0* [2] dezvoltată de Facebook AI Research. Am folosit atât modelul *facebook/wav2vec2-base-960h* antrenat pe setul de date *LibriSpeech* [9], cât și modelul preantrenat *facebook/wav2vec2-base* pe care am continuat să-l antrenez pe seturile de date *Mini LibriSpeech* (subset din LibriSpeech) și *Common Voice Delta Segment 16.1* (subset din Common Voice) [1].

2.1.1 Arhitectura modelului

Modelul *wav2vec 2.0* este un model de învățare profundă alcătuit din 4 componente principale: Latent Feature Encoder (Convolutional Network), Context Network (Transformer Encoder), Quantization Module (Gumbel Softmax) și Contrastive Loss. 2.1

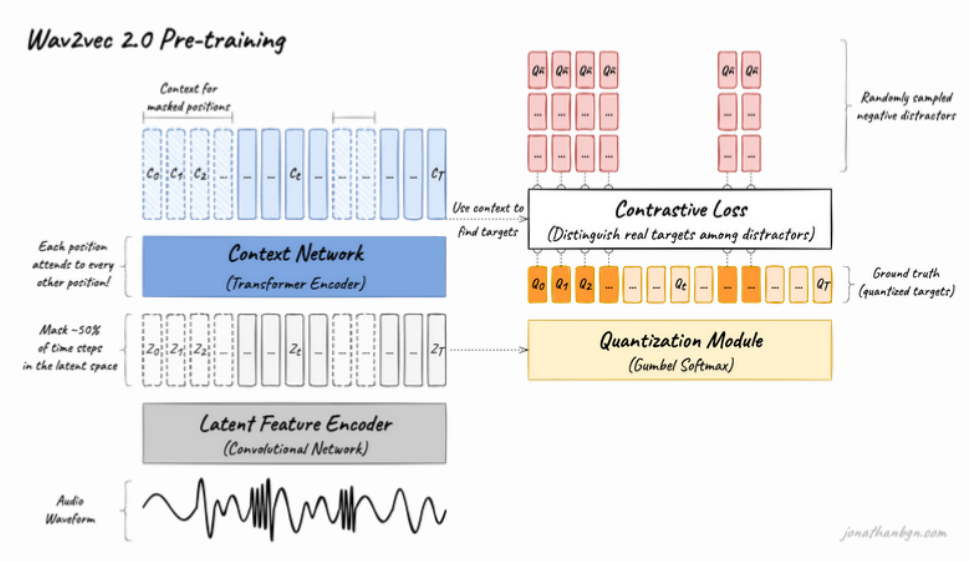


Figura 2.1: Arhitectura modelului *wav2vec 2.0* [1]

Latent Feature Encoder

Componenta Latent Feature Encoder este o rețea convoluțională care primește ca intrare un semnal audio și aplică o serie de operații de convoluție, normalizare și activări GELU pentru a extrage caracteristici latente din semnalul audio. 2.2

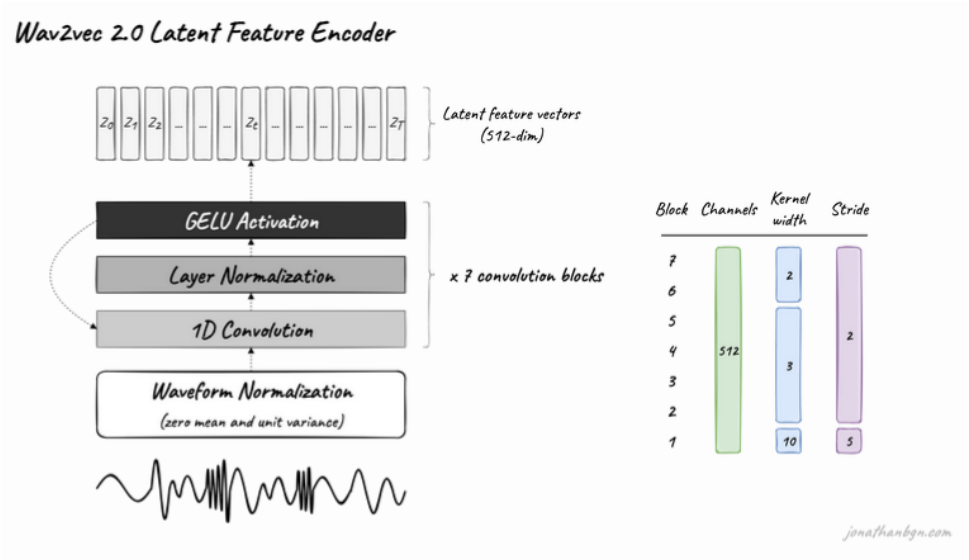


Figura 2.2: Arhitectura componentei Latent Feature Encoder [1]

Context Network

Componenta Context Network este un encoder de tip Transformer care primește ca intrare caracteristicile latente extrase de componenta Latent Feature Encoder și le procesează pentru a obține o reprezentare contextuală a semnalului audio. Aducând aminte de arhitectura modelului anterior *wav2vec* [11], care folosea tot o rețea convoluțională la acest pas, ar părea că se aseamană cu componenta anterioară. Diferența constă în faptul că Latent Feature Encoder urmărește să reducă dimensiunea semnalului audio, în timp ce Context Network urmărește să înțeleagă un context mai larg al semnalului audio. 2.3

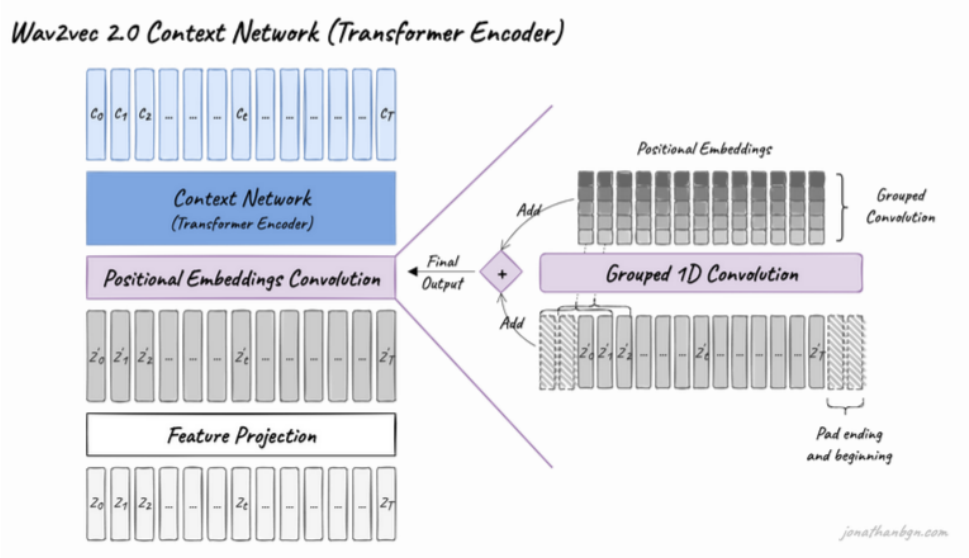


Figura 2.3: Arhitectura componentei Context Network [1]

Quantization Module

Deoarece modelul *wav2vec 2.0* folosește pentru partea de Context Network un encoder de tip Transformer, ne confruntăm cu problema structurii continue a semnalului audio. Limbajul scris poate fi discretizat într-un set finit de simboluri, în timp ce semnalul audio nu permite în mod direct acest lucru. Astfel, modelul *wav2vec 2.0* folosește un modul de cuantizare care învață automat unități de vorbire din semnalul audio. Intuitiv, se încearcă găsirea unor sunete fonetice finite și reprezentative pentru ieșirile din Latent Feature Encoder. De asemenea, se aplică funcția Gumbel Softmax [8], funcție diferențiabilă care permite antrenarea modelului prin backpropagation. 2.4

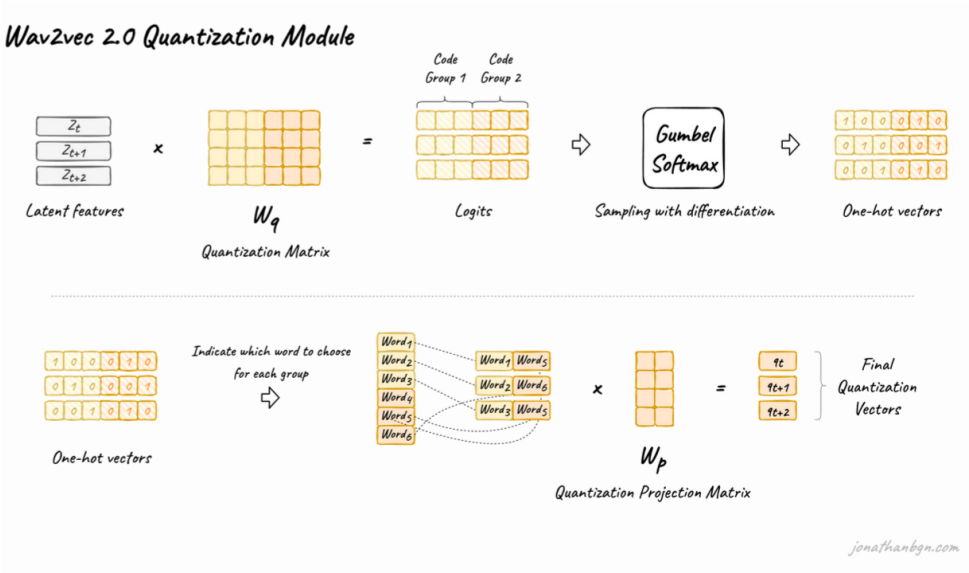


Figura 2.4: Arhitectura componentei Quantization Module [1]

Contrastive Loss

Pentru antrenarea modelului folosește o mască care ascunde 50% din vectorii proiectați din spațiul latent înainte să fie trecuți prin Context Network. Acest lucru forțează modelul să învețe reprezentări între vectorii proiectați și vectorii ascunși. Pentru fiecare poziție mascată, se aleg uniform aleator 100 de exemple negative de la alte poziții și se compară similaritatea cosinus între vectorul proiectat și vectorii aleși. Astfel, funcția de pierdere contrastivă încurajează similaritatea cu exemplele true positive și penalizează similaritatea cu exemplele false positive.

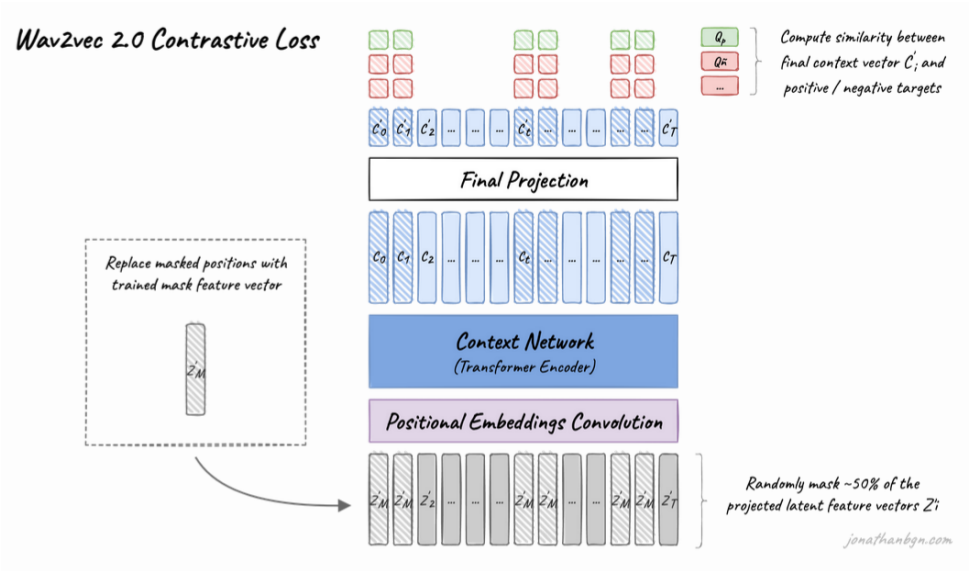


Figura 2.5: Arhitectura componentei Contrastive Loss [1]

2.1.2 Setul de date

Modelul oficial a fost preantrenat pe setul de date *LibriSpeech*, iar eu am continuat antrenarea pe seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1*.

Mini-LibriSpeech

Mini LibriSpeech este un subset al setului de date *LibriSpeech* care conține aproximativ 2 ore de înregistrări audio la o frecvență de eșantionare de 16 kHz. În medie, fiecare înregistrare are o durată de 6.72 secunde, cel mai lung audio având o durată de 31.5 secunde.

[1] Imaginile au fost preluate de pe site-ul lui Jonathan Bgn, "Illustrated Wav2Vec 2.0", disponibil la: https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html.

Common Voice Delta Segment 16.1

Common Voice Delta Segment 16.1 este un subset al setului de date *Common Voice* care conține aproximativ 2 ore de înregistrări audio la o frecvență de eșantionare de 48 kHz. A fost nevoie să reduc frecvența de eșantionare la 16 kHz pentru a putea folosi aceste date la antrenarea modelului. În medie, fiecare înregistrare are o durată de 5.63 secunde, cel mai lung audio având o durată de 10.47 secunde.

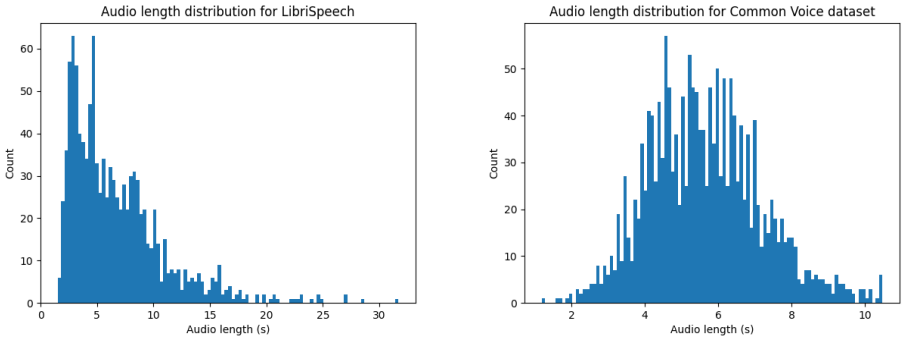


Figura 2.6: Distribuția duratelor audio-urilor din seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1*

Concluzie

Menționez aceste detalii deoarece pentru generarea subtitrărilor vom avea nevoie de audio-uri mult mai lungi decât cele folosite pentru antrenare care nu ar încăpea în memorie. Astfel, va trebui să folosim o tehnică de segmentare a audio-urilor în bucăți mai mici pentru a putea procesa audio-urile mai lungi. Mai multe detalii despre această tehnică vor fi prezentate în secțiunea *Subtitrări*.

2.1.3 Antrenarea modelului

Pentru antrenarea modelului am folosit limbajul de programare Python și biblioteca Hugging Face care pune la dispoziție o serie de pachete precum *transformers* și *datasets*.

Hiperparametrii folosiți pentru fine-tuning-ul modelului sunt:

- **Learning rate** - pentru a controla cât de mult se modifică gradienții în timpul antrenării
- **Weight decay** - pentru a controla cât de mult se penalizează valorile mari ale parametrilor
- **Warmup steps** - pentru a controla cât de mult se modifică learning rate-ul în primele pași ai antrenării

- **Batch size** - câte exemple se procesează în același timp

Hiperparametru	**Valoare**
Rata de învățare (*learning rate*)	1×10^{-4}
Descreșterea greutății (*weight decay*)	0.005
Pași de încălzire (*warmup steps*)	1000
Dimensiunea lotului (*batch size*)	4

Tabela 2.1: Hiperparametrii folosiți pentru fine-tuning-ul modelului *wav2vec2*

Am antrenat modelul pe seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1* pe o placă grafică NVIDIA Tesla V100 pusă la dispoziție de Google Colab. Am făcut un checkpoint la fiecare 500 de pași pentru a putea monitoriza evoluția modelului. Rezultatele checkpoint-urilor pentru modelul *facebook/wav2vec2-base* sunt prezentate în cele două tabele de mai jos.

Tabela 2.2: Mini LibriSpeech

Step	**Train loss**	**Val loss**
500	3.840	3.099
1000	1.202	0.586
1500	0.360	0.352
2000	0.231	0.333
2500	0.163	0.357
3000	0.136	0.331
3500	0.114	0.369
4000	0.104	0.348
4500	0.094	0.335
5000	0.083	0.284
5500	0.078	0.332
6000	0.072	0.356
6500	0.069	0.393
7000	0.066	0.380

Tabela 2.3: Common Voice Delta 16.1

Step	**Train loss**	**Val loss**
500	3.919	3.236
1000	1.287	0.570
1500	0.368	0.400
2000	0.226	0.371
2500	0.166	0.402
3000	0.136	0.475
3500	0.116	0.445
4000	0.097	0.448
4500	0.096	0.404
5000	0.079	0.459
5500	0.080	0.400
6000	0.069	0.445
6500	0.073	0.421
7000	0.064	0.440

Note

Se observă că modelul a început să învețe destul de repede, scăzând pierderea de antrenare de la 3.840 la 0.066, respectiv de la 3.919 la 0.064 în doar 7000 de pași. Urmărind graficul, se observă fenomenul de *overfitting* care apare în jurul pașilor 5000-6000, motiv pentru care am ales să opresc antrenarea la 7000 de pași și sa folosesc checkpoint-ul de la pasul 5000, respectiv 5500.

2.1.4 Îmbunătățire cu n-gram language model

Pentru a îmbunătăți recunoașterea vorbirii, am folosit un n-gram language model care mărește performanța modelului de la **wer 4.2%** la **wer 2.9%**, aducând o îmbunătățire de **1.3%**.

N-gram Language Model

Un n-gram language model este un model statistic care estimează, în cazul nostru, probabilitatea apariției unui caracter având în vedere cele n-1 caractere anterioare. Modelul se bazează pe ipoteza Markov de ordinul n, conform căreia putem aproxima probabilitatea apariției unui caracter folosind doar ultimele n caractere. Formula de mai jos ilustreaza această idee.

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{m} P(w_i | w_1, w_2, \dots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) \quad (2.1)$$

Am folosit un context de 5 caractere și am antrenat modelul pe setul de date *Helsinki-NLP/europarl* [12] deoarece conține și texte în limba engleză și putem avea certitudinea că textele sunt corecte din punct de vedere gramatical. Cu ajutorul librăriei *KenLM* [7], am antrenat modelul de n-grame și apoi am creat un procesor specific Hugging Face pentru modelele *wav2vec 2.0*.

În mod normal, modelul *wav2vec 2.0* ia argumentul maxim din distribuția de probabilitate pentru a prezice caracterul următor. În schimb, cu ajutorul n-gram language model, aceste probabilități sunt alterate pentru a se apropia de limbajul natural.

2.2 Clasificarea videoclipurilor

Videoclip-urile încărcate pe platform vin însoțite de metadate precum: titlu, descriere și, folosind modelul prezentat anterior, subtitrări. Pentru o experiță mai personalizată, am ales să clasific videoclipurile în funcție de subiectul abordat în topicuri precum: politics, sport, entertainment tehnologie și afaceri. Am folosit un modelul de clasificare *BERT*, Bidirectional Encoder Representations from *Transformers*, dezvoltat de *Google* [4] pe care l-am antrenat pe setul de date *BBC News* [6].

2.2.1 Arhitectura modelului

Modelul *BERT* este un model de învățare profundă care are la bază partea de encoder a unui *Transformer* [13], scopul lui fiind de a înțelege contextul cuvintelor într-o propoziție. Modelul *BERT* vine în două variante: *BERT-base* și *BERT-large*, cele două diferă prin

numărul de blocuri de encoder (12 și 24), numărul de neuroni din stratul fully-connected (768 și 1024) și numărul de attention heads (12 și 16), dar conceptual sunt la fel.

Spre deosebire de arhitectura standard a unui *Transformer*, *BERT* adaugă un token special la începutul fiecărei propoziții, *[CLS]*, folosit pentru clasificare. Intuitiv, acest token va reține informații despre întreaga propoziție și va putea fi folosit pentru rețeaua de clasificare care va determina topicul propoziției. 2.7

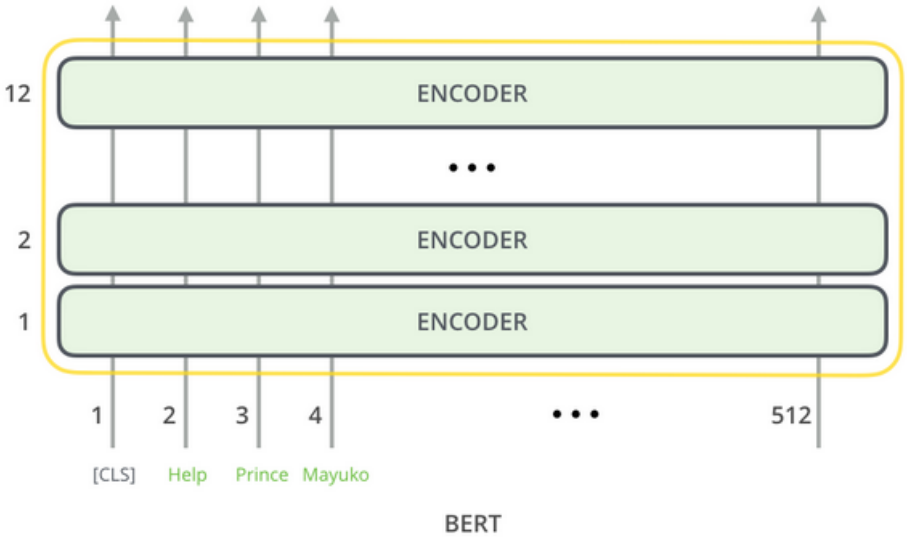


Figura 2.7: Arhitectura modelului *BERT* [2]

Ieșirile modelului *BERT* au dimensiunea 768, iar pentru partea de clasificare sunt trecute prin un strat fully-connected cu 5 neuroni, câte unul pentru fiecare clasă. Cu ajutorul funcției *softmax*, se obține o distribuție de probabilitate peste cele 5 clase, iar clasa cu cea mai mare probabilitate este cea aleasă.

2.2.2 Setul de date

Pentru antrenarea modelului am folosit setul de date *BBC News* care conține 2225 de articole între anii 2004-2005. Setul de date este împărțit în 5 clase: politics, sport, entertainment, technology și business. Mai jos am prezentat distribuția datelor din setul de date din punct de vedere al exemplelor pe clasă și al lungimii medii a articolelor. 2.8

[2]Imaginea a fost preluată de pe site-ul lui Jay Alammar, "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)", disponibil la: https://jalammar.github.io/illustrated-bert/.

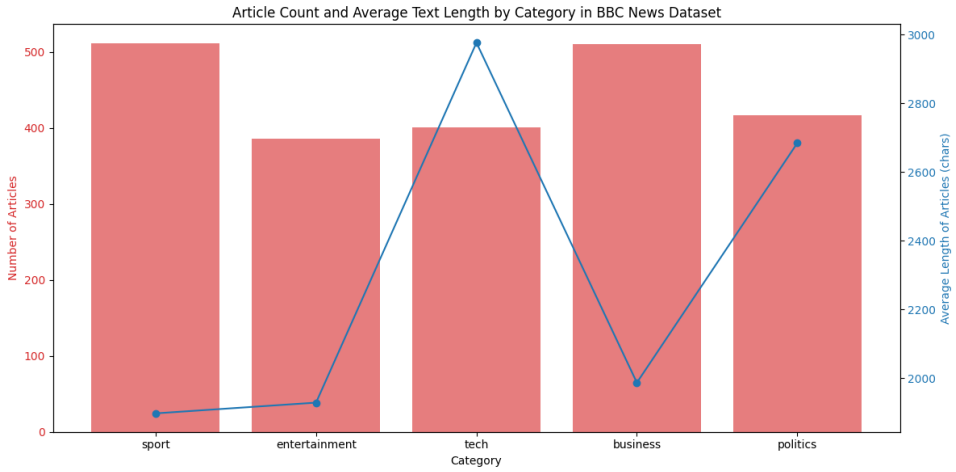


Figura 2.8: Distribuția claselor din setul de date *BBC News*

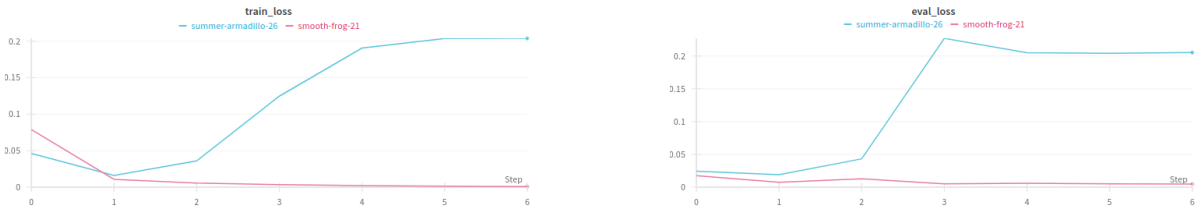
2.2.3 Antrenarea modelului

Antrenarea modelului a fost făcută folosind limbajul de programare Python, biblioteca PyTorch. [10] și plecând de la modelul preantrenat *bert-base-uncased* pus la dispoziție de Hugging Face. Codul a fost structurat în:

- **dataset.py** - creează un Dataset specific PyTorch pentru setul de date *BBC News*, implementând metodele *___len___* și *___getitem___*; metoda *___getitem___* aplică tokenizer-ul pe text cu lungimea maximă de 512 token-uri și returnează un tuplu de input_ids, attention_mask și label
- **model.py** - definește arhitectura modelului *BERT* și a rețelei de clasificare: inițializează modelul preantrenat *bert-base-uncased*, extrage token-ul special *[CLS]* și adaugă un strat liniar cu 5 neuroni pentru clasificare
- **trainer.py** - antrenează modelul pe setul de date *BBC News* folosind hiperparametrii din primiți ca parametru în linia de comandă și salvează modelul de fiecare dată când obține o acuratețe mai bună pe setul de validare

În figurile de mai jos 2.9 sunt ilustrate pierderile de antrenare și de evaluare pentru modelul *BERT* folosind ca rată de învățare 1×10^{-4} și 1×10^{-5}. Se observă fenomenul de *overfitting* pentru rata de învățare 1×10^{-4}, motiv pentru care am ales să experimentez cu valori mai mici.

Am încercat mai multe experimente plecând de la aceeași hiperparametri, cu rata de învățare 1×10^{-5}, pentru a vedea cum se comportă modelul la diferite inițializări ale ponderilor. În figura 2.10 sunt ilustrate valorile obținute pentru acuratețea modelului pe setul de validare.



(a) Training Loss (b) Evaluation Loss

Figura 2.9: Training and Evaluation Loss Graphs

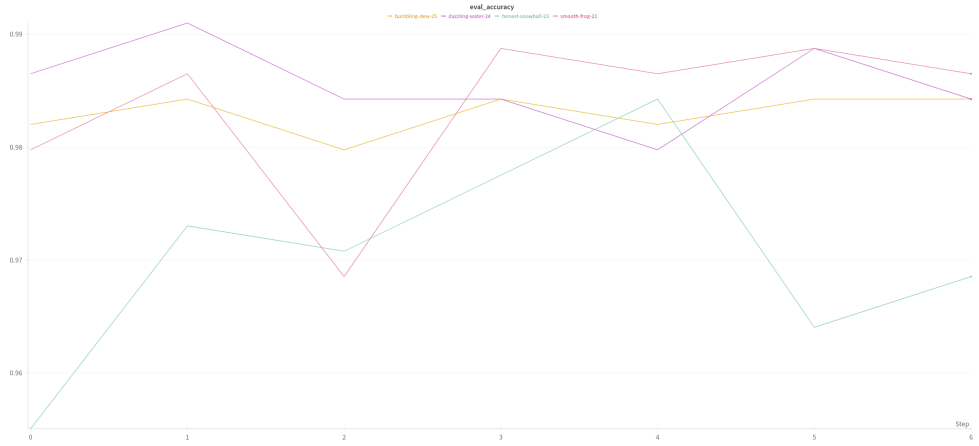


Figura 2.10: Evaluation Accuracy Graph

Hiperparametrii

Pentru modelul final pe care l-am folosit pentru clasificarea videoclipurilor am ales următorii hiperparametrii:

Hiperparametru	**Valoare**
Rata de învățare *(learning rate)*	0.00001
Dimensiunea lotului *(batch size)*	8
Optimizator *(optimizer)*	Adam
Funcția de pierdere *(loss function)*	CrossEntropyLoss

Tabela 2.4: Hiperparametrii modelului *BERT*

2.3 Concluzie

Cele două modele prezentate anterior, *wav2vec 2.0* și *BERT*, au fost încărcate pe platforma Huggingface și sunt disponibile în pachetul *transformers*.

wav2vec 2.0

Utilizarea modelului de recunoaștere a vorbirii necesită folosirea modulului *Wav2Vec2ForCTC* și a modulului *Wav2Vec2Processor* sau *Wav2Vec2ProcessorWithLM* pentru a folosi varianta îmbunătățită cu n-grame. Codul de mai jos ilustrează cum se pot folosi modelele prezentate anterior.

```
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor,
Wav2Vec2ProcessorWithLM

asr_repos = {
    "minilibrispeech": "3funnn/wav2vec2-base-minilibrispeech",
    "common-voice": "3funnn/wav2vec2-base-common-voice",
}
lm_boosted_repos = {
    "minilibrispeech": "3funnn/wav2vec2-base-minilibrispeech-lm",
    "common-voice": "3funnn/wav2vec2-base-common-voice-lm",
}

repo_name = "minilibrispeech"
boost_lm = True
model = Wav2Vec2ForCTC.from_pretrained(asr_repos[repo_name])

if boost_lm:
    processor = Wav2Vec2ProcessorWithLM.from_pretrained(
lm_boosted_repos[repo_name])
else:
    processor = Wav2Vec2Processor.from_pretrained(asr_repos[repo_name])

```

BERT

Modelul de clasificare a videoclipurilor este disponibil în pachetul *transformers* sub numele *3funnn/bert-topic-classification*. Modelul primește ca intrare un text care poate fi o concatenare a titlului și descrierii videoclipului și returnează o distribuție de probabilitate pentru fiecare din cele 5 clase. Huggingface pune la dispoziție funcția *pipeline* care permite folosirea modelului într-un mod simplu. Codul de mai jos ilustreaza acest lucru.

```
from transformers import pipeline

classifier = pipeline("text-classification", model="3funnn/bert-topic-
classification")

text = "This is a video about the latest technology in AI."
result = classifier(text)
print(result)
# Output: {'label': 'tech', 'score': 0.897}
```

Capitolul 3

Inginerie software

3.1 Frontend

3.2 Backend

3.2.1 Server

3.2.2 Recunoașterea vorbirii

3.2.3 Clasificarea videoclipurilor

3.3 Baze de date

3.3.1 MongoDB

3.3.2 Elasticsearch

Bibliografie

[1] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers și Gregor Weber, *Common Voice: A Massively-Multilingual Speech Corpus*, 2020, arXiv: 1912.06670 [cs.CL].

[2] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed și Michael Auli, *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*, 2020, arXiv: 2006.11477 [cs.CL].

[3] Pamela Bump, „How Video Consumption is Changing in 2023", în *HubSpot* (Apr. 2023), URL: https://blog.hubspot.com/marketing/how-video-consumption-is-changing.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee și Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019, arXiv: 1810.04805 [cs.CL].

[5] Elasticsearch, URL: https://www.elastic.co/.

[6] Derek Greene și Pádraig Cunningham, „Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", *Proc. 23rd International Conference on Machine learning (ICML'06)*, ACM Press, 2006, pp. 377–384.

[7] Kenneth Heafield, „KenLM: Faster and Smaller Language Model Queries", *Proceedings of the Sixth Workshop on Statistical Machine Translation*, ed. de Chris Callison-Burch, Philipp Koehn, Christof Monz și Omar F. Zaidan, Edinburgh, Scotland: Association for Computational Linguistics, Iul. 2011, pp. 187–197, URL: https://aclanthology.org/W11-2123.

[8] Eric Jang, Shixiang Gu și Ben Poole, *Categorical Reparameterization with Gumbel-Softmax*, 2017, arXiv: 1611.01144 [stat.ML].

[9] Vassil Panayotov, Guoguo Chen, Daniel Povey și Sanjeev Khudanpur, „Librispeech: An ASR corpus based on public domain audio books" (2015), pp. 5206–5210, DOI: 10.1109/ICASSP.2015.7178964.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai și Soumith Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019, arXiv: 1912.01703 [cs.LG].

[11] Steffen Schneider, Alexei Baevski, Ronan Collobert și Michael Auli, *wav2vec: Unsupervised Pre-training for Speech Recognition*, 2019, arXiv: 1904.05862 [cs.CL].

[12] Jörg Tiedemann, „Parallel Data, Tools and Interfaces in OPUS", *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, ed. de Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk și Stelios Piperidis, Istanbul, Turkey: European Language Resources Association (ELRA), Mai 2012, pp. 2214–2218, URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser și Illia Polosukhin, *Attention Is All You Need*, 2023, arXiv: 1706.03762 [cs.CL].