Recunoaștere de cuvinte scurte

Georgiana- Emanuela Dura
Universitatea Tehnică din Cluj-Napoca
durageorgiana@yahoo.com

Abstract— **Clasificarea automată a textului este o sarcină importantă în domeniul procesării limbajului natural (PLN), cu o gamă largă de aplicații. Recunoașterea de cuvinte scurte are o multitudine de aplicații de la sisteme de monitorizare, interfață om-mașină, totodată are aplicabilități și în robotică pentru Interpretarea comenzilor simple pentru controlul roboților sau a altor sisteme automate. Acest tip de proiect mai poate fi folosit si pentru recunoașterea comenzilor jucătorilor din jocurile bazate pe text. Spre deosebire de recunoașterea continuă a vorbirii, recunoașterea cuvintelor scurte se concentrează pe identificarea rapidă și precisă a cuvintelor sau frazelor scurte predefinite.**

Cuvinte cheie—***CNN, Recunoaștere de cuvinte scurte, Warden, 2018***

I. INTRODUCERE

Apariția *machine learning* a revoluționat modul în care semnalele audio sunt prelucrate. În trecut, aceste prelucrări se bazau pe algoritmi matematici și statistici, dar noile metode au făcut ca abordarea problemelor să fie mult mai flexibilă și precisă.

În prelucrarea semnalelor audio, *machine learning* este folosit pentru a extrage caracteristici precum amplitudinea și frecvența, care sunt apoi utilizate pentru a antrena diverse modele. Acest proiect este dedicat dezvoltării unui model de recunoaștere automată a vorbirii (ASR) pentru a identifica zece cuvinte scurte diferite. Obiectivul principal este de a procesa semnalele audio și de a antrena un model capabil să distingă între aceste cuvinte. Printre cuvintele de interes se numără: "no", "yes", "stop", "left", "right", "go", "down", "up". Aplicațiile sunt variate cum ar fi asistenții virtuali, dispozitive smart home, și diverse aplicații mobile, unde utilizatorii pot controla dispozitivele prin comenzi vocale simple. Printre altele, recunoașterea sunetului include exemple precum lătratul câinilor, clasificare muzică/ vorbire, bebelușii care plâng, detecție de emoții etc. Este o tehnologie cheie care va fi încorporată în majoritatea dispozitivelor care oferă capabilități de inteligență artificială. De exemplu, în prezent toată lumea are telefoane inteligente cu asistenți mobili precum *Google Assistant*, Amazon Alexa sau Apple Siri. Aceste aplicați domină și invadează interacțiunile umane. Sistemele controlate prin voce au fost, de asemenea, utilizate cu succes în educația și medicina pentru persoanele cu handicap sau nevăzători [1]. Aceste modele sunt dezvoltate prin intermediul rețelelor neuronale convoluționale, care au demonstrat o performanță superioară metodelor tradiționale.

II. METODE UTILIZATE PENTRU RECUNOAȘTEREA DE CUVINTE SCURTE

A. *Întrebuințării*

Recunoașterea de cuvinte scurte este un subiect destul de necesar și discutat în prezent. Deoarece aceasta automatizare a sistemelor de recunoaștere automată a vorbirii. De exemplu o bună întrebuințare a acestor sisteme ar fi la jocurile video. Pentru recunoașterea cuvintelor scurte, pe lângă rețelele neuronale convoluționale (CNN), [1] se pot implementa și alte metode avansate de *machine learning* și tehnici de prelucrare a semnalelor. Iată câteva dintre cele mai relevante: În loc de utilizarea unui controler tradițional sau a unei tastaturi, jucătorii pot folosi comenzi vocale scurte pentru a interacționa cu jocul. De exemplu:

- Navigare în meniuri: Jucătorii pot spune „start”, „stop” , „yes”, „no” pentru a naviga prin meniuri.
- Comenzi în joc: Comenzi precum „go”, „up”, „down” pot fi folosite pentru acțiuni rapide și precise în timpul jocului.

B. *Metode posibile*

1) *Rețele Neuronale Convoluționale (CNN)*

Rețelele neuronale convoluționale sunt frecvent utilizate pentru extragerea caracteristicilor din semnale audio. Aceste rețele procesează spectrogramele generate din semnale audio și identifică trăsături distinctive care pot diferenția cuvintele scurte. CNN-urile sunt eficiente în captarea caracteristicilor locale din datele de intrare și sunt bine adaptate pentru clasificarea imaginilor și, prin extensie, a spectrogramelor audio. Altfel spus Rețeaua CNN (rețea neuronală convoluțională) este inspirată de modul în care funcționează ochii noștri. Ne imaginăm că analizăm o poză cu o pisică. Ochii nu văd imaginea dintr-odată, ci scanează zone mici din ea pentru a identifica margini, colțuri și alte detalii simple. O rețea CNN analizează o imagine prin divizarea ei în zone mai mici și aplicarea de filtre specializate pentru a identifica caracteristici specifice, precum margini sau texturi.

Filtrul acesta se plimbă pe toată imaginea, analizând fiecare zonă.

În Figura 1 este reprezentată o rețea CNN de dimensiuni reduse.

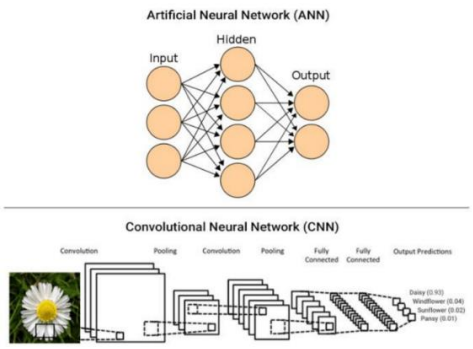

Figura 1. Rețea CNN [1]

Un exemplu de utilizare a CNN pentru recunoașterea de cuvinte scurte este propus de autorii articolului Speech-music discrimination using deep visual feature extractors [2]. Chiar daca numele articolului ne spune că este vorba despre discriminare dintre muzica si vorbire, aceleași metode și principii se folosesc si pentru recunoașterea de cuvinte scurte. Ei propun o metodă inovatoare în care semnalele audio sunt reprezentate sub formă de spectrograme și sunt tratate ca imagini. Aceste spectrograme sunt apoi procesate folosind extractoare de caracteristici vizuale adânci, cum ar fi rețelele neuronale convoluționale (CNN), pentru a extrage caracteristici semnificative din semnalele audio [1].

2)Rețele Neuronale Recurente (RNN) și LSTM

Rețelele neuronale recurente, în special cele bazate pe arhitectura Long Short-Term Memory (LSTM), sunt capabile să captureze dependențele temporale în datele audio. Aceasta le permite să rețină informații relevante de-a lungul secvențelor de timp, fiind astfel foarte potrivite pentru recunoașterea cuvintelor scurte care pot apărea în contexte variate. LSTM-urile evită problema gradientului exploziv sau disipativ, permițând rețelei să învețe pe termen lung. La acest tip de rețele ordinea în care sunt introduse datele contează foarte mult deoarece daca datele ar fi inversate am obține un rezultat eronat. De exemplu daca spunem că astăzi ploua, iar mâine va fi soare, nu este tot una cu mâine ploua și astăzi va fi soare. Astfel că daca nu introducem datele în ordinea corectă, calculatorul v-a învăța greșit.

Rețelele neuronale recurente funcționează pe principiul următor: ieșirea fiecărui strat este salvată și reintrodusă în intrarea sistemului pentru a prezice ieșirea acelui strat. Celulele recurente sunt folosite de către RNN pentru pentru a memora informații anterioare și a le integra în procesarea informațiilor curente. Astfel, rețelele pot detecta modele temporale și dependențe între cadrele temporale succesive ale semnalului audio [3].

Pentru a clasifica cele 8 tipuri de sunete, rețeaua poate fi antrenată pe un set de date care conține semnale audio cu cele 8 clase, etichetate corespunzător. Rețeaua RNN învață apoi să recunoască și să clasifice aceste tipuri de semnale pe baza caracteristicilor temporale și secvențiale. Arhitectura specifică utilizată poate fi LSTM (Long Short-Term Memory), care este proiectată pentru a evita problema gradientului care apare în RNN tradiționale și pentru a reține informații pe termen lung.

LSTM permite o clasificare mai precisă deoarece este capabilă să memoreze și să utilizeze informații relevante din trecut în procesarea curentă a semnalului audio.

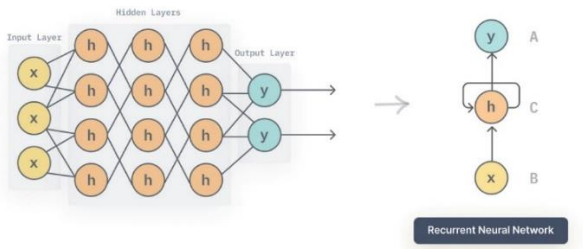

Figura 2. Rețea neuronală recurentă [3]

În Figura 2 este reprezentat principiul unei rețele neuronale recurente.

C. Descrierea setului de date și procesare

Pentru realizarea acestui proiect am folosit un set de date, care conține 8000 de fișiere audio, care sunt împărțite in cele 8 clase: "no", "yes", "stop", "left", "right", "go", "down", "up". Fiecare clasa este denumită în funcție de numele fișierului pe care îl conține. În fiecare clasă se află câte 1000 de înregistrări în formatul .wav. Acestea sunt fișiere audio de 16000 Hz pe 32 biți. Sarcina acestui algoritm este de a prezice eticheta corectă. .

III. IMPLEMENTARE ALGORITM

Acest algoritm este implementat pe baza unei rețele neuronale CNN. O rețea neuronală este un model matematic inspirat de funcționarea creierului uman, care este utilizat pentru a învăța și a face predicții pe baza datelor de intrare. Este compusă dintr-o colecție de noduri numite neuroni, care sunt conectați în straturi și prelucrează informațiile în mod paralel.

Rețelele neuronale convoluționale (CNN) sunt un tip de rețele neuronale specializate în prelucrarea datelor cu o structură de tip matrice, cum ar fi imagini sau semnale audio. Ele sunt denumite astfel deoarece utilizează operația de convoluție pentru a extrage caracteristici relevante din datele de intrare. Metoda implementată în lucrarea de față se bazează pe antrenarea unei rețele neuronale convoluționale, folosind ca date de intrare spectrograme audio. Setul de date utilizat pentru antrenarea, validarea și testarea rețelei conține 8000 fișiere audio, fiecare având o lungime de lungime 30 secunde. Toate acestea sunt fișiere audio de 16000 Hz pe 32 biți în format .wav.

Implementarea algoritmului pentru discriminarea între fișierele de muzică și vorbire cuprinde patru etape:

A. Preprocesarea datelor de intrare

Din cele 8000 de fișiere, 6400 (80%) vor fi folosite pentru antrenarea modelului. Acestea sunt datele cele mai importante, deoarece modelul învață din ele cum să recunoască diferitele comenzi, 1600 (20%) de fișiere vor fi folosite pentru validare. Datele de validare nu sunt folosite direct la antrenament, ci pentru a evalua performanța modelului pe măsură ce acesta învață. Acest lucru ajută la prevenirea supra-antrenamentului, care apare când modelul învață prea bine datele de antrenament și nu se generalizează bine la date noi.

De fiecare dată, datele se amestecă aleator în lista rezultată, folosind funcția "shuffle" din TensorFlow.

Plotez cateva forme de undă, care sunt luate aleator din setul de antrenare.

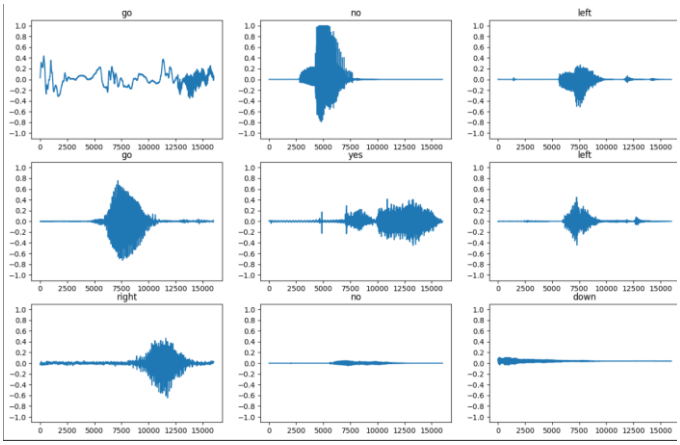

Figura 3. Secvențe audio de vorbire

În figura 3 sunt reprezentate 9 semnale sonore luate aleator din setul de antrenament. În cod aceste date pot fi și ascultate. Apoi am realizat o funcție pentru a putea calcula și afișa spectrogramele semnalelor audio.

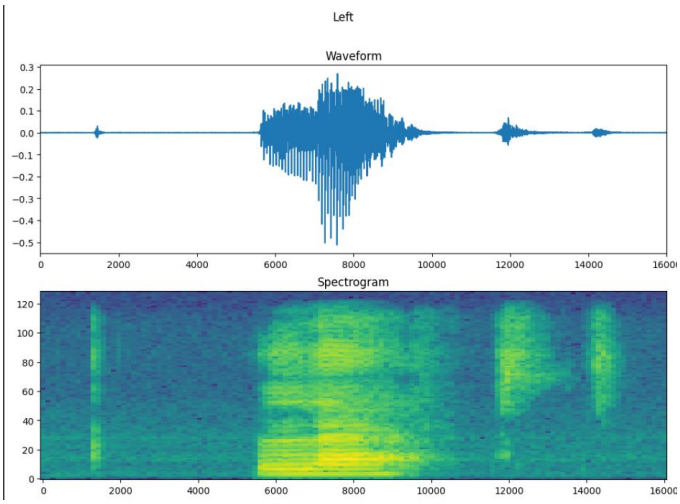

Figura 4. Secvență audio de vorbire și spectrograma ei

În Figura 4 este reprezentat semnalul audio (Waveform) al secvenței "left" și spectrograma (spectrogram) aferentă acestuia.

Imaginea prezentată în Figura 4 conține două grafice: forma de undă a semnalului audio și spectrograma acestuia. Graficul de sus, forma de undă, arată variația amplitudinii semnalului în timp, indicând o intensitate mare între 4000 și 10000 de eșantioane, sugerând prezența unui sunet puternic, posibil vorbire. Graficul de jos, spectrograma, detaliază cum frecvențele semnalului variază în timp, unde culorile mai deschise indică intensități mai mari. În spectrogramă, se observă frecvențe active între 2000 și 10000 de eșantioane, predominant în intervalele joase și medii, în timp ce segmentele inițiale și finale arată intensități mai mici, sugerând zgomot de fundal sau tăcere.

În Figura 5 sunt reprezentate semnalele audio alese aleator, dar sub forma de spectrograme.

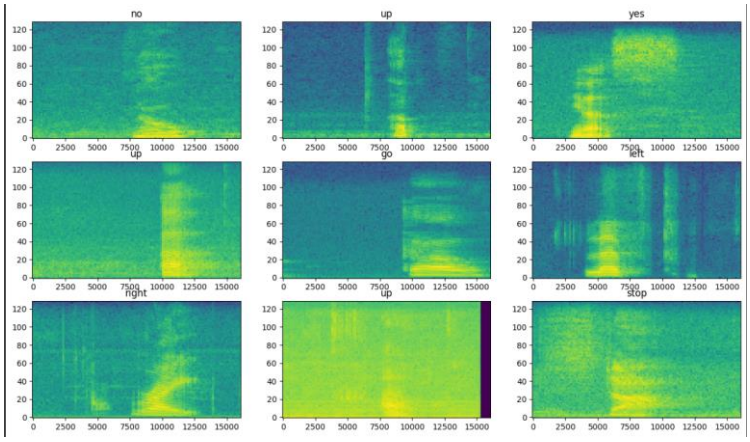

Figura 5. Spectrograme audio de vorbire

B. Construirea, compilarea și antrenarea modelului

Această etapă este una foarte important, deoarece dacă antrenarea nu se realizează corect, atunci și datele finale de la ieșire vor fi eronate.

Adaug operațiuni Dataset.cache și Dataset.prefetch pentru a reduce latența de citire în timp ce antrenez modelul. Pentru model, voi folosi o simplă rețea neuronală convoluțională (CNN), deoarece am transformat fișierele audio în imagini de spectrogramă.

Pentru model, voi folosi o simplă rețea neuronală convoluțională (CNN), deoarece am transformat fișierele audio în imagini de spectrogramă [5].

Modelul tf.keras.Sequential va folosi următoarele straturi de preprocesare Keras:

- tf.keras.layers.Resizing: pentru a reduce eșantionarea intrării pentru a permite modelului să se antreneze mai rapid.
- tf.keras.layers.Normalization: pentru a normaliza fiecare pixel din imagine pe baza mediei și a deviației standard.Sau altfel spus fiecare pixel să fie cuprins în intervalul [0,1)

Clasa *Sequential* din biblioteca open-source *Keras* include un numar de opt straturi diferite. Acest model este potrivit atunci când fluxul de date trece prin rețeaua neuronală într-o singură direcție, de la intrare la ieșire, fără conexiuni recurente. Este cel mai des utilizat pentru rețelele neuronale cu straturi dense (fully connected) și convoluționale[4].

Straturile utilizate în această lucrare sunt următoarele:

- **Input**: strat de intrare care specifică forma datelor de intrare în rețea.
- **Resizing:** redimensionează fiecare spectrogramă la o dimensiune de 64x64 pixeli. Este folosit pentru a asigura că toate imaginile de intrare au aceeași dimensiune.
- **Normalization:** normalizează datele de intrare. Se calculează media și deviația standard a datelor din setul de spectrograme, date care mai apoi sunt folosite pentru normalizarea datelor de intrare când se utilizează modelul.
- **Conv2D**: strat de convoluție 2D cu 32 și 64 de filtre de dimensiune 3x3. Funcția de activare utilizată este ReLU (Rectified Linear Unit). Acest strat aplică filtre pentru

a extrage caracteristici din imaginea de intrare, caracteristici care sunt salvate mai apoi într-o hartă de caracteristici.

○ **Stratul ReLU** aceasta setează toate valorile negative ale ieșirii la zero, iar valorile pozitive sunt păstrate nemodificate la ieșire. Eliminarea valorilor negative poate accelera antrenarea rețelei neuronale și îmbunătățirea performanțelor acesteia [4].

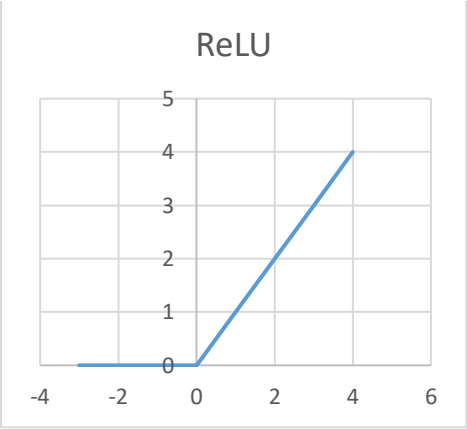

Figura 6. Funcția Relu

○ **MaxPooling2D:** reduce dimesiunea feature map-ului, ceea ce conduce la scurtarea timpului de antrenament. Glisează peste fereastra de intrare și ia valoarea maximă .

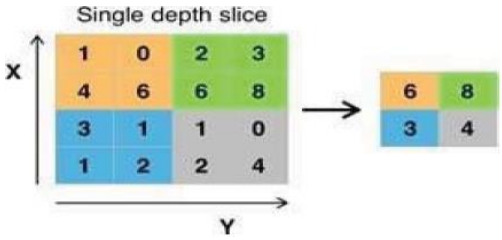

Figura 7. MaxPooling2D [4]

În figura 7 este reprezentat principiul pe care se bazează MaxPooling2D.

○ **Dropout:** anulează contribuția unor pixeli la următorul strat, cu o anumită probabilitate. Ajută la prevenirea overfitting-ului (supraantrenarea). Acest lucru este reprezentat si în Figura 8.

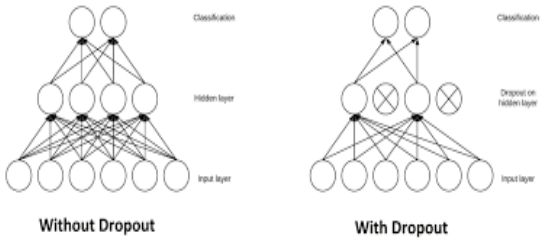

Figura 8. Principiul Dropout [4]

○ **Flatten**: tensorul multidimensional rezultat în urma straturilor convoluționale și MaxPooling este transformat într-un vector unidimensional, pentru a putea fi utilizat într-un strat dense (fully connected).

○ **Dense, cu funcție de activare ReLU:** strat dens (fully connected) cu 128 de neuroni și funcția de activare ReLU. Acest strat aplică operații liniare și funcții de activare pentru a combina caracteristicile extrase într-un vector de caracteristici mai abstract.

○ **Dense:** ultimul strat dens care produce ieșirea finală a rețelei, având un număr de neuroni egal cu numărul de clase sau etichete (8 etichete în cazul nostru). Nu este specificată o funcție de activare, deoarece se va utiliza una implicită. Funcția ne permite să obținem o distribuție de probabilități pentru clasele posibile, ceea ce este util în problemele de clasificare în care trebuie să alegem una din mai multe clase posibile [4].

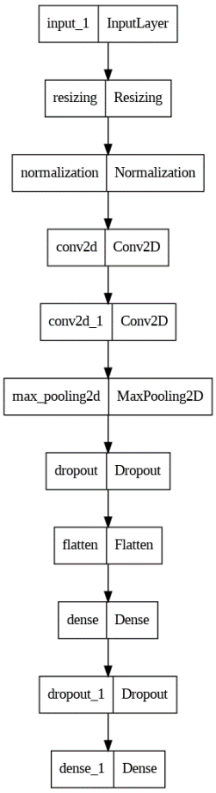

Figura 9. Arhitectura Modelului

În Figura 9, este reprezentată arhitectura modelului creat. Se poate observa că fiecare strat are 2 denumiri, cea din stânga reprezintă numele variabilei din codul scris, iar cea din dreapta este numele stratului.

După crearea modelului cu toate specificațiile de mai sus urmează configurare modelul Keras cu optimizatorul Adam și pierderea de entropie încrucișată, acest lucru se realizează prin functia *compile()*. Funcția configurează modelul pentru antrenament, definește cum se vor ajusta parametrii modelului (optimizator), măsoară performanța modelului (funcție de pierdere) și monitorizează acuratețea predicțiilor (metrici).

Algoritmul de optimizare utilizat este *Adam* (Adaptive Moment Estimation) din TensorFlow, care va fi folosit în timpul antrenării pentru ajustarea ponderilor rețelei neuronale. Utilizează informațiile de moment de ordinul întâi și de ordinul doi pentru a adapta rata de învățare în timpul

antrenării. Procesul de optimizare Adam implică actualizarea iterativă a parametrilor modelului pentru a minimiza funcția de cost.

Funcția de pierdere în acest caz este *SparseCategoricalCrossentropy* din TensorFlow și specifică măsura discrepanței dintre valorile prezise de model și valorile reale, atunci când avem sarcina de a clasifica între mai multe clase cu etichete specifice. Această funcție de cost calculează diferența dintre distribuția de probabilitate a claselor prezise de model și distribuția de probabilitate a claselor reale.

Metricile sunt utilizate pentru evaluarea performanței modelului în timpul antrenării și evaluării. În acest caz, se utilizează metrica accuracy, care calculează proporția corectă de exemple clasificate în raport cu totalul de exemple. Acuratețea este o măsură comună utilizată pentru evaluarea modelelor de clasificare și este interpretată ca fracțiunea de exemple clasificate corect.

Odată ce a fost compilat, modelul poate fi antrenat cu ajutorul metodei *fit()*. Metoda primește un set de date de antrenare care conține atât spectrogramele semnalelor audio cât și etichetele asociate acestora. Pentru acest proces de antrenare au fost suficiente 10 epoci. Antrenarea modelului se oprește dacă nu se obține o îmbunătățire semnificativă a performanței după numărul specificat de epoci consecutive, așa cum este definit în parametrul *patience* al obiectului *EarlyStopping*. Aceasta ajută la evitarea supraantrenării (overfitting) și permite selectarea celui mai bun model în funcție de performanță.

Este utilizat de asemenea și un set de date de validare pentru a evalua performanța modelului pe date noi în timpul antrenării.

După ce modelul este antrenat se trasează curbele de pierdere de antrenament și validare pentru a verifica cum s-a îmbunătățit modelul în timpul antrenamentului, acest lucru este reprezentat în Figura 9.

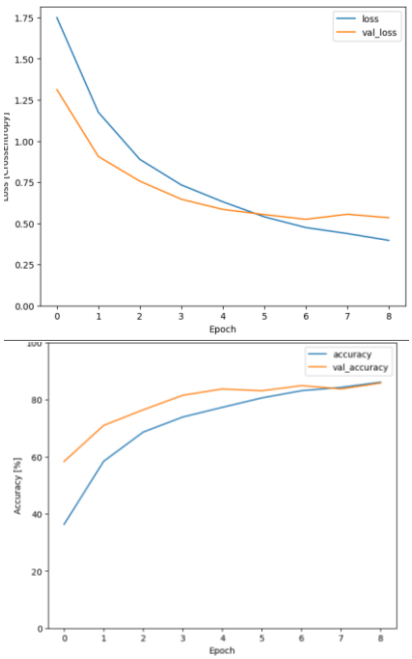

Figura 9. Acuratețea și pierderile modelului după antrenare

C. Evaluarea modelului pe un set de date de test

Evaluarea modelului se realizează prin model.evaluate acesta ia modelul antrenat deja și îl testează pe date noi (pe care nu le-a văzut niciodată la antrenament). Compară predicțiile modelului cu valorile reale și îți oferă un scor (de obicei acuratețe sau pierdere) pentru a vedea cât de bine generalizează modelul la date necunoscute.

D. Matricea de confuzie

O matrice de confuzie este un instrument esențial pentru evaluarea performanței unui model de clasificare. Aceasta este o matrice N x N (unde N este numărul de clase) care afișează numărul de predicții corecte și incorecte făcute de model comparativ cu valorile reale ale datelor testate. Fiecare rând al matricei reprezintă instanțele din clasa reală, iar fiecare coloană reprezintă instanțele din clasa prezisă.

Pentru că am 8 clase diferite "no", "yes", "stop", "left", "right", "go", "down", "up" utilizez o matrice de confuzie de dimensiune 8x8 pentru evaluarea performanței modelului. Fiecare rând al matricei reprezintă instanțele din clasa reală, iar fiecare coloană reprezintă instanțele din clasa prezisă de model.[5]

Matricea de confuzie pentru un model de clasificare cu 8 clase poate fi reprezentată prin Tabelul 1.

În tabelul cu numarul 1: TPi reprezintă numărul de instanțe corect clasificate pentru clasa i, iar FPi,j reprezintă numărul de instanțe din clasa reală i, incorect clasificate ca fiind din clasa *j*. Variabila i merge de la 1 la 8,reprezentând liniile matricei, iar variabila j merge de la 1 la 8, reprezentând coloanele matricei.

Clasa	Prezis 1	Prezis 2	Prezis 3	Prezis 4	Prezis 5	Prezis 6	Prezis 7	Prezis 8
Real 1	TP1	FP1,2	FP1,3	FP1,4	FP1,5	FP1,6	FP1,7	FP1,8
Real 2	FP2,1	TP2	FP2,3	FP2,4	FP2,5	FP2,6	FP2,7	FP2,8
Real 3	FP3,1	FP3,2	TP3	FP3,4	FP3,5	FP3,6	FP3,7	FP3,8
Real 4	FP4,1	FP4,2	FP4,3	TP4	FP4,5	FP4,6	FP4,7	FP4,8
Real 5	FP5,1	FP5,2	FP5,3	FP5,4	TP5	FP5,6	FP5,7	FP5,8
Real 6	FP6,1	FP6,2	FP6,3	FP6,4	FP6,5	TP6	FP6,7	FP6,8
Real 7	FP7,1	FP7,2	FP7,3	FP7,4	FP7,5	FP7,6	TP7	FP7,8
Real 8	FP8,1	FP8,2	FP8,3	FP8,4	FP8,5	FP8,6	FP8,7	TP8

Tabel 1. Matrice de confuzie 8x8

IV. Rezultate

În această secțiune sunt prezentate acuratețea si pierderea din timpul antrenării modelului, rezultatul evaluării performanței modelului pe datele de testare cât și rezultatul predicției modelului folosind matricea de confuzie.

A. Acuratețea și pierderea în timpul antrenării modelului

În urma evaluării acurateței și pierderii în timpul antrenării modelului, s-au obținut următoarele rezultate, așa cum se poate vedea în graficele de mai jos.

În Figura 10. sunt reprezentate valorile funcției de pierdere în fiecare epocă, atât pentru *Loss*, cât și pentru *val_loss*. Acestea reprezintă o măsură a erorii modelului în timpul antrenării. Valoarea *loss* (pierderea) este calculată pe baza diferenței dintre valorile prezise de model și valorile reale din setul de date de antrenare, iar *val_loss* se calculează pe baza datelor de validare.

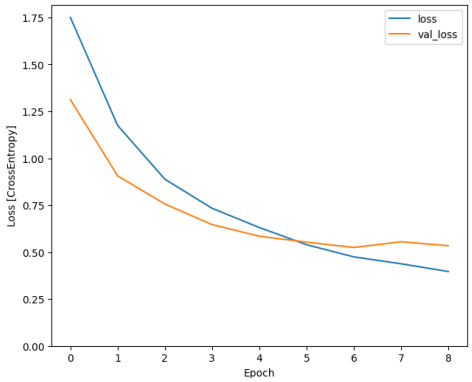

Figura 10. Pierderile de după antrenare

În Figura 10 prima linie (albastră) arată pierderea din antrenament (metrics['loss']) pe parcursul epocilor. A doua linie (portocalie) arată pierderea din validare (metrics['val_loss']) pe parcursul epocilor. Se observă ca aceasta scade pe parcursul procesului de antrenare și tinde spre 0% ceea ce reprezintă un lucru bun.

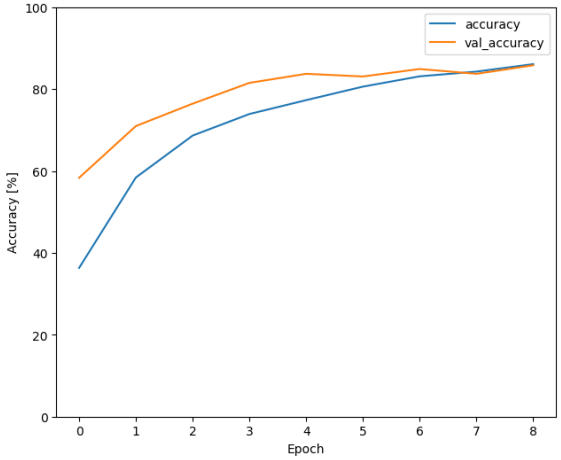

Figura 11. Acuratețea după antrenare

În Figura 11 prima linie (albastră) arată acuratețea din antrenament (metrics['accuracy']) pe parcursul epocilor. A doua linie (portocalie) arată pierderea din validare (metrics['val_accuracy']) pe parcursul epocilor. Se observă ca aceasta crește pe parcursul procesului de antrenare și tinde spre 100% ceea ce reprezintă un lucru bun.

Valoarea *accuracy (*acuratețea*)* este calculată ca raportul dintre numărul de predicții corecte și numărul total de exemple, iar *val_accuracy* reprezintă acuratețea modelului pe datele de validare.

Din graficul anterior se poate observa faptul că de la epocă la epocă valorile acurateței cresc, atingând 1.0000 [%]. Pierderile mici și valorile mari ale acurateței sunt principalii indicatori ai unui model bine antrenat, cum se întâmplă și în cazul modelului de față.

B. Evaluare performanță model

Evaluarea modelului se realizează prin model.evaluate acesta ia modelul antrenat deja și îl testează pe date noi (pe care nu le-a văzut niciodată la antrenament). Compară predicțiile modelului cu valorile reale și îți oferă un scor (de obicei acuratețe sau pierdere) pentru a vedea cât de bine generalizează modelul la date necunoscute. Astfel am obținut o pierdere de 49% și o acuratețe de 84%.

```
{'loss': 0.4951765835285187, 'accuracy': 0.8413461446762085}
```

C. Matrice de confuzie

După cum s-a explicat în secțiunea *Implementare*, matricea de confuzie este folosită pentru a verifica cât de bine a clasificat modelul cele 8 clase "no", "yes", "stop", "left", "right", "go", "down", "up" folosind setul de date de testare. În Figura 12. este o matrice de confuzie pentru un modelul de clasificare a cuvintelor scurte. Fiecare rând reprezintă eticheta reală, iar fiecare coloană reprezintă predicția modelului. Matricea arată cât de bine sau cât de prost a clasificat modelul diferitele cuvinte.

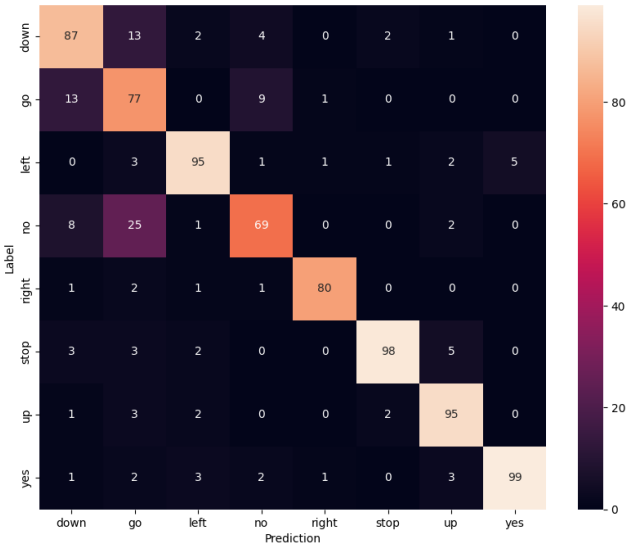

Figura 12. Matricea de confuzie obținută

Fiecare rând indică eticheta reală a datelor, iar fiecare coloană arată predicțiile făcute de model. Diagonala principală, de la stânga sus la dreapta jos, indică numărul de instanțe corect clasificate pentru fiecare cuvânt. De exemplu, modelul a clasificat corect 87 de instanțe de „down”, 77 de „go”, 95 de „left”, și așa mai departe.

Valorile din afara diagonalei principale reprezintă erorile de clasificare. De exemplu, modelul a clasificat greșit 13 instanțe de „down” ca „go” și 8 instanțe de „no” ca „go”. Aceste confuzii indică faptul că modelul are dificultăți în a distinge între anumite cuvinte, cum ar fi „go” și „no”, sau „down” și „go”.

Observăm că modelul are o performanță ridicată pentru cuvintele „left”, „stop”, „up” și „yes”, care au fost corect clasificate în peste 95% din cazuri. Pe de altă parte, cuvintele „no” și „down” sunt frecvent confundate cu „go”, ceea ce sugerează că modelul ar putea beneficia de mai multe date de antrenament sau de ajustări ale parametrilor pentru a îmbunătăți precizia în aceste cazuri.

Astfel, matricea de confuzie arată că modelul funcționează bine pentru majoritatea cuvintelor, dar există anumite perechi de cuvinte pentru care performanța poate fi îmbunătățită. Aceste informații sunt esențiale pentru a direcționa eforturile viitoare de optimizare a modelului, fie prin colectarea de mai multe date, fie prin ajustarea arhitecturii și parametrilor modelului.

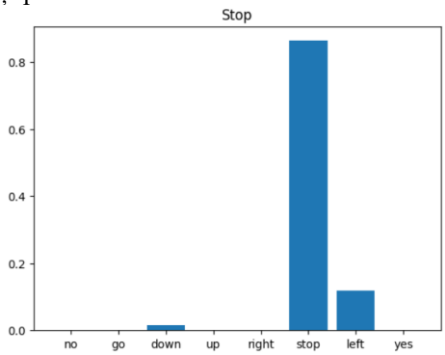

Figura 13. Interferența cu *stop*

Pentru a vedea cât de buna este funcționalitatea algoritmului se testează interferența între semnalele audio prezente în baza de date clasele: "no", "yes", "stop", "left", "right", "go", "down", "up".

După testele făcute se poate observa că în unele cazuri sunt prezente următoarele interferențe: stop cu left, go cu up, no cu go și up , right cu down. În funcție de nivelele de intensitate se poate observa cât de tare se inferferează semnalele, uneori mai mult, alteori mai puțin, acest lucru l-am representat vizual în figurile 13, 14, 15 , 16, 17.

În Figura 13 se poate observa că semnalul de bază *stop* în cea mai mare parte este clasificat corect (80%), dar are si mici interferențe cu *left* în prcente de aproximativ 15%, iar cu down mult mai puțin 5%.

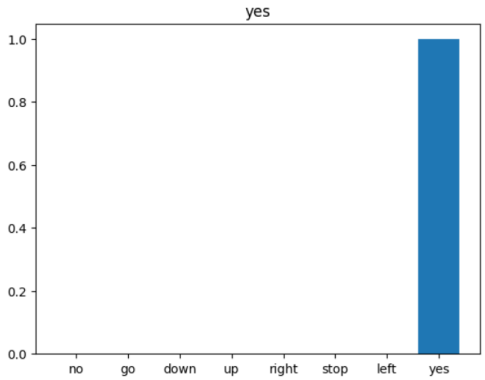

Figura 14. Interferența cu *yes*

În Figura 14 se poate observa că semnalul de bază *yes* este mereu clasificat corect (100%).

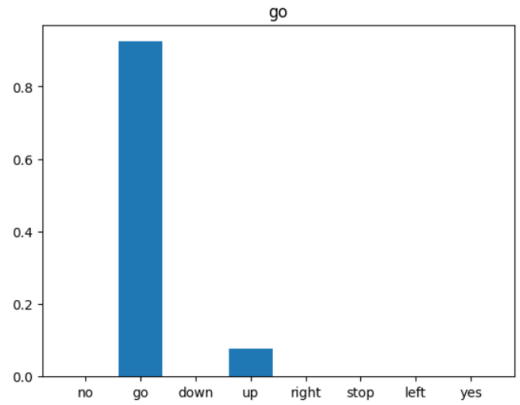

Figura 15. Interferența cu *go*

În Figura 15 se poate observa că semnalul de bază *go* în cea mai mare parte este clasificat corect (90%), dar are si mici interferențe cu *up* în prcente de aproximativ 10%.

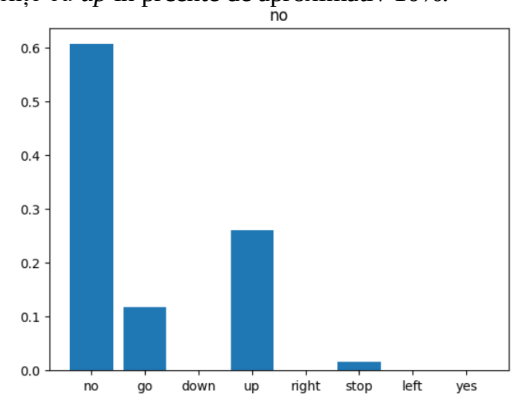

Figura 16. Interferența cu *no*

În Figura 15 se poate observa faptul că este cel mai defavorabil caz dacă se compară acest caz cu cele anterioare deoarece semnalul de bază *no* este clasificat corect doar în proporție de 60%, iar acesta mai are 3 interferențe cu go, up, stop. Cea mai mare interferență o are cu *up* (25%), cu go (10%), iar cu stop procentajul de interferență este foarte mic aproximativ 2%- 3%.

V. Concluzii

Proiectul de clasificare a cuvintelor scurte demonstrează o aplicare eficientă a tehnicilor de machine learning în recunoașterea vocală. Modelul dezvoltat a reușit să clasifice corect majoritatea cuvintelor scurte cu o precizie remarcabilă, ceea ce subliniază potențialul acestuia în aplicații reale. Matricea de confuzie indică faptul că, în general, modelul are performanțe excelente, cu rate de clasificare corectă de peste 95% pentru cuvinte precum „left”, „stop”, „up” și „yes”.

Cu toate acestea, anumite confuzii între cuvintele „go” și „no”, precum și între „down” și „go”, sugerează că există spațiu pentru îmbunătățiri. Aceste confuzii pot fi abordate prin colectarea de mai multe date de antrenament specifice acestor clase sau prin ajustarea parametrilor modelului și optimizarea arhitecturii acestuia.

În ansamblu, proiectul a demonstrat că utilizarea rețelelor neuronale convoluționale pentru clasificarea cuvintelor scurte este nu doar fezabilă, ci și eficientă, oferind o bază solidă pentru dezvoltări ulterioare. Implementarea

acestui model poate avea aplicații variate, de la sisteme de control vocal pentru jocuri și dispozitive inteligente, până la îmbunătățirea interfețelor utilizatorilor în diverse tehnologii. Continuitatea în optimizarea modelului și extinderea dataset-ului vor asigura performanțe și mai bune, consolidând utilitatea și aplicabilitatea soluției în diverse domenii.

VI. REFERINȚE BIBLIOGRAFICE.

[1] "Performance Analysis of Text Classification Algorithms using Confusion Matrix .pdf."

[2] "Papakostas and Giannakopoulos - 2018 - Speech-music discrimination using deep visual feat.pdf."

[3] "Solovyev et al. - 2020 - Deep Learning Approaches for Understanding Simple .pdf."

[4] "Ajit et al. - 2020 - A Review of Convolutional Neural Networks.pdf."

[5]URL: https://pytorch.org/

[6] URL: https://keras.io/

[7] URL: https://kaggle.com/