# Reconeixement de Comandaments de Veu (Speech Command Recognition)

**Roger Baiges Trilla i Pau Hidalgo Pujol**

TVD - GIA @ UPC

30 de Desembre de 2024

Aquest treball explora el disseny, implementació i avaluació de models d'aprenentatge profund per al reconeixement de comandes de veu. L'objectiu principal és classificar amb alta precisió gravacions d'àudio en un conjunt predefinit de comandes, emprant tècniques de preprocessament d'àudio com l'espectrograma, el MEL-espectrograma i MFCC, juntament amb xarxes neuronals convolucionals (CNN), recurrents (RNN), i models basats en atenció i transformers. A través d'una sèrie d'experiments, s'analitza l'impacte de diferents arquitectures, tècniques de normalització, regularització i augment de dades (SpecAugment) en el rendiment dels models. També s'analitzen els resultats de realitzar *transfer learning* amb models entrenats per àudio. Els resultats mostren que els models, entrenats suficientment, obtenen 97% en el conjunt de validació. L'estudi també demostra la importància de la normalització, la regularització, i l'ús de SpecAugment per evitar l'*overfitting* i millorar la capacitat de generalització dels models. El model final entrenat des de zero (un ResNet2 amb 1.6M paràmetres) assoleix una accuracy de **0.980** en el conjunt de test i realitzar un finetune de Hubert **0.986**, demostrant el potencial dels models.


## 1 Introducció

El reconeixement de comandes de veu és una àrea fonamental en el camp del tractament de la veu i la interacció home-màquina, amb aplicacions que van des d'assistents virtuals fins a sistemes domòtics. En aquest projecte, el nostre objectiu és dissenyar, implementar i avaluar models capaços de classificar amb alta precisió gravacions d'àudio en un conjunt predefinit de comandes, com ara "up", "left", "bird", "on" i "five", entre d'altres.
En aquest treball s'exploraran diferents tècniques i arquitectures, incloent xarxes convolucionals, xarxes recurrents i transformers, amb l'objectiu de determinar com poden contribuïr a la millora del rendiment en la tasca. A més, també es provaran diferents preprocessaments de la senyal d'àudio inicial (espectograma lineal, MEL i MFCC) i tècniques de normalització i regularització.
Aquest document descriu el procés seguit en la realització d'aquesta pràctica, detallant els models proposats, els resultats obtinguts i l'anàlisi d'aquests, així com les conclusions derivades del nostre treball. Al final, intentarem determinar quin és el millor model possible, basant-nos sobretot en l'*accuracy* obtinguda.

## 2 Experiments

En aquesta secció, detallarem els experiments inicials realitzats i els models provats. Es descriuran les arquitectures escollides, intentant justificar-les, i es presentaran les diferents proves realitzades així com la metodologia.

### 2.1 Espectrograma

Per poder realitzar els experiments correctament, ens calia determinar quines dades d'entrada utilitzar. Per tant, els primers tests van servir per escollir el *preprocessat* realitzat a l'àudio. Les diferents opcions escollides han sigut: espectrograma, espectrograma al quadrat (*power spectrogram)*, MEL-espectrograma amb 32, 64 i 128 bins i MFCC (*Mel-Frequency Cepstral Coefficients*) amb 13, 40 i 64 features (tant per MEL 64 com per 128).
Per provar el seu rendiment, vam utilitzar dos *baselines*:

- **Model *Small***:
  - Redimensionament de les dades d'entrada a 32x32. + Normalització (*norm_layer*).
  - Tres capes *Conv2D* amb 32, 64 i 128 filtres respectivament, cada una amb un kernel de 3x3, activació *ReLU* i padding *same*. MaxPooling2d després de cada una.
  - Una capa *Flatten* + capa densa (*Dense*) amb 64 unitats i activació *ReLU*.
- **Model *Big***: Model més gran per poder estudiar millor els efectes:
  - Redimensionament de les dades d'entrada a 64x64 + Normalització (*norm_layer*).
  - Quatre blocs convolucionals:
    * *Conv2D* amb 64, 128, 256 i 256 filtres respectivament, un kernel de 3x3, activació *ReLU* i padding *same*.
    * Una capa de *BatchNormalization*.
    * Una capa *MaxPooling2D*.
    * Una capa *Dropout* de 0.25 per als dos primers blocs i 0.3 per als dos darrers.
  - Una capa de *GlobalAveragePooling2D* + Dues capes denses (*Dense*) amb 256 i 128 neurones respectivament, amb activació *ReLU* i regularització L2 (*kernel_regularizer=regularizers.l2(0.01)*). Després de cada capa densa, s'aplica *Dropout* de 0.3.

## 2.2 Normalització

Un altre experiment que vam optar per realitzar aviat va ser analitzar l'efecte de la normalització. En general aquest procés ... i per aquest motiu vam decidir provar-lo aviat. D'aquesta manera, als models definits posteriorment podiem utilitzar-la.
En general, la majoria de models consten de dues parts: una primera de "comunicació" entre les features de l'espectrograma (per exemple, usant convolucionals) i una segona de còmput (usant denses). S'ha optat per estudiar els efectes de la normalització en 4 llocs diferents: a l'inici, entre convolucionals (comunicació), abans de les denses i entre les denses. S'han aplicat a un model molt similar al baseline *small*, però amb 16, 32 i 32 filtres i 64, 32 neurones a les denses.

## 2.3 Convolucionals

Una arquitectura que té basant sentit aplicar als espectrogrames són les xarxes convolucionals. Per aquest motiu hem realitzat un estudi bastant exhaustiu d'aquest tipus de models, provant bastantes combinacions diferents.
Per simplificar els experiments, tots els models s'han provat amb resize de 32 a l'inici i amb l'espectrograma que millors resultats va donar al primer experiment. A més, s'han entrenat durant 15 epochs amb l'optimizer Adam per defecte. Més al final, agafarem els millors models i intentarem obtenir millors resultats amb ells, fent-los més grans, aplicant millors tècniques...
A continuació, podem observar una taula amb les arquitectures provades 1.

| Model Name | Type | Architecture Details |
|---|---|---|
| **Conv1D Models** | conv1d_1 | 2 layers, filters: [16, 32], kernel size: [5, 3], strides: [1, 1], dense units: 64 |
| | conv1d_2 | 3 layers, filters: [32, 64, 128], kernel size: [7, 5, 3], strides: [1, 1, 2], dense units: 128 |
| | conv1d_3 | 4 layers, filters: [16, 32, 64, 32], kernel size: [9, 5, 3, 3], strides: [2, 1, 1, 1], dense units: 32 |
| | conv1d_4 | 3 layers, filters: [32, 64, 32], kernel size: [3, 5, 3], strides: [1, 2, 1], dense units: 128 |
| | conv1d_5 | 3 layers, filters: [16, 64, 16], kernel size: [5, 7, 5], strides: [1, 2, 1], dense units: 64 |
| **Conv2D Models** | conv2d_1 | 2 layers, filters: [16, 32], kernel size: [(3, 3), (3, 3)], strides: [(1, 1), (1, 1)], dense units: 64 |
| | conv2d_2 | 3 layers, filters: [32, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1)], dense units: 128 |
| | conv2d_3 | 4 layers, filters: [16, 32, 64, 128], kernel size: [(5, 5), (3, 3), (3, 3), (3, 3)], strides: [(2, 2), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_4 | 3 layers, filters: [32, 64, 32], kernel size: [(3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1)], dense units: 128 |
| | conv2d_5 | 3 layers, filters: [16, 64, 16], kernel size: [(3, 3), (5, 5), (3, 3)], strides: [(1, 1), (2, 2), (1, 1)], dense units: 128 |
| | conv2d_6 | 4 layers, filters: [128, 64, 32, 16], kernel size: [(5, 5), (3, 3), (3, 3), (3, 3)], strides: [(2, 2), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_7 | 4 layers, filters: [32, 32, 32, 32], kernel size: [(5, 5), (3, 3), (3, 3), (3, 3)], strides: [(2, 2), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_8 | 4 layers, filters: [16, 32, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_9 | 4 layers, filters: [16, 32, 64, 128], kernel size: [(7, 7), (5, 5), (3, 3), (3, 3)], strides: [(2, 2), (2, 2), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_10 | 5 layers, filters: [32, 32, 64, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| | conv2d_11 | 4 layers, filters: [32, 64, 128, 128], kernel size: [(3, 3), (3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1), (1, 1)], dense units: 64 |
| **Conv2D with Pooling / Normalization** | conv2d_3_avg | conv2d_3 + average pooling |
| | conv2d_7_avg | conv2d_7 + average pooling |
| | conv2d_8_avg | conv2d_8 + average pooling |
| | conv2d_11_avg | conv2d_11 + average pooling |
| | conv2d_3_norm | conv2d_3 + layer normalization |
| | conv2d_7_norm | conv2d_7 + layer normalization |
| | conv2d_8_norm | conv2d_8 + layer normalization |
| | conv2d_11_norm | conv2d_11 + layer normalization |
| **Mixed Models** | mixed_1 | 1 conv1d and 2 conv2d layers, kernel size: [(3), (3,3), (3,3)], dense units: 64 |
| | mixed_2 | 2 conv1d and 2 conv2d layers, kernel_size: [(5), (3), (3, 3), (3, 3)], dense units: 128 |
| | mixed_3 | 2 conv1d and 1 conv2d layers, kernel_size: [(5), (3), (3, 3)], dense units: 128 |
| | mixed_4 | 1 conv1d and 1 conv2d layers, kernel_size: [(5), (3, 3)], dense units: 64 |
| | mixed_5 | 3 conv1d and 1 conv2d layers,kernel_size: [(9), (5), (3), (3, 3)], dense units: 32 |
| **Residual Models** | resnet_1 | 2 layers, filters: [16, 32], kernel size: [(3, 3), (3, 3)], strides: [(1, 1), (1, 1)], residual: [T, T], dense units: 64 |
| | resnet_2 | 3 layers, filters: [32, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1)], residual: [T, F, T], dense units: 128 |
| | resnet_3 | 4 layers, filters: [16, 32, 64, 32], kernel size: [(5, 5), (3, 3), (3, 3), (3, 3)], strides: [(2, 2), (1, 1), (1, 1), (1, 1)], residual: [T, T, F, T], dense units: 32 |
| | resnet_4 | 3 layers, filters: [32, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3)], strides: [(1, 1), (2, 2), (1, 1)], residual: [T, T, T], dense units: 128 |
| | resnet_5 | 3 layers, filters: [16, 64, 16], kernel size: [(3, 3), (5, 5), (3, 3)], strides: [(1, 1), (2, 2), (1, 1)], residual: [F, T, F], dense units: 64 |
| | resnet_8 | 4 layers, filters: [16, 32, 64, 128], kernel size: [(3, 3), (3, 3), (3, 3), (3, 3)], strides: [(1, 1), (1, 1), (1, 1), (1, 1)], residual: [T, T, T, T],, dense units: 64 |
| **Residual2 Models** | resnetv2_1 | 4 layers, filters: [16, 32, 64, 128], dense units: 64 |
| | resnetv2_2 | 4 layers, filters: [32, 64, 128, 128], dense units: 64 |
| | resnetv2_3 | 4 layers, filters: [32, 64, 128, 128], dense units: 128 |
| | resnetv2_4 | 3 layers, filters: [32, 64, 128], dense units: 64 |
| | resnetv2_5 | 3 layers, filters: [32, 64, 128], dense units: 32 |
| | resnetv2_6 | 2 layers, filters: [16, 32], dense units: 64 |
| | resnetv2_7 | 1 layer, filters: [32], dense units: 64 |

Table 1: Arquitectures de CNN provades

Com pot observar-se, hem provat tant convolucionals 1d com 2D. A més, hem afegit alguns models que incorporen connexions residuals. Els models Residual poden tenir connexions entre cada una de les convolucions (si les dimensions no encaixen aplica una convolució 1x1) i utilitza un flatten al final. Els Residual2 utilitzen blocks amb dues convolucions per block (és a dir, el nombre de capes realment és el doble), així com LayerNormalitzation i GlobalAveragePooling2D.

## 2.4 Xarxes Recurrents

Les xarxes recurrents (RNN) també són molt utilitzades en aquest camp, permetent modelar dependències temporals.

| Model Name | Type | Architecture Details |
|---|---|---|
| **RNN Models** | rnn_1 | 1 layer, units: [32], dense units: 64 |
| | rnn_2 | 2 layers, units: [64, 32], dense units: 64 |
| | rnn_3 | 3 layers, units: [128, 64, 32], dense units: 64 |
| | rnn_4 | 2 layers, units: [64, 64], dense units: 128 |
| **LSTM Models** | lstm_1 | 1 layer, units: [32], dense units: 64 |
| | lstm_2 | 2 layers, units: [64, 32], dense units: 64 |
| | lstm_3 | 3 layers, units: [128, 64, 32], dense units: 64 |
| | lstm_4 | 2 layers, units: [64, 64], dense units: 128 |
| **GRU Models** | gru_1 | 1 layer, units: [32], dense units: 64 |
| | gru_2 | 2 layers, units: [64, 32], dense units: 64 |
| | gru_3 | 3 layers, units: [128, 64, 32], dense units: 64 |
| | gru_4 | 2 layers, units: [64, 64], dense units: 128 |
| **Bidirectional LSTM** | bilstm_1 | 1 layer, units: [32], dense units: 64 |
| | bilstm_2 | 2 layers, units: [64, 32], dense units: 64 |
| | bilstm_3 | 3 layers, units: [128, 64, 32], dense units: 64 |
| | bilstm_4 | 2 layers, units: [64, 64], dense units: 128 |
| **Bidirectional GRU** | bigru_1 | 1 layer, units: [32], dense units: 64 |
| | bigru_2 | 2 layers, units: [64, 32], dense units: 64 |
| | bigru_3 | 3 layers, units: [128, 64, 32], dense units: 64 |
| | bigru_4 | 2 layers, units: [64, 64], dense units: 128 |
| | bigru_5 | 2 layers, units: [128, 64], dense units: 32, dropout: 0.2 |
| **ConvLSTM2 Models** | convlstm2_1 | 1 layer, units: [64], filters: [32], kernel size: [(3, 3)], dense units: 64 |
| | convlstm2_2 | 2 layers, units: [64], filters: [64, 32], kernel size: [(3, 3), (3, 3)], dense units: 64 |
| | convlstm2_3 | 3 layers, units: [64], filters: [64, 32, 32], kernel size: [(3, 3), (3, 3), (3, 3)], dense units: 64 |
| **ConvGRU2 Models** | convgru2_1 | 1 layer, units: [64], filters: [32], kernel size: [(3, 3)], dense units: 64 |
| | convgru2_2 | 2 layers, units: [64], filters: [64, 32], kernel size: [(3, 3), (3, 3)], dense units: 64 |
| | convgru2_3 | 3 layers, units: [64], filters: [64, 32, 32], kernel size: [(3, 3), (3, 3), (3, 3)], dense units: 64 |
| **ConvBidirGRU2** | convbidirgru1_1 | 1 layer, units: [64], filters: [32], kernel size: [(3, 3)], dense units: 64 |
| | convbidirgru2_2 | 2 layers, units: [64], filters: [64, 32], kernel size: [(3, 3), (3, 3)], dense units: 64 |
| | convbidirgru2_3 | 3 layers, units: [64], filters: [64, 32, 32], kernel size: [(3, 3), (3, 3), (3, 3)], dense units: 64 |
| **ConvCRNN Models** | convcrnn_1 | 2 Conv2D layers (32), kernel: [(20, 20), (5, 5)], strides: [(8, 8), (2, 2)], 2 Bidirectional GRU layers (32 units), dense units: 64 |
| | convcrnn_2 | 2 Conv2D layers (16), kernel: [(3, 3), (5, 3)], strides: [(1, 1), (1, 1)], 1 GRU layer (256 units), dense units: 128 |
| | convcrnn_3 | 2 Conv2D layers (16), kernel: [(3, 3), (5, 3)], strides: [(1, 1), (1, 1)], 1 GRU layer (256 units), 2 dense layers [128, 256] |
| **Attention RNN** | mhatt_rnn_1 | units: [128], dense units: 64, num heads: 4, key dim: 64, ffn units: 128 |
| | mhatt_rnn_2 | units: [128], dense units: 64, num heads: 4, key dim: 64, ffn units: 128, dropout: 0.2 |
| **Multi-Head Attention RNN** | mhatt_rnn_1 | units: [128], dense units: 64, num heads: 4, key dim: 64, ffn units: 128 |
| | mhatt_rnn_2 | units: [128], dense units: 64, num heads: 4, key dim: 64, ffn units: 128, dropout: 0.2 |

Table 2: Arquitectures de RNN provades

En aquest cas, hem provat tant xarxes recurrents soles (LSTM, GRU; també bidireccionals) com combinant-les amb convolucions i attention 2. Els ConvCRNN estan inspirats en [1], att-rnn en [2] i mhatt-rnn en [3], així com les implementacions d'aquest últim paper a github [4].

## 2.5 Transformers

Una vegada ja experimentat amb xarxes convolucionals i recurrents, era hora de provar amb els models més poderosos que existeixen actualment, els *transformers*.

Es va començar amb una versió molt simplificada amb un únic bloc Transformer, amb un nombre reduït de paràmetres, per tal d'assegurar que el model pogués entrenar-se adequadament sense problemes de sobreajustament. Posteriorment, es va decidir augmentar el nombre de capes i la dimensió de *embedding*, passant a arquitectures més complexes com el Transformer Model 2. En aquest cas, per evitar problemes de memòria, es va decidir aplicar un *batch size* efectiu de 512 gràcies a la tècnica de *gradient accumulation*, amb un *batch size* inicial de 64 multiplicat per 8 passos acumulats.

Degut a les dificultats inherents dels transformers, com l'alt cost computacional i la seva tendència a requerir grans quantitats de dades per no sobreajustar-se o inclús si són massa complexes dificultats per aprendre i no quedar-se estancats, es va decidir explorar combinacions amb xarxes convolucionals. Aquestes convolucions s'utilitzen per reduir la dimensionalitat inicial de les dades i extreure característiques locals rellevants. Això simplifica l'entrada al transformer i redueix la càrrega computacional. Aquest enfocament és molt similar a un ViT (*Vision Transformer*), ja que, tot i no tenir *embeddings* de tokens, es treballa amb *patches* de posició i característiques extretes per convolucionals.

## 2.6 Taula de Paràmetres de les Arquitectures Transformer

| Model | Embedding Dim | Nº de Heads | Feed-Forward Dim | Blocs Transformer |
|---|---|---|---|---|
| Model 1 | 128 | 8 | 256 | 1 |
| Model 2 | 128 | 8 | 256 | 2 |
| CNN-Transformer | 128 | 8 | 256 | 1 |
| CNN-Transformer gran | 256 | 8 | 512 | 2 |

Table 3: Resum de les arquitectures de Transformers provades.

## 2.7 Models pre-entrenats

Finalment l'únic que faltava a provar era treballar amb models ja pre-entrenats per veure si erem capaços de millorar els resultats. Pel que fa al seu ús podem dividir en dues tasques el nostre treball:

- **Embeddings congelats + Classificadors:** Utilitzar les representacions ja apreses per entrenar al final un classificador basat en *xarxes neuronals*, *SVM*, *Logistic Regression*, *LightGBM*.
- **Finetuning:** Ajustar els pesos dels diferents models per especialitzar-los en la tasca.

El model provat per comparar les dues maneres va ser **DistilHubert** on posteriorment es va finetunejar aquest model alhora que **Hubert-Large** i **Hubert-XLarge**.

## 2.8 Millores addicionals

Un cop realitzats els experiments, i considerant que haviem mantingut molts dels models relativament senzills, vam escollir els millors models de cada apartat per realitzar-ne proves addicionals, amb models més complexos, resizes més grans (64x64 enlloc de 32x32), i més epochs d'entrenament (+50).

**Callbacks** Una altra millora, relacionada amb el fet d'entrenar durant més temps, va ser la incorporació de schedules pel learning-rate i altres callbacks. Concretament, en alguns models hem provat d'afegir *ReduceLROnPlateau*, *lr cosine decay* , *ExponentialDecay* i *EarlyStopping*. Aquesta part, més que un experiment, és un afegit que hem incorporat als models més elaborats per tal de poder obtenir el màxim del seu potencial, i per tant no hem quantificat els seus efectes.

**SpecAugment** Una tècnica molt utilitzada en tasques de reconeixement d'àudio és SpecAugment [5]. Hem optat per implementar una capa de Keras que realitza aquesta tècnica (sense el time warping, només el masking tant per temps com per features) per analitzar si millora el rendiment dels models. Hem provat diverses configuracions de SpecAgument (1 sola màscara o 2).

**Regularització** Al ser més complexos, aquests models patien més de problemes com overfitting. És per aquest motiu que els experiments relacionats amb la regularització (especialment *Dropout* i *l2*) vam realitzar-los en aquest apartat. El model utilitzat és el baseline *Big* del principi.

# 3 Resultats

## 3.1 Espectrograma

| Spectrogram Type | Small Model | | Big Model | |
|---|---|---|---|---|
| | Val Loss | Val Accuracy | Val Loss | Val Accuracy |
| Spectrogram | 0.6066 | 0.8977 | 0.3008 | 0.9573 |
| Power Spectrogram | 0.7981 | 0.7995 | 0.7212 | 0.8441 |
| Mel Spectrogram 32 | 0.4466 | **0.9203** | 0.2992 | 0.9590 |
| Mel Spectrogram 64 | 0.4374 | 0.9149 | 0.2961 | **0.9615** |
| Mel Spectrogram 128 | 0.4111 | 0.9162 | 0.2914 | 0.9604 |
| MFCC 13 | 0.5543 | 0.8931 | 0.3495 | 0.9474 |
| MFCC 13 (64) | 0.5122 | 0.8993 | 0.3609 | 0.9440 |
| MFCC 40 | 0.5157 | 0.9056 | 0.3262 | 0.9514 |
| MFCC 64 | 0.7106 | 0.8578 | 0.3322 | 0.9494 |
| MFCC 64 (64) | 0.6698 | 0.8685 | 0.3195 | 0.9528 |

Table 4: Validation Loss i Accuracy per diferents tipus d'espectrograma en els models Small i Big

Clarament, podem observar en 4 com el power-spectrogram és el pitjor amb diferència. El millor sembla ser el Mel-spectrogram (en qualsevol de les seves variants). Per simplicitat, en la resta d'experiments utilitzarem el mel-spectrogram de 64, ja que és el que obté millor resultat amb el model més gran.

## 3.2 Normalització

| Normalization | Val Loss | Val Accuracy |
|---|---|---|
| none | 0.4111 | 0.8942 |
| proposed_start | 0.4049 | 0.9045 |
| layer_start | 3.3938 | 0.0367 |
| batch_start | 0.4237 | 0.8984 |
| layern_start_between_conv | 0.3729 | 0.9114 |
| batch_start_between_conv | 0.4096 | 0.9110 |
| layer_start_after_conv | 3.3939 | 0.0367 |
| batch_start_after_conv | 0.4037 | 0.9100 |
| layer_between_conv | 0.3820 | 0.9118 |
| batch_between_conv | 0.4756 | 0.8957 |
| mlayer_between_dense | 0.4148 | 0.8963 |
| batch_between_dense | 0.4475 | 0.9008 |
| layer_before_dense | 0.3904 | 0.9086 |
| batch_before_dense | 0.4731 | 0.8900 |
| layer_start_between_conv_before_dense | 3.3938 | 0.0367 |
| batch_start_between_conv_before_dense | 0.4373 | 0.9011 |
| layern_start_after_conv | 0.4765 | 0.8889 |
| layern_start_between_conv_before_dense | 0.3608 | 0.9140 |

Table 5: Validation Loss i Accuracy per a diferents configuracions normalització.

Els resultats 5 són complicats d'evaluar. Per començar, sembla ser que la regularització millora en pràcticament tots els casos, tot i que utilitzar LayerNorm al principi sembla ser que provoca que el model no entreni. La normalització té un gran efecte sobretot entre convolucionals (més que entre les capes denses). El model amb millor accuracy final és el que utilitza LayerNormalization en tots els llocs menys a l'inici (també té el loss més baix).

## 3.3 Convolucionals

| Model | Parameters | Val Loss | Val Accuracy |
|---|---|---|---|
| conv1d_1 | 528031 | 1.0932 | 0.7948 |
| conv1d_2 | 4233695 | 1.2316 | 0.8165 |
| conv1d_3 | 81727 | 0.7757 | 0.8124 |
| conv1d_4 | 545023 | 1.1485 | 0.8153 |
| conv1d_5 | 276687 | 1.4183 | 0.7792 |
| conv2d_1 | 137954 | 0.5908 | 0.8827 |
| conv2d_2 | 1145378 | 0.5177 | 0.9100 |
| conv2d_3 | 132258 | 0.4323 | 0.9196 |
| conv2d_4 | 303554 | 0.4869 | 0.9124 |
| conv2d_5 | 71954 | 0.4128 | 0.9033 |
| conv2d_6 | 106386 | 0.3803 | 0.9111 |
| conv2d_7 | 38850 | 0.3693 | 0.9121 |
| conv2d_8 | 230306 | 0.4129 | 0.9315 |
| conv2d_9 | 116258 | 0.4782 | 0.8917 |
| conv2d_10 | 173698 | 0.4313 | 0.9188 |
| conv2d_11 | 373410 | 0.3755 | 0.9299 |
| conv2d_3_avg | 107682 | 0.3447 | 0.9230 |
| conv2d_7_avg | 32706 | 0.3407 | 0.9067 |
| conv2d_8_avg | 107426 | 0.2951 | 0.9250 |
| conv2d_11_avg | 250530 | 0.3135 | 0.9250 |
| conv2d_3_norm | 132258 | 0.3624 | 0.9198 |
| conv2d_7_norm | 38850 | 0.3387 | 0.9124 |
| conv2d_8_norm | 230306 | 0.3464 | 0.9326 |
| conv2d_11_norm | 373410 | 0.3035 | 0.9397 |
| mixed_1 | 285119 | 0.6172 | 0.9011 |
| mixed_2 | 4287967 | 0.7650 | 0.8943 |
| mixed_3 | 4214175 | 0.8480 | 0.8730 |
| mixed_4 | 2118175 | 0.8975 | 0.8509 |
| mixed_5 | 1067935 | 0.8788 | 0.8604 |
| resnet_1 | 138530 | 0.6464 | 0.8786 |
| resnet_2 | 1153762 | 0.5134 | 0.9191 |
| resnet_3 | 62114 | 0.3990 | 0.9043 |
| resnet_4 | 369442 | 0.5046 | 0.9108 |
| resnet_5 | 54610 | 0.3825 | 0.8977 |
| resnet_8 | 241314 | 0.4383 | 0.9235 |
| resnetv2_1 | 321298 | 0.1996 | 0.9485 |
| resnetv2_2 | 613410 | 0.2089 | 0.9506 |
| resnetv2_3 | 623650 | **0.1715** | **0.9568** |
| resnetv2_4 | 317730 | 0.1807 | 0.9500 |
| resnetv2_5 | 312610 | 3.3917 | 0.0312 |
| resnetv2_6 | 26706 | 0.4049 | 0.8792 |
| resnetv2_7 | 23138 | 0.8744 | 0.7476 |

Table 6: Training Results de models CNN

La conclusió principal de la taula 6 és que les convolucions 1D no aporten gaire. Els seus models tenen molts paràmetres i obtenen *accuracies* molt més baiges (tant soles com combinades amb 2D). Les convolucionals 2D semblen prometedores, i amb més paràmetres solen obtenir millors resultats. Un nombre de capes més elevat també sembla ser beneficiós, així com strides de tan sols 1 i kernels de 3. Les que utilitzen Residual obtenen resultats bastant similars a convolucionals del mateix nombre de paràmetres (o pitjors), però les Residual2 en mlts casos són capaçes d'obtenir resultats molt millors (en part perquè tenen normalització, que podem veure que millora lleugerament els resultats).

Utilitzar GlobalAveragePooling2D redueix el nombre de paràmetres sense afectar massa al rendiment.

Un altre gràfic interessant és el de resultats en funció del nombre de paràmetres 1.

Els models semblen escalar amb el nombre de tamanys. Observem clarament que convolucionals 1D són pitjors, i Resnetv2 sembla ser el millor model. És important notar que aquests experiments han sigut realitzats amb un resize de 32 (fet que explica, per exemple, que el model de convolucionals baseline *Big* obtingués millors resultats al provar els espectrogrames).

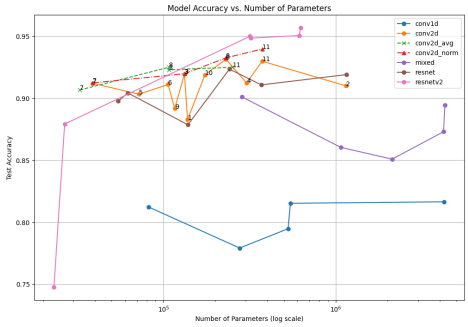


Figure 1: Accuracy en funció de log(params)

## 3.4 Recurrents

| Model | Val Loss | Val Accuracy | Parameters |
|---|---|---|---|
| **RNN** | | | |
| rnn_1 | 3.2633 | 0.0603 | 7231 |
| rnn_2 | 3.3405 | 0.0497 | 15487 |
| rnn_3 | 3.3903 | 0.0347 | 44287 |
| rnn_4 | 2.7892 | 0.1441 | 28831 |
| **LSTM** | | | |
| lstm_1 | 0.5973 | 0.8256 | 16543 |
| lstm_2 | 0.3501 | 0.8974 | 49567 |
| lstm_3 | 0.2294 | 0.9397 | 164767 |
| lstm_4 | 0.2653 | 0.9270 | 78367 |
| **GRU** | | | |
| gru_1 | 0.5071 | 0.8536 | 13535 |
| gru_2 | 0.2809 | 0.9218 | 38495 |
| gru_3 | 0.2269 | 0.9407 | 125279 |
| gru_4 | 0.2991 | 0.9145 | 62239 |
| **BiLSTM** | | | |
| bilstm_1 | 0.4876 | 0.8556 | 31007 |
| bilstm_2 | 0.2182 | 0.9401 | 113439 |
| bilstm_3 | 0.2072 | 0.9426 | 409375 |
| bilstm_4 | 0.2270 | 0.9380 | 185375 |
| **BIGRU** | | | |
| bigru_1 | 0.3933 | 0.8908 | 24991 |
| bigru_2 | 0.2285 | 0.9369 | 87199 |
| bigru_3 | 0.2155 | 0.9485 | 309919 |

| Model | Val Loss | Val Accuracy | Parameters |
|---|---|---|---|
| **ConvLSTM2** | | | |
| convlstm2_1 | 0.2413 | 0.9364 | 285279 |
| convlstm2_2 | 0.2006 | 0.9480 | 172991 |
| convlstm2_3 | 0.1781 | 0.9548 | 116703 |
| **ConvGRU2** | | | |
| convgru2_1 | 0.2449 | 0.9360 | 215775 |
| convgru2_2 | 0.1993 | 0.9478 | 136255 |
| convgru2_3 | 0.2050 | 0.9486 | 96351 |
| **ConvBidirGRU** | | | |
| convbidirgru2_1 | 0.2283 | 0.9415 | 429151 |
| convbidirgru2_2 | 0.1862 | 0.9551 | 251327 |
| convbidirgru2_3 | 0.2158 | 0.9469 | 162271 |
| **ConvCRNN** | | | |
| convcrnn_1 | 0.5350 | 0.8574 | 80223 |
| convcrnn_2 | 0.2198 | 0.9529 | 435663 |
| convcrnn_3 | 0.2223 | 0.9497 | 480750 |
| **AttRNN** | | | |
| att_rnn_1 | **0.1539** | 0.9539 | 167822 |
| att_rnn_2 | 0.1569 | **0.9577** | 167822 |
| **mhatt_rnn** | | | |
| mhatt_rnn_0 | 0.1682 | 0.9557 | 3892703 |
| mhatt_rnn_1 | 0.1808 | 0.9528 | 777422 |
| mhatt_rnn_2 | 0.1959 | 0.9460 | 777422 |

Table 7: Training Results de models RNN

S'observa (7) clarament com les convolucionals simples no són capaces de realitzar la tasca correctament. GRU sembla ser millor que LSTM, en la majoria de casos (tant combinacions simples, com bidireccionals). Afegir convolucionals abans de les capes recurrents també millora els resultats. El millor model és el que utilitza attention després de les recurrents, tot i que sorprenentment és millor *attention* directament que *multihead* (encara que en el paper original era al revés).

## 3.5 Transformers

Els resultats obtinguts amb les diferents arquitectures de transformers mostren algunes dificultats importants en l'entrenament i en l'obtenció de resultats competitius. Tot seguit es detallen els resultats i es proporcionen hipòtesis sobre els factors que poden haver influït en el seu rendiment:

- **Simple Transformer (1 bloc):** Aquest model va aconseguir una accuracy de 0.72 en el conjunt de validació, però va presentar grans dificultats durant l'entrenament, resultant en una parada anticipada pel *early stopping*. Això suggereix que l'arquitectura és insuficient per capturar les característiques del conjunt de dades o massa complexa.
- **Transformer avançat (2 blocs amb gradient accumulation):** Tot i les millores teòriques respecte al model simple, aquest model no va aconseguir aprendre res significatiu, també petant per *early stopping*. Aquesta limitació podria ser deguda a l'elevada complexitat de l'arquitectura en comparació amb la quantitat i diversitat de dades disponibles.

- **Transformer amb CNN (simple):** Aquest enfocament va obtenir resultats molt més prometedors, amb una accuracy de 0.94 en validació. Tot i això, encara es va observar un alt cost computacional, i els resultats van ser lleugerament inferiors als models CNN simples.
- **Transformer amb CNN (més complex):** Aquesta arquitectura no va aconseguir superar els resultats del model CNN simple, amb una accuracy màxima de 0.93. Això indica que l'increment en complexitat no va ser beneficiós, probablement per l'excés de paràmetres i la dificultat d'entrenament.

Es creu que el principal obstacle d'aquestes arquitectures es troba en la seva elevada complexitat, que fa que requereixin més dades per evitar problemes de sobreajustament. La combinació amb convolucions va ajudar a millorar els resultats gràcies a l'extracció de característiques locals.

| Model | Accuracy Val | Comentaris |
|---|---|---|
| Simple Transformer (1 bloc) | 0.72 | Early stopping, rendiment insuficient |
| Transformer avançat (2 blocs) | N/A | Early stopping, no aprenia res |
| Transformer amb CNN (simple) | 0.94 | Bon rendiment però costós |
| Transformer amb CNN (més complex) | 0.93 | Similar o lleugerament pitjor que CNN simple |

Table 8: Resum dels resultats obtinguts amb les diferents arquitectures de transformers.

## 3.6 Models pre-entrenats

Els models pre-entrenats van mostrar una millora significativa en els resultats, especialment quan es va aplicar *fine-tuning*, en comparació amb l'ús d'*embeddings* congelats. A continuació, es detallen els resultats obtinguts:

### 3.6.1 Embeddings Congelats

En aquesta configuració, es van utilitzar diverses arquitectures de classificadors per treballar amb les representacions pre-apreses:

- **Feed Forward (1 capa):** Accuracy de 0.9693 en validació.
- **Feed Forward (2 capes, dropout 0.2):** Accuracy de 0.9724 en validació.
- **Feed Forward (3 capes, dropout 0.4):** Accuracy de 0.9735 en validació.
- **SVM:** Accuracy de 0.9653 en validació.
- **Logistic Regression:** Accuracy de 0.9694 en validació.

Tot i que els *embeddings* congelats van produir resultats decents, l'ús de *fine-tuning* va demostrar ser superior, ja que permet optimitzar totes les capes del model.

### 3.6.2 Fine-tuning

Els resultats obtinguts amb *fine-tuning* dels models pre-entrenats són els següents:

- **DistilHuBERT:** Accuracy de 97.69% en test.
- **HuBERT-Large:** Accuracy de 98.57% en test (millor model de la competició, *outperforming* a la resta).
- **HuBERT-XLarge:** Accuracy de 98.55% en test.

## 3.7 SpecAugment i Regularització

En aquest apartat, els experiments no estan tan ben definits com en els anteriors i ha consistit més en un procés de prova i error. Tot i això, podem observar els efectes de SpecAugment en 9.

| Model and Configuration | Train Loss | Train Accuracy | Val Loss | Val Accuracy |
|---|---|---|---|---|
| ConvNet without Spec Aug | 0.0591 | 0.9952 | 0.2254 | 0.9530 |
| ConvNet with Spec Aug (1 Time, 1 Freq) | 0.0665 | 0.9946 | 0.2039 | 0.**9582** |
| ConvNet with Spec Aug (2 Time, 2 Freq) | 0.0921 | 0.9893 | 0.2161 | 0.9559 |
| ConvNet with Spec Aug (2 Time, 1 Freq) | 0.0758 | 0.9906 | 0.2116 | 0.9561 |
| Att-MH-RNN without Spec Aug | 0.0307 | 0.9916 | 0.2419 | 0.9491 |
| Att-MH-RNN with Spec Aug (1 Time, 1 Freq) | 0.0285 | 0.9926 | 0.2193 | 0.9520 |
| Att-MH-RNN with Spec Aug (2 Time, 2 Freq) | 0.0689 | 0.9802 | 0.1523 | 0.**9636** |
| Att-MH-RNN with Spec Aug (2 Time, 2 Freq) + Dropout (0.2) | 0.0739 | 0.9777 | 0.1460 | 0.9624 |
| Att-MH-RNN without Spec Aug + Dropout (0.2) | 0.0288 | 0.9918 | 0.2029 | 0.9593 |

Table 9: Comparació de Train i Validation Loss per diferents combinacions de SpecAugment

Sembla ser que SpecAugmentation realment millora els resultats del models. En els dos casos provats, el model inicial realitzava overfitting (i per aquest motiu han estat escollits), com podem observar amb l'accuracy i el loss del train. Spec Augmentation el redueix, permetent assolir resultats millors. En el cas del model d'attention multihead, la implementació del paper utilitzava Dropout, i hem observat també els efectes de treure'l. En definitiva, sembla ser que Spec Augmentation realment permet obtenir millors resultats i en general generalitzar millor. Mencionar que, en d'altres experiments realitzats, hem detectat com obteniem millors resultats de manera consistent amb 2 masks tant al temps com a les features, i que per resizes de 64 (o models encara més grans) sembla funcionar millor tamanys de màscara més grans (temps 20 i features 20, versus 10 i 5).

| Model and Configuration | Train Loss | Train Accuracy | Val Loss | Val Accuracy |
|---|---|---|---|---|
| No Regularization (no_reg) | 0.0525 | 0.9836 | 0.2795 | 0.9435 |
| Dense L2 (0.001) (dense_l2_001) | 0.0822 | 0.9843 | 0.2845 | 0.9403 |
| Conv L2 (0.001) (conv_l2_001) | 0.3604 | 0.9560 | 0.4606 | 0.9327 |
| Conv L1 (0.001) (conv_l1_001) | 0.7033 | 0.9203 | 0.7561 | 0.9047 |
| Conv L1 + L2 (0.001, 0.0001) (conv_l1_l2_001_0001) | 0.7277 | 0.9161 | 0.7357 | 0.9118 |
| Dropout (0.2) (dropout_02) | 0.0700 | 0.9791 | 0.2402 | 0.9472 |
| Dropout (0.1) (dropout_01) | 0.0605 | 0.9824 | 0.2416 | 0.9468 |
| Dropout (0.3) (dropout_03) | 0.0844 | 0.9761 | 0.2410 | 0.9491 |
| Dropout (0.5) (dropout_05) | 0.1087 | 0.9719 | 0.2306 | 0.9497 |
| Conv Dropout (0.1) (conv_dropout_01) | 0.1112 | 0.9666 | 0.1812 | 0.9530 |
| Conv Dropout (0.2) (conv_dropout_02) | 0.1066 | 0.9668 | 0.1703 | **0.9534** |
| Conv Dropout (0.5) (conv_dropout_05) | 0.2852 | 0.9121 | 0.1821 | 0.9460 |
| Dense L2 (0.001) + Dropout (0.2) (dense_l2_001_dropout_02) | 0.1171 | 0.9798 | 0.2610 | 0.9452 |
| Conv L2 (0.001) + Dense L2 (0.001) (conv_l2_001_dense_l2_001) | 0.4048 | 0.9557 | 0.5699 | 0.9132 |
| Conv Dropout (0.2) + Dense L2 (0.001) (conv_dropout_02_dense_l2_001) | 0.2128 | 0.9500 | 0.1703 | 0.9533 |

Table 10: Performance del model amb diferents tècniques de regularització.

Pel que fa a la regularització, sembla ser que sí que permet millorar els resultats d'un model (especialment si fa overfitting) 10. Té un millor impacte, almenys amb l'arquitectura posada, situar aquest dropout entre les convolucionals enlloc de fer-ho tan sols a la densa. Regularització l2 i l1 semblen ser pitjors.

## 3.8 Entrenaments més llargs

Un fet que creiem important a comentar és que també hem provat entrenar els models (els millors de convolucionals i recurrents especialment) durant més temps, amb els callbacks comentats anteriorment. Aquests experiments, si bé no tant significatius com els comentats, ens han permés esbrinar encara amb més profunditat els models.

Les conclusions principals són que els models amb recurrents són capaços d'assolir aproximadament 96.8 d'accuracy al validation (les nostres implementacions). Això inclou també el que utilitza multi-head attention, un resultat menor al del paper (hem assolit, com a molt, 97.01 d'accuracy al validation amb aquell model). Sorprenentment, els basats purament en convolucionals eren capaços d'assolir accuracies lleugerament més elevades, especialment els basats en el nostre Resnet2 (amb 50 epochs, i de forma relativament consistent, assolien valors propers 0.97).
Un cas especial és el d'un model de Resnet2 que utilitza 64, 64, 128, 128, 256 filtres i 256 neurones a la densa: un total de 1.594.658 (aproximadament el doble que els provats). Aquest ha arribat a assolir una accuracy al **test** de **0.9800** (val 0.9777). Escalar l'arquitectura a més paràmetres sembla millorar-la, o sigui que segurament es podrien obtenir millors resultats amb un model similar però amb 3M params. Utilitzant Resnet2 + BidireccionalGRU + attention (com att-RNN canviant les convolucions, 1.8M params) hem assolit valors similars, **0.9802** a test.

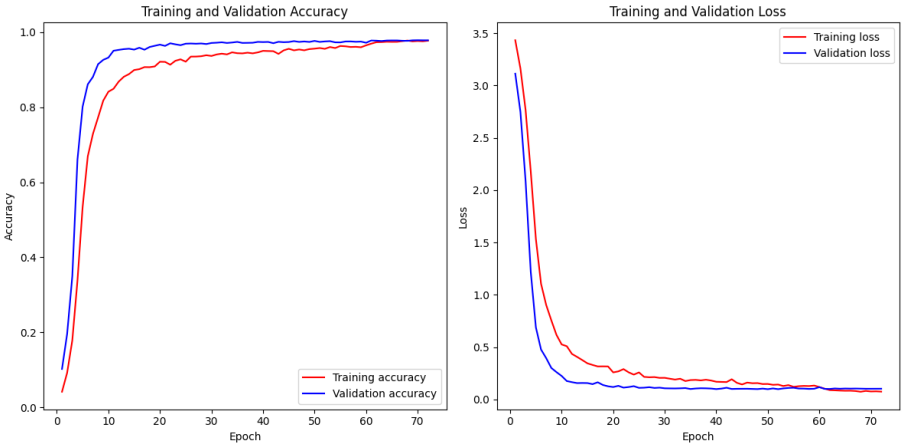


Figure 2: Training curves of 1.5M Resnet2

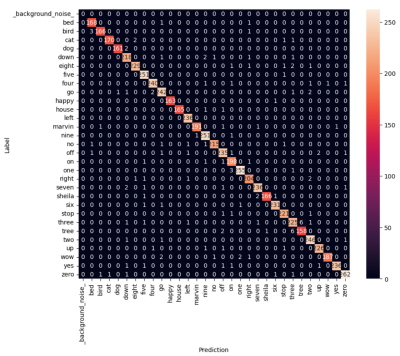


Figure 3: Confusion matrix of 1.5M Resnet2

# 4 Conclusions

En aquest treball hem pogut explorar el disseny de diversos models per al reconeixement de comandes de veu. A través dels experiments, hem sigut capaços d'identificar l'impacte de diverses decisions a l'hora de crear-los, així com el tamany dels models. Un dels primers passos va ser determinar el millor preprocessament de l'àudio. Els experiments ens han mostrat que el MEL-espectrograma, especialment amb 64 bins, supera a la resta. La normalització s'ha demostrat que és un component vital per a un entrenament efectiu. Aplicar LayerNormalization entre les capes convolucionals i abans de les capes denses permet obtenir una millora notable en el rendiment dels models.

Pel que fa a les arquitectures de xarxes neuronals, les xarxes convolucionals 2D (CNN) han superat clarament les convolucionals 1D. Les arquitectures basades en ResNetv2 han demostrat ser molt bones, especialment si incorporaven normalització i GlobalAveragePooling2D. Les xarxes recurrents (RNN), sobretot les GRU bidireccionals, també han mostrat un bon rendiment (combinades amb convolucionals i *attention*). Tot i la seva popularitat en altres tasques, els models Transformer han sigut ser més difícils d'entrenar en aquest context, i els resultats obtinguts són pitjors en comparació amb els models basats en CNN i RNN.

Hem pogut observar com els resultats seguien millorant a mesura que entrenàvem els models més temps (i amb models més grans), utilitzant tècniques com SpecAugment i regularització, mitjançant l'ús de Dropout. Un model ResNet2 de 1.6 milions de paràmetres va aconseguir un 98.00% d'accuracy en el conjunt de test. Un altre model similar, però combinant Resnet2, bidireccional GRU i atenció va obtenir resultats similars. Aquests resultats són molt significatius, ja que comparant-los amb leaderboards del dataset i tasques similars [6][7][8] podem considerar que són molt bons. A més, aquest *score* seria el millor resultat a la competició de l'assignatura, [9] (considerant models entrenats des de zero).

Finalment, vam realitzar transfer learning amb models pre-entrenats per àudio, específicament DistilHubert, Hubert i XLHubert. Els models més grans han demostrat un potencial encara major, assolint una accuracy de 98.6% en el conjunt de test. A la mateixa competició de [9], obté la primera posició.

En resum, aquest estudi ha demostrat que els models d'aprenentatge profund, quan s'entrenen adequadament, són capaços d'aconseguir un reconeixement de comandes de veu amb una alta precisió. La selecció de la representació d'àudio correcta, l'ús de tècniques de normalització i regularització, i l'aplicació de mètodes d'augment de dades com SpecAugment són essencials per assolir aquest objectiu. També és molt important el fet d'utilitzar models preentrenats i fer transfer learning. El model de 1.6M, així com els models pre-entrenats, han demostrat el seu potencial per a aplicacions de reconeixement de comandes de veu en entorns del món real.

# References


[1] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] A. Mesaros, T. Heittola, and T. Virtanen, "An attention-based neural network for detecting rare sound events," *ICASSP 2018 - 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[3] L. Lugosch, O. Rybkin, V. Voleti, O. von Grünberg, S. Keselj, and H. Palangi, "Streaming keyword spotting on mobile devices," *arXiv preprint arXiv:2005.06720*, 2020.

[4] Google Research, "Keyword spotting using streaming architectures," 2020.

[5] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *Interspeech 2019*, 2019.

[6] Papers with Code, "Audio classification on speech commands v1," 2024. State-of-the-art results.

[7] Papers with Code, "Keyword spotting on google speech commands," 2024. State-of-the-art results.

[8] Papers with Code, "Speech recognition on speech commands v2," 2024. State-of-the-art results.

[9] Kaggle, "Tvd 2024 - reconocimiento de comandos de voz," 2024. Kaggle Competition.