# Slide 1 - Intro

Hello everyone. My name is Radu. I know my English is not very good, but today I will present you my project. In short, my task was to create a voice command classification model that would work on an embedded system. In order to solve this task I used different techniques for processing the voice signal and improving the performance of the model. I also made a comparison between 3 established architectures in the field of classification and found which is the best solution.

# Slide 2 – The problem

This embedded system is Raspbberry Pi 4 model B. It will process the voice signal and classify the command in real time. The result will be the voice actuation of a smart light bulb or other hardware devices.

Also, I want to achieve a performance similar to systems such as Amazon Alexa, Google Assistant, Siri, but the processing should be done locally.

# Slide 3 – Window\_size, Window\_step, number MFCC

1. Find a suitable feature matrix size. Choose the window size, window step and number of extracted features wisely.

The features are calculated in real time on the embedded system and based on them the prediction is made. Using window size = 20ms and window step = 20ms, the size of the feature matrix will be (32,50). With window size = 30ms and window step = 10ms, the shape is almost double (32,98).

Although I have more features, there is the problem of their calculation cost.

# Slide 4 – My choose for window\_size and window\_step

I found that using a window size - 30ms and a window step - 10ms gave me better accuracy, so I sacrificed computational resources for a more confident prediction. I will compensate on the architecture side.

# Slide 5 – Complexity of the model

I hope that many of us have heard of convolution at least once. The complexity of a model is determined by the number of convolutional layers used, the number of filters applied, kernel\_size, the use of a bias, and others. This complexity determines the number of parameters, the size of the model and even the training and prediction time. A combination of smaller convolution filters can produce the same result as a large filter but with a lower consumption of computational resources and number of parameters.

# Slide 6 – Arhitectures

In this paper I analyzed, implemented and compared three established convolutional neural networks: VGG16, AlexNet and SqueezeNet. The most complex being VGG16, and the most robust SqueezeNet. For my task, SqueezeNet was the best fit.

# Slide 7 – Customize -> citit +

A batch of training data is used to estimate the gradient descent error. Thus the weights of the model are updated. So the larger the lot size, the more accurate this statistical estimate is. Too small batch sizes are noisier in weight updates. A workaround for steeper updates is to use a lower learning rate.

# Slide 8 – Post-training cuantization - > citit + explicații

# Slide 9 – Quantization benefits -> citit + explicații

# Slide 10 – Results(local performance) -> prezentare

# Slide 11,12,13 – Speed of live predict -> explain +

The surprise was for models quantized with 16x8 integer, which performed the prediction in a higher average time. This is because this quantization mode is an experimental, post-training mode, that does not have kernel-optimized implementations or hardware accelerators to improve execution performance. Therefore, this type of quantization is likely to be slower than the floating-point interpreter.

# Slide 14 – Video 30s

# Slide 15 – Conclusions -> citit

# Slide 16 – Thank you!

Română

**Slide1**

Salutare tuturor. Numele meu este Radu. Eu știu că engleza mea nu este foarte bună, dar eu azi vă voi prezenta proiectul meu. Pe scurt, sarcina mea a fost să realizez un model de clasificare al comenzilor care să funcționeze pe un sistem embedded. Pentru a rezolva această sarcină eu am folosit diferite tehnici de prelucrare a semnalului vocal și de eficientizare a performanțelor modelului. De asemenea am realizat o comparație între 3 arhitecturi consacrate în domeniul clasificării și am găsit care este soluția cea mai bună.

**Slide2**

Acest sistem embedded este Raspbberry Pi 4 model B. Acesta va procesa semnalul vocal și va clasifica în timp real comanda. Rezultat va fi acționarea vocală a unui bec inteligent sau alte dispozitive hardware.

De asemnea, îmi doresc să obțin o performanță asemănătoare cu sisteme precum Amazon Alexa, Google Assistant, Siri, dar procesarea să fie realizată local.

**Slide 3 – Soluții 1**

1. Găsește o dimensiune potrivită a matricii de caracteristici. Alege înțelept dimensiunea ferestrei, pasul și numărul de caracteristici extrase.

Aceste matrici de caracterstici sunt calculate în timp real pe sistemul embedded și pe baza acestora se realizează predicția. Folosind window size = 2ms și window step =2ms, dimensiunea matricei de caractersitici va fi (32,50). Cu window size= 3ms și window step = 1ms, shape-ul este aproape dublu (32,98).

Deși am mai multe cadre, se pune problema costului de calcul al acestora.

**Slide 4 – Soluții 1**

Am descoperit că utilizând un window size - 3 ms și un window step - 1 ms mi-a oferit acuratețe mai bună, așa că am sacrificat resursele de calcul pentru o predicție mai sigură. Voi compensa pe partea de arhitectură.

**Slide 5 – Soluții 2 Complexity of the model**

Sper că mulți dintre noi au auzit cel puțin o dată de convoluție. Complexitatea unui model este determinată de numărul de straturi convoluționale folosite, numărul de filtre aplicate, kernel\_size, utilizarea unui bias și altele. This complexitate determină numărul de parametri, dimensiunea modelului și chiar timpul de antrenare și predicție. O combinație de filtre de convoluție mai mici poate produce același rezultat cu un filtru mare dar cu un consum de resurse computaționale și număr de parametri mai mic.

**Slide 6 – Soluții 3 - Arhitectures**

În această lucrare am analizat, implementat și comparat trei rețele neuronale convoluționale consacrate: VGG16 , AlexNet și SqueezeNet. Cea mai complexă fiind VGG16, iar cea mai robustă SqueezeNet. Pentru sarcina mea, cel mai bine s-a potrivit SqueezeNet.

**Slide 7 – Solutii 4 - Customize**

**Slide 8**

**Slide 9**

**Slide 10**

Un lot de date de antrenament este utilizat pentru estimarea erorii de coborâre a gradientului. Astfel sunt actualizate ponderile modelului. Deci cu cât dimensiunea lotului este mai mare cu atât această estimare statistică este mai precisă. Dimensiunile prea mici ale loturilor sunt mai zgomotoase în actualizările ponderilor. Ca rezolvare pentru actualizările mai abrupte este utilizarea unei rate de învățare mai mică.

**Slide 11,12,13 – Speed of live predict -> explain +**

**S**urpriza a fost pentru modele cuantizate cu 16x8 integer, care au realizat predicția într-un timp mediu mai mare. Acest lucru se datorează faptului că această modalitate de cuantizare este un mod experimental, post-antrenament, care nu are implementări optimizate pentru nuclee sau acceleratoare hardware care să îmbunătățească performanța execuției. Prin urmare, acest tip de cuantizare este posibil să fie mai lentă decât interpretorul cu numere în virgulă mobilă.

Slide 14 Video

Slide 15 Concluzii

Slide 16 Thank you.