HW3:

Machine Learning for IOT

Group 2

Dri Emanuele s280133

Malan Erich s267475

Nicolì Alessandro s278091

Exercise 1

Objective: Develop a Big/Little model for Keyword Spotting composed by a Big neural network running on a notebook and a Little neural network running on a Raspberry Pi.

Execution: For this exercise we used REST protocol to implement the communication (we used the PUT method because it is *idempotent* and so it can be applied multiple times without changing the result beyond the initial application). We chose this solution because we don't need to keep always alive the connection between the two devices and so the intermittent asynchronous REST calls are suitable for our needs, in fact we need to establish a connection only when the "success checker" (for which we used the Score Margin method) doesn't accept the prediction of the Little model. So, with the REST protocol, the client can invoke the web service only when needed. In a real case scenario, a client might not need to use the Big model for minutes, so a continuous connection (MQTT) is a bad choice.

To find suitable models for our task, we started with the DSCNN modified in hw2 which made use of pruning and WO quantization. For the Little model that was sufficient and this model alone achieves an accuracy of 89.325%. For the big model instead we worked on two knobs to improve accuracy: depth and width, in particular we quadrupled the number of filters for the convolutional layers and on the other hand we also added one main block of this net (DepthWiseConv2D+Conv2D+BatchNormalization+ReLu). Given the considerable size of the resulting model it was very important here to put a dropout layer (before the dense layer and with 0.25 as parameter) to regularize our model and avoid overfitting. All this resulted in a big model with an accuracy of 94.5% (note that no pruning or quantization were applied this time).
The final step was then to tune the value of the Score Margin threshold, we found 0.44 (44%) as the best value in order to respect both constraints. The accuracy obtained with this setup was 93.25%.

Note: To correctly use the little_client.py script one must explicit the server ip using *--server_ip <string with the ip of the device hosting big_service.py>* (the port is set by default to 8080 but can be changed with *--server_port*)

Use Case scenario: an operator gives a vocal command to a forklift and, if the forklift's client does not reach the score margin threshold, it sends the command to the server in order to correctly understand it.

Exercise 2

Objective: Develop a Cooperative Inference distributed application for Keyword Spotting composed by: a cooperative client that reads a an audio signal, applies some pre-processing, sends the result to the inference clients and retrieve the neural networks outputs; an inference client that receives pre-processed audio signal, classify its content and sends the output back.

Execution: For this exercise we used MQTT protocol to implement the communication. Differently from the previous exercise, in this case we need to exchange data among clients every time we need to label an audio file. With the REST approach, the cooperative client should contact every server individually (knowing a priori how many and exact addresses) and probably would also increase the communication cost. With the MQTT approach, instead, the client has to send the data to the broker only once and then the broker will spread it to the inference clients subscribed to the topic, which is far more independent and robust towards connection errors / inference clients unavailable. So in this case (and in similar IoT application cases) we think that MQTT is the best approach between the two.

Since we are using the eclipse broker, all the topics have "/PoliTO/ML4IOT/Group2/" at the beginning to avoid receiving data from other applications. The cooperative client sends its data in ".../{clientID}/data/"

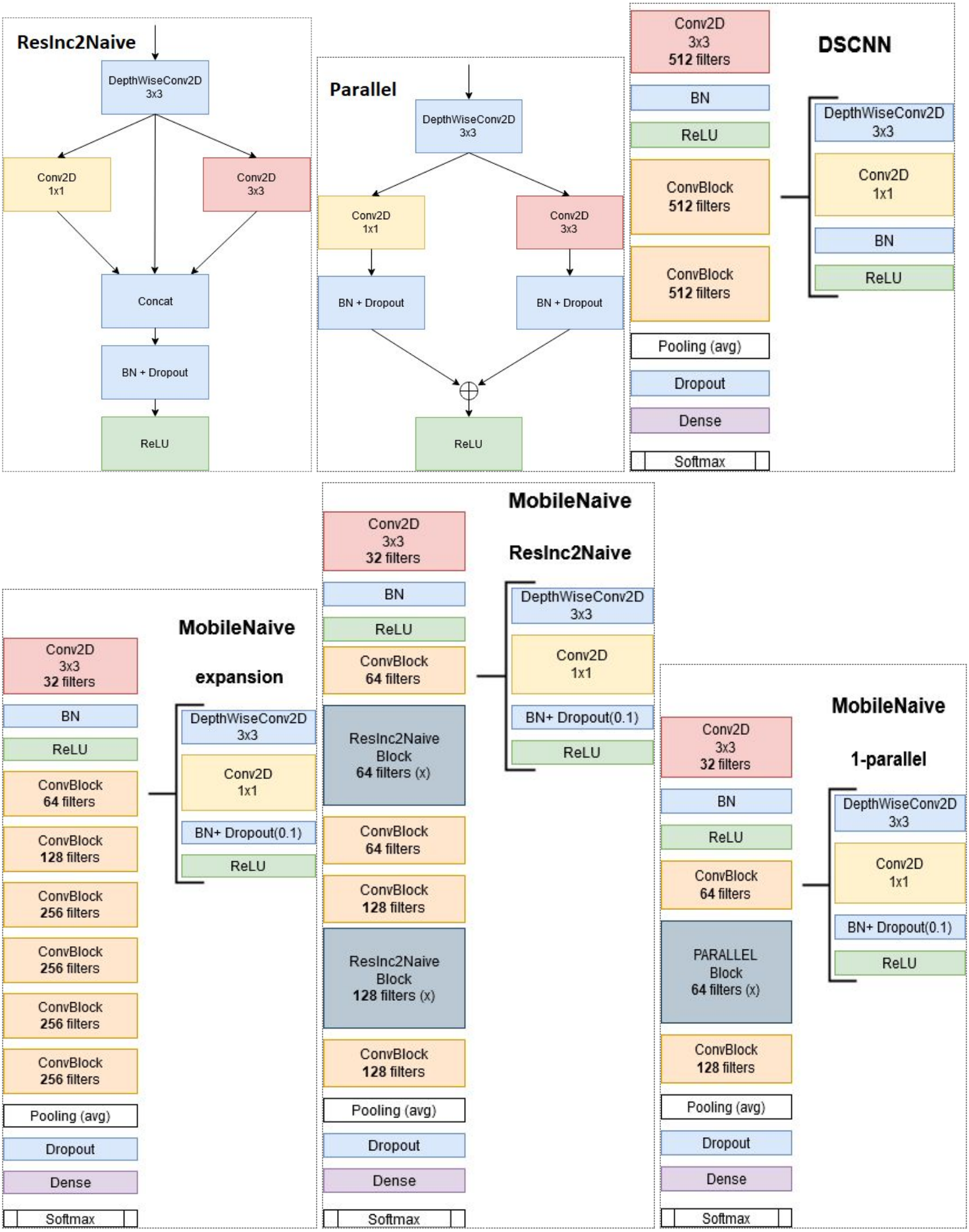
and retrieves results from ".../{clientID}/results/#" where "#" is a number from 1 to N. The inference client retrieves data from ".../+/data/" and sends the results in ".../+/results/{version}" where "+" is the client ID and "{version}" is the version of the model that it is using. In this way a single inference client can serve different cooperative clients. For all the transmissions we use QOS = 2.

However, as it is now, we optimized the parameters to handle only one client that sends a new message almost immediately after it sent the previous one. This means that, if we connect two (or more) cooperative clients, they send too much data and some of it may be lost. To overcome this problem, it is sufficient to slow down the publication rate.

Model definitions: The five inference models presented have been designed according to the distributed scenario explained above, the models are designed to run singularly in medium/low capability hardware, or in a parallel way in a singular performing machine. The backbone is the DSCNN *[fig. DSCNN]*, whereas for the variations we followed the approach of the MobileNet (one of the most used networks for mobile devices inference), which poorly performed over 20 epochs because of its overloaded capacity with respect to the task. So we decided to match the two models (which both use the Depth-wiseConv2D) by modifying the DSCNN, instead of maintaining the same number of Conv2D filters in every layer, the model starts with 32 filters and doubles it up to 256, which will be repeated 4 times, moreover the convolutional filters shape is the same of the MobileNet: 1x1 (far more cheaper) *[fig. MobileNaive-expansion]*. In the end we considered that the features extracted would be different with respect to the filter shape, so we implemented some variations that provide mixed types of filters. The NN common literature presents the inception networks and in particular the inception blocks which follow such a strategy. Obviously the whole inception block is too big and deep for our task, so we decided to subdivide the result of the depth conv into three routes, one leave it as it is, one perform a 1x1 Conv2D and the last 3x3Conv2D, in the end they are concatenated *[fig. ResInc2Naive-block && MobileNaive-ResInc2Naive]*. For saving some computation in the end we modified this version by only summing up the Conv2D-layers output *[fig. Parallel-block]*, and we provided two different versions, with 1 *[fig. MobileNaive-1parallel]* and 2 respectively "parallel" blocks.

Cooperative Policy: each model provides the softmax vector for labels and the cooperative_client script computes for each the most probable label and the relative score margin. Then a Majority Voting system is used (to decide the definitive label), where each "vote" is weighted by its score margin.

	Model 1	Model 2	Model 3	Model 4	Model 5	**Cooperative**
Accuracy	91.5%	93,5%	91,875%	92,625%	92,875%	**95,625%**


ResInc2Naive
DepthWiseConv2D 3x3
Conv2D 1x1
Conv2D 3x3
Concat
BN + Dropout
ReLU
Parallel
DepthWiseConv2D 3x3
Conv2D 1x1
Conv2D 3x3
BN + Dropout
BN + Dropout
ReLU
Conv2D 3x3 512 filters
BN
ReLU
ConvBlock 512 filters
ConvBlock 512 filters
Pooling (avg)
Dropout
Dense
Softmax
DSCNN
DepthWiseConv2D 3x3
Conv2D 1x1
BN
ReLU
MobileNaive expansion
Conv2D 3x3 32 filters
BN
ReLU
ConvBlock 64 filters
ConvBlock 128 filters
ConvBlock 256 filters
ConvBlock 256 filters
ConvBlock 256 filters
ConvBlock 256 filters
Pooling (avg)
Dropout
Dense
Softmax
DepthWiseConv2D 3x3
Conv2D 1x1
BN+ Dropout(0.1)
ReLU
MobileNaive ResInc2Naive
Conv2D 3x3 32 filters
BN
ReLU
ConvBlock 64 filters
ResInc2Naive Block 64 filters (x)
ConvBlock 64 filters
ConvBlock 128 filters
ResInc2Naive Block 128 filters (x)
ConvBlock 128 filters
Pooling (avg)
Dropout
Dense
Softmax
DepthWiseConv2D 3x3
Conv2D 1x1
BN+ Dropout(0.1)
ReLU
MobileNaive 1-parallel
Conv2D 3x3 32 filters
BN
ReLU
ConvBlock 64 filters
PARALLEL Block 64 filters (x)
ConvBlock 128 filters
Pooling (avg)
Dropout
Dense
Softmax
DepthWiseConv2D 3x3
Conv2D 1x1
BN+ Dropout(0.1)
ReLU