Spectrum Awareness at the Edge: Modulation Classification using Smartphones

Nasim Soltani, Kunal Sankhe, Stratis Ioannidis, Dheryta Jaisinghani, and Kaushik Chowdhury
Department of Electrical and Computer Engineering
Northeastern University
Boston, USA
Email: {soltani.n,sankhe.ku}@husky.neu.edu, {ioannidis,dheryta,krc}@ece.neu.edu

***Abstract*—As spectrum becomes crowded and spread over wide ranges, there is a growing need for efficient spectrum management techniques that need minimal, or even better, no human intervention. Identifying and classifying wireless signals of interest through deep learning is a first step, albeit with many practical pitfalls in porting laboratory-tested methods into the field. Towards this aim, this paper proposes using Android smartphones with TensorFlow Lite as an edge computing device that can run GPU-trained deep Convolutional Neural Networks (CNNs) for modulation classification. Our approach intelligently identifies the SNR region of the signal with high reliability (over 99%) and chooses grouping of modulation labels that can be predicted with high (over 95%) detection probability. We demonstrate that while there are no significant differences between the GPU and smartphone in terms of classification accuracy, the latter takes much less time (down to $\frac{1}{870}$x), memory space ($\frac{1}{3}$ of the original size), and consumes minimal power, which makes our approach ideal for ubiquitous smartphone-based signal classification.**

***Index Terms*—TensorFlow Lite, Modulation Classification, Low SNR, Smartphone, Edge Computing**

I. Introduction

Wireless spectrum is getting more crowded, with several billions of devices, from large base stations to tiny IoT sensors jostling for efficient utilization of the spectrum. The future points towards a shared ecosystem, where devices from different vendors and with different priority levels will share a common set of spectrum bands. Already cellular operators are pushing for access to unlicensed bands [1], while opportunistic use of TV whitespace (400-700 MHz) has now become a reality [2] [3]. In all such use-cases detecting other signals of interest, or at least recognizing existing signals of a specific modulation type, is of paramount importance. Importantly, coexisting devices will likely be unable to demodulate each other's signals, and worse, may only have few symbol-length sensing duration to infer such information (to minimize any outage for a higher priority transmission). Thus, fast and lightweight modulation recognition techniques are needed.

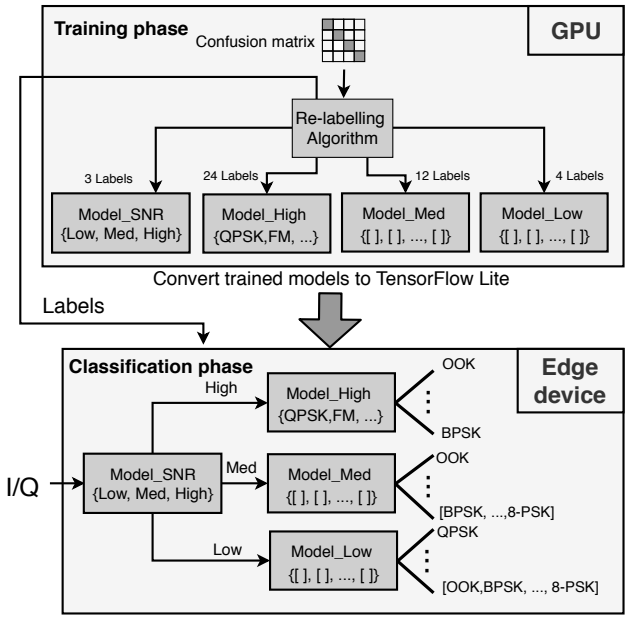


Fig. 1: Overview of the approach with separate processing in the GPU (training, label grouping) and edge device (classification).

A. Problem: Signal classification at the wireless edge

While the research community has demonstrated remarkable success in signal classification using deep learning techniques [4] [5] [6], many of these approaches cannot be directly ported to the field. Firstly, a rigorous study of how deep learning architectures trained on dedicated GPUs perform in terms of accuracy on resource-constrained wireless edge devices is needed. Secondly, the impact on classification time and power consumption becomes important. Finally, real world receivers observe noisy conditions, with common occurrence of very low SNR values. In such situations, there is a possibility of some compromise– e.g., could some form of coarse classification (with fewer labels) be obtained with high detection probability vs. large number of labels with very low probability? More importantly, can all these decisions

be automated, without involving human in-the-loop?

Given the scale and the density of today's wireless networks [7], we need solutions that are not only scalable, but also cost-effective with a shorter time-to-market. Thus, we propose to solve this problem by leveraging the most ubiquitous edge platform currently available: *smartphones*. Recent reports indicate that over 81% of the US adult population use smartphones, with this number rising to 96% in the age range of 18-29 years old [8]. Furthermore, the geographical breakdown into urban/rural categories shows this percentage to be around 83% and 71%, respectively. We wish to demonstrate how deep learning on smartphones can democratize signal classification ability; where any such handheld device can classify modulation schemes used in signals of interest.

B. Approach: Deep learning on smartphones

There exist several works on using machine learning on smartphones [9]. However, to the best of our knowledge, we are the first to demonstrate the use of a smartphone as a modulation classifying device, which was previously only possible with software defined radios or expensive specialized spectrum analyzers. Given that wireless spectrum is noisy and that previous works [10] [11] (summarized in Section III) only permit as low as 10% accuracy in low SNR conditions, we propose a secondary learning network architecture that detects and activates coarse classifications, i.e., fewer labels, in such adverse situations. Thus while our approach gives 97% accuracy in high SNR ($\geq$10dB) for 24 different modulation labels, it can also return a similar accuracy in low SNR (-10 to -2dB), when these labels are grouped into 4 classes. We argue that there are situations, such as detecting an intelligent network intrusion, where *some* information but with reliably high detection probability is needed. As shown in Figure 1, our approach uses deep CNN trained on a cluster of NVIDIA Tesla V100 GPUs, but then ported to multiple different Android smartphone devices. Figure 2 shows the structure and layers of the CNN (described in more details in the following sections). The CNN model is compressed and then executed in the TensorFlow Lite environment [12].

Our main contributions are as follows:

- We identify how flexible labeling strategies for modulation classification based on perceived SNR conditions can boost the detection probability of that *label* category (e.g., transitioning flexibly from 24 distinct labels of modulations in high-SNR to 4 labels in low-SNR conditions). We propose an automated label-assignment algorithm that uses raw I/Q samples in conjunction with CNN for training. [Section III]
- We demonstrate, with the help of TensorFlow Lite, how smartphones can be enabled as the capable edge platforms for implementing deep CNNs for several wireless applications, such as SNR detection and modulation classification. [Section IV]
- We experimentally show that not only our algorithm improves prediction accuracy for a reduced number of labels for low SNR conditions up to 93%, but also using smartphones as the classification engine results in 870x speedup in time compared to GPUs. Interestingly, we achieve this without reducing the accuracy or consuming extensive power (less than 2W including screen and WiFi module power). [Section V]

We discuss related work in Section II, limitations, and future research directions in Section VI, and conclude in Section VII.

II. Related Work

Modulation classification: Understanding the wireless environment is a first step towards building an intelligent radio that has myriad of military, day-to-day civilian applications, as well as use-cases in first-responder networks where dissimilar protocols/agencies operate in a shared space [13]. The most common scenarios include device authorization and network anomaly detection that can be done either at the physical or the link layer [14] [15]. Signal analysis with modulation classification is a way to ensure only authorized transmitters are present, and apart from more classical, ground-based networks, such techniques have also been demonstrated for flying UAVs [16]. The goal of classifying users using physical layer raw I/Q samples has been attempted in the state-of-the-art works [5] [10] [11], from pure modulation classification to RF device fingerprinting.

Learning the spectrum: Deep learning schemes for wireless networking have emerged in the last few years given the abundance of data available today [17] [4] [18]. Such techniques are used in data aggregation, localization, interference management, performance optimization of mmWave communication, and spectrum management. Our focus is to classify the modulation schemes for better spectrum management. Current deep learning solutions result in low accuracy ($\approx$ 10%) in noisy (SNR<0 dB) environments [10]. Hence, through this work, we aim to improve the detection accuracy for low-SNR regions.

Bringing spectrum learning to the edge: Customized edge platforms with micro-architectures for deep learning are previously proposed for classification on FPGAs [19] [20]. This platforms can be used for classifying streaming data in different applications ranging from wireless signal processing to image processing. Edge solutions for classification of wireless signals is proposed by Restuccia and Melodia [21]. They develop an FPGA implementation of a CNN for two example applications – modulation classification (for 5 classes) and detecting number of FFT points (for 3 classes). FPGA-based approaches are good candidates for implementing deep learning architectures because of their high performance. However, designing customized architectures for deep learning using hardware description languages such as, VHDL, and Verilog requires

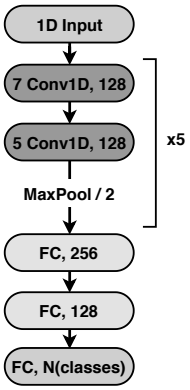


Fig. 2: Baseline CNN for modulation classification

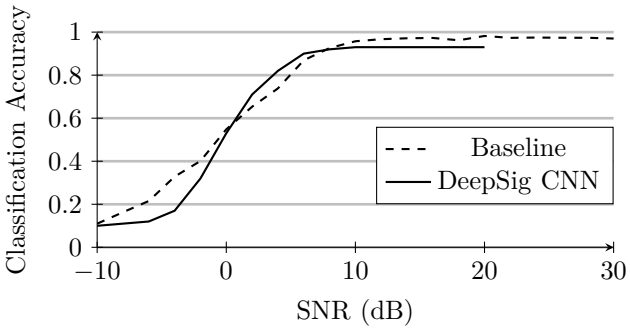


Fig. 3: Classification accuracy for our baseline and DeepSig CNN. Note that for both classifiers average accuracy remains <24% for SNRs<0 dB.

a lot of effort. Vivado HLS, which is a high level synthesis tool and obviates VHDL/Verilog designing, also needs the architecture to be described in C++, which is still a step away from the algorithm described in python. Therefore, the need for a candidate platform that minimizes the step between algorithm and edge implementation, is sensed.

Learning on smartphones: Smartphones are now commonly used as a learning platform for a variety of use-cases, such as, on-device QnA [9], personalized text input [22], and image processing [23]. Google has a software framework for running deep learning algorithms on embedded devices called as TensorFlow Lite [12]. TensorFlow Lite consists of a set of tools that allow execution of deep learning algorithms in the constrained environments including smartphones. Python APIs provided by TensorFlow Lite optimize the size of model binaries to efficiently execute on the low-end edge devices. We leverage this tool to port our deep learning algorithm on Android smartphones. We believe this is the first experimental work on wireless modulation classification at the edge using smartphones.

III. SNR-Based Modulation Classification

We show in this section how low-SNR regions pose a significant challenge in modulation classification, as well as present our approach towards implementing a deep learning CNN architecture. Then, we explain how the SNR itself can be an input towards grouping modulation labels together so that the confusion across groups is minimized.

A. Classification problem in the low-SNR region

We start with a deep CNN classifier, shown in Figure 2 to improve the classification accuracy in the low-SNR region.

This network, henceforth called as *baseline*, consists of multiple convolutional layers, each with 128 one dimensional filters of sizes of 7×1 or 5×1.

For purposes of comparison with prior work and for repeatability, we choose the *difficult* dataset provided in [10]. The dataset has 24 classes of modulation, shown in Table I, each having SNRs ranging from -10 to 30 dB with increments of 2 dB. Each modulation/SNR pair has 4096 sequences of each 1024 I and Q samples. The dataset thus contains over 2 million examples of `<I/Q, SNR, Modulation>` with the labels (prediction classes) as the modulation schemes. We partition the dataset into 60%, 20%, and 20% train, validation and test sets, respectively. We train and test our baseline architecture shown in Figure 2 with this dataset using Keras libraries with TensorFlow backend. Figure 3 shows classification accuracy for the baseline architecture. We compare the accuracy with the results from [10] shown as DeepSig CNN in Figure 3. We observe that our classification accuracy is slightly better than the DeepSig CNN, however more importantly, classification accuracy is poor at low SNRs for both models. On average modulation classification accuracy is $\approx 24\%$ when SNR is below 0 dB.

To gain a deeper insight on the issues impacting accuracy, we plot confusion matrices for each individual SNR<10 dB. Such a representation helps us better identify the specific classes that are being confused with each other, and ultimately drop the overall accuracy.

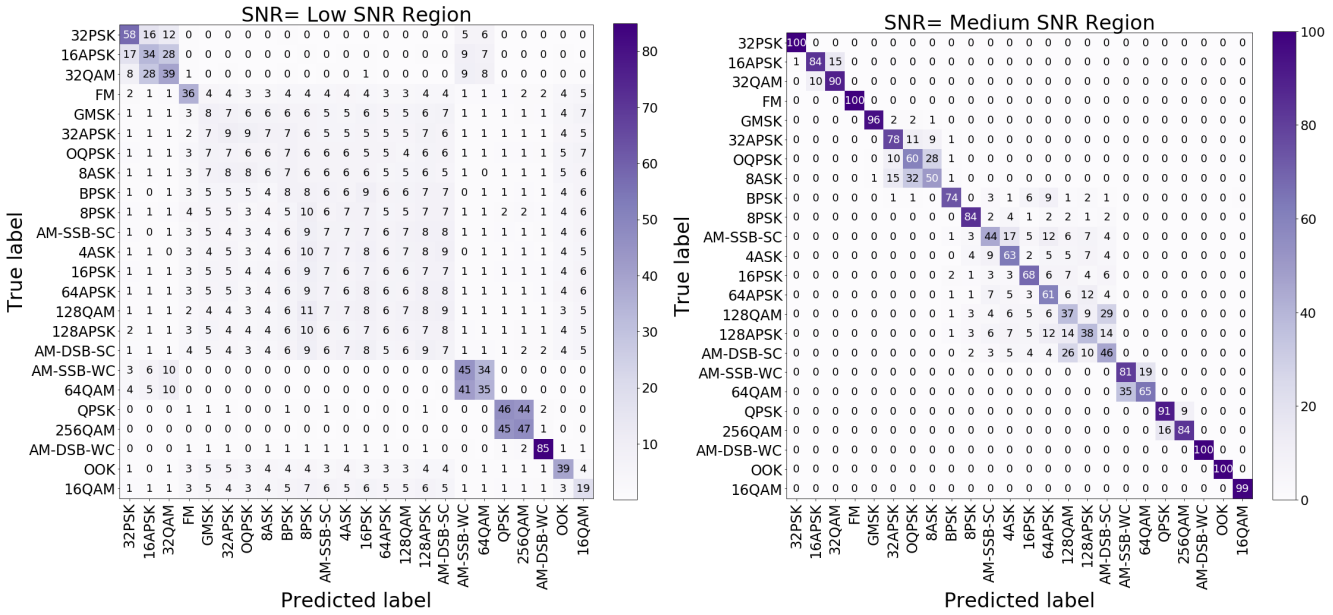
By studying the confusion matrices for each SNR level, we see how they demonstrate similar patterns in the confused classes within each SNR range of [-10,-2] dB, [0,8] dB and [10,30] dB. The difference of 2 dB for each range comes from 2 dB increment between values in the original dataset. We define these ranges as Low, Medium and High SNR regions, respectively.

In the next step, average confusion matrices for Low and Medium SNRs are plotted and shown in Figures 4a and 4b. The number in each element shows the fraction of test sequences where the two labels are confused with each other (multiplied by 100). For example, in Figure 4a the value in intersection of row 256QAM and column QPSK shows 45, which means the true label 256QAM is confused with QPSK in 45% of test sequences. Note that, we do not show the matrix for High SNR as we get considerably high, $\approx 97\%$, classification accuracy for that region.

Based on this observation, we regroup the fine-grained

TABLE I: Class to modulation scheme mapping

Class	1	2	3	4	5	6	7	8	9	10	11	12
Modulation	32PSK	16APSK	32QAM	FM	GMSK	32APSK	0QPSK	8ASK	BPSK	8PSK	AM-SSB-SC	4ASK
Class	13	14	15	16	17	18	19	20	21	22	23	24
Modulation	16PSK	64APSK	128QAM	128APSK	AM-DSB-SC	AM-SSB-WC	64QAM	QPSK	256QAM	AM-DSB-WC	OOK	16QAM



(a) Low SNR region

(b) Medium SNR region

Fig. 4: Confusion matrices that show the percentage of testing set where a particular modulation class is confused with another class. In an all-correct classification, all highlighted cells will be on the diagonal.

individual modulation schemes into course-grained aggregated modulation schemes to help the neural network detect groups of modulation schemes with a high accuracy.

B. Compromising detection resolution for higher accuracy

We aim to improve the accuracy of modulation detection by compromising the finer per-modulation classification. Hence, when we regroup the modulation schemes, we assign them our custom labels. For example, Figure 4a shows that 256QAM and QPSK are generally confused with each other, while they are rarely confused with other classes. Therefore, we group these two classes into one single class. The grouping and re-labelling helps the CNN to more easily classify the signal.

To automate the grouping process, we develop an approach described in Algorithm 1. We run the algorithm separately for each SNR region of Low and Medium. The inputs of this algorithm are the confusion matrices shown in Figures 4a and 4b, and the output is *cluster_list* that gives the new grouping with multiple modulation labels fused together into a *single* class for the given SNR region.

In our algorithm, to define new classes for each SNR region, first, independent labels that are easily distinguished from others are added to a *cluster_list*. They form clusters with only one member. Following this, 2-4 member clusters are formed and added to a *candidate_list* along with a `sum` value. The `sum` value is the summation of elements in the confusion matrix for all the members of the cluster and shows how strongly these labels were confused. Once we have the *candidate_list* ready, the clusters inside it compete based on the `sum` value to become a final class. The winning clusters are the ones with larger `sum` values. If two clusters have mutual member and the same `sum` value, the one with fewer members wins, to provide us higher detection resolution. In our algorithm, e_i is an element in confusion matrix and L_i is the true label associated with e_i. Variable *acc* shows the lower limit of overall expected accuracy after re-labelling. For our evaluation, we set it as 90%. Additional variables such as $margin = 5$ imply that if there is an element $e_i \geq 85 (= 90 - 5)$ and there is no other elements in that row ≥ 5, then the true label L_i associated with that element e_i will be added to the *cluster_list*. Variables *acc* and *margin* can be tuned by the user as a trade-off between detection accuracy and resolution.

The *cluster_list* output of Algorithm 1, is a list of new labels that is composed of clusters of old labels for each SNR range. For the given dataset that we use in this paper, the labels obtained are as shown in Figure 5. We note, however, that our approach is generic in nature and will form different label groups when given other datasets.

Figure 5 shows that as the output of re-labelling algorithm, I/Q sequences in Low-SNR region can now be identified with 4 labels, which is a hierarchical grouping of

Algorithm 1 Re-labelling Algorithm

Set acc; Set margin;
for all row in conf_matrix **do**
  for all e_i in row **do**
    if $e_i \geq (acc + margin)$ **then**
      cluster_list.append(L_i)
    else if $e_i \geq (acc - margin)$ **and** no e_n in row≥ 5 **then**
      cluster_list.append(L_i)
    end if
  end for
  ignore $e_i \leq 5$
  sort e_is in descending order as $e_i, e_j, e_k, e_m, ...$
  if $sum(e_i, e_j) \geq acc$ **then**
    candidate_list.append([L_i,L_j, sum])
  else if $sum(e_i, e_j, e_k) \geq acc$ **then**
    candidate_list.append([L_i,L_j,L_k, sum])
  else if $sum(e_i, e_j, e_k, e_m) \geq acc$ **then**
    candidate_list.append([L_i,L_j,L_k,L_m, sum])
  end if
end for
Sort candidate_list with descending sums
for all clusters in candidate_list **do**
  if $sum \geq acc$ **then**
    if two clusters have same sums **and** have mutual members **then**
      cluster_list.append(cluster with fewer members)
    end if
    if None of members are in candidate_list **then**
      cluster_list.append(cluster)
    end if
  end if
end for

the old labels. Similarly, Medium-SNR region is identified with 12 labels and High-SNR region with 24 labels (as before).

Following the grouping process, the CNN architecture in Figure 2 is re-trained independently, in our case with 3 new datasets (each associated with an SNR range and each with new labels) at the output layer. Thus, we have 3 separately trained models, named `Model_High`, `Model_Med` and `Model_Low`, each for one specific SNR region. Classification accuracy obtained on GPU testing for these models is shown in Figure 6. The plot shows average accuracies as 94%, 93% and 97% in three regions of Low, Medium, High SNR, respectively. Such higher accuracies are achieved at the expense of lower resolution in detecting the modulation scheme compared to classical baseline CNN.

To summarize, for SNR<10 dB using the baseline method, the classifier returns one of the 24 modulation schemes in Table I as the predicted label, with a very high (up to 90%) chance that the prediction is incorrect.

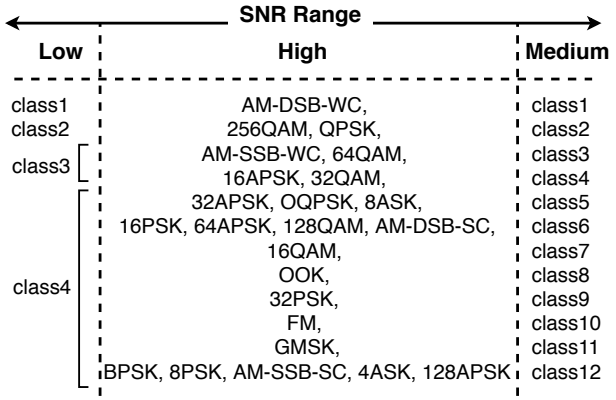


Fig. 5: New classes after re-labelling. For High-SNR region, each modulation scheme is a label (class)

The SNR-based re-labeling scheme reduces the resolution of detecting 24 different modulation schemes for SNR<10 dB, but in a way that the predicted label is correct with up to 94% probability.

C. SNR-level based modulation classification

In order to select the proper model among `Model_High`, `Model_Med`, or `Model_Low`, we need an automated method to detect SNR level from raw I/Q samples. Figure 7 conceptualizes this flow. We describe our approach of automated SNR detection as follows.

1) SNR-level detection: In a deterministic approach for measuring SNR, first an estimation of the noise floor is acquired, in a specified bandwidth that requires collecting I/Q samples when no information signal is being transmitted. Then, the absolute power level of the information signal is measured and its ratio with estimated noise power is taken. This deterministic approach generates an exact number as SNR value.

Note that our approach does not require an exact SNR computation; but only needs to detect which of the 3 classes of Low, Medium or High is a better descriptor of the observed SNR level. Thus, we do not need to measure the noise floor if we can simply obtain these labels from the signal itself. Towards this goal, we train and test on GPUs, a shallow neural network architecture shown in Figure 8 with the same dataset described in III-A, this time with SNR labels instead of modulation labels. Our neural network distinguishes between 3 SNR classes – Low, Medium, High with an accuracy of 99.9%. We refer to this model as `Model_SNR` through the rest of the paper.

Next, we discuss the method of porting the trained models on Android smartphones.

IV. System Implementation

The process of SNR-based modulation classification described in the previous section, must now be ported for an edge implementation. This is complicated given the

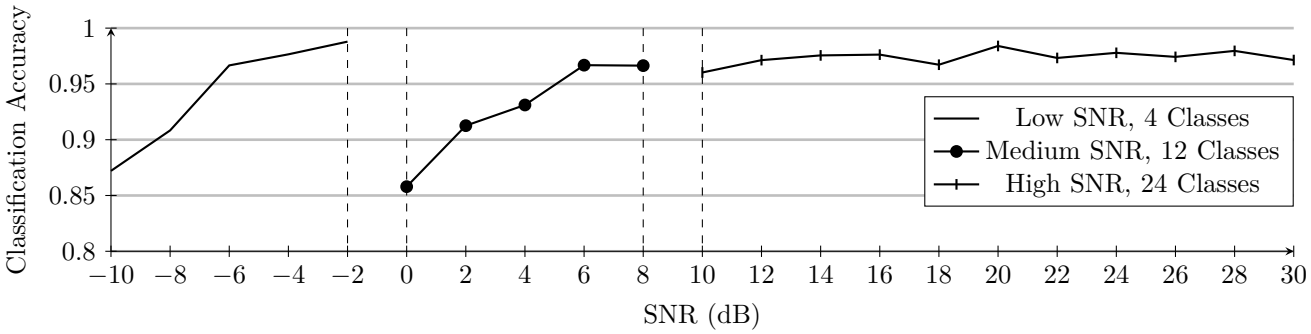


Fig. 6: Classification accuracy after re-labelling for each SNR level has a minimum of 86%

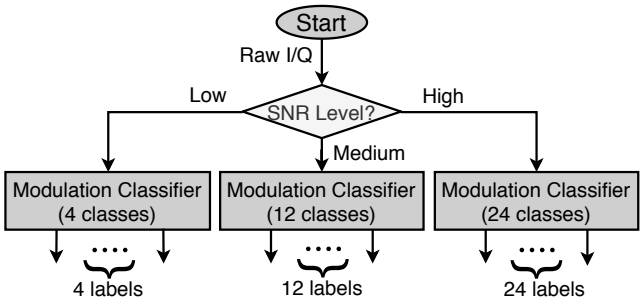


Fig. 7: Modulation classifier selection based on SNR level

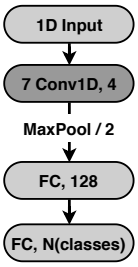


Fig. 8: Proposed CNN architecture for SNR classification

memory size required for multiple models and timing constraints, which makes edge implementation difficult without frameworks that compress models. Thus, we choose Google's framework for Android devices, TensorFlow Lite, as a possible solution.

A. From TensorFlow to TensorFlow Lite

To transfer a model from the laboratory environment to the smartphone, we first train models on regular GPUs using Keras libraries with TensorFlow backend. Next, these trained models are saved in an *.hdf5* files that contain the neural network architecture, the optimizer configuration, and model weights. Then, these trained models are cross-compiled for Android, using TensorFlow Lite converter. The result is a file with *.lite* extension. Cross compilation using TensorFlow Lite converters results in a compression process that stores the model as a flat buffer, which allows serializing data without unpacking/parsing at the time of de-serialization [24]. The *.lite* file is used in Java code

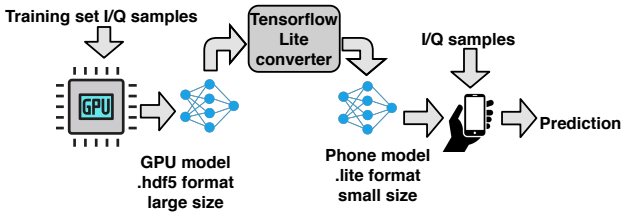


Fig. 9: Converting a model from TensorFlow to TensorFlow Lite

on an Android device, as described next. Figure 9 shows the process of a model trained by regular TensorFlow on GPUs, and then converted to a TensorFlow Lite classifier on an Android device.

B. Executing classifiers via Android app

After the TensorFlow Lite file is generated, we begin the process of porting the classifier to the edge device. TensorFlow Lite libraries make it possible for the Android device to interpret and operate on the model contained in the *.lite* file. An *Interpreter* driver class creates a high level Application Programming Interface (API) for the user to run the model. We create an Android app using Android Studio 3.3.2. App execution needs – *.lite* models, the labels associated with each model – *Label.txt*, and the test set of raw I/Q samples – for the purposes of evaluation. We provide files corresponding to models and labels to the app through it's assets folder. Files containing raw I/Q samples will be saved on a general folder in internal memory of the phone. When the app is launched, the user is asked through GUI to select an input file that contains the dataset of raw I/Q samples. In the background, all model files in the assets folder are loaded and mapped to the memory of the device. Next, a *Lite Classifier* object is created for each model using the *Interpreter* driver class. The input file is memory-mapped to a flat buffer, as needed by the Lite Classifier object. When the flat buffer is fed to the Lite Classifier Object, a probability vector is generated whose length is equal to the number of classes in the model. This tells us the probability of each predicted label, with the sum of the elements in the probability vector

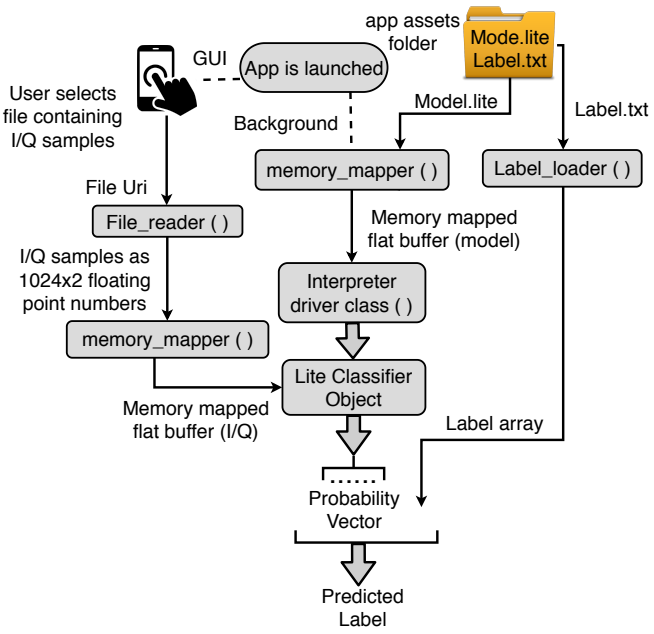


Fig. 10: Sequence of function calls on the smartphone, from loading the target file containing I/Q samples to the final predicted label by the classifier.

equal to one. Taking the simple maximum of the elements in the probability vector gives the prediction decision of the model for the input sequence. Depending on the index of the maximum, the corresponding label from *Label.txt* file is reported as the predicted label. Figure 10 shows this process for an example model.

V. Evaluation

In this section we first describe our evaluation methodology and provide the hardware and software details of our platforms. Next, we evaluate and compare our GPU and smartphone models against model size, classification time, given that both return similar detection probability. We also report power consumption on the smartphone.

A. Methodology

We implement our design on 3 smartphone devices shown in Table II, and we present our results in the next subsection. The results are compared with GPU implementation that runs on NVIDIA Tesla V100 with 640 Tensor Cores and 100 teraFLOPS.

TABLE II: Specifications for the three smartphones of LG, HTC and Lenovo used in our experimental study.

	LG	HTC	Lenovo
Model	K20	Desire 610	Note K3
Android version	8.1.0	4.4.2	6.0
Processor	MSM8917 Quad-core 1.4 GHz	1.2 GHz Quad-core Cortex-A7	MT6753 8-core 1.7 GHz
RAM	2 GB	1GB	2 GB

TABLE III: Number of parameters in trained models

Model	Number of trained parameters
`Model_SNR`	262,719
`Model_Low`	1,953,668
`Model_Med`	1,954,700
`Model_High`	1,956,248

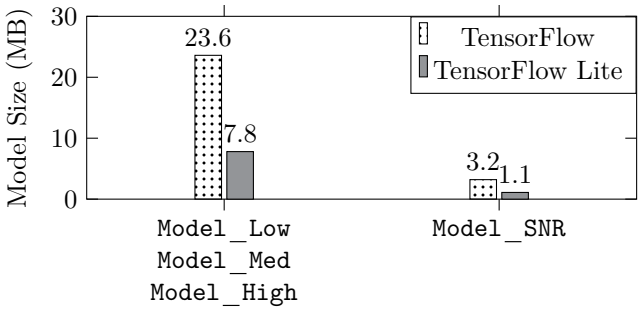


Fig. 11: Memory size of the model on the GPU and the smartphone. Compressed model, with TensorFlow Lite, occupies $\frac{1}{3}$ of the original model.

Consider the CNNs for SNR and modulation classification described in earlier sections and ported to the smartphone. We use the term *number of parameters*, as shown in Table III, as a representative measure of the model size and the number of operations that will be done at classification time. `Model_Low`, `Model_Med` and `Model_High` have the same architecture but different number of classes in the last layer. This results in slightly different number of parameters.

B. Results

We present quantitative results for our metrics here.

1) Model size: Model size is the memory space occupied in the device. We recall that TensorFlow Lite conversion compresses the original model as a special format in a flat buffer, as explained in Section IV-A. The size of the compressed models in our case is approximately $\frac{1}{3}$ that of the original Keras model. We show a comparison in Figure 11. `Model_Low`, `Model_Med` and `Model_High` have the same size because of having the same architecture.

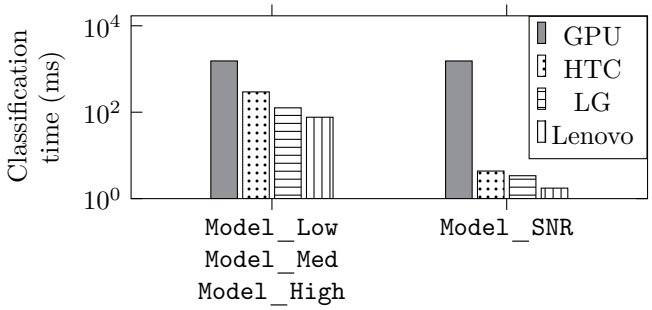
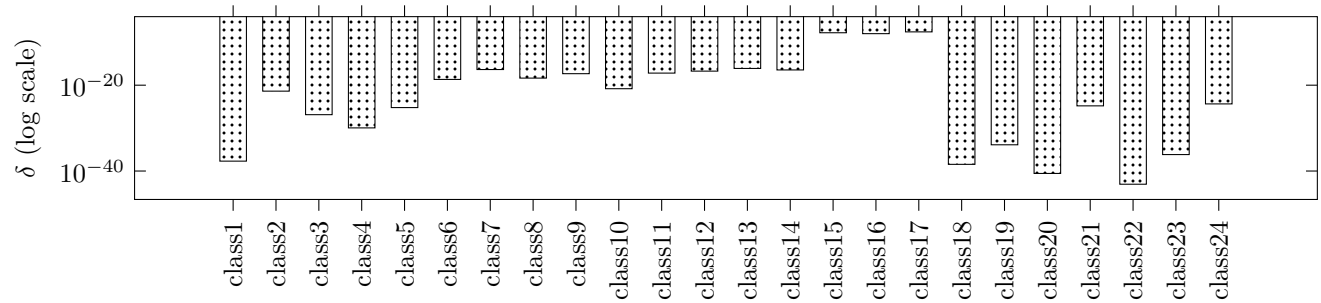
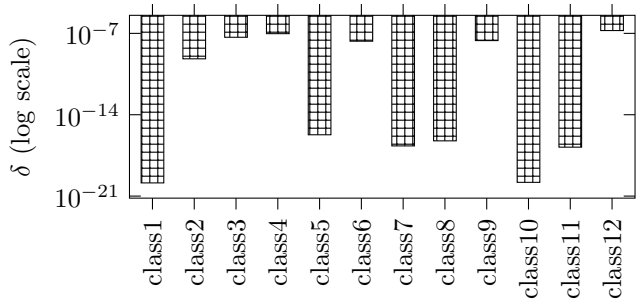


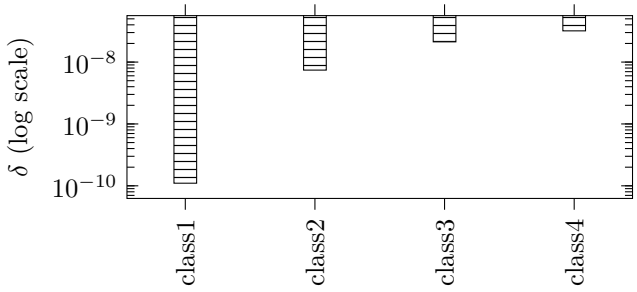
Fig. 12: Classification time shows up to a 870x boost in smartphone compared to GPU



(a) High-SNR range with classes from Table I



(b) Medium-SNR range with classes from Figure 5



(c) Low-SNR range with classes from Figure 5

Fig. 13: Average of the difference between probability vectors in GPU (Keras) model and smartphone model per class

2) Classification time: The time taken by the classifier on any platform to classify a single I/Q sequence from the instance that sequence is fed to it, to the instance the probability vector is generated is called *classification time*. This time is measured by feeding in the same I/Q sequence to the model on GPU and on smartphones 10 times to smooth out the impact of interrupts on the operating system.

The mean classification time on different platforms is shown in Figure 12. Here, y-axis has log-scale to better show the large difference between classification time on the platforms. We observe a boost ranging from $\frac{1533}{295} \approx 5$x to $\frac{1533}{1.75} \approx 870$x in classification time compared to GPU.

The reason for the shorter classification time is that GPUs are massively parallel, and their architecture is perfect for working on large batch sizes of data, e.g. for training process or testing on a large batch, however, as edge platforms work on streaming data (only one sequence or frame of the input) GPUs have a considerably lower performance. Classification time for each of our 4 models on the GPU is significant, regardless of the size of the model. This is due to massive under-utilization of GPU resources in all these cases.

3) Detection probability vector comparison as a measure of accuracy: We obtain classification accuracy on GPUs by testing large batch sizes of data in the classification phase. For each sequence, we save a predicted label at the end of this phase. After that, we compare the true label against the predicted label and decide whether or not that sequence is predicted correctly. For measuring accuracy, we divide the number of correctly predicted sequences by the total number of sequences in the test set. On edge devices, however, the same process might not be possible, because of constraints of memory on the device. Therefore, we explore other alternatives for checking how accurately deep learning at the edge device is performing. Thus, instead of directly measuring accuracy on the smartphone, we compare detection probability vectors between the smartphone and the GPU models. Note that such a vector is the output of the CNN, whose length equals to the number of classes and whose sum of elements equals to 1. Probability vector tells us the probability of prediction of each class for a single input sequence.

We feed 100 different I/Q sequences to our modulation classifiers on the GPU and smartphone, and we save the probabilities. We calculate the difference between probabilities on GPU and at the edge for each class using Equation 1.

$$\delta_i = \frac{1}{N} \sum_{j=0}^{N-1} |p_{i,j} - q_{i,j}| \qquad i = 0, 1, ..., L-1 \qquad (1)$$

Where L is the number of classes in each SNR range, $N = 100$ is the number of sequences used for testing and $p_{i,j}$ is element i from probability vector associated with sequence j on the GPU and q is similar vector on the phone. Equation 1 leads to a δ between probability vectors per class. For High-SNR range we have $\delta_1, ..., \delta_{24}$ each corresponding to a class. For Medium-SNR range and Low-SNR range, we have $\delta_1, ..., \delta_{12}$ and $\delta_1, ..., \delta_4$, respectively. These δ_is can be illustrated as Figure 13a,

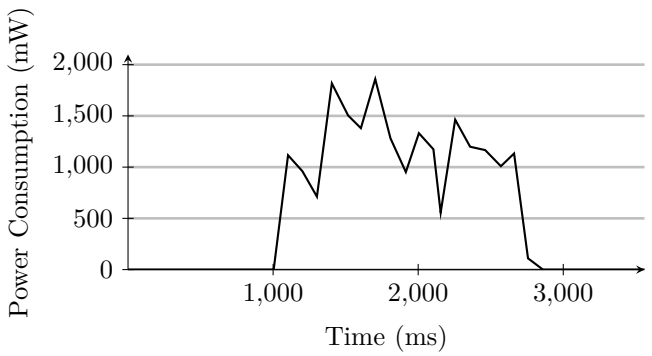


Fig. 14: Power consumption snapshot for TensorFlow Lite on Lenovo phone

13b and 13c for `Model_High`, `Model_Med` and `Model_Low`, respectively. Theoretically the δ would be very small, as seen in practice. This is because the compressed model is not quantized and it is still a floating-point model.

4) Power Consumption: We use Trepn power profiler [25], provided by Qualcomm, to estimate the power consumption on Snapdragon processors. As advised by Trepn documentation [26], to measure app power, we put the profiler in delta power mode, run the profiler and then we choose our classifier app. In this manner, the power consumption level before launching the app would be considered as the baseline power consumption of the phone. In our test, we have fed to the classifier, 10 sequences of 1024 I/Q samples. Figure 14 shows a snapshot of delta power consumption graph for the Lenovo phone. The achieved power consumption for the learning operation on the phone are comparable with the state-of-the-art [27]. Note that the majority of the power consumed is due to the screen and WiFi module, while the classifier takes minimal time.

To calculate the average, we add all the non-zero points in graph of Figure 14 and divide the sum by 10 which is the number of times the classifier is run in our test. The average power consumption is 1964.2 mW. This number includes the power consumed for all the app stages described in Figure 10, not only the stage running Tensorflow Lite.

VI. Discussion and Future Work

Our study revealed some non-intuitive results. For instance, we observe that the classification process is faster on smartphones than GPUs, mainly because, unlike GPUs, smartphones are not customized for operating on large batch sizes of data, which needs specialized hardware resources. Moreover, TensorFlow Lite makes the CNN model compact enough to be ported to a smartphone and the model execution is not memory and battery intensive. Although, we have made concrete forays in this rich area of research, there remain interesting open problems to be solved. We identify those problems that will require collective efforts from the community to address them.

Real-time I/Q Samples: In its current shape, the I/Q samples are read from a local file stored on the smartphone. In a practical end-system, the phone should be able to record live I/Q samples from the spectrum and detect the modulation classification in real-time. We are working on a more advanced version of the presented framework that integrates the real-time processing capabilities [28].

Detailed resource analysis: Our focus in this paper is to demonstrate experimentally that the concept of an intelligent modulation classifier at the edge is feasible. A deeper evaluation is needed that helps analyze the performance of model in varied platforms across operating system, device memory, computation power, and battery consumption. To that end, we are experimenting with different smartphones that include both Android and iOS versions.

Model robustness: The baseline CNN used in this paper is based on the dataset collected in a particular environment. While it may help benchmarking with other results, we still do not know how this model will perform when exposed to numerous other conditions with a multitude of heterogeneous device vendors. There is a possibility of channel impairments affecting the accuracy of model. Ideally, an apt model is the one that can automatically adapt to its environment. An in-depth experimental evaluation is needed to answer these questions and develop a robust learning that is independent of any channel effects.

VII. Conclusion

In this paper we proposed an SNR-based classification scheme to characterize modulation codes for the purpose of spectrum management. We proposed a new convolutional neural network for the modulation classification task and trained the network with data labelled in a novel manner. The labelling mechanism is derived with the insights about modulation classes learned from the confusion matrices. We validated the proposed mechanism and the models empirically with multiple Android smartphones running TensorFlow Lite framework. Our implementation showed considerable boost in classification time, without any loss of accuracy as compared to the same neural network running on a GPU.

Acknowledgement

This work is supported by DARPA under the Young Faculty Award grant N66001-17-1-4042. We are grateful to Dr. Tom Rondeau, program manager at DARPA, for his insightful comments and suggestions that significantly improved the quality of the work.

References

[1] H. Cui, V. C. M. Leung, S. Li, and X. Wang, "Lte in the unlicensed band: Overview, challenges, and opportunities," *IEEE Wireless Communications*, 2017.

[2] A. Saeed, K. A. Harras, E. Zegura, and M. Ammar, "Local and low-cost white space detection," in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[3] FCC, "White Space - Federal Communications Commission." https://www.fcc.gov/general/white-space.

[4] S. C. Hauser, W. C. Headley, and A. J. Michaels, "Signal detection effects on deep neural networks utilizing raw iq for modulation classification," in *IEEE Military Communications Conference (MILCOM)*, 2017.

[5] K. Sankhe, M. Belgiovine, F. Zhou, S. Riyaz, S. Ioannidis, and K. Chowdhury, "Oracle: Optimized radio classification through convolutional neural networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[6] S. Riyaz, K. Sankhe, S. Ioannidis, and K. Chowdhury, "Deep learning convolutional neural networks for radio identification," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 146–152, 2018.

[7] T. Banerjee, K. R. Chowdhury, and D. P. Agrawal, "Using polynomial regression for data representation in wireless sensor networks," *International Journal of Communication Systems*, vol. 20, no. 7, pp. 829–856, 2007.

[8] M. F. S. by PEW Report, June 2019.

[9] Q. Cao, N. Weber, N. Balasubramanian, and A. Balasubramanian, "Deqa: On-device question answering," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 27–40, ACM, 2019.

[10] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[11] T. J. O'shea and N. West, "Radio machine learning dataset generation with gnu radio," in *Proceedings of the GNU Radio Conference*, 2016.

[12] Google, "TensorFlow Lite." https://www.tensorflow.org/lite.

[13] M. Zheleva, R. Chandra, A. Chowdhery, A. Kapoor, and P. Garnett, "Txminer: Identifying transmitters in real-world spectrum measurements," in *IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2015.

[14] Q. Xu, R. Zheng, W. Saad, and Z. Han, "Device fingerprinting in wireless networks: Challenges and opportunities," *IEEE Communications Surveys Tutorials*, 2016.

[15] J. Hua, H. Sun, Z. Shen, Z. Qian, and S. Zhong, "Accurate and efficient wireless device fingerprinting using channel state information," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[16] D. Zhang, W. Ding, B. Zhang, C. Xie, H. Li, C. Liu, and J. Han, "Automatic modulation classification based on deep learning for unmanned aerial vehicles," *Sensors*, 2018.

[17] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE Transactions on Cognitive Communications and Networking*, 2018.

[18] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys Tutorials*, 2019.

[19] J. Sanchez, N. Soltani, P. Kulkarni, R. V. Chamarthi, and H. Tabkhi, "A reconfigurable streaming processor for real-time low-power execution of convolutional neural networks at the edge," in *International Conference on Edge Computing*, pp. 49–64, Springer, 2018.

[20] J. Sanchez, N. Soltani, R. Chamarthi, A. Sawant, and H. Tabkhi, "A novel 1d-convolution accelerator for low-power real-time cnn processing on the edge," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2018.

[21] F. Restuccia and T. Melodia, "Big data goes small: Real-time spectrum-driven embedded wireless networking through deep learning in the rf loop," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[22] M. Xu, F. Qian, Q. Mei, K. Huang, and X. Liu, "Deeptype: On-device deep learning for input personalization service with minimal privacy concern," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2018.

[23] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017.

[24] "Flat Buffers." https://google.github.io/flatbuffers/flatbuffers_white_paper.html.

[25] "Qualcomm trepn power profiler." https://developer.qualcomm.com/software/trepn-power-profiler, 2018.

[26] "Qualcomm trepn power profiler questions and answers." https://developer.qualcomm.com/software/trepn-power-profiler/faq, 2018.

[27] B. Zhou, J. Lohokare, R. Gao, and F. Ye, "Echoprint: Two-factor authentication using acoustics and vision on smartphones," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.

[28] M. Schulz, *Teaching Your Wireless Card New Tricks: Smartphone Performance and Security Enhancements Through Wi-Fi Firmware Modifications*. PhD thesis, Technische Universität, 2018.