2ND YEAR INTERNSHIP REPORT

2018 INTERN SOFTWARE ENGINEER - MACHINE LEARNING

# ArmNN: Software interface for high-performance neural network inference on Arm hardware

Morgane ROUVROY
2nd year - Specialization MMIS

4 June 2018 - 24 August 2018 (12 weeks)

**Arm Ltd**
110 Fulbourn Road
Cambridge, CB1 9NJ
United Kingdom

**Internship Supervisor**
Matthew BENTHAM
**Ensimag Reviewer**
Wojciech BIENIA

## Acknowledgements

The internship opportunity Arm gave me was a great chance for learning and professional development. I consider myself as very lucky to have been be a part of it. I am also grateful for having a chance to meet so many wonderful people who welcomed and helped me though this internship period.

I would like to give special thanks to my manager, Matthew Bentham, who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on the correct path and allowed me to carry out my tasks as best as possible.
I'm also very thankful for the whole ArmNN team for their welcome and advice on my work. In particular, thanks to David Beck for his patience and helping me see the value of unit testing.

Last but not least, an internship is not only about work and I was lucky enough to meet and make great friends with some of the others interns.
Thank you all !

## Abstract

This report describes my summer internship in the Cambridge (UK) headquarters of Arm, from the 4th of June 2018 to the 24th of August 2018.
Arm is a leading semiconductor IP company and has lastly been developing its Machine Learning branch to enhance inference experience on all its hardware. As part of this division, I was assigned to the ArmNN project, an open-source translation interface from leading Neural Network frameworks to Arm GPUs and CPUs.
In the setting of this project I could develop a whole new feature for the software : integrating a new language standard for Machine Leaning models to the interface so that ONNX models could be used on Arm hardware. I also participate to the team day to day activities and duties by performing small tasks needed for the release, like peer reviewing and android cross-compilation development.
By the end of this internship, two ONNX models - MNIST and MobileNet_v2 - were fully supported in ArmNN and giving correct predictions.

# Contents

# 1. Introduction

Incorporated to the ENSIMAG engineering program, the Assistant Engineer internship's main aim is to introduce students to the main skills of an engineer. This moment allow students to know not only the theoretical part of their study but also the practical one. This internship is also an occasion to refine their professional project : field of application, type of company (start-up, big company, small companies) or location.

Arm Ltd proposed this year (2018) an internship in its Machine Learning team in Cambridge to work on one of their open-source project : ArmNN. This project aims at giving users all over the world an easy interface to run inferences from Machine Learning models on Arm hardware.

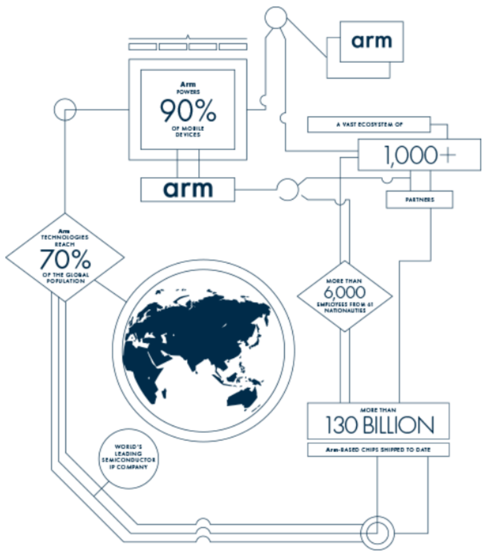
After giving some information on this internship context - company, division... -, we will detail the main objectives for the release, along with the used methodology. Finally, we will have a look at the tasks completed during these three months.

# 2. Internship environment

## 2.1 Arm

Arm is a British world's leading semiconductor IP (Intellectual Property) company. Unlike many other companies of this field, Arm creates and licenses its technology as IP rather than manufacturing and selling it themselves. Thus, each product containing Arm technology sold allows Arm to touch royalties, adding up to the license selling benefits.

A striving feature of Arm technology is its low electric power consumption and memory usage optimization, making it particularly useful for portable devices.


arm
Arm POWERS 90% OF MOBILE DEVICES
arm
A VAST ECOSYSTEM OF
1,000+
PARTNERS
Arm TECHNOLOGIES REACH 70% OF THE GLOBAL POPULATION
MORE THAN 6,000 EMPLOYEES FROM 61 NATIONALITIES
MORE THAN 130 BILLION
Arm-BASED CHIPS SHIPPED TO DATE
WORLD'S LEADING SEMICONDUCTOR IP COMPANY

Arm impact worldwide[1]

## 2.2 The Machine Learning Division

As Artificial Intelligence spreading creates a new kind of innovation, with new features like speech recognition, predictive photography or predictive text, Arm technologies needs to adapt. Accordingly, the Machine Learning team was created in 2017, with the main purpose of allowing Machine Learning development on Arm-based products.

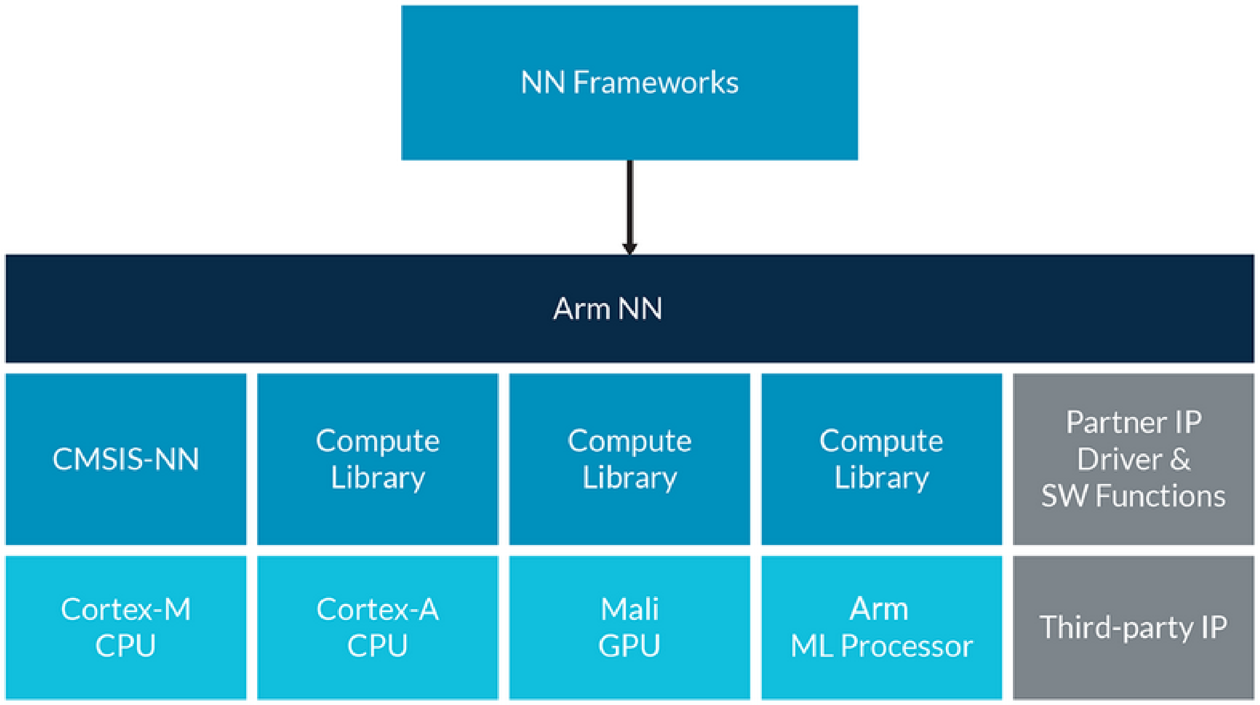
As of today, the team is based both in Cambridge (UK) and Galway (Ireland) and counts over 50 members, from software engineers to performance analysts.

The team projects include low-level, optimized Machine Learning and Neural Network functions for CPU and GPU hardware but also a software interface in between these functions and existing frameworks, named ArmNN.

## 2.3 Work context

As Machine Learning evolves, so do frameworks and hardware needs. Yet, each one has its own standards and target different platforms. In this context, ArmNN software interface acts as a bridge in between existing neural networks frameworks - like TensorFlow or Caffe - and the processing hardware, making it easy for the programmer to continue using his favourite framework while still developing high performance Neural Networks for mobiles and embedded Linux devices.

More, as ArmNN relies on the Arm Compute Library (set of low-level, optimized functions for ML and NN), the programmer also gets an optimized version of its graph, resulting in performance improvement - 15x or more over equivalent OpenCV functions [2]-.



ArmNN overview [3]

# 3. ArmNN project

## 3.1 Objectives

### 3.1.1 For release 18.08

As part of the ArmNN team, I worked on release 18.08 of the software - 18.05 was released one week before my arrival- . ArmNN 18.08 release main features are :

1. **Marvin support** :
   Adding support for accelerating Neural Network inference on the Marvin NPU in ArmNN.

2. **Memory usage reduction** :
   To keep up with both Arm's policy of low memory usage and main consumers requirements - Samsung, MediaTek... -, memory improvements in Android NN driver are scheduled.

3. **LSTM operator** :
   Another important point is the integration of audio and text operator - used and tested for instance on DeepSpeech -. Among those operators, the LSTM operator should be implemented on ArmNN and the ArmNN Android driver.

4. **TFLite support** :
   As ArmNN is aiming at mobiles and embedded Linux devices, it needs to support TensorFlow's lightweight solution for mobile and embedded devices, TensorFlow Lite. It enables on-device machine learning inference with low latency and a small binary size. This means being able to parse and infer tflite files on Linux but also on Android.

5. **Android API** :
   One of the main users of ArmNN being Samsung, the interface should provide an easy way to use it on Android. More, it should also fulfill the Android requirement such as the VTS tests - set of tests one should pass before being able to be commercialized on an Android device - .

### 3.1.2 ONNX file format

Besides working to complete coding tasks for the next release (18.08), I was also tasked with the making of a whole new feature for ArmNN : supporting Onnx file format. ONNX - which stands for Open Neural Network Exchange -, is a Deep Learning file format, which goal is to allow interoperability between the mains AI frameworks - TensorFlow, Caffe2, etc. -. Supporting this spreading format in ArmNN would allow Arm to focus on writing code to support new operators and optimizations instead of doing separated parsers for each of the frameworks.

Thus, my goal was to add an Onnx parser to the ArmNN software, allowing users to use this new format. As a first end-goal, it was decided the new parser should be able to parse and run the onnx version of the Mnist model. This model, one of the most simple Neural Network, take an image of number and output the identified number.

## 3.2 Methodology

ArmNN being a wide and complex project gathering many engineers, standards and conventions had to be fixed regarding the work methodology so as everyone could work properly.

### 3.2.1 Scrum

An agile methodology - Scrum - was applied during the entire project. Thus, the project was divided in two-weeks long sprints, each with its own features and content to be added to the software. This is made possible by using many tools of the Scrum method.

#### Jira task board

Every functionality to be added during a sprint is divided into tasks and sub-tasks. One can see the different tasks and their state of completeness (open, under progress, under review...) on Jira.

#### Daily Meetings

So as to know how the project is going and if anyone needs help with its tasks, daily meetings are scheduled. Since half the team is in Galway, these meetings take the shape of 10 minutes phone conferences where every team member explains what he did the day before, his goals for the oncoming day and if he needs any help.

#### Planning Poker

At the end of each sprint, a meeting is scheduled to sum up every task done and the advancement of the project. During this meeting, new tasks are chosen and prioritized for the next sprint using planning poker. Each team member has cards and has to vote for the difficulty of the task being prioritized - 0 being trivial and 8 much more complicated -.

#### Retrospective sessions

Every 3 months, just after the release, a retrospective (or *lesson's learned* session takes place. These allow to identify changes that need to be done, in any aspect of the working life such as Communication, planning, meetings, process, infrastructure, Customer engagement etc. and ensure everyone feels empowered to make these changes.

### 3.2.2 Continuous Integration

With code being written by many person on the same time, sometimes even on the same files, continuous integration is a must. Yet, as code not compiling, or failing a basic set of tests, shouldn't be uploaded to Git master branch, verification is required.

**Gerrit and peer-reviewing**

Gerrit is a code review tool. Before pushing any code into the project Git, the code is actually uploaded on Gerrit, where it can be reviewed by other team members and finally approved for merging. This prevents uploading broken code to the project.

**Jenkins**

Jenkins is an automation server. It automates various processes, and is mainly used for building and running a set of tests on each new commit on Gerrit, on various platforms. It makes it easier finding errors and allows for faster integration and debugging.

### 3.2.3 Onnx specifics

As Onnx support was a more consequent project than the usual sprint tasks, the working methodology had to be adapted, and I was taken out of the sprint during 1 month and a half. The work was therefore divided into 3 main steps :

- **Design note** :
  Before getting into the implementation, it is important to know exactly what tasks are going to be needed and have a rough approximation of the resulting timeline. The design note, available on ARM intranet for every ML team member outlines the implementation main steps and their testing methods, based on Onnx documentation.

- **Prototype** :
  Once the design note was approved by the team manager, next step was to produce a Onnx parser prototype. While still being able to parse a Onnx file, the prototype does not need to be security proof nor very well tested. This allows to get a better grasp of how to implement each step of the design note and detect possible implementation problems - unsupported datatypes or operations for instance- . The results of the prototyping should then be summed up on the intranet to give some feedback on how implementing such feature is possible.

- **Productising** :
  Last step is productising the parser, ie convert the prototype into a finalized product and adding it definitely to the software. Each feature of the parser - initialization, parsing a specific operation.... - needs to be refactored and tested completely with relevant Unit tests. More, meaningful exceptions need to be thrown when something goes wrong or is not currently supported. The parser is thus divided into small Jira tasks, making it easier to review and test them separately.

## 3.3 Tasks completed

During this internship, I worked on many parts of the ArmNN project, allowing me to get a rather global view of the software and its features.

### 3.3.1 Improving memory efficiency

**Evaluating the impact of zero-copy operation**

Before implementing a feature, it can be useful to evaluate its efficiency on some small parts of the code. One of my tasks was to run some tests regarding the use of zero-copy method on tensors. Zero-copy is an efficient way of transferring data by using the same copy of memory between the users. Yet, this method needs memory alignment conditions so as to be usable. In this context, I had to identify which cases allowed for zero-copy use and how much memory savings it would allow, so as to see if implementing this feature was worth it.

### 3.3.2 LSTM operator

**Supporting LSTM operator in Android VTS Tests**

To ensure new content and application could run smoothly on Android, testing resources - named Vendor Testing Suite - are available and need to be runned successfully before any commercialization. These tests include Neural Network support and, among various operators, the LSTM. For these test to be passed, ArmNN needed to be able to parse the Android implementation of LSTM and returning the correct output. My work thus involved parsing in the right order the inputs of the Android LSTM operator and passing them to the ArmNN software so it could create the corresponding layer.

### 3.3.3 TensorFlow Lite Parser

**Making inferences on a given network**

To execute a network, the first step is to parse the given file and build an Inference Model that could be used by the Arm Compute Library functions. To achieve this, every kind of file supported has its own parser. Yet, the steps and options needed to build an Inference Model are pretty much the same. Therefore, to keep the code as general as possible, the InferenceModel class is a template of two parameters : the Parser and the data type it uses. However, because of small differences between Tensorflow, Caffe and TensorFlow Lite, the parser code couldn't be totally generalized for TensorFlow Lite. Indeed, unlike the two others, TFLite does not support text format and can be divided in sub-graphs. To allow for TensorFlow Lite support, I thus choose to make a partial specialization of this template, changing the Inference Model creation, while still keeping the global pattern of the code.

**Deploying TensorFlow Lite to Android**

The main purpose of TensorFlow Lite is to be used on mobiles, thanks to its small binary size. One of ArmNN feature is therefore to allow for cross-compilation between Linux x86-64 and Android. This is made possible by using specific tools chains and compilation flags in the cmake, so that the compiler would know which format to use. Once that's done, another step is to deploy the built files and libraries to the Android devices. The environment needs to be aware of the path to these files so that it can link the executables to the libraries and find the files when needed. To do this, I exported these information in the device environment using an adb shell - special debug shell for android -.

### 3.3.4 Onnx parser

#### Design note

First step to implement the new parser was to make its design note. After some research on the ONNX implementation and other examples of ONNX front-end implementation for frameworks like Caffe or Tensorflow, it appears there are 2 different routes to implement such a feature :

1. Use the back-end abstraction in ONNX - coded in python - and derive a back-end that builds an ArmNN graph directly via the ArmNN graph building API. This implies an onnx model is always loaded and the inference processes is driven via a python program.

2. Implement a ONNX parser from scratch in C++ within ArmNN, implying Arm has to maintain it as ONNX evolves.

Even if the latter option would firstly look like more work, it is actually much less work than the other for ArmNN since onnx files use google library protobuf, a library which ArmNN already support from previous parser needs. Thus, supporting ONNX files can be divided into 3 main implementation tasks :

1. **Loading** :
   Before parsing a given onnx file, we need to load it into memory so we can read it. As these files are similar to protobuf files, one can read an onnx file as a text file or a binary file, depending of the encoding. From the returned string, we can then create the corresponding ArmNN network, by extracting the ONNX graph from the model.

2. **Parsing** :
   Once the graph is extracted, we can use it to create the corresponding ArmNN network. This means mapping and parsing every node of the graph in the right order, with each node containing its own type and operation. More, each supported operation need to be converted into an ArmNN layer by converting its parameters to something usable to ArmNN.

3. **Building** :
   Being able to parse a file is not sufficient for ensuring ArmNN supporting ONNX. Indeed, to access the parsing code and know how to deal with such files, the parsing files for onnx format should be regrouped under a library and linked to the project in the Cmake file. More, this library need to be added to the Android deployment script so it works on Android devices like phones.

The design note also need to state how the testing of the feature should go. While ONNX provides a good test suite for back-end implementation, choosing to make the parser from scratch instead of deriving it impeded us from using them. Completing the ONNX parser feature therefore includes coding unit tests for every node supported, making an End-to-End test but also adding onnx support to the ExecuteNetwork program made for doing inferences.

Finally, as ONNX supports over 70 different operators, this represent quite q bit of work and need to be prioritized for ArmNN needs. Indeed, the time line should be done as to focus on the operators presents in our test models - mnist, resnet50... - and develop from those with the end goal of supporting all ONNX operators. Plus, maintaining the parser updated with ONNX latest changes also needs to be done regularly.

**Prototype**

The prototype for the Onnx parser had for main requirement to be able to run the MNIST Neural Network and give the correct results. This implied being able to parse every operation present in the Onnx model - Convolution, Add, Reshape, MaxPooling, Matmul, Relu and Constant - and make the corresponding layers in the ArmNN network.

To achieve this result, the parser proceeds by steps:

- **Loading the model file:**
  Models can be loaded from text and binary files but also strings (used for unit testing). This is done using google protobuf library.

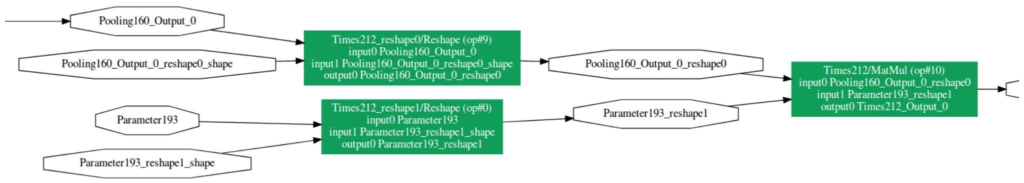
- **Preprocessing the graph:**
  Before being able to parse the graph, some preprocessing needs to be done, involving sorting the nodes in topological order, mapping the names of each node to their output for future use and creating output and inputs placeholders.

- **Processing each node:**
  For each node in the model, we need to make a corresponding layer in the ArmNN network. This means checking and creating the corresponding tensors from the inputs of the node, computing the output shape, creating the corresponding operation layer and connecting it to the right inputs.

While that is not really difficult in itself, some requirements from the armNN side made some of these operation more difficult to parse.

**Fully Connected layers:** Fully Connected layers are layers where all the nodes are connected between each other, and compute the weighted sum of their input nodes. One of the main differences between ArmNN and Onnx is how to deal with such layers. While ArmNN has a special layer for this operation, Onnx doesn't. Indeed, fully connected layers can be divided into a MatMul node, followed by an Add node.
To parse these correctly from Onnx to ArmNN it is thus necessary to detect such combinations of nodes. The solution for the prototype was to delay the creation of any Matmul layer until we know if it is followed by an Add node. If it is the case, we treat it as a Fully Connected layer. Otherwise, it is a simple Matmul.

**Insuring constantness of some inputs:** ArmNN require for some inputs - weights and bias for fully connected layer for instance - to be constants. An easy way to identify a constant input is if it has a default value in the graph initializer list. Yet, in some cases, that is not enough. When a constant input is reshaped for example, its value is still constant but the corresponding tensor has an input, labelling it as not constant. To fix that, inputs of nodes had to be tested for being constant and, if that was the case, a new constant layer is made instead of the considered layer. However, this implies calculating the output of the operation during parsing, resulting in long and ineffective code.


Pooling160_Output_0
Pooling160_Output_0_reshape0_shape
Times212_reshape0/Reshape (op#9)
input0 Pooling160_Output_0
input1 Pooling160_Output_0_reshape0_shape
output0 Pooling160_Output_0_reshape0
Pooling160_Output_0_reshape0
Parameter193
Parameter193_reshape1_shape
Times212_reshape1/Reshape (op#0)
input0 Parameter193
input1 Parameter193_reshape1_shape
output0 Parameter193_reshape1
Parameter193_reshape1
Times212/MatMul (op#10)
input0 Pooling160_Output_0_reshape0
input1 Parameter193_reshape1
output0 Times212_Output_0

Example of constant input not labelled as constant (MNIST model)

**Productising**

Once the prototype works, it is time to productise the feature. This first involves separating the coding into small Jira tasks, easy to test - via Unit tests - and review. Since the coding was done in the prototyping phase, the productising phase mostly involves code re-factoring for efficiency, feature testing and proper exception checks.

Here, the re-factoring touched the very steps of parsing a file :

- **Loading the model file**
- **Preprocessing the graph:**
  Processing was simplified, with only the creation of inputs and outputs layers and the populating of the Info map, used to keep track of the properties of every tensor in the graph.
- **Detecting FullyConnected layers:**
  cf Fully Connected layers paragraph.
- **Creating every layer:**
  For each node, the inputs datatypes and size are checked, the corresponding layer created in ArmNN network and the Info map updated with the newly created tensor.
- **Connecting every layer**
  Connections between layers are made once every layer is created, allowing not to sort the layers before parsing.

This new implementation allows for better reviewing and maintenance as the global steps to deal with a layer are the same, whatever the layer. This was made possible using mainly two data structures : a mapping of information, values (if constant) and data type to name for every tensor in the model and a map of the inputs and outputs layers for every layer in the final graph.

**Fully Connected layer:** Instead of delaying the creation of Matmul nodes until knowing it is not a Fully Connected layer, those are now detected in the first pass and assigned a value telling if the layer creation pass should consider this layer as a Fully Connected along with the next Add operator or if its a simple Matmul nod.

**Insuring constantness of some inputs:** The new mechanic of the parser made detecting constant nodes much easier. Indeed, as each tensor of the graph is mapped to an Info structure, keeping track of its shape, data-type and associated value, when both inputs of a layer are constants, the resultant tensor is then considered constant, by moving the input value to the new tensor.

**ONNX integration achieved**

By the end of the 18.08 release, ONNX integration has reached the point were both MNIST and Mobilenet_v2 models are supported in ArmNN and infer correct results. Moreover, preparatory work has been done for the future integration of model Inception_v3, with the identifying of the required new operators and the creation of the corresponding Jira story and its subtasks. Finally, proper documentation has been written to allow other people to take over the implementation of this feature.

### 3.3.5 Preparation for next release

#### DeepSpeech operator usage analysis

One of the requirements for release 18.11 is the ability to run DeepSpeech model. However, DeepSpeech uses many operators which are not currently supported by ArmNN, mainly related to flow control - while loops for instance - . One possibility could be to "unroll" such loops getting rid of the need for flow control operators. As a preparation for next release, it was necessary to analyze how much work supporting DeepSpeech would be, ie which unsupported operators could not be gotten rid of. This was mostly done by using visualizing tools like TensorBoard and running the model on paper by hand.

## 3.4 Challenges

This whole internship brought its share of challenges, some team wide and others more intern specific.

### 3.4.1 Communication in between teams

The ArmNN project is divided into 2 main teams : one in Cambridge (UK) and one in Galway (Ireland). Thus, communication needs to be constant, so that everyone knows what the others are doing or if there is a problem. Yet, the many platforms used - including but not restricted to Skype for business, Microsoft teams, Jira, Gerrit - made keeping track of everything going on pretty difficult as people could forget to keep the task board updated or won't commit until a feature is fully finished, letting days go without any updates.

### 3.4.2 Getting a global overview of the code

As a newcomer to a year old project, understanding the code and its global functioning was pretty challenging. As going through every file was impossible, I actually found working on small functionalities, widely spread in the code, enabled me to get an overall idea of how the code worked as a whole and finally get my bearings.

# 4. Conclusion

## 4.1 Internship assessment

### 4.1.1 Lessons learned

This internship was a great opportunity to be part of a software engineering team and develop my skills, both technical (C++) and non-technical.

Through the duration of my stay, I got to improve my coding skills from the advice of the other team members, allowing for a code more optimized and elegant. One of the most important - and yet often discarded by students - part of coding a project is testing. Working on the integration of ONNX format to ArmNN made that pretty clear. With so many possible cause for problems, not testing a function thoroughly - even the simplest ones - could mean days and days of painful debugging, passing through each layer of a 150-layer model using GDB.
Let's just say : " Testing is to coding what washing the dishes is to cooking ".

ArmNN is a wide project, with over 30 engineers assigned to its development, in the UK but also in Ireland. More, it relies on code made by other teams and needs to respond to clients and other teams requirements. This made very clear to me how a project such as this one is not only about coding but also about keeping delays, task prioritizing, keeping user experience as easy as possible - changing the API as little as possible for instance -, and much more... Thinking out a feature by making prototypes or impact evaluations before pushing through with the full implementation is therefore a must.

Another aspect working on this project made clear to me was the importance of teamwork. With people from various horizons, everyone has an area of predilection - android compilation, machine learning, assembly, etc... - . Asking people for help or discussing some implementation idea not only makes the code sturdier and more efficient but also teach valuable lessons on topics one may not know very much. This can be pretty impressing at the beginning, with the fear of bothering and interrupting someone else's work but is definitely worth overcoming shyness.

As an engineering student, working for a company such as Arm gave me a whole new perspective on the impact my work could have all around me. The armNN project, in particular, is an open-source project, allowing for feedback from all over the world. More, it recently opened to external contributions and is at the center of a new partnership with Linaro's Machine Learning Intelligence to further enhance worldwide collaboration in the ML field.

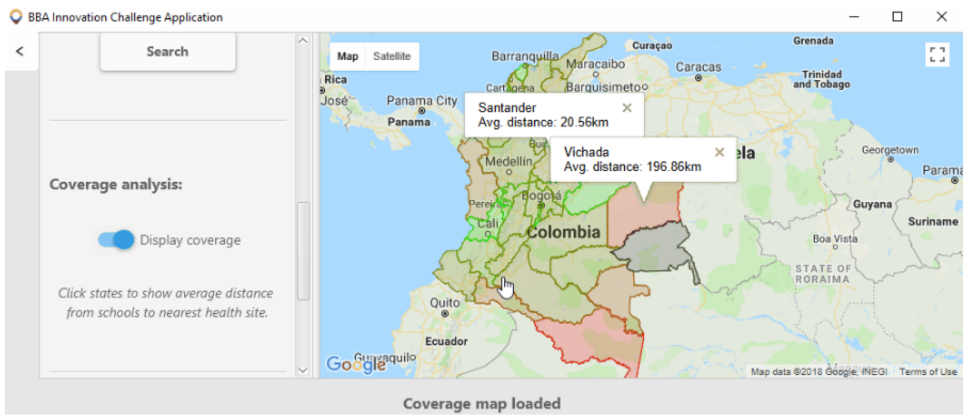
On a more personal level, this internship also helped me define my professional project better.
While Machine Learning is definitely a field I enjoy, I believe I would prefer a more tangible application of

it - object recognition app, genetic algorithms for AI, etc.. -. More, going for such application may allow me to apply my skills in 3D and rendering, which is another field I deeply enjoy.

### 4.1.2 2018 Intern Innovation Challenge

One of the outstanding moments of this internship was the 2018 Intern Innovation Challenge. This competition - in partnership with UNICEF - in between Arm interns worldwide presented us with a challenge to solve in two-weeks time, by team of 5. This year, the challenge was to help Colombia government detect locations in dire need of health and scholar infrastructures. Each team had thus to develop an application enabling the user to visualize easily the data extracted from shapefiles - specific coordinates file format - and display information as the nearest health facility.

My team submission used JavaFX and an online hosted PostGreSQL database to try and solve this challenge. I was personally responsible for the SQL related functionalities and one of the application feature : coverage analysis. This feature enabled the user to colour each state of Colombia according to the average distance in between a school and its nearest health facility.
Full project can be seen on Github: `https://github.com/MorganeRouvroy/ARMChallenge`



The coverage analysis feature in action

This challenge was an amazing opportunity to work with interns from other places but also tests our skills at designing a simple solution to a real world problem and implement it within short delays while still keeping up with work.

Our work was evaluated by a jury of Arm Software Engineers and UNICEF Innovation team members and we actually won this year contest.

## 4.2 Final thoughts

I can honestly say that my time spent interning at Arm resulted in one of the best summers of my life. Not only was it a great opportunity to grow, both professionally and personally, but I also got to meet many fantastic people. The atmosphere at the Cambridge office was always welcoming which made me feel right at home. Additionally, I felt like I was able to contribute to the company by assisting and working on the armNN project throughout the summer. Overall, this experience gave me a taste of what a software engineer job could be and I am very thankful for this chance of confirming that's what I want to do in the future.

# Bibliography

[1] Arm, “Arm impact worldwide,” 2018. [Online]. Available: https://www.arm.com/company

[2] T. Hartley, “Arm nn: Build and run ml apps seamlessly on mobile and embedded devices,” 2018. [Online]. Available: https://community.arm.com/tools/b/blog/posts/arm-nn-sdk

[3] Arm, “About arm nn sdk,” 2018. [Online]. Available: https://developer.arm.com/products/processors/machine-learning/arm-nn

[4] R. Theart, “Getting started with pytorch for deep learning.” [Online]. Available: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

# Appendices

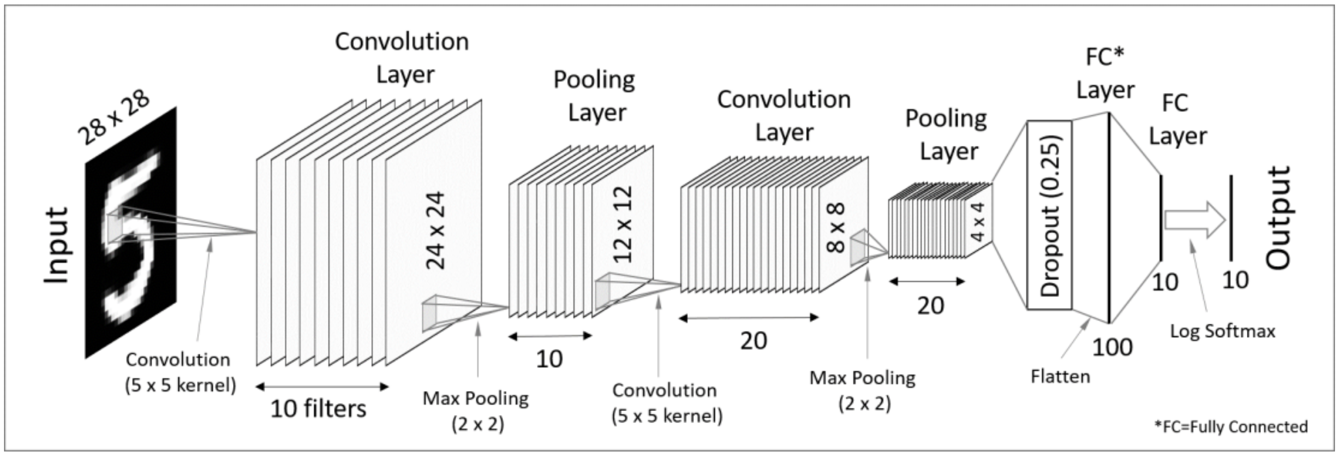
# A. Machine Learning Cheatsheet

This appendix aims at giving a quick overview of Machine Learning vocabulary and state so as to help understanding this report better.

## A.1 Reference Models

During this internship, I mostly worked with 4 reference machine learning models.

- **Simple MNIST:**
  MNIST is a simple computer vision dataset, consisting of images of handwritten digits and labels. This is one of the easiest dataset to work with, making it one of the most used in Machine Learning, especially by beginners. The MNIST data are 28x28 images and corresponding labels, split into a training, validation and test dataset. Each image is a gray-scale 28x28 image.
  The MNIST model used during this internship classify these images into the correct class (0-9), using Convolution and pooling operations, followed by a Fully Connected layer, as shown here :



MNIST model architecture [4]

- **MobileNet v2:**
  MobileNet is a family of classifying network made to run efficiently on mobile devices. The one used during this internship is version 2, and was trained on the ImageNet dataset to recognize 1000 different classes of images, ranging from animals, to plants or even daily objects likes nails or car.
  To run efficiently on mobile devices, MobileNet, contrary to other image classifying models - cf. Inception-, uses depthwise convolutions followed by pointwise convolution. This results in fewer parameters but also a slight decrease in performances.

- **Inception v3:**
  Like MobileNet, Inception is a classifier network. Although it is trained on the same dataset, Inception v3 has slightly better performances at the expense of more parameters. This difference is mostly due to the use of standard convolutions in Inception, as opposed to depthwise convolutions in MobileNet.

- **DeepSpeech:**
  DeepSpeech is a Speech-to-Text engine developed by Mozilla. This model uses bidirectional recurrent neural network (BRNN) trained to ingest speech spectrograms and generate English text transcriptions.