Deployment and Mobile Friendly DL Survey

1.Introduction

Οι υλοποίηση ενός μοντέλου Deep Learning σε έναν ισχυρό υπολογιστή με GPU αποτελεί μόνο το πρώτο βήμα για τον κύκλο ζωής και χρήσης του μοντέλου. Τα μοντέλα πολλές φορές είναι πολύ μεγάλα σε μέγεθος και το και το inference είναι απαιτητικό σε πόρους όπως ενέργεια και χρόνος εκτέλεσης και φυσικά κεντρικό ρόλο παίζει το είδος του inference όπως πχ τον αν είναι real time όπως μία εφαρμογή αναγνώρισης αντικειμένων με χρήση κάμερας. Επίσης παίζει ρόλο η συσκευή στην οποία θα εκτελείται το μοντέλο. Μία εφαρμογή που τρέχει στο cloud ή σε έναν ιδιωτικό server μπορεί να αντιμετωπίσει τις αυξημένες ανάγκες σε πόρους. Μία mobile συσκευή όμως μπορεί να αντιμετωπίσει προβλήματα ακόμα και αν έχει αρκετό χώρο μνήμη και επεξεργαστική ισχύ λόγω μεγάλης κατανάλωσης ενέργειας. Ένας microcontroller αντιμετωπίζει όλα τα παραπάνω προβλήματα καθώς επίσης και άλλα που έχουν να κάνουν με το σύνολο των εντολών που μπορεί να εκτελέσει. Επίσης πολλά μοντέλα είναι προσβάσιμα μέσω web και αν και εκτελούνται στην συσκευή ο χρόνος του download αποτελεί μία σημαντική παράμετρο. Τέλος λόγω των διαφορών που υπάρχουν ανάμεσα σε desktop/servers και κινητές συσκευές είναι καλό το μοντέλο να είναι βελτιστοποιημένο για εκτέλεση σε αυτά τα περιβάλλοντα με τις ιδιαιτερότητες του είτε θετικές είτε αρνητικές. Μία αρνητική είναι ότι οι βιβλιοθήκες python δεν είναι άμεσα διαθέσιμες σε κινητές συσκευές και αν τρέξουμε τα μοντέλα σε αυτό το περιβάλλον η απόδοση είναι υποβέλτιστη. Μία θετική ιδιαιτερότητα είναι πχ η ύπαρξη DSP ή TPU σε smartphones, παρόλο που πολλές φορές δεν υπάρχει διαθέσιμη GPU ο DSP αποτελεί μία πολύ καλή εναλλακτική που ένα DL framework πρέπει να μπορεί να εκμεταλλευτεί. Σε αυτήν την εργασία θα παρουσιάσουμε τους τρόπους που χρησιμοποιούν τα πιο δημοφιλή frameworks, Pytorch και TensorFlow, για να αντιμετωπίσουν τους περιορισμούς και να εκμεταλλευτούν τις ιδιαιτερότητες των κινητών συσκευών. Θα παρουσιάσουμε αναλυτικά όλο το pipeline του deployment απο την εκπαίδευση μέχρι το inference και θα περιγράψουμε αναλυτικά τις μεθόδους βελτιστοποίησης που εφαρμόζονται. Το υπόλοιπο της εργασίας είναι δομημένο ως εξής. Στην ενότητα 2 θα περιγράψουμε το documentation για τα frameworks tfLite,tf_js και pytorchLite συνοπτικά ώστε να μπορεί ο αναγνώστης να βρει περισσότερες πληροφορίες για κάθε διαδικασία. Στην ενότητα 3 θα περιγράψουμε την μεθοδολογία που ακολουθούν τα frameworks για να αντιμετωπίσουν περιορισμούς και ιδιαιτερότητες όπως περιγράφηκαν παραπάνω. Στην ενότητα 4 θα παρουσιάσουμε τις επιλογές που υπάρχουν για το deployment για κάθε framework και θα τις αναλύσουμε. Στην ενότητα 5 θα κάνουμε μία σύντομη σύγκριση των frameworks και στην ενότητα 6 θα αναφέρουμε λίγα πράγματα για την υλοποίηση και τα αρχεία που συνοδεύουν την εργασία σχετικά με το implementation.

2.Documentation

Εδώ θα αναφέρουμε το documentation για κάθε framework και θα περιγράψουμε σύντομα τις λειτουργίες που φέρνει εις πέρας η κάθε λειτουργία. Θα ξεκινήσουμε με την tfLite, στην συνέχεια με την tf_js και τέλος pytorch mobile.

2.1 TensorFlow Lite

Η κεντρική σελίδα όπου μπορούν να βρεθούν πληροφορίες για framework είναι εδώ[https://www.tensorflow.org/lite/guide] .Εδώ αναφέρονται τα κύρια χαρακτηριστικά και θα κάνουμε μία σύντομη αναφορά σε αυτά.

Βελτιστοποίηση για φορητές συσκευές.

Αυτό επιτυγχάνεται αντιμετωπίζοντας 5 σημεία. Το πρώτο είναι το **latency**. Ο χρόνος απόκρισης δεν επιβαρύνεται από από roundtrip σε κάποιο server στο cloud η αλλού χαρακτηριστικό που μπορεί να είναι απαραίτητο πλέον από επιθυμητό. Για παράδειγμα μία εφαρμογή ενός αυτόνομου οχήματος που οδηγεί το όχημα ή ανιχνεύει πεζούς δεν μπορεί να ανεχτεί μία μεγάλη καθυστέρηση. Η απόδοση του κάτω από τέτοιες συνθήκες θα ήταν μη αποδεκτή. Το δεύτερο σημείο είναι το **privacy.** Προσωπικά δεδομένα δεν φεύγουν ποτέ από την συσκευή. Αυτό το σημείο μπορεί να είναι προαπαιτούμενο για κάποια use cases. Το τρίτο σημείο είναι ότι δεν χρειάζεται σύνδεση στο **internet.** Το κόστος πχ για κάποια μοντέλα που επεξεργάζονται real time video θα ήταν μεγάλο και επίσης η κατανάλωση ενέργειας θα ήταν επίσης μεγάλη. Το τέταρτο σημείο είναι μικρό **μέγεθος** μοντέλου και το πέμπτο σημείο είναι η χαμηλή κατανάλωση **ενέργειας** . Αυτά που αναφέρθηκαν σε αυτή την παράγραφο αφορούν και τα δύο frameworks και δεν θα αναφερθούμε ξανά σε αυτά στην συνέχεια.

Multiple platform support

Όπως αναφέρθηκε παραπάνω υποστηρίζονται όλες οι μεγάλες πλατφόρμες πράγμα που την καθιστά διαθέσιμη τεχνολογία plug and play. Παρόλο που εδώ θα ασχοληθούμε μόνο με τις πλατφόρμες για smartphones αξίζει να σημειωθεί ότι η tfLite υποστηρίζει και embedded devices καθώς και microcontrollers.

- Android[https://www.tensorflow.org/lite/android]
- IOS[https://www.tensorflow.org/lite/guide/ios]
- Embedded Linux[https://www.tensorflow.org/lite/guide/python]
- Microcontrollers[https://www.tensorflow.org/lite/microcontrollers]

Diverse Language support

Ή tfLite υποστηρίζει Java,SWIFT,Objective-C,Python.

High Performance

Η υψηλή επίδοση επιτυγχάνεται αντιμετωπίζοντας το πρόβλημα σε δύο διαστάσεις. Η πρώτη είναι το **hardware acceleration** και η δεύτερη το **model optimization.** Στην δεύτερη επειδή αφορά πολλές παραμέτρους θα αναφερθούμε σε ξεχωριστή ενότητα στην συνέχεια. Σχετικά με το πρώτο τα κύρια σημεία περιγράφονται παρακάτω.

Hardware Acceleration

[https://www.tensorflow.org/lite/performance/delegates]

Η tfLite είναι σε θέση να χρησιμοποιήσει το διαθέσιμο υλικό της συσκευής αν αυτό υπάρχει. Για παράδειγμα μπορεί να χρησιμοποιήσει πέρα από την CPU την GPU ή τον DSP ή TPU. Αυτό το hardware αν υπάρχει είναι σε θέση να πετύχει υψηλή απόδοση όσον αφορά τον χρόνο εκτέλεσης αλλά και διάρκεια ζωής της μπαταρίας διότι ενδεικνύεται για συχνές αριθμητικές πράξεις της διαδικασίας των νευρωνικών δικτύων που μπορούν να επιβαρύνουν την CPU. Πέρα από την απόδοση όσον αφορά τον χρόνο και την κατανάλωση ενέργειας η χρήση του hardware συνήθως δίνει πιο ακριβή αποτελέσματα καθώς η ακρίβεια των υπολογισμών είναι καλύτερη από μιας απλής CPU.

Model Optimization

[https://www.tensorflow.org/lite/performance/model_optimization]

Υπάρχουν πολλοί λόγοι για τους οποίους πρέπει να γίνει. Μερικοί από τους λόγους που ένα μοντέλο θα πρέπει να βελτιστοποιηθεί είναι οι ακόλουθοι

Size reduction

Αυτό το optimization σχετίζεται με μικρότερο χώρο αποθήκευσης μικρότερο χρόνο κατεβάσματος και λιγότερη χρήση μνήμης. Αυτοί οι περιορισμοί είναι παντού στις φορητές συσκευές.

Latency reduction

Αυτό το optimization σχετίζεται με τον χρόνο που χρειάζεται να τρέξει το inference. Μειώνοντας την επεξεργασία που γίνεται σε επίπεδο εντολών που χρειάζονται μειώνουμε το latency και μεγιστοποιούμε την διάρκεια της μπαταρίας.

Accelerator compatibility

Όταν ξέρουμε για το διαθέσιμο hardware μπορούμε να βελτιστοποιήσουμε το μοντέλο για το συγκεκριμένο hardware με αποτέλεσμα το inference να τρέχει πολύ γρήγορα.

Φυσικά όταν βελτιστοποιούμε ένα μοντέλο όσον αφορά το size reduction και το latency reduction είναι πολύ πιθανό να δούμε κάποια επίδραση στην ακρίβεια του μοντέλου. Το αν είναι αποδεκτό κάτι τέτοιο στα πλαίσια της κάθε εφαρμογής είναι δύσκολο να απαντηθεί από πριν. Το πώς μπορούν αυτοί οι στόχοι να πραγματοποιηθούν σε τεχνικό επίπεδο περιγράφεται παρακάτω.

End to end examples

[https://www.tensorflow.org/lite/examples]

Η tfLite προσφέρει πολλά έτοιμα παραδείγματα που μπορούν να χρησιμοποιηθούν. Πρόκειται για πολλά υποστηριζόμενα tasks όπως image classification, object detection, pose estimation, question answering, text classification, etc.

Support Libraries

[https://www.tensorflow.org/lite/inference_with_metadata/lite_support]

[https://www.tensorflow.org/lite/inference_with_metadata/task_library/overview]

Επίσης πρέπει να αναφέρουμε δύο βιβλιοθήκες που υπάρχουν και παρέχουν βοηθητικά εργαλεία. Η πρώτη παρέχει διάφορα εργαλεία τα οποία μπορούν να υποστηρίξουν διάφορα tasks ‘οπως Image Classification, Object Detection και άλλα. Αυτά τα APIs προσφέρουν λύσεις που είναι απλές αποδοτικές και εύκολα επεκτάσιμες. Η βιβλιοθήκη αυτή ονομάζεται Tensorflow Task Library. Η δεύτερη είναι σχεδιασμένη να διευκολύνει την επεξεργασία των input/output των μοντέλων ονομάζεται Tensorflow Support Library.

API reference
[https://www.tensorflow.org/lite/api_docs]

2.2 TensorFlow JS

[https://www.tensorflow.org/js/guide]
Όπως και η tfLite έτσι και η tf_js έχει τα πλεονεκτήματα που αναφέρθηκαν παραπάνω. Περισσότερες πληροφορίες για το framework από το πώς να δημιουργεί tensors μέχρι εκπαίδευση μοντέλων.Να σημειωθεί ότι το framework σε αντίθεση με την πρώτη

μέθοδο(tfLite) μπορεί να χρησιμοποιηθεί και για να εκπαιδεύσουμε μοντέλο εξ ολοκλήρου [https://www.tensorflow.org/js/guide/train_models]. Εδώ θα ασχοληθούμε μόνο με την χρήση για inference τοπικά στην συσκευή. Θεωρούμε το σενάριο όπου έχουμε ήδη ένα εκπαιδευμένο μοντέλο. Ενδεικτικά αναφέρουμε κάποιες από τις βασικές ενότητες,
Εκπαίδευση μοντέλου[https://www.tensorflow.org/js/guide/train_models]
Αποθήκευση και φόρτωση μοντέλου[https://www.tensorflow.org/js/guide/save_load]
Platform and Enviroment[https://www.tensorflow.org/js/guide/platform_environment]

2.2 Pytorch Mobile

[https://pytorch.org/mobile/home/].

Η pytorch αν και καθυστερημένα σε σχέση με την tensorFlow έδωσε την δυνατότητα τα προ εκπαιδευμένα μοντέλα να τρέχουν σε inference mode σε συσκευές Android και Ios. Μέχρι στιγμής δεν υποστηρίζονται embedded devices ή microcontrollers. Το workflow είναι παρόμοιο με αυτό της tensorflow και τα πλεονεκτήματα και οι ιδιαιτερότητες που αναφέρθηκαν για τα δύο προηγούμενα ισχύουν και εδώ. Την διαδικασία της μετατροπής θα την περιγράψουμε αναλυτικά στην συνέχεια. Εδώ αξίζει να σημειώσουμε ότι υπάρχει υποστήριξη για συσκευές που έχουν GPU σε IOS[https://pytorch.org/tutorials/prototype/ios_gpu_workflow.html] GPU για Android[https://pytorch.org/tutorials/prototype/vulkan_workflow.html] καθώς και DSP και NPU για Android[https://pytorch.org/tutorials/prototype/nnapi_mobilenetv2.html]. Η υποστήριξη για τα τελευταία hardware backends θα είναι σύντομα διαθέσιμη και για IOS.

3.Optimization Methodology

Σε αυτήν την ενότητα περιγράφουμε την μεθοδολογία που ακολουθείται απο τα frameworks για να αντιμετωπίσουν το πρόβλημα όπως το ορίσαμε στην ενότητα 1. Δηλαδή περιορισμένοι πόροι σε επεξεργαστική ισχύ, μνήμη, αποθηκευτικό χώρο, συνδεσιμότητα στο διαδίκτυο και φυσικά κατανάλωση ενέργειας. Με βάση αυτά αναπτύχθηκαν μέθοδοι που αντιμετωπίζουν το πρόβλημα. Σαν παράδειγμα ένα μοντέλο των 600MB που μπορεί να μετατραπεί σε 150MB χωρίς να χάσουμε κάτι σημαντικό στην ακρίβεια του αποτελεί μεγάλο κέρδος, όπως επίσης και η δυνατότητα να εκμεταλευτούμε την ύπαρξη DSP ή TPU. Γενικά για την αντιμετώπιση του προβλήματος και η TensorFlow και η Pytorch ακολουθούν ίδια προσέγγιση αλλά υπάρχουν κάποιες διαφορές για αυτό θα παρουσιάσουμε αναλυτικά την διαδικασία και για τα δύο. Να σημειωθεί ότι όλα τα optimization που είναι διαθέσιμα για tfLite και tf_js είναι ίδια οπότε θα τα παρουσιάσουμε μία φορά στην ενότητα για την TensorFlow.

3.1 TensorFlow Optimizations

Η tfLite υποστηρίζει βελτιστοποίηση μέσω **quantization**, **pruning** και **clustering**. Στην συνέχεια θα αναλύσουμε αυτές τις μεθόδους.

3.1.1 Quantization. Η μέθοδος αυτή λειτουργεί μειώνοντας την ακρίβεια των αριθμών που αναπαριστούν τις παραμέτρους του μοντέλου. Αυτό έχει ως αποτέλεσμα την μείωση του μεγέθους του μοντέλου καθώς και την μείωση του χρόνου εκτέλεσης. Γενικά η tfLite υποστηρίζει τις ακόλουθες μεθόδους.

Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android)
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Small accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP

Post training quantization float16.
[**https://www.tensorflow.org/lite/performance/post_training_float16_quant**]

Η πρώτη μέθοδος λειτουργεί ως εξής. Πρόκειται για πολύ απλή μέθοδο όπου απλώς αντικαθιστούμε την ακρίβεια των αριθμών(παραμέτρων και αποτελεσμάτων) από 32bit σε16bit. Αυτό οδηγεί σε μείωση του μεγέθους και γρηγορότερο inference καθώς το hardware μπορεί να πραγματοποιεί υπολογισμούς με αυτήν την μειωμένη ακρίβεια.

Post training dynamic range quantization.
[https://www.tensorflow.org/lite/performance/post_training_quant]

Πρόκειται για παρόμοια μέθοδο με την προηγούμενη άλλα εδώ δυναμικά μετατρέπονται οι float32 παράμετροι σε int8 κάτι που μειώνει το μέγεθος κατά 75%.Αυτό συμβαίνει για

ενέργειες που το υποστηρίζουν και αφού γίνει η επεξεργασία με τύπο int8 οι μεταβλητές μετατρέπονται και πάλι σε τύπο float. Δηλαδή οι μετατροπές γίνονται δυναμικά κατά την διάρκεια της εκτέλεσης. Αυτή η μέθοδος χρησιμοποιείται μόνο όταν έχουμε διαθέσιμη μόνο CPU και αδυνατεί να εκμεταλλευτεί hardware τύπου GPU,DSP,TPU όπως αναφέρθηκε παραπάνω.

Post training integer quantization.
[https://www.tensorflow.org/lite/performance/post_training_integer_quant]

Αυτή η μέθοδος χρησιμοποιείται για microcontrollers και μετατρέπει στατικά όλες τις folat32 παραμέτρους σε int8. Αυτή η μέθοδος έρχεται σε δύο εκδόσεις, στην πρώτη όλες οι παράμετροι μετατρέπονται σε int8. Στην δεύτερη τα βάρη μετατρέπονται σε int8 και τα αποτελέσματα των activations μετατρέπονται σε int16.

Post training integer quantization second edition.
[https://www.tensorflow.org/model_optimization/guide/quantization/post_training]

Η προηγούμενη περίπτωση μπορεί να μην λειτουργεί σωστά για κάποια μοντέλα. Γενικά προτείνεται για μοντέλα που είναι ευαίσθητα στην ακρίβεια των activations. Έτσι εδώ όλα τα βάρη αναπαριστώνται με 8 bit αλλά τα activations με 16bit.

Quantization aware training.
[http://www.tensorflow.org/model_optimization/guide/quantization/training]

Η τελευταία αυτή μέθοδος διαφέρει από τις προηγούμενες καθώς οι προηγούμενες μετατρέπουν ένα ήδη δημιουργημένο μοντέλο. Σε αυτήν την μέθοδο χρησιμοποιώντας συγκεκριμένα APIs και γνωρίζοντας ότι το μοντέλο θα μετατραπεί εκπαιδεύεται εξομοιώνοντας την διαδικασία σαν να έτρεχε ήδη με άλλο τύπο παραμέτρων. Αυτό τις περισσότερες φορές οδηγεί σε μοντέλα που κατά την διάρκεια του inference είναι πιο ακριβή σε σχέση με αυτά που προκύπτουν από τις παραπάνω μεθόδους αφού μαθαίνουν παραμέτρους που είναι ανθεκτικές σε σενάρια απώλειας αριθμητικής ακρίβειας.

Operation Fusion.

Σε αυτή την μέθοδο η οποία δεν αναφέρεται ρητά σαν μέθοδος βελτιστοποίησης από την tensorflow αλλά αναφέρεται ως τέτοια από την pytorch συγχωνεύονται εντολές. Στην tensorflow υπάρχουν οι primitive εντολές και οι composite εντολές. Η διαδικασία παράγει ένα operation που συγχωνεύει όλες τις primitive εντολές από τις οποίες αποτελείται μία composite εντολή. Τα πλεονεκτήματα από αυτή την διαδικασία είναι το ότι βελτιστοποιεί τους υπολογισμούς και μειώνει το I/O της μνήμης βελτιώνοντας το latency σε συσκευές

με περιορισμένους πόρους. Φυσικά και δεν είναι μια απλή διαδικασία και υπάρχουν προκλήσεις που πρέπει να αντιμετωπιστούν.

3.1.2 Pruning Αυτή είναι η δεύτερη μέθοδος.[https://www.tensorflow.org/model_optimization/guide/pruning]

Αυτή η μέθοδος βελτιστοποίησης διαγράφει παραμέτρους που έχουν μικρό αντίκτυπο στην ακρίβεια του μοντέλου. Παρόλο που δεν μειώνει σημαντικά το μέγεθος του μοντέλου επιτρέπει την πιο αποδοτική συμπίεση του μειώνοντας το download time. Δεν προσφέρει κάποια άλλα πλεονεκτήματα και η μέθοδος αυτή υλοποιείται κατα την διάρκεια της εκπαίδευσης.

3.1.3 Clustering. Αυτή είναι η τρίτη μέθοδος.[https://www.tensorflow.org/model_optimization/guide/clustering]

Η τελευταία αυτή μέθοδος έχει ίδια αποτελέσματα με την προηγούμενη χωρίς να προσφέρει κάποια άλλα πλεονεκτήματα.Με βάση αυτά συμπεραίνουμε ότι η μέθοδος του quantization είναι η πιο σημαντική πετυχαίνοντας καλύτερη επίδοση και στο μέγεθος του μοντέλου αλλά κυρίως στην βελτίωση του inference time σημαντικό χαρακτηριστικό για κινητές συσκευές.Να σημειωθεί ότι τα optimizations που περιγράφηκαν και εφαρμόστηκαν για την μέθοδο μετατροπής σε tfLite είναι διαθέσιμα και στην περίπτωση της δεύτερης μεθόδου tf_js που ακολουθεί και δεν θα τα αναφέρουμε ξανά.

3.2 Pytorch Optimizations

Να σημειώσουμε ότι η pytorch προσφέρει και optimizations για την συγχώνευση operations όπως θα περιγραφεί παρακάτω. Αναφέρονται ως optimization σε αντίθεση με την tensorflow που αναφέρονται αυτόνομα και όχι μαζί με τα άλλα optimizations. Η λογική παραμένει ίδια και ο σκοπός που επιτυγχάνεται είναι η μείωση του latency μέσω της μείωσης της πολυπλοκότητας του μοντέλου.

3.2.1 Operation Fusion

Το πρώτο optimization που εφαρμόζει η pytorch για να μειώσει το inference time είναι η συγχώνευση υπολογισμών σε επίπεδα. Οι πράξεις(operations) που μπορούν να συγχωνευτούν είναι οι παρακάτω.[https://pytorch.org/tutorials/recipes/mobile_perf.html]

1)Convolution και Batch Normalization

2)Convolution και Batch Normalization Relu

3)Convolution Relu

4)Linear Relu

3.2.2 Quantization

Το δεύτερο optimization αφορά την αντίστοιχη λειτουργικότητα που προσφέρει και η tensorflow με την ίδια ονομασία, δηλαδή quantization.

Η σχετική διαδικασία που ακολουθείται περιγράφεται παρακάτω. Υπάρχουν τρία είδη αντίστοιχα με της tensorflow αλλά έχουν διαφορές.[https://pytorch.org/blog/introduction-to-quantization-on-pytorch/]

Dynamic Quantization

Αυτή η μέθοδος όπως και οι επόμενες μετατρέπει όλα τα βάρη σε int8 από floating point. Να σημειωθεί ότι δεν υπάρχει επιλογή για μετατροπή σε float16 όπως στην tensorflow. Ο όρος dynamic προκύπτει από το ότι εκτός από τα βάρη μετατρέπονται την ώρα της εκτέλεσης και τα activations σε int8. Να σημειωθεί ότι ενώ τα βάρη μετατρέπονται σε αναπαράσταση int8 τα activations γράφονται και διαβάζονται στη μνήμη σε μορφή floating point.

Post-Training Static Quantization

Αυτή η μέθοδος κάνει ότι και η παραπάνω όσον αφορά τα βάρη και την αναπαράσταση των αριθμών. Εκτός από τα βάρη και τα memory accesses γίνονται σε μορφή int8 κάτι που μειώνει το latency άρα και την απόδοση του μοντέλου ακόμα περισσότερο. Αυτά τα δύο αναφέρονται ως integer arithmetic και integer memory accesses. Στην πράξη αυτό επιτυγχάνεται τροφοδοτώντας με δεδομένα τα οποία θα μοιάζουν με πραγματικά δεδομένα που θα τροφοδοτούν το δίκτυο και συλλέγοντας στατιστικές για το εύρος των τιμών που παίρνουν τα activations. Αυτό επιτυγχάνεται με το να εισάγονται observer modules μετά τα activations. Στην συνέχεια μπορούμε να περνάμε quantized values στα επιμέρους operations χωρίς το overhead της μετατροπής από/σε float/int. Αυτή η πληροφορία μπορεί να χρησιμοποιηθεί για να βρούμε το ιδανικό quantization για διαφορετικά activations και να το χρησιμοποιήσουμε στην φάση του inference. Η pytorch συνοδεύεται από features που δίνουν την δυνατότητα στον χρήστη να βελτιστοποιήσει το quantization. Αυτά είναι observer modules που μπορούν να παραμετροποιηθούν. Operator fusion όπως αναφέρθηκε και παραπάνω που μειώνει δραστικά τα memory accesses και συνεπώς την απόδοση. Τέλος προσφέρεται η δυνατότητα per channel quantization όπου τα βάρη όπου τα βάρη ενός επιπέδου convolutional/linear γίνονται quantised ανεξάρτητα το ένα από το άλλο δίνοντας καλύτερη απόδοση με το ίδιο overhead.

Quantization Aware Training

Ο τελευταίος αυτός τρόπος είναι και ο πιο αποδοτικός από την οπτική ότι βελτιστοποιεί την ακρίβεια που μπορεί να επιτύχει. Εδώ στην φάση της εκπαίδευσης γνωρίζουμε ότι θα γίνει quantization και παρόλο που όλες οι πράξεις και στο forward και στο backward pass γίνονται με floating point arithmetic οι τιμές στρογγυλοποιούνται σαν να είχαν αναπαρασταθεί σε μορφή int8. Πρόκειται για παρόμοια διαδικασία με την προηγούμενη άλλα γίνεται για όλη την διάρκεια της εκπαίδευσης πράγμα που μεγιστοποιεί την απόδοση της τεχνικής.

4.Deployment

4.1 Διαδικασία deployment tfLite μοντέλου

Η διαδικασία του deployment ενός tfLite model σε μια φορητή συσκευή περιγράφεται παρακάτω.

Αρχικά πρέπει να έχουμε το μοντέλο σε format .tflite.

Μπορούμε να βρούμε ένα έτοιμο μοντέλο σε format .tflite από αυτά που έχει διαθέσιμα η tensorflow ή από κάποια άλλη πηγή.

Με την βιβλιοθήκη TensorFlow Model Maker έχουμε την δυνατότητα χρησιμοποιώντας transfer learning να εκπαιδεύσουμε ένα υπάρχον μοντέλο με custom dataset χρησιμοποιώντας ένα από τα έτοιμα μοντέλα tfLite. Να σημειωθεί εδώ ότι η tfLite δεν δίνει την δυνατότητα εκπαίδευσης μοντέλου παρά μόνο transfer learning. Η μόνη επιλογή για μοντέλο που δεν βασίζεται σε κάποιο έτοιμο είναι η επόμενη. Τα tasks που υποστηρίζονται είναι τα παρακάτω.[https://www.tensorflow.org/lite/guide/model_maker]

Supported Tasks	Task Utility
Image Classification: tutorial, api	Classify images into predefined categories.
Object Detection: tutorial, api	Detect objects in real time.
Text Classification: tutorial, api	Classify text into predefined categories.
BERT Question Answer: tutorial, api	Find the answer in a certain context for a given question with BERT.
Audio Classification: tutorial, api	Classify audio into predefined categories.
Recommendation: demo, api	Recommend items based on the context information for on-device scenario.

Μπορούμε να μετατρέψουμε ένα υπάρχον tf μοντέλο σε μοντέλο tfLite χρησιμοποιώντας το TensorFlow Lite Converter. Αυτό μπορεί να γίνει είτε με το python API ή με εργαλείο γραμμής εντολών. Σχετικά με το εργαλείο γραμμής εντολών αναφέρεται ότι υπάρχουν περιορισμοί σε σχέση με το API οπότε ενδείκνυται η χρήση του python API. Στην φάση αυτή μπορούμε να υλοποιήσουμε optimizations όπως περιγράφηκαν προηγουμένως ή/και να προσθέσουμε μετα-δεδομένα στο μοντέλο χαρακτηριστικό που μπορεί να φανεί χρήσιμο όταν δημιουργούνται αυτόματα code-wrappers για συγκεκριμένες πλατφόρμες. Να σημειωθεί ότι σε αυτό το βήμα μπορούν να προκύψουν προβλήματα στα οποία θα αναφερθούμε στην φάση της υλοποίησης.[https://www.tensorflow.org/lite/convert/index]

Inference

Για να χρησιμοποιήσουμε το tflite μοντέλο υπάρχουν δύο τρόποι.Για μοντέλα χωρίς μεταδεδομένα χρησιμοποιείται το Tensor Flow Interpreter API[https://www.tensorflow.org/lite/guide/inference]. Περισσότερα στην φάση της υλοποίησης.

Για δεδομένα με μεταδεδομένα υπάρχουν βιβλιοθήκες για custom ή out of the box APIs και χρησιμοποιούνται για δημιουργία inference pipelines[https://www.tensorflow.org/lite/inference_with_metadata/task_library/overview]

Technical Constraints

Να σημειωθεί ότι δεν μπορούν να μετατραπούν όλα τα μοντέλα σε tfLite. Περισσότερα αναφέρονται εδώ.[https://www.tensorflow.org/lite/guide/ops_compatibility].

4.2 Διαδικασία Deployment tf_js

Η δεύτερη μέθοδος είναι η εκτέλεση του μοντέλου με χρήση javascript τοπικά στην συσκευή. Η tensorFlow δίνει την δυνατότητα να γίνει αυτό με χρήση του tensorFlow.js [https://www.tensorflow.org/js/guide]. Να σημειωθεί ότι το framework σε αντίθεση με την πρώτη μέθοδο(tfLite) μπορεί να χρησιμοποιηθεί και για να εκπαιδεύσουμε μοντέλο εξ ολοκλήρου [https://www.tensorflow.org/js/guide/train_models]. Εδώ θα ασχοληθούμε μόνο με την χρήση για inference τοπικά στην συσκευή. Θεωρούμε το σενάριο όπου έχουμε ήδη ένα εκπαιδευμένο μοντέλο.

Μετατροπή του μοντέλου

Η tensorflow.js μας δίνει την δυνατότητα να μετατρέψουμε μοντέλο που έχει αποθηκευτεί είτε ως SavedModel, Keras model ή tensorflow hub model σε αντίστοιχο μοντέλο .js[https://www.tensorflow.org/js/guide/conversion]. Περισσότερα για την

μετατροπή στην φάση της υλοποίησης. Να σημειωθεί ότι τα optimizations που αναφέρθηκαν παραπάνω μπορούν να εφαρμοστούν και σε αυτή τη μέθοδο παρόλα αυτά είναι βέλτιστη τακτική να δημιουργούμε το μοντέλο με γνώση του ότι θα τρέξει σε περιβάλλον με περιορισμένους πόρους συνεπώς θα πρέπει να αποφεύγονται υλοποιήσεις πολύπλοκων αρχιτεκτονικών και ελαχιστοποίηση παραμέτρων ώστε να είναι πιο απλό το μοντέλο. Ο js tensorflow Converter [https://github.com/tensorflow/tfjs/tree/master/tfjs-converter] αποτελείται από δύο μέρη. Το πρώτο είναι ένα εργαλείο γραμμής εντολών που μετατρέπει το μοντέλο σε tf.js. Υπάρχει και API που μπορεί να χρησιμοποιηθεί για την μετατροπή αλλά ενδύκνειται το εργαλείο γραμμής εντολών λόγω του ότι δεν βρήκαμε πως να εφαρμόσουμε τα optimizations από το API ενώ με το εργαλείο είναι απλή διαδικασία. Το δεύτερο μέρος είναι ένα API για να φορτωθεί το μοντέλο και να εκτελεστεί το inference. Κατά την διαδικασία της μετατροπής ελέγχεται αν η κάθε διαδικασία(ops) υποστηρίζεται από το tensorFlow.js. Να σημειωθεί ότι παρόλο που όταν μετατρέπεται το μοντέλο γίνεται ότι είναι δυνατόν για την βελτιστοποίηση της απόδοσης το μοντέλο είναι καλό να έχει αναπτυχθεί με την γνώση ότι θα τρέχει σε περιβάλλον με περιορισμένους πόρους. Συνεπώς η αρχιτεκτονική πρέπει να είναι όσο το δυνατόν απλή και ο αριθμός των παραμέτρων ελάχιστος.

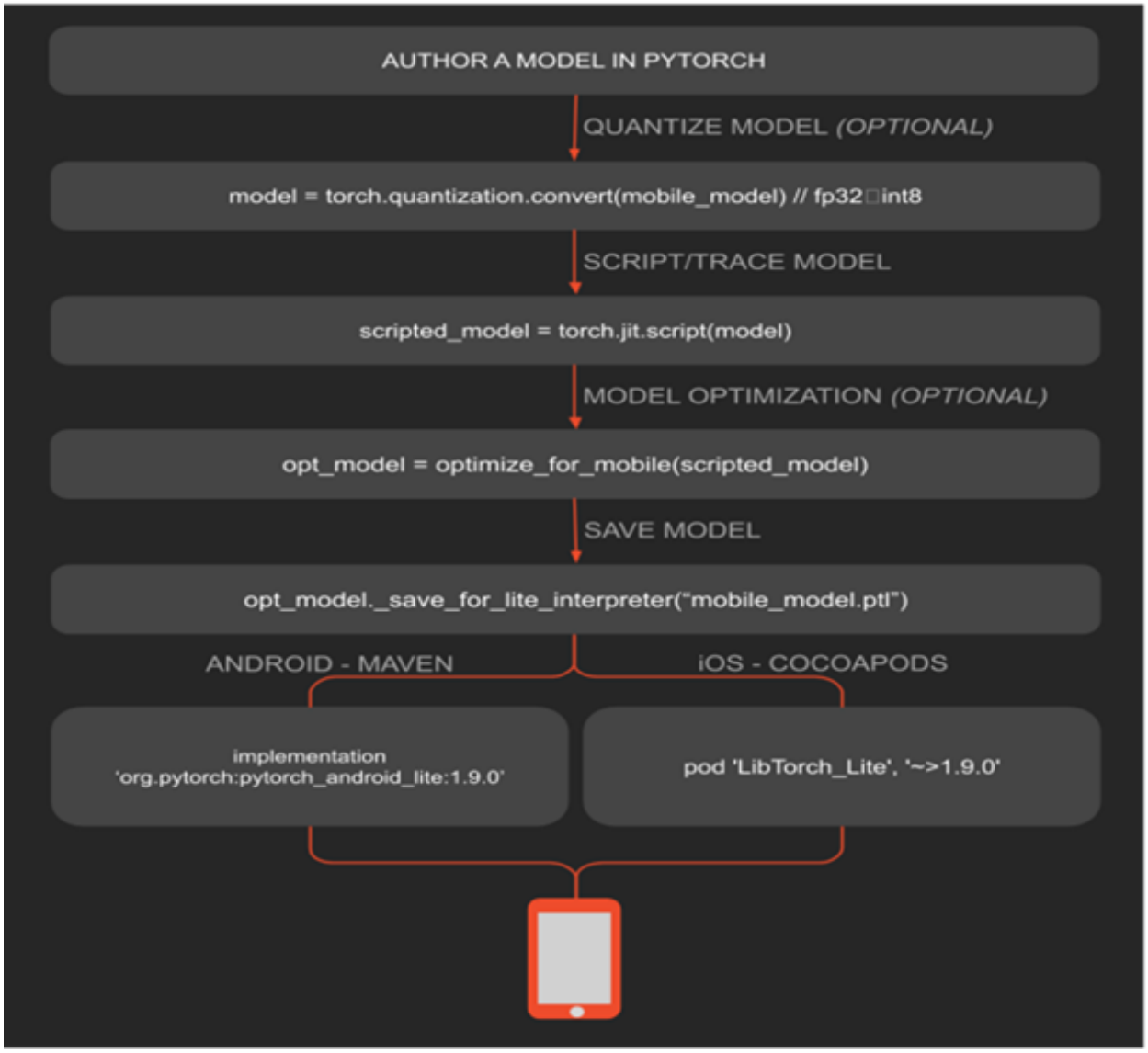
Εκτέλεση του μοντέλου

Για να τρέξουμε το μοντέλο πρέπει να φορτώσουμε εκτός από το μοντέλο και το js API το οποίο είναι διαθέσιμο εδώ https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis@1.5.1/dist/tfjs-vis.umd.min.js .

Από την στιγμή που φορτώσουμε το προηγούμενο script και το μοντέλο μας είμαστε έτοιμοι να τρέξουμε το inference. Να σημειωθεί ότι πλέον υπάρχει βιβλιοθήκη και για React Native[https://blog.tensorflow.org/2020/02/tensorflowjs-for-react-native-is-here.html], framework που αναπτύχθηκε για να δημιουργούνται εφαρμογές γραμμένες εξ ολοκλήρου σε javascript που τρέχουν και σε Android και σε iOS. Εδώ θα ασχοληθούμε με την υλοποίηση σε Android μόνο χωρίς χρήση React. Περισσότερα στην υλοποίηση.

4.3 Διαδικασία Deployment Pytorch μοντέλου

Η παρακάτω εικόνα περιγράφει το workflow από την συγγραφή έως το deployment ενός μοντέλου για mobile.



Για το πρώτο βήμα δεν θα πούμε κάτι εδώ θεωρούμε ότι έχουμε ένα έτοιμο μοντέλο. Το δεύτερο βήμα αφορά αυτά που περιγράφηκαν στην προηγούμενη ενότητα.

Script/Trace Model

Το επόμενο βήμα στο workflow είναι η μετατροπή του μοντέλου το οποίο είναι βασισμένο σε python σε μοντέλο βασισμένο σε torchscript[https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html]. Αυτή η διαδικασία μπορεί να είναι απλή επιτυγχάνοντας τον σκοπό με ένα API call ή μπορεί να είναι πολύπλοκη και να αλλάξει το μοντέλο σε πολλά σημεία. Αφού γίνει η μετατροπή και προαιρετικά μπορεί να ακολουθήσει δεύτερο optimization του μοντέλου. Αρχικά πρέπει να πούμε τι είναι η torchScript. Η torchscript είναι μια ενδιάμεση αναπαράσταση ενός μοντέλου pytorch. Παρόλο που το περιβάλλον σε python προτιμάται και ενδείκνυται

για τα περισσότερα use cases στο σενάριο του deployment σε φορητές συσκευές η στο edge του δικτύου δεν ισχύει αυτό. Για αυτό τον λόγο η torchscript δημιουργήθηκε για να εκτελεί το μοντέλο σε περιβάλλον C++ το οποίο πετυχαίνει γενικά καλύτερη επίδοση συνεπώς το μοντέλο θα μπορεί να τρέξει οπουδήποτε μπορεί να τρέξει η libtorch. Με λίγα λόγια η torchscript ορίζεται ως high performance deployment runtime. Υπάρχουν δύο τρόποι για να μετατρέψουμε το μοντέλο σε κατάλληλη μορφή και περιγράφονται παρακάτω.

- Tracing

Σε αυτή την μέθοδο ως είσοδο έχουμε ένα μοντέλο και δείγματα δεδομένων. Κατά την διάρκεια της διαδικασίας τα δεδομένα εισάγονται στο μοντέλο σαν απλή διαδικασία inference και οι εντολές που εκτελούνται καταγράφονται σε torchscript.

- Scripting

Σε αυτή την μέθοδο είσοδος είναι μόνο το μοντέλο που δημιουργήθηκε στην python. Εδώ με static inspection του κώδικα του μοντέλου κάθε εντολή αντιστοιχείται στην αντίστοιχη torchscript.

Με βάση αυτά τα ερωτήματα που εγείρονται είναι πότε χρησιμοποιούμε την κάθε μία και για πιο λόγο υπάρχουν δύο μέθοδοι. Η απάντηση είναι ότι συνήθως η δεύτερη επιλογή είναι η καλύτερη λύση άλλα δεδομένων περιορισμών. Αυτοί οι περιορισμοί είναι.

1)Το μοντέλο δεν χρησιμοποιεί μη υποστηριζόμενες λειτουργίες pytorch. Περισσότερα εδώ[https://pytorch.org/docs/stable/jit_unsupported.html]

2)Η λογική του μοντέλου είναι περιορισμένη σε ένα υποσύνολο python functions που υποστηρίζονται. Περισσότερα εδώ[https://pytorch.org/docs/stable/jit_builtin_functions.html#python-built-in-functions]

3)Η σύνταξη του μοντέλου περιορίζεται σε ένα υποστηριζόμενο υποσύνολο. Περισσότερα εδώ[https://pytorch.org/docs/stable/jit_python_reference.html]

Ένα σημαντικό πλεονέκτημα της μεθόδου αυτής είναι ότι αν αποτύχει θα αποτύχει για έναν συγκεκριμένο λόγο κάτι που θα έχει ως αντιμετώπιση μια συγκεκριμένη μετατροπή του αρχικού κώδικα.

Παρόλα αυτά η πρώτη μέθοδος υπάρχει για συγκεκριμένους λόγους. Ο πρώτος είναι για το σενάριο όπου δεν έχουμε πρόσβαση στον κώδικα του μοντέλου για λόγους π.χ ιδιοκτησίας. Και ο δεύτερος λόγος είναι η περίπτωση των περιορισμών που αναφέρθηκαν για την μέθοδο scripting. Για την περίπτωση που χρησιμοποιήσουμε την μέθοδο trace θα πρέπει να αντιμετωπίσουμε και το θέμα του device pinning όπου ο

κώδικας που παράγεται μπορεί να χρησιμοποιεί παραπάνω από μία συσκευές(CPU,GPU,DSP,TPU).

Οι δύο παραπάνω μέθοδοι δεν αποκλείουν η μία την άλλη. Υπάρχουν σενάρια όπου συνδυάζονται ώστε να επιτευχθεί βέλτιστο αποτέλεσμα. Περισσότερα εδώ[https://pytorch.org/docs/stable/jit.html#mixing-tracing-and-scripting].

Torchscript Optimization

Το επόμενο βήμα στο workflow είναι το optimization του script που προέκυψε από το προηγούμενο βήμα. Εδώ δεν βρήκαμε πολλές πληροφορίες για το τι ακριβώς επιτυγχάνεται. Το πρώτο που συμβαίνει είναι να εφαρμοστούν optimizations που εφαρμόστηκαν και στο βήμα 2 με την λογική ότι έχουν νόημα δεδομένου ότι ο κώδικας του μοντέλου έχει μετατραπεί. Το δεύτερο είναι να βελτιστοποιηθούν τα ops εκείνα που μεταφράστηκαν από περιβάλλον python σε περιβάλλον C++.

Inference

Το τελευταίο βήμα είναι να σώσουμε το μοντέλο που δημιουργήθηκε στο κατάλληλο format και να χρησιμοποιήσουμε τα APIs που υπάρχουν για Android και IOS. Επίσης ενδύκνειται η χρήση maven και cocoapods για διαχείριση των dependencies.

4.3 Διαδικασία Deployment Pytorch/TensorFlow μοντέλου σε Python

Η τρίτη μέθοδος δεν περιλαμβάνει μετατροπή του μοντέλου. Αντί για αυτό στόχος είναι να τρέξουμε το μοντέλο στην συσκευή χρησιμοποιώντας python. Υπάρχουν πολλές εφαρμογές για να τρέξει κώδικας python. Ένας τρόπος είναι να χρησιμοποιήσουμε το micro-web framework Flask και να δημιουργήσουμε ένα application που θα εκτελεί το μοντέλο μας. Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε node.js. Ο πιο απλός τρόπος είναι να εγκαταστήσουμε το termux και από εκεί να εγκαταστήσουμε την python. Στην συνέχεια θα τρέξουμε μία εφαρμογή που ακούει για συνδέσεις σε ένα port στο localhost δημιουργώντας ένα socket στο localhost όπου θα συνδέεται η εφαρμογή java θα στέλνει το request στο μοντέλο που θα εκτελείται στο python script το οποίο θα επιστρέφει την απάντηση. Αυτή η μέθοδος δεν προσφέρει κάποιο πλεονέκτημα σε σχέση με τις δύο προηγούμενες αλλά αποτελεί επιλογή για την περίπτωση που δεν μπορούμε ή δεν θέλουμε να μετατρέψουμε το μοντέλο για κάποιο λόγο. Αυτό το σενάριο υπάρχει στην βιβλιογραφία αλλά είναι δύσκολο να ξέρουμε τις περιπτώσεις που συμβαίνει και είναι μάλλον σπάνιο. Αν δεν μας ενδιαφέρει να μετατρέψουμε το μοντέλο επίσης μπορούμε να επιλέξουμε αυτήν την μέθοδο. Αυτή η μέθοδος αφορά και την pytorch και την tensorflow.

5.Conclusion and key points

Κλείνοντας την παρουσίαση θα κάνουμε μία σύγκριση των 2 frameworks και θα εστιάσουμε σε κάποια σημαντικά σημεία. Όπως αναφέρεται και στην βιβλιογραφία η tensorflow και η tensorflowLite είναι δύο διαφορετικά frameworks με το ίδιο όνομα ενώ η pytorch και η pytorch mobile είναι το ίδιο framework και μοιράζονται το ίδιο codebase.

Model Conversion

Αυτό έχει σαν αποτέλεσμα η διαδικασία της μετατροπής να είναι γενικά πιο straightforward για την pytorch σε αντίθεση με την tensorflow όπου κάποιος που δημιουργεί ένα μοντέλο που θα μετατραπεί σε tfLite πρέπει να είναι πιο προσεκτικός σε σχέση με τα υποστηριζόμενα operations που προσφέρει η TensorFlow.

Supported Operations

Κάθε operator που είναι διαθέσιμος(CPU) στην pytorch είναι διαθέσιμος και στην pytorch mobile. Αντίθετα η tensorflow έχει ένα μόνο υποσύνολο operations που είναι διαθέσιμες στην tensorflow lite[https://www.tensorflow.org/lite/guide/op_select_allowlist]. Επίσης μπορούμε να το εντοπίσουμε κατά την διάρκεια της εγγραφής του μοντέλου ώστε να μην χρειαστεί να ξαναγράψουμε το μοντέλο ώστε να είναι συμβατό με tfLite.

Model Optimization/Quantization

Και τα δύο frameworks υποστηρίζουν τις ίδιες τεχνικές optimization/quantization για μείωση του μεγέθους του μοντέλου και φυσικά του inference time. Η tensorflow φαίνεται να είναι πιο ευέλικτη προσφέροντας περισσότερες επιλογές.

Metadata Support

Και τα δύο frameworks υποστηρίζουν μεταδεδομένα κάτι που χρησιμεύει ώστε να παραχθούν code wrappers αυτόματα.

6.Experiments and Results

Τέλος για λόγους επίδειξης δημιουργήσαμε τρείς εφαρμογές όπου δημιουργούμε δύο απλά μοντέλα linear regression, το ένα tensorFlow και το άλλο pytorch. Στην συνέχεια τα μετατρέψαμε εφαρμόζοντας τα default optimizations σε tfLite, tf_js και pytorch mobile. Ο κώδικας για την δημιουργία και την μετατροπη βρίσκεται στα δύο .ipynb που συνοδεύουν την εργασία το TFLite.ipynb και το Pytorch_mobile.ipynb το μοντέλο για την javascript είναι το .bin και το .json αρχείο, τα αρχεία .html και script.js είναι αυτά που φορτώνουν το μοντέλο και το tfjs@3.7.0 είναι η βιβλιοθήκη javascript. Το .ptl είναι το

μοντέλο pytorch mobile.Τέλος το model.tflite είναι το tflite μοντέλο. Το Implementation.ipynb περιγράφει τα βασικά API calls για κάθε μέθοδο από τις τρείς. Όλα τα αρχεία συνοδεύουν την εργασία. Στην συνέχεια δημιουργήσαμε τρείς εφαρμογές Android μία για κάθε περίπτωση tfLite, tf_js και pytorch mobile. Την τελευταία μέθοδο deployment, δηλαδή να τρέξουμε κατευθείαν το μοντέλο σε python χωρίς καμία μετατροπή δεν την εφαρμόσαμε. Οι τρείς Android εφαρμογές συνοδεύουν την εργασία. Τέλος όπως ζητήθηκε δημιουργήσαμε ένα app όπου τρέχει ένα μοντέλο image classification με χρήση κάμερας. Μετατρέψαμε το μοντέλο το οποίο ήταν 540MB σε tfLite. Το μέγεθος μειώθηκε στα 50MB. Ο κώδικας για την χρήση κάμερας είναι λίγο πολύπλοκος λόγω του ότι εμπλέκονται αρκετά callbacks αλλά η όλη ιδέα είναι ότι όταν όλα είναι έτοιμα τραβάμε φωτογραφία και στο callback που εκτελείται όταν η φωτογραφία είναι διαθέσιμη καλούμε το μοντέλο και δημιουργούμε ένα toast με τα αποτελέσματα. Περισσότερα στην παρουσίαση.

References

https://pytorch.org/mobile/home/

https://pytorch.org/tutorials/recipes/recipes_index.html

https://towardsdatascience.com/deep-learning-on-your-phone-pytorch-lite-interpreter-for-mobile-platforms-ae73d0b17eaa

https://pytorch.org/tutorials/recipes/mobile_perf.html?highlight=mobile

https://pytorch.org/tutorials/recipes/script_optimized.html

https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

https://www.kdnuggets.com/2021/11/on-device-deep-learning-pytorch-mobile-tensorflow-li17te.html

https://www.mathworks.com/company/newsletters/articles/what-is-int8-quantization-and-why-is-it-popular-for-deep-neural-networks.html

https://towardsdatascience.com/pytorch-jit-and-torchscript-c2a77bac0fff

https://paulbridger.com/posts/mastering-torchscript/

https://github.com/pytorch/pytorch/blob/master/android/README.md

https://www.analyticsvidhya.com/blog/2021/11/how-to-deploy-machine-learningml-model-on-android/

https://towardsdatascience.com/deploying-a-deep-learning-model-on-mobile-using-tensorflow-and-react-4b594fe04ab

https://medium.com/joytunes/deploying-a-tensorflow-model-to-android-69d04d1b0cba

https://towardsdatascience.com/the-essential-guide-to-learn-tensorflow-mobile-and-tensorflow-lite-a70591687800

https://www.tensorflow.org/lite/guide/inference

https://www.tensorflow.org/lite/guide

https://www.tensorflow.org/lite/inference_with_metadata/task_library/overview

https://towardsdatascience.com/run-tensorflow-models-in-the-browser-84280b3c71ad

https://js.tensorflow.org/api/latest/#Models-Loading

https://stackoverflow.com/questions/5749569/load-html-file-into-webview

https://developer.android.com/guide/webapps/webview#java

https://www.freecodecamp.org/news/building-a-node-js-application-on-android-part-1-termux-vim-and-node-js-dfa90c28958f/

https://sbrocks.medium.com/how-to-run-node-js-express-server-in-android-230459a6f3c4

https://github.com/christianversloot/machine-learning-articles/blob/main/tensorflow-model-optimization-an-introduction-to-quantization.md

https://blog.tensorflow.org/2020/02/tensorflowjs-for-react-native-is-here.html

https://towardsdatascience.com/step-by-step-yolo-model-deployment-in-localhost-using-python-8537e93a1784

https://www.geeksforgeeks.org/deploy-machine-learning-model-using-flask/

https://medium.com/@umerfarooq_26378/tools-to-run-python-on-android-9060663972b4

https://www.tensorflow.org/lite/convert/operation_fusion

https://pytorch.org/mobile/android/