

# Distributed Deep Learning using BigDL

Athanasiou Ioannis  
School of Electrical and Computers Engineering  
National Technical University of Athens  
Athens, Greece  
el17041@mail.ntua.gr

Karavangelis Athanasios  
School of Electrical and Computers Engineering  
National Technical University of Athens  
Athens, Greece  
el17022@mail.ntua.gr

**Abstract**— Η ανάπτυξη ενός συστήματος καταναμημένης βαθιάς μάθησης με τεχνολογίες παράλληλης επεξεργασίας αποτελεί μία ιδιαίτερα σύνθετη διαδικασία. Παρόλο που η βιβλιογραφία είναι ελλιπής σε τεχνικές λεπτομέρειες, η μελέτη του προβλήματος της καταναμημένης εκπαίδευσης παρουσιάζει ιδιαίτερο ενδιαφέρον. Στην παρούσα έρευνα, χρησιμοποιούμε το framework BigDL για το σκοπό αυτό. Δημιουργούμε έναν cluster στο cloud με χρήση του Apache Spark, αποτελούμενο από 3 κόμβους, και επανεκπαιδεύουμε μια πληθώρα μοντέλων. Πειραματιζόμαστε με διαφορετικούς συνδυασμούς χρησιμοποιούμενων κόμβων/πυρήνων του cluster, εστιάζοντας στην επίδραση που είχε η παραμετροποίηση αυτή στον χρόνο εκπαίδευσης.

**Keywords**— *καταναμημένη βαθιά μάθηση, BigDL, Apache Spark, cluster, batch size*

## I. ΕΙΣΑΓΩΓΗ

Ο σκοπός της εργασίας μας ήταν η εφαρμογή και μελέτη της καταναμημένης εκπαίδευσης νευρωνικών δικτύων, με την χρήση τεχνολογιών αιχμής.

Συγκεκριμένα, χρησιμοποιήσαμε το framework BigDL της Intel. Το framework αυτό, επιτρέπει την καταναμημένη εκπαίδευση νευρωνικών δικτύων, δίνοντας της δυνατότητα να παραμετροποιούμε την χρήση του διαθέσιμου infrastructure.

Στην μελέτη μας, στήσαμε ένα cluster με χρήση του spark, αποτελούμενο από 3 κόμβους, και επανεκπαιδεύσαμε μια πληθώρα μοντέλων, πειραματιζόμενοι με διαφορετικούς συνδυασμούς χρησιμοποιούμενων κόμβων/πυρήνων του cluster. Εστίασαμε στην επίδραση που είχε η παραμετροποίηση αυτή στον χρόνο εκπαίδευσης των μοντέλων, χωρίς να δώσουμε προσοχή στην ποιότητα των προβλέψεων που παρήγαγαν τα μοντέλα.

Εργαστήκαμε με μοντέλα στον τομέα της εικόνας, και πιο συγκεκριμένα με το πρόβλημα του image classification, χρησιμοποιώντας το σύνολο δεδομένων MNIST και PatchCamelyon.

## II. ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ CLUSTER

Το στήσιμο του ζητούμενου συστήματος ήταν από τα πιο απαιτητικά τμήματα της μελέτης μας. Το framework BigDL παρέχει μία πληθώρα τρόπων για να στηθεί το απαιτούμενο περιβάλλον, προτού ξεκινήσει η εκπαίδευση των μοντέλων. Υποστηρίζει την εκπαίδευση μοντέλων τόσο σε έναν κόμβο (local mode), όσο και σε πολλούς κόμβους (cluster mode).

Προφανώς, κύριος σκοπός της εργασίας μας ήταν η μελέτη της χρήσης πολλών κόμβων. Αρχικά, εργαστήκαμε

σε απλά παραδείγματα με την χρήση του local mode, με σκοπό την εξοικείωσή μας με το framework.

**Local mode:** Για την επιλογή αυτή, ακολουθήσαμε δύο διαφορετικές τεχνικές.

Η πρώτη περίπτωση ήταν η χρήση του περιβάλλοντος Google Colab, που επιτρέπει την εκπαίδευση μοντέλων με την χρήση του BigDL, μετά από τις εγκαταστάσεις των απαραίτητων βιβλιοθηκών.

Πιο συγκεκριμένα, απαιτείται η εγκατάσταση των εξής:

- Java (jdk8)
- python module “bigdl-orca”
- python modules “tensorflow” και “tensorflow-datasets”

Μετά από τις εγκαταστάσεις αυτές, το περιβάλλον που χρειαζόμασταν ήταν έτοιμο.

Η δεύτερη περίπτωση που εξετάσαμε, ήταν η χρήση των επίσημων docker images του framework. Πιο συγκεκριμένα, μετά από την εγκατάσταση του docker στα τοπικά μας μηχανήματα, κατεβάσαμε το image “intelanalytics/bigdl:2.1.0-SNAPSHOT”. Προφανώς, οι δημιουργοί του image είχαν φροντίσει ώστε το απαραίτητο περιβάλλον να είναι ήδη στημένο. Στην συνέχεια, για να ξεκινήσουμε την εκπαίδευση των μοντέλων, χρησιμοποιήσαμε την διεπαφή του Jupyter Notebook, όπου και προχωρήσαμε σε απλά πειράματα.

### Cluster mode:

Όσον αφορά την λειτουργία σε cluster mode, το BigDL παρέχει την δυνατότητα να χρησιμοποιηθούν τρεις κύριοι διαφορετικοί τρόποι:

1. Η εγκατάσταση του BigDL πάνω από ένα ήδη χτισμένο cluster με Hadoop/YARN
2. Η χρήση της πλατφόρμας Databricks πάνω από ένα cluster του AWS
3. Η χρήση της πλατφόρμας Dataproc του Google Cloud

Κατά την μελέτη μας, πειραματιστήκαμε με όλες τις ανωτέρω μεθόδους. Το κυριότερο πρόβλημα που αντιμετωπίσαμε ήταν η δύσκολη εύρεση συμβατότητας των εκδόσεων του Apache Spark με αυτές της BigDL αλλά και της Python. Αυτό μας οδήγησε σε πολλές αποτυχημένες προσπάθειες για τη δημιουργία ενός cluster που λειτουργεί ορθά.

Γενικά, παρατηρήσαμε ότι το documentation ήταν ελλιπές στην πλειονότητα των μεθόδων, και υπήρχε σημαντική έλλειψη περιγραφής των απαιτούμενων εκδόσεων των διάφορων βιβλιοθηκών που έπρεπε να

χρησιμοποιηθούν. Η έλλειψη αυτή, μας ώθησε να στραφούμε προς την λύση του Databricks, η οποία παρέχει το πιο ολοκληρωμένο documentation για το στήσιμο του cluster.

Προφανώς, το πρώτο βήμα για την κατασκευή του cluster ήταν η δέσμευση των απαραίτητων υπολογιστικών κόμβων και η οργάνωσή τους σε ένα cluster. Αρχικά, έπρεπε να επιλέξουμε έναν πάροχο cloud. Μπορούσαμε να επιλέξουμε ανάμεσα στα AWS, Microsoft Azure και Google Cloud Platform. Επιλέξαμε το AWS, καθώς παρέχει το πιο σαφές documentation για την διαδικασία που θα έπρεπε να ακολουθήσουμε.

Αρχικά, δημιουργήσαμε ένα Databricks Workspace, από την κεντρική διαχειριστική σελίδα του λογαριασμού μας.

Στην συνέχεια, μέσα από την διαχειριστική σελίδα του workspace που κατασκευάσαμε, προχωρήσαμε στην δημιουργία του Cluster. Στο σημείο αυτό, χρειάστηκε ιδιαίτερη προσοχή στην επιλογή των κατάλληλων προδιαγραφών του cluster:

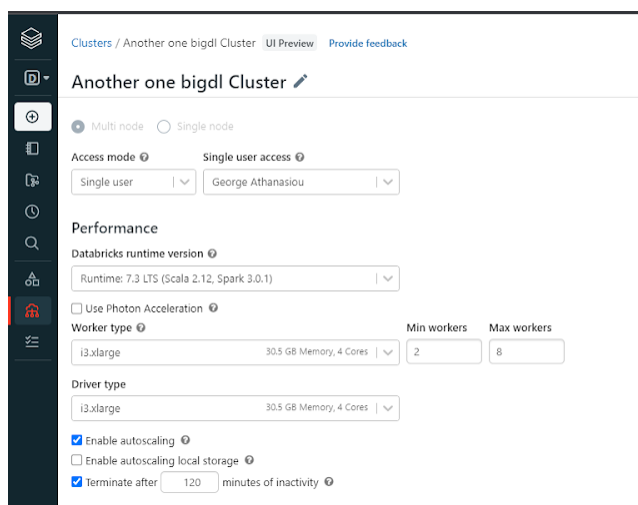


Fig. 1 Στιγμιότυπο οθόνης από το UI του Databricks για cluster configuration

Όπως φαίνεται στην εικόνα, χρειάστηκε να επιλέξουμε το runtime version του cluster, το οποίο στην συνέχεια καθόρισε τις εκδόσεις των κύριων βιβλιοθηκών που θα έπρεπε να εγκαταστήσουμε. Επιλέξαμε το runtime 7.3 LTS, λόγω της έκδοσης της Python με την οποία ήταν συμβατό (3.7.5), η οποία ήταν αναγκαία για την συμβατότητα με βιβλιοθήκες μηχανικής μάθησης που θα χρησιμοποιούσαμε στην συνέχεια της μελέτης μας.

Μία επιπρόσθετη σχεδιαστική απόφαση την οποία κληθήκαμε να λάβουμε, ήταν η δέσμευση των κατάλληλων πόρων. Πιο συγκεκριμένα, μας δινόταν η δυνατότητα να διαλέξουμε από μία πληθώρα κόμβων, με διαφορετική υπολογιστική ισχύ (διαφορετική διαθέσιμη μνήμη και διαφορετικός αριθμός πυρήνων). Η επιλογή μας βασίστηκε, κυρίως, στον αριθμό των διαθέσιμων πυρήνων ώστε να μπορούσαμε να μελετήσουμε μία πληθώρα διαφορετικών συνδυασμών τους, και έτσι επιλέξαμε τον τύπο worker i3.xlarge, που διαθέτει 30.5GB μνήμης και 4 πυρήνες.

Το επόμενο βήμα που έπρεπε να ακολουθήσουμε, προτού να έχουμε ένα λειτουργικό cluster όπου μπορούμε να εκτελέσουμε τα πειράματά μας, ήταν η εγκατάσταση των απαραίτητων βιβλιοθηκών στα μηχανήματα του cluster. Σε αυτό το σημείο χρειάστηκε ιδιαίτερη προσοχή, αφού οι

εκδόσεις των python wheels και των jar αρχείων που εγκαταστήσαμε, ήταν αναγκαίο να συμπίπτουν με τις αντίστοιχες εκδόσεις του runtime περιβάλλοντος που είχαμε επιλέξει για το cluster.

Πιο συγκεκριμένα, χρειάστηκε να βρούμε online τα Python Wheels για την εγκατάσταση των βιβλιοθηκών bigdl\_orca\_spark και bigdl\_dllib\_spark, και να τα εγκαταστήσουμε από το UI του Databricks για τον cluster μας. Με παρόμοιο τρόπο, έπρεπε να εγκαταστήσουμε στην συνέχεια και τα αρχεία αντίστοιχα αρχεία jar για τις παραπάνω βιβλιοθήκες.

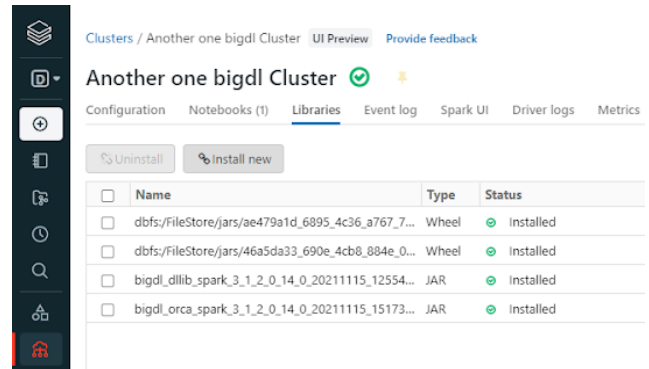


Fig. 2 Οι εγκατεστημένες βιβλιοθήκες στο cluster μας όπως φαίνονται στο UI του Databricks

Τελικά, καταλήξαμε να χρησιμοποιήσουμε τις εξής εκδόσεις:

- Apache Spark 3.1.0
- BigDL 0.14.0

Κάποιες περαιτέρω βιβλιοθήκες που χρησιμοποιήσαμε, ήταν αρκετό να εγκατασταθούν κατά την διάρκεια των πειραμάτων, από την διεπαφή των Jupyter Notebooks που παρέχει το Databricks.

Προτού προχωρήσουμε στην εκτέλεση των πειραμάτων, παρατηρήσαμε ότι το Databricks μας δίνει πρόσβαση στο User Interface του Spark Cluster που έχει χτιστεί, που διαθέτει έναν master και δύο slave workers.

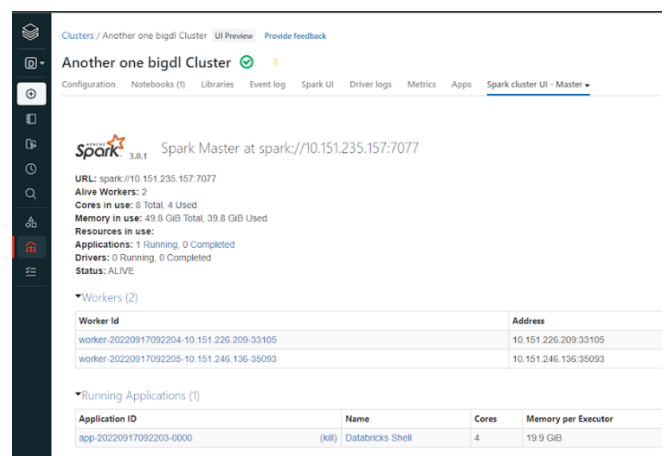


Fig. 3 Στιγμιότυπο οθόνης από το Spark UI του cluster μας

### III. FRAMEWORKS - ΒΙΒΛΙΟΘΗΚΕΣ

#### A. BigDL

Το BigDL είναι ένα framework καταναμημένης βαθιάς μάθησης για το Apache Spark. Με την χρήση του, οι προγραμματιστές μπορούν να αναπτύσσουν εφαρμογές βαθιάς μάθησης ως τυπικά προγράμματα Spark, τα οποία μπορούν να εκτελεστούν απευθείας πάνω σε υπάρχοντα Spark/Hadoop clusters.

Παρουσιάζει πολλά πλεονεκτήματα ανάμεσα στα οποία είναι τα εξής:

- Πλούσια υποστήριξη βαθιάς μηχανικής μάθησης - Είναι σχεδιασμένο σύμφωνα με την βιβλιοθήκη μηχανικής μάθησης Torch, και παρέχει ολοκληρωμένη υποστήριξη για βαθιά μάθηση, συμπεριλαμβανομένων αριθμητικών υπολογισμών μέσω των Tensors, και νευρωνικών δικτύων υψηλού επιπέδου. Ταυτόχρονα, επιτρέπει την φόρτωση προ-εκπαιδευμένων μοντέλων σε προγράμματα Spark.
- Εξαιρετικά υψηλή απόδοση - Για να επιτύχει υψηλή απόδοση, χρησιμοποιεί πολυ-νηματικό προγραμματισμό (multi-threading) σε κάθε πρόγραμμα Spark. Ως αποτέλεσμα, είναι τάξεις μεγέθους ταχύτερο από τις open source εκπαιδεύσεις του Torch σε έναν κόμβο.
- Αποτελεσματική κλιμάκωση - Μπορεί να κλιμακωθεί (scale-out) και να χρησιμοποιηθεί για data analytics σε κλίμακα μεγάλων δεδομένων, λόγω της χρήσης του Apache Spark, καθώς και αποτελεσματικών υλοποιήσεων Stochastic Gradient Descent[1].

#### B. Apache Spark

Το framework Apache Spark είναι μια open-source ενοποιημένη μηχανή ανάλυσης (unified analysis engine) για επεξεργασία δεδομένων μεγάλης κλίμακας και μηχανική μάθηση.

Πρόκειται για ένα framework το οποίο χρησιμοποιείται από πολλές εταιρείες που λειτουργούν με πολλά δεδομένα όπως Netflix, eBay, Yahoo κ.α. Είναι το μεγαλύτερο open source project στην κοινότητα των big data.

Το Apache Spark χαρακτηρίζεται από την ταχύτητα που παρέχει για επεξεργασία δεδομένων μεγάλης κλίμακας (100 φορές πιο γρήγορο από το Hadoop), από την ευκολία χρήσης του και από το γεγονός ότι είναι μια ενοποιημένη μηχανή ανάλυσης, καθώς περιέχει βιβλιοθήκες υψηλότερου επιπέδου όπως για SQL queries, data streaming, μηχανική μάθηση κ.α.

Η υψηλή απόδοση του Apache Spark προέρχεται από την ικανότητα του να επεξεργάζεται in-memory μέσω memory-persistent RDDs ή DataFrames αντί να σώζει τα δεδομένα σε κάποιο σκληρό δίσκο όπως στην αρχιτεκτονική που ορίζει το Hadoop MapReduce[2].

Γενικά, ένας Spark cluster όπως ο δικός μας αποτελείται από έναν master(driver) κόμβο και πολλούς worker κόμβους. Στους master κόμβους τρέχουν τα Spark jobs τα οποία αποτελούνται από πολλά Spark tasks, καθένα εκ των οποίων δουλεύει σε ένα τμήμα των δεδομένων (data partition). Ο master, λοιπόν, είναι υπεύθυνος για τον προγραμματισμό και διαμοιρασμό των tasks στους workers όπου τρέχουν πραγματικά τα Spark tasks.

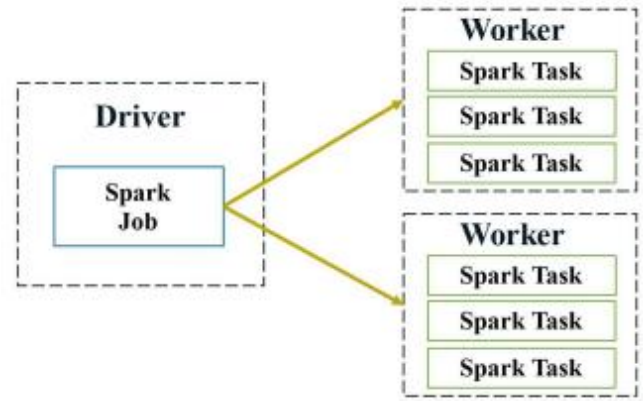


Fig. 4 Γραφική απεικόνιση ενός driver και των workers του καθώς και των Spark jobs και Spark tasks

Στη δική μας περίπτωση χρησιμοποιούμε το framework BigDL σε έναν Apache Spark cluster όπου κάθε επανάληψη της εκπαίδευσης γίνεται στη μορφή ενός Spark job και κάθε Spark task τρέχει το ίδιο μοντέλο σε ένα υποσύνολο των δεδομένων (batch)[3].

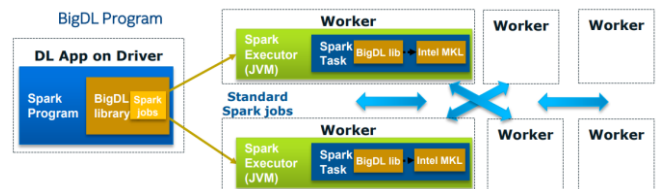


Fig. 5 Ο driver κόμβος καθώς και οι workers κόμβοι περιέχουν τις βιβλιοθήκες BigDL για την κατάλληλη επεξεργασία των δεδομένων της εκπαίδευσης

#### C. Orca

Η πιο σημαντική βιβλιοθήκη, που κατέστησε δυνατή την καταναμημένη εκπαίδευση των μοντέλων, μέσα από την διεπαφή ενός απλού Python Notebook, ήταν η βιβλιοθήκη Orca.

Η βιβλιοθήκη αυτή, ουσιαστικά αποτελεί ένα πλαίσιο ενορχήστρωσης pipelines, που επιτρέπει στον προγραμματιστή να ορίζει δυναμικές πηγές δεδομένων και να τις συνδέει ρητά με λειτουργίες επεξεργασίας. Παρέχει πολλές δυνατότητες για την εργασία με δομές δεδομένων της βιβλιοθήκης Pandas της Python, αλλά μπορεί να χρησιμοποιηθεί και με οποιοδήποτε άλλο είδος δεδομένων. Στην πειραματική μελέτη μας, χρησιμοποιήθηκε για να καθορίζουμε το πλήθος των κόμβων και πυρήνων που θα χρησιμοποιηθούν κατά την εκπαίδευση των μοντέλων[4].

#### D. bigdl-orca

Η βιβλιοθήκη αυτή μας επιτρέπει να αξιοποιήσουμε τις δυνατότητες της Orca ενώ δουλεύουμε με το framework BigDL, και παρέχει μία υψηλού επιπέδου προγραμματιστική διεπαφή. Χαρακτηριστικά η προαναφερθείσα δήλωση του πλήθους των πόρων που θέλουμε να χρησιμοποιήσουμε, γίνεται προγραμματιστικά με την εξής εντολή (σε γλώσσα Python):

```
init_orca_context(  
    cluster_mode="spark-submit",  
    num_nodes=num_nodes,  
    cores=cores  
)
```

Fig. 6 Υλοποίηση σε Python για την εντολή init\_orca\_context

Ταυτόχρονα, η βιβλιοθήκη, μας έδωσε πρόσβαση, μέσω του submodule της, `bigdl.orca.learn.tf.estimator`, στην κλάση `Estimator`, η οποία αναλαμβάνει την εκπαίδευση του μοντέλου που καθορίζουμε, με ένα μικρό και κατανοητό τμήμα κώδικα (σε Python), εφαρμόζοντας τις επιλογές που έχουμε ήδη κάνει με την εντολή `init_orca_context[5]`.

```
est =
Estimator.from_keras(keras_model=model)
est.fit(
    data=train_data,
    validation_data=validation_data
    batch_size=320,
    epochs=epochs,
)
```

Fig. 7 Υλοποίηση της διαδικασίας του fitting σε Python με χρήση Orca Estimator

#### E. Tensorflow – Tensorflow datasets

Η βιβλιοθήκη Tensorflow αποτέλεσε την μόνη βιβλιοθήκη που χρησιμοποιείται καθαρά για μηχανική μάθηση, που αξιοποιήσαμε.

Μέσω του submodule `tensorflow.keras.applications`, αποκτήσαμε πρόσβαση σε μία πληθώρα ήδη εκπαιδευμένων μοντέλων[6][8].

Μέσω του submodule `tensorflow.keras.layers` αποκτήσαμε πρόσβαση σε πολλά ήδη υλοποιημένα στρώματα νευρωνικών δικτύων (`Dense`, `GlobalAveragePooling2D` κ.α.), τα οποία χρησιμοποιήσαμε για να επεκτείνουμε τα μοντέλα που είχαμε ήδη φορτώσει, και να προβούμε στην διαδικασία του fine-tuning τους για τα πειράματά μας.

Τέλος, με την χρήση της βιβλιοθήκης `tensorflow-datasets`, μπορέσαμε να αποκτήσουμε πρόσβαση σε σύνολα δεδομένων, μέσω streaming από το Google Storage, διαδικασία η οποία αναλύεται και παρακάτω[7].

### IV. ΠΕΙΡΑΜΑΤΙΚΗ ΕΚΠΑΙΔΕΥΣΗ ΤΩΝ ΜΟΝΤΕΛΩΝ

Όσον αφορά την μελέτη μας μετά από το στήσιμο του Infrastructure, στηριχτήκαμε στην μέθοδο του fine-tuning προ-εκπαιδευμένων μοντέλων από το keras, στο task του Image Classification με τα datasets MNIST και PatchCamelyon.

Τα μοντέλα που εκπαιδεύσαμε είναι τα:

- Ένα δικό μας απλό συνελκτικό μοντέλο (CNN)
- MobileNetV2
- ResNet50

#### A. Σύνολα Δεδομένων (Datasets)

**MNIST:** Το σύνολο δεδομένων MNIST αποτελείται από εικόνες χειρόγραφων αριθμητικών ψηφίων, και στόχος είναι η αναγνώριση του κάθε ψηφίου, δηλαδή η κατηγοριοποίησή του σε μία από τις δέκα πιθανές κλάσεις (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)[9].

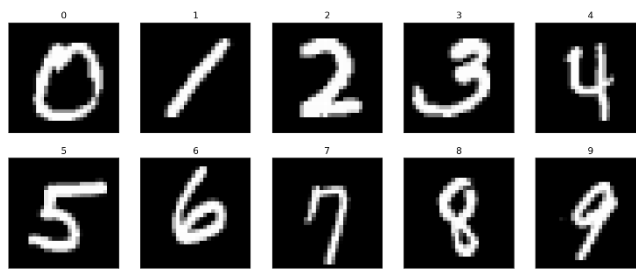


Fig. 8 Παραδείγματα εικόνων χειρόγραφων ψηφίων από το σύνολο δεδομένων mnist

**PatchCamelyon:** Το σύνολο δεδομένων PatchCamelyon αποτελείται από 327.680 έγχρωμες φωτογραφίες που έχουν εξαχθεί από ιστοπαθολογικές σαρώσεις τομών λεμφαδένων, με κάθε δείγμα να είναι χαρακτηρισμένο με την παρουσία (ή απουσία) μεταστατικού ιστού[10].

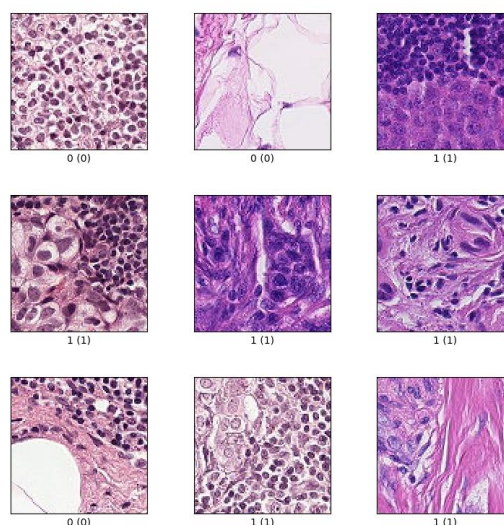


Fig. 9 Παραδείγματα εικόνων σαρώσεων από το σύνολο δεδομένων patch camelyon με label 0 ή 1 ανάλογα με την παρουσία μεταστατικού ιστού

Προσπαθήσαμε να επιλέξουμε δύο σύνολα δεδομένων τα οποία να διαφέρουν σημαντικά ως προς το μέγεθός τους, ώστε να μελετήσουμε τον ρόλο που διαδραματίζει το μέγεθος αυτό στον χρόνο εκπαίδευσης συναρτήσει του αριθμού των διαθέσιμων κόμβων-πυρήνων.

Ακόμα, ιδιαίτερη αναφορά αξίζει στον τρόπο αποθήκευσης του συνόλου δεδομένων και του διαμοιρασμού του στους κόμβους του cluster. Αρχικά, δοκιμάσαμε να αποκτήσουμε πρόσβαση στο dataset με ένα απλό download μέσω της βιβλιοθήκης `tensorflow-datasets`, αλλά στην περίπτωση αυτή, αντιμετωπίσαμε πρόβλημα με τον διαμοιρασμό του dataset στο cluster. Συγκεκριμένα, το dataset έπρεπε να είναι προσβάσιμο και από τον master κόμβο άλλα και από τους υπόλοιπους workers.

Για να λύσουμε αυτό το πρόβλημα, χρησιμοποιήσαμε την υπηρεσία Google Storage. Πιο συγκεκριμένα, ανοίξαμε έναν χώρο αποθήκευσης (bucket) στο Google Storage, και φορτώσαμε τα datasets που θέλαμε να χρησιμοποιήσουμε μέσω αυτού του bucket με απευθείας streaming στους κόμβους του cluster μας[13][14].



mybucket-bigdl-ece-ntua


Location

us (multiple regions in United States)

Storage class

Standard

Public access

 Public to internet

Protection

None

OBJECTS

CONFIGURATION

PERMISSIONS


PROTECTION

LIFECYCLE

Buckets

>

mybucket-bigdl-ece-ntua



UPLOAD FILES

UPLOAD FOLDER

CREATE FOLDER


TRANSFER DATA

MANAGE HOLDS


DOWNLOAD

DELETE

Filter by name prefix only

 Filter

Filter objects and folders

 Show deleted objects





<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access	Version ID
<input type="checkbox"/>	 <a href="#">ipynb_checkpoints/</a>	—	Folder	—	—	—	—	—
<input type="checkbox"/>	 <a href="#">google-cloud-dataproc-metainfo/</a>	—	Folder	—	—	—	—	—
<input type="checkbox"/>	 <a href="#">mnist/</a>	—	Folder	—	—	—	—	—
<input type="checkbox"/>	 <a href="#">patch_cameleon/</a>	—	Folder	—	—	—	—	—

Fig. 10 Στιγμιότυπο οθόνης από το bucket μας όπου φαίνονται τα datasets mnist και patch\_cameleon

## B. Τα μοντέλα που χρησιμοποιήσαμε

### Το δικό μας CNN:

Στο δικό μας συνελκτικό νευρωνικό δίκτυο, χρησιμοποιήσαμε 2 Conv2D layers του Keras ακολουθούμενα από ένα MaxPooling layer το κάθε ένα. Με τα Conv2D layers περνάμε των τριών διαστάσεων διάνυσμα (μια εικόνα σε RGB μορφή) μέσα από feature maps (τον αριθμό των οποίων καθορίζουμε εμείς και είναι 20 και 50 αντίστοιχα) κι έπειτα η έξοδος από αυτό το στρώμα μειώνεται σε μέγεθος (downsampling) μέσω των MaxPooling layers. Η διαδικασία αυτή ονομάζεται feature extraction.

Έπειτα από το feature extraction, χρησιμοποιούμε ένα Flatten layer ώστε να ενοποιήσουμε τα δεδομένα σε ένα διάνυσμα μιας διάστασης και το διάνυσμα αυτό τροφοδοτείται στα hidden Dense layers (πυκνά στρώματα). Χρησιμοποιούμε 500 Dense layers και έπειτα 1 output layer με softmax συνάρτηση ενεργοποίησης (activation function) ώστε να έχουμε ως έξοδο τις 10 ή 2 (ανάλογα το dataset) κατηγορίες που θέλουμε να διαχωρίσουμε.

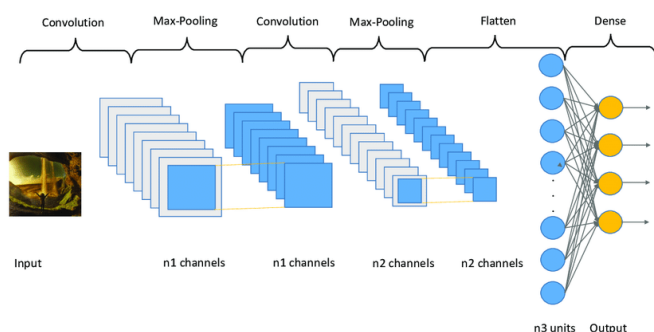


Fig. 11 Γραφική απεικόνιση του μοντέλου μας

**MobileNetV2:** Η οικογένεια των μοντέλων MobileNets αποτελεί μία αποδοτική έκδοση των κλασικών (για τα tasks μηχανικής μάθησης σχετικά με τις εικόνες) Συνελκτικών Νευρωνικών Δικτύων, με εφαρμογές στην Όραση των Κινητών Συσκευών και Ενσωματωμένων Συστημάτων. Η λειτουργία τους βασίζεται σε μία βελτιστοποιημένη αρχιτεκτονική που χρησιμοποιεί συνελίξεις που μπορούν να διαχωριστούν σε βάθος, για την δημιουργία ελαφρών νευρωνικών δικτύων που μπορούν να έχουν χαμηλή καθυστέρηση για εφαρμογές σε κινητές και ενσωματωμένες συσκευές.

Το βαθύ νευρωνικό δίκτυο MobileNetV2 αποτελείται από 3.5 εκατομμύρια παραμέτρους και έχει βάθος 105 στρωμάτων[12]. Αποτελεί εξέλιξη του μοντέλου MobileNet, το οποίο αποτελείται από 4.3 εκατομμύρια υπερπαραμέτρους, και έχει βάθος 55 στρωμάτων[11].

Ένας από τους βασικούς λόγους που επιλέξαμε το MobileNetV2 ως ένα από τα μοντέλα που θα κάναμε fine-tuning, είναι το μικρό του μέγεθος που επιτρέπει την γρήγορη εκπαίδευσή του, και η πρόσβασή μας σε προ-εκπαιδευμένη έκδοσή του, στο πιο ευρέως γνωστό dataset για image classification “ImageNet”.

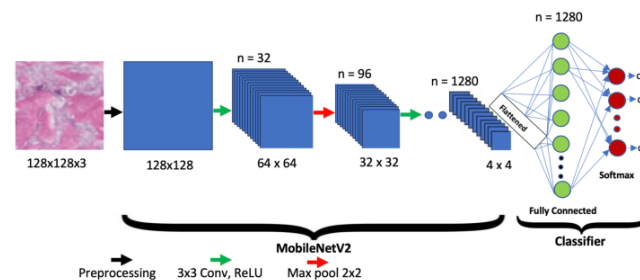


Fig. 12 Η αρχιτεκτονική του μοντέλου MobileNetV2

**ResNet50:** Το μοντέλο ResNet50 είναι ένα συνελκτικό νευρωνικό δίκτυο και αποτελεί μία παραλλαγή του μοντέλου ResNet. Χρησιμοποιεί 50 layers (48 συνελκτικά στρώματα 1 MaxPool και 1 Average Pool στρώμα). Ένα ResNet είναι ένα «Residual neural network» είναι ένα τεχνητό νευρωνικό δίκτυο το οποίο «στοιβάξει» residual blocks για τη δημιουργία ενός δικτύου.

Το ResNet50 έχει πάνω από 23 εκατομμύρια παραμέτρους, γεγονός που καθιστά την εκπαίδευσή του χρονοβόρα και απαιτητική ως προς τους πόρους. Πρόκειται για το πιο μεγάλο σε όγκο μοντέλο που θα εκπαιδευσουμε.

Ενδεικτικά, το ResNet50 pretrained στο σύνολο δεδομένων “ImageNet” σε πάνω από 1 εκατομμύριο εικόνες μπορεί να κατηγοριοποιήσει αυτές σε 1000 κατηγορίες αντικειμένων.

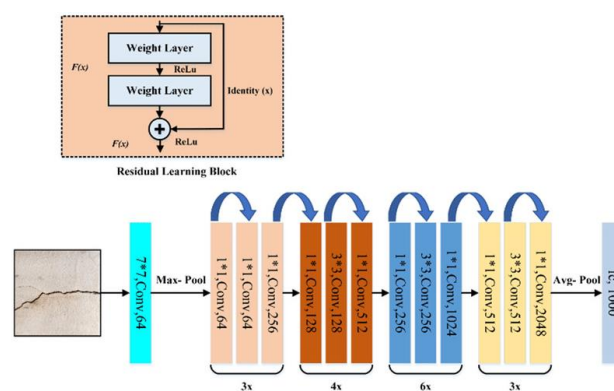


Fig. 13 Η αρχιτεκτονική του μοντέλου ResNet50

## C. Mini-Batch Stochastic Gradient Descent

Για να εξηγήσουμε την παρακάτω τεχνική στην οποία βασιστήκαμε στην πειραματική διαδικασία μας είναι χρήσιμο να αναφέρουμε πρώτα τις παρακάτω μεθόδους.

## Gradient Descent

Στα μαθηματικά, η μέθοδος gradient descent (η αλλιώς steepest descent) είναι ένας πρώτης τάξης επαναληπτικός αλγόριθμος βελτιστοποίησης για την εύρεση ενός τοπικού ελάχιστου μιας διαφοροποιήσιμης συνάρτησης. Η ιδέα είναι ότι εκτελούμε επαναλαμβανόμενα βήματα προς την αντίθετη κατεύθυνση της κλίσης (ή κατά προσέγγιση κλίσης) της συνάρτησης στο τρέχον σημείο, επειδή αυτή είναι η κατεύθυνση της πιο απότομης καθόδου.

Σε ένα πρόβλημα μηχανικής μάθησης ας υποθέσουμε πως θέλουμε να εκπαιδύσουμε τον αλγόριθμό μας με gradient descent για να ελαχιστοποιήσετε τη συνάρτηση κόστους  $J(w, b)$  και να φτάσουμε στο τοπικό της ελάχιστο τροποποιώντας τις παραμέτρους του ( $w$  και  $b$ ).

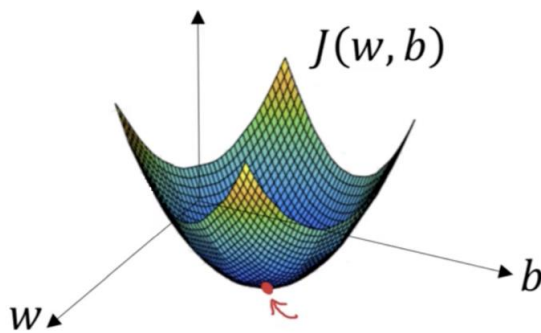


Fig. 15 Η παρακάτω εικόνα δείχνει τους οριζόντιους άξονες που αντιπροσωπεύουν τις παραμέτρους ( $w$  και  $b$ ), ενώ η συνάρτηση κόστους  $J(w, b)$  αναπαρίσταται στους κατακόρυφους άξονες. Η κατάβαση κλίσης είναι μια κυρτή συνάρτηση.

Γνωρίζουμε ότι θέλουμε να βρούμε τις τιμές των  $w$  και  $b$  που αντιστοιχούν στο ελάχιστο της συνάρτησης κόστους. Για να αρχίσουμε να βρίσκουμε τις σωστές τιμές αρχικοποιούμε τα  $w$  και  $b$  με κάποιους τυχαίους αριθμούς. Στη συνέχεια, η κατάβαση με κλίση ξεκινά από αυτό το σημείο και ακολουθεί επαναληπτικά προς την πιο απότομη κατεύθυνση προς τα κάτω μέχρι να φτάσει στο σημείο όπου η συνάρτηση κόστους είναι όσο το δυνατόν μικρότερο [15].

## Stochastic Gradient Descent(SGD)

Η Stochastic Gradient Descent (σε συντομογραφία SGD) μπορεί να θεωρηθεί ως μια «στοχαστική» προσέγγιση της gradient descent, καθώς αντικαθιστά την πραγματική κλίση (υπολογισμένη από ολόκληρο το σύνολο δεδομένων) με μια εκτίμηση αυτής (υπολογισμένη από ένα τυχαία επιλεγμένο υποσύνολο δεδομένων). Ειδικά σε προβλήματα βελτιστοποίησης υψηλών διαστάσεων, αυτό μειώνει την πολύ υψηλή υπολογιστική επιβάρυνση, επιτυγχάνοντας ταχύτερες επαναλήψεις με αντάλλαγμα χαμηλότερο ποσοστό σύγκλισης [16].

Στο Gradient Descent, υπάρχει ένας όρος που ονομάζεται "batch" που υποδηλώνει τον συνολικό αριθμό δειγμάτων από ένα σύνολο δεδομένων που χρησιμοποιείται για τον υπολογισμό του "gradient" για κάθε επανάληψη. Στην κλασσική Gradient Descent, το batch μας είναι ολόκληρο το σύνολο δεδομένων. Αν και η χρήση ολόκληρου του συνόλου δεδομένων μας βοηθάει αρκετά για να φτάσουμε στα ελάχιστα με λιγότερους θορύβους και λιγότερη τυχαιότητα,

υπάρχει πρόβλημα όταν το σύνολο δεδομένων μας μεγαλώσει.

Στη μέθοδο Stochastic Gradient Descent, το μέγεθος του batch είναι ένα δείγμα από το σύνολο δεδομένων. Το γεγονός αυτό μας προϋποθέτει για ένα μονοπάτι προς την εύρεση ελαχίστου με περισσότερο θόρυβο, αλλά αυτό δε μας ενδιαφέρει καθώς το ελάχιστο θα βρεθεί και σε σημαντικά μικρότερο χρόνο εκπαίδευσης. Πρέπει να παρατηρήσουμε ότι παρά τις περισσότερες επαναλήψεις που θα απαιτηθούν, η μέθοδος SGD είναι υπολογιστικά πολύ λιγότερο «ακριβή» από την κλασσική Gradient Descent και προτιμάται για βελτιστοποίηση αλγορίθμων μάθησης [17].

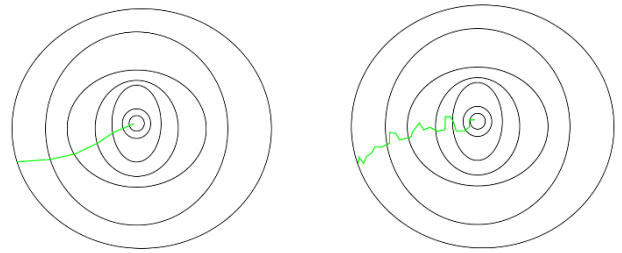


Fig. 14 Παράδειγμα εύρεσης ελαχίστου με Gradient Descent(αριστερά) και Stochastic Gradient Descent(δεξιά). Παρατηρούμε τον θόρυβο που οφείλεται στην τυχαιότητα της μεθόδου SGD.

## Mini-Batch SGD

Η μέθοδος Mini Batch SGD, η οποία χρησιμοποιούμε στις εκπαιδεύσεις μας, θεωρείται ότι είναι η διασταύρωση μεταξύ κλασσικής Gradient Descent και SGD. Σε αυτήν την προσέγγιση, αντί να επαναλαμβάνουμε για ολόκληρο το σύνολο δεδομένων ή για ένα δείγμα τη φορά, χωρίζουμε το σύνολο δεδομένων σε μικρά υποσύνολα (batches) και υπολογίζουμε τις gradients για κάθε batch.

Στην πειραματική μας διαδικασία ξεκινήσαμε με σταθερό μέγεθος batch για όλους τους αριθμούς κόμβων. Ωστόσο, η βέλτιστη υπολογιστική επιλογή είναι ένα διαφορετικό μέγεθος batch ανάλογα με τον αριθμό των κόμβων.

Η state-of-the-art τεχνική υπαγορεύει ένα γραμμικά αυξανόμενο μέγεθος batch που δίνεται από τον τύπο:

$$total\_batch\_size = worker\_batch\_size \cdot number\_of\_workers$$

Αυτό το μοτίβο ακολουθείται καθώς στην εκπαίδευση σε ένα Spark cluster ένα Spark job δημιουργείται για κάθε επανάληψη, δηλαδή για κάθε batch, το οποίο αντίστοιχα διασπάται σε tasks στους workers.

Έτσι αυξάνοντας το batch εκμεταλλευόμαστε τον παραλληλισμό στην εκτέλεση και έτσι μειώνεται ο χρόνος εκπαίδευσης όπως θα δούμε και στα αποτελέσματα. Αυτό συμβαίνει διότι ο αριθμός των επαναλήψεων και αντίστοιχα των Spark jobs μειώνεται και συνεπώς έχουμε μείωση του overhead που οφείλεται σε διαδικασίες του Spark (προγραμματισμός, διαμοιρασμός tasks στους κόμβους κ.τ.λ.).

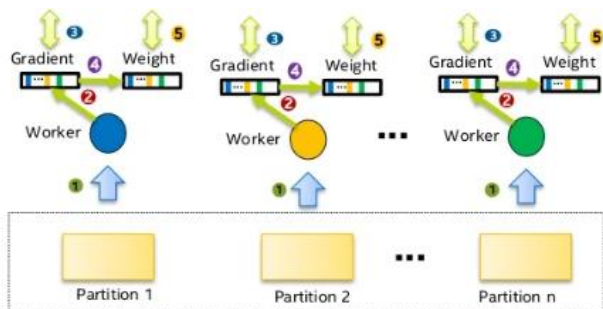


Fig. 16 Αρχιτεκτονική Parameter Server του BigDL πάνω από τον Spark Block Manager

Ωστόσο, με υψηλότερο batch παρατηρείται λογική μείωση στην ακρίβεια του μοντέλου καθώς έχουμε λιγότερες επαναλήψεις λόγω του αυξημένου batch. Το παραπάνω θέτει ένα άνω όριο στην κλιμάκωση που μπορεί να γίνει για το batch καθώς η ακρίβεια του μοντέλου είναι σημαντικός παράγοντας. Στα πλαίσια της εργασίας μας προσπαθήσαμε να κρατήσουμε τη μείωση αυτή σε χαμηλά επίπεδα, σε συνδυασμό με ικανοποιητική μείωση του χρόνου εκπαίδευσης.

#### D. Fine Tuning

Η διαδικασία του fine-tuning έγκειται στην τροποποίηση προ-εκπαιδευμένων μοντέλων, ώστε να αξιοποιήσουμε την εκπαίδευση που έχουν λάβει σε γενικού περιεχομένου tasks, για την εξειδίκευσή τους σε ένα πιο ειδικού σκοπού task.

Η γενικότερη λογική πίσω από το fine-tuning, είναι ότι στα βαθιά νευρωνικά δίκτυα, τα πρώτα στρώματα νευρώνων είναι σχεδιασμένα ώστε να εξάγουν τα πιο γενικά features των δεδομένων εισόδου. Τα χαρακτηριστικά αυτά, είναι χρήσιμα για κάθε παρόμοιο task, και επομένως είναι ωφέλιμο να μπορούμε να χρησιμοποιήσουμε τα στρώματα που τα εξάγουν.

Επομένως, κατά το fine-tuning, επιλέγουμε ένα ποσοστό των αρχικών στρωμάτων του προ-εκπαιδευμένου δικτύου και τα “παγώνουμε” (freezing), δηλαδή δεν μεταβάλλουμε τα βάρη τους. Μπορούμε να τροποποιήσουμε το μοντέλο, προσθέτοντας κάποια επιπλέον στρώματα στο τέλος του, και στην συνέχεια εκπαιδεύουμε το νέο δίκτυο (εκτός των “παγωμένων” στρωμάτων) στο task που θέλουμε να λύσουμε στην συγκεκριμένη περίπτωση.

#### E. Model Parallelism - Data Parallelism

Στην σύγχρονη εποχή, κατά την εκπαίδευση Βαθιών Νευρωνικών Δικτύων, έχουν αυξηθεί σημαντικά τόσο το μέγεθος των δικτύων (αριθμός των βαρών τους), όσο και το μέγεθος των συνόλων δεδομένων (datasets) που χρησιμοποιούνται. Με βάση τις αυξήσεις σε αυτά τα δύο μεγέθη, έχουν επικρατήσει δύο κύριες τεχνικές για την κατανεμημένη εκπαίδευση, η παραλληλοποίηση των δεδομένων (data parallelism) και η παραλληλοποίηση του μοντέλου (model parallelism).

##### Synchronous Data parallelism

Στην σύγχρονη εποχή της βαθιάς μάθησης, τα σύνολα δεδομένων είναι υπερβολικά ογκώδη ώστε να χωρέσουν στην μνήμη του επεξεργαστή. Επομένως, η μόνη λύση θα

ήταν να χρησιμοποιήσουμε την τεχνική Stochastic Gradient Descent με μικρό batch size. Το αποτέλεσμα θα ήταν η εκτίμηση των παραγώγων κατά το forward propagation να μην ήταν ακριβής σε κάθε βήμα, με αποτέλεσμα να χρειάζονται πολλά περισσότερα epochs εκπαίδευσης για να αρχίσει το μοντέλο να συγκλίνει σε ορθές προβλέψεις. Ως συνέπεια, η εκπαίδευση του μοντέλου θα έπαιρνε πολύ περισσότερο χρόνο.

Η τεχνική του data parallelism έρχεται να λύσει αυτό το πρόβλημα. Στην τεχνική αυτή, κατά το στάδιο του forward propagation της εκπαίδευσης, τα βάρη του μοντέλου αντιγράφονται σε κάθε κόμβο. Κάθε κόμβος, δηλαδή, λαμβάνει ένα αντίγραφο του τρέχοντος μοντέλου. Κάθε κόμβος λαμβάνει, επίσης, ένα διαφορετικό τμήμα του τρέχοντος batch, γεγονός που μας επιτρέπει να διαλέγουμε batch size πολλαπλάσιο, του αντίστοιχου batch size στην μη κατανεμημένη εκπαίδευση, κατά το πλήθος των κόμβων του cluster. Οι παράγωγοι που υπολογίζονται από το forward propagation σε κάθε κόμβο, συγκεντρώνονται στην συνέχεια σύγχρονα (synchronous data parallelism) στον κύριο κόμβο (master), όπου και υπολογίζονται οι αλλαγές που πρέπει να γίνουν στα βάρη του μοντέλου, κατά το βήμα του backward propagation. Τα νέα βάρη που υπολογίζονται, θα αντιγραφούν σε όλους τους κόμβους για να ξεκινήσει ο επόμενος κύκλος εκπαίδευσης κοκ. Μία σημαντική απαίτηση του synchronous data parallelism είναι η ύπαρξη ενός all-reduce αλγορίθμου, που συλλέγει όλες τις υπολογισμένες παραγώγους από τα forward propagation βήματα όλων των κόμβων. Η τεχνική του synchronous data parallelism, είναι η τεχνική που χρησιμοποιεί το framework που μελετάμε στην παρούσα εργασία, BigDL.

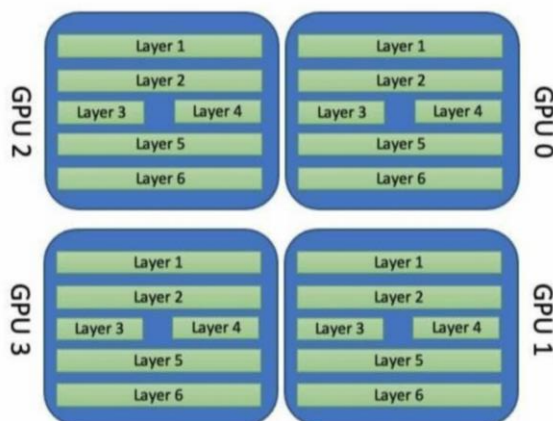


Fig. 17 Απεικόνιση Data Parallelism

##### Synchronous Data Parallelism on BigDL

Το BigDL υλοποιεί την τεχνική αυτή με την επαναληπτική διαδικασία του εξής αλγορίθμου.

Σε κάθε επανάληψη, τρέχουν κάποια Spark Jobs και υπολογίζουν τις τοπικές παραγώγους σε κάθε κόμβο, χρησιμοποιώντας το τρέχον mini-batch. Στην συνέχεια γίνεται μία σύνοψη των παραγώγων αυτών στον κύριο κόμβο, και ενημερώνονται τα βάρη του μοντέλου. Τα δεδομένα εκπαίδευσης και το μοντέλο αναπαριστώνται στο BigDL ως ένα Spark RDD που μοιράζεται στο Spark cluster [18].



Algorithm 1 Data-parallel training in BigDL

```
1: for  $i = 1$  to  $M$  do
2:   // "model forward-backward" job
3:   for each task in the Spark job do
4:     read the latest weights;
5:     get a random batch of data from local Sample partition;
6:     compute local gradients (forward-backward on local model
       replica);
7:   end for
8:   // "parameter synchronization" job
9:   aggregate (sum) all the gradients;
10:  update the weights per specified optimization method;
11: end for
```

Fig. 18 Ο αλγόριθμος για Data Parallelism όπως εφαρμόζεται στο framework BigDL[18]

### Asynchronous Data Parallelism

Ένα σημαντικό μειονέκτημα της σύγχρονης υλοποίησης της τεχνικής του Data Parallelism είναι η πιθανότητα οι κόμβοι να μένουν ανενεργοί για σημαντικό χρονικό διάστημα μετά την ολοκλήρωση του forward propagation βήματός τους, περιμένοντας να ολοκληρωθεί το αντίστοιχο βήμα και στους υπόλοιπους κόμβους, προκειμένου ο master κόμβος να συγκεντρώσει όλες τις παραγώγους και να αρχίσει την διαδικασία του Gradient Descent. Στην ασύγχρονη εκπαίδευση, η αναμονή αυτή μπορεί να παραλείπεται, καθιστώντας τον χρόνο αναμονής κάθε κόμβου ανεξάρτητο από τις υπολογιστικές δυνατότητες των υπόλοιπων κόμβων, και βοηθώντας έτσι στο scaling του συστήματος.

### Model Parallelism

Η δεύτερη κύρια τεχνική κατανεμημένης βαθιάς μάθησης που αναφέραμε, είναι η τεχνική του Model Parallelism. Η τεχνική αυτή αντιμετωπίζει ένα ιδιαίτερα σημαντικό πρόβλημα της τεχνικής του Data Parallelism στην σύγχρονη εποχή. Το πρόβλημα αυτό σχετίζεται με το αυξανόμενο μέγεθος των μοντέλων. Πιο συγκεκριμένα, πλέον υπάρχουν μοντέλα βαθιάς μηχανικής μάθησης που διαθέτουν εκατοντάδες εκατομμύρια εκπαιδευσιμες παραμέτρους (πχ το μοντέλο BERT με 110 εκατομμύρια παραμέτρους). Το μέγεθος αυτό καθιστά πολύ απαιτητική σε μνήμη την εκπαίδευση σε έναν μεμονωμένο κόμβο, αφού θα πρέπει ολόκληρο το μοντέλο να φορτωθεί στην μνήμη του επεξεργαστή του κόμβου. Η τεχνική του Model Parallelism λύνει το πρόβλημα αυτό, "μοιράζοντας" τις παραμέτρους του μοντέλου στους κόμβους του cluster. Όπως φαίνεται στην ανωτέρω εικόνα, διαδοχικά στρώματα του μοντέλου μοιράζονται στον ίδιο κόμβο, με αποτέλεσμα οι απαιτήσεις του κόμβου σε μνήμη να μειώνονται. Τα στρώματα του κάθε κόμβου εκπαιδεύονται, μεταφέροντας τις εξόδους τους στον επόμενο κόμβο κατά το βήμα του forward propagation. Ομοίως, κατά το βήμα του backward propagation, αναμένουν τις μεταβολές των βαρών που πρέπει να εφαρμόσουν, από τον κόμβο που διαθέτει το επόμενο στην σειρά στρώμα τους [19].

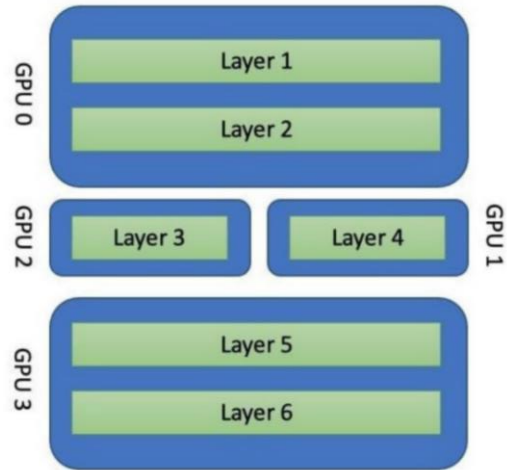


Fig. 19 Απεικόνιση Model Parallelism

### F. Πειραματική Διαδικασία

Η πειραματική διαδικασία στηρίχθηκε στην συγγραφή παραμετροποιήσιμων και επαναχρησιμοποιήσιμων τμημάτων κώδικα στα Python Notebooks που προσφέρει η πλατφόρμα Databricks.

Συντάξαμε μικρού μεγέθους συναρτήσεις, οι οποίες δέχονταν ένα πλήθος ορισμάτων - υπερπαραμέτρων, και, καλούμενες επαναληπτικά, αναλάμβαναν την διεκπαιρέωση ολόκληρου του κύκλου ζωής κάθε πειράματος, ξεκινώντας από την αρχικοποίηση το περιβάλλοντος με την δέσμευση των κατάλληλων πόρων, και καταλήγοντας στην οπτικοποίηση των αποτελεσμάτων υπό την μορφή πινάκων και διαγραμμάτων. Χαρακτηριστικό παράδειγμα των ορισμάτων που δέχονταν αυτές οι συναρτήσεις είναι το παράθυρο των διαθέσιμων πυρήνων και κόμβων, το μοντέλο προς εκπαίδευση, τα training και test sets, ο αριθμός των epochs της εκπαίδευσης, κλπ.

Βασίζόμενοι, λοιπόν, σε αυτές τις συναρτήσεις, δημιουργήσαμε μία πληθώρα από Python Notebooks στο Jupyter, σε κάθε ένα από τα οποία μελετούσαμε την επίδραση του διαθέσιμου αριθμού των κόμβων/πυρήνων στην διαδικασία του fine-tuning ενός συγκεκριμένου μοντέλου σε ένα συγκεκριμένο dataset.

Σημαντική αναφορά πρέπει να γίνει στους χρόνους εκπαίδευσης που εκτείνονται από μερικά λεπτά έως και πολλές ώρες ανάλογα το πλήθος των κόμβων, το σύνολο δεδομένων, τον αριθμό των εποχών εκπαίδευσης καθώς και το μοντέλο που χρησιμοποιούμε για την εκπαίδευση. Τα παραπάνω αποτέλεσαν τις βασικές υπερπαραμέτρους της πειραματικής διαδικασίας που ακολουθήσαμε.

### G. Github repository

Ο πηγαίος κώδικας των Notebooks αυτών, μαζί με τα αποτελέσματα και τις οπτικοποιήσεις, μπορεί να βρεθεί στο github repository του project (<https://github.com/John-Atha/distributed-deep-learning-NTUA-2022>).



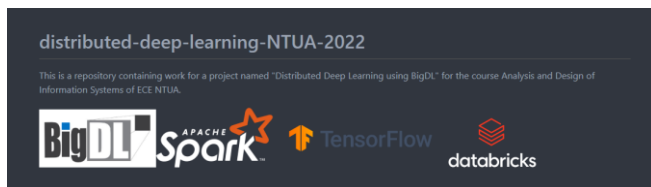


Fig. 20 To github repository για το project

## V. ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Κατά τη διάρκεια της πειραματικής διαδικασίας είχαμε συνεχώς νέα αποτελέσματα ανάλογο τις μεθόδους και το Spark mode που χρησιμοποιούσαμε που μας οδηγούσαν σε νέες δοκιμές και προσπάθειες. Αποφασίσαμε να διατυπώσουμε τα συμπεράσματα μας με μία χρονική αναδρομή, ξεκινώντας από τις πρώτες προσπάθειες μας και καταλήγοντας στις πιο ολοκληρωμένες τελικές μας δοκιμές. Αρχικά ελέγξαμε τη συμπεριφορά της εκπαίδευσης των διάφορων μοντέλων στο local mode του Spark, μέσω του Google Collab, εστιάζοντας στις διαφορές που έγκεινται στον αριθμό των πυρήνων. Έπειτα, μεταφέραμε το πρόβλημα σε έναν Spark cluster (cluster mode) εστιάζοντας κυρίως στις διαφορές που επιφέρουν οι κόμβοι στην εκπαίδευση μας, αλλά όχι μόνο.

### A. Local mode (Google Collab)

#### a) Δοκιμές

Στην περίπτωση του local mode στηριχτήκαμε κυρίως στην επίδραση του αριθμού των cores στην εκπαίδευση διαφόρων μοντέλων με διάφορα datasets.

Αρχικά, πειραματιστήκαμε στο πρόβλημα του Image Classification με την εκπαίδευση του μοντέλου Lenet στο σύνολο δεδομένων MNIST. Δοκιμάσαμε εκπαυδύσεις για 1 και 10 εποχές και αριθμό πυρήνων από 1 έως 4.

Έπειτα, πειραματιστήκαμε στο πρόβλημα του Text Classification, χρησιμοποιώντας ένα απλό Sequential μοντέλο με embeddings στο σύνολο δεδομένων IMDB. Και πάλι, δοκιμάσαμε εκπαυδύσεις για 1 και 10 εποχές και αριθμό πυρήνων από 1 έως 4.

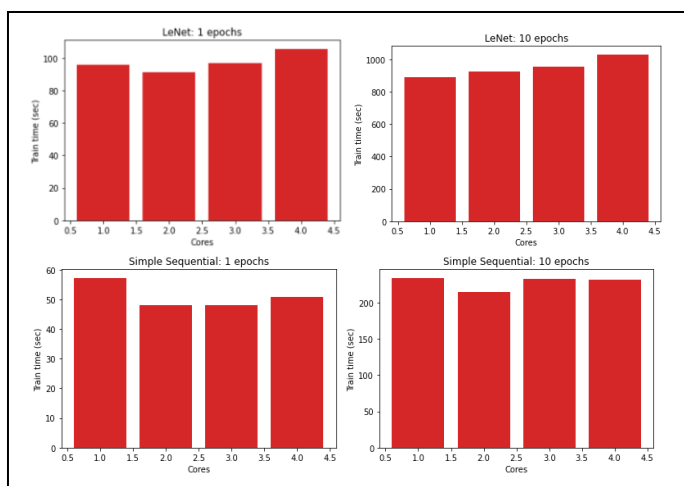


Fig. 21 Ραβδογράμματα για τους χρόνους εκτέλεσης, τις εποχές και τον αριθμό των πυρήνων για το LeNet και το απλό Sequential μοντέλο με embeddings

Ακόμη εκπαιδύσαμε και κάποια μοντέλα που χρησιμοποιήσαμε αργότερα στο cluster mode ώστε να έχουμε ένα μέτρο σύγκρισης.

Αυτά ήταν, πρώτον, το δικό μας CNN και δεύτερον το MobileNetV2 για το MNIST dataset.

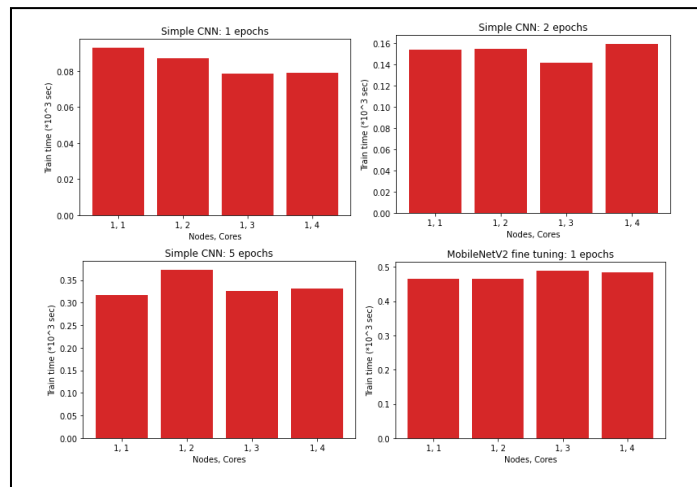


Fig. 22 Ραβδογράμματα για τους χρόνους εκτέλεσης ανάλογο το μοντέλο, τις εποχές και τον αριθμό των πυρήνων για το δικό μας CNN και το MobileNetV2

Αριθμός πυρήνων	Χρόνος εκπαίδευσης (sec)							
	Lenet with MNIST		Simple Sequential with IMDB		Το δικό μας CNN with MNIST			MobileNetV2 with MNIST
	1 epoch	10 epochs	1 epoch	10 epochs	1 epoch	2 epochs	5 epochs	1 epoch
1	96	891*	57	234	93	154	316*	464*
2	91*	925	48*	214*	87	155	373	465
3	97	956	48*	232	79*	142*	326	490
4	106	1031	51	231	79*	160	331	484

Fig. 23 Συγκεντρωτικά αποτελέσματα από τις εκπαιδεύσεις των μοντέλων σε local mode. Με \* σημειώνονται οι καλύτεροι χρόνοι ανά μοντέλο όπου βλέπουμε πως δεν υπάρχει κάποιο σταθερό μοτίβο σχετικά με τον αριθμό των πυρήνων

### B. Cluster mode (Spark cluster on Databricks)

Μεταβαίνοντας σε cluster mode είχαμε πλέον από 1 έως 3 κόμβους για να πειραματιστούμε με τα διάφορα datasets και μοντέλα. Στο cluster mode πλέον μας ενδιαφέρει ο συνολικός χρόνος εκπαίδευσης συναρτήσει των κόμβων. Κρατήσαμε σταθερό το πλήθος των πυρήνων και ίσο με 4.

Αρχικά, σκεφτήκαμε να δοκιμάσουμε κάποιο απλό Dataset και μοντέλο, ώστε να μπορέσουμε να βγάλουμε κάποια πρώτα συμπεράσματα σε σχετικά μικρό χρόνο, αφού θα είχαμε λίγες παραμέτρους και δεδομένα για την εκπαίδευση. Έτσι εκκινήσαμε από ένα απλό συνελκτικό δίκτυο (Το δικό μας CNN, σελ.5) με το MNIST dataset.

### Πρώτες εκπαιδεύσεις MNIST

Συγκεκριμένα, εκπαιδεύσαμε το δικό μας CNN στο MNIST, αυξάνοντας παράλληλα και τον αριθμό των εποχών(1, 2 και 5 εποχές), με σκοπό να δούμε αν υπάρχουν διαφορές για μεγαλύτερες χρονικά εκπαιδεύσεις. Τα αποτελέσματα που είχαμε, δεν ήταν αυτά που αναμέναμε, καθώς δεν είδαμε κάποια μείωση χρόνου έπειτα από την κατανομή της εκπαίδευσης σε περισσότερους του ενός κόμβους.

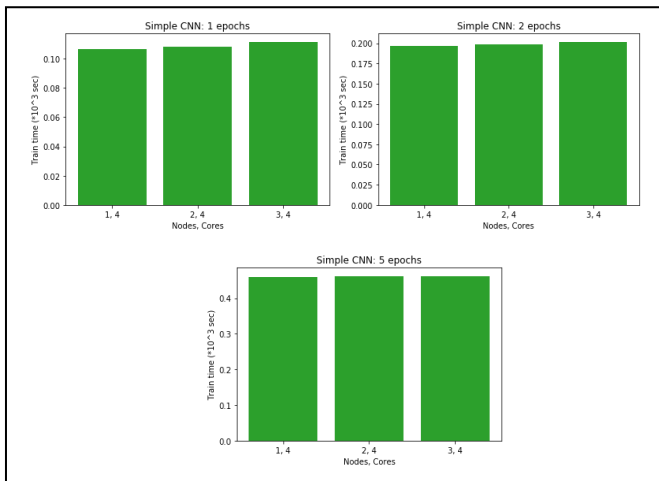


Fig. 24 Αποτελέσματα σε μορφή ραβδογραμμάτων για 1,2 και 3 κόμβους και για 1,2 και 5 εποχές. Βλέπουμε σταθερούς και ελαφρά αυξανόμενους χρόνους με την αύξηση των κόμβων

Έτσι, σκεφτήκαμε πως ίσως οι διαδικασίες του Spark υπερτερούσαν χρονικά σε σχέση με αυτές της εκπαίδευσης (ενημέρωση βαρών, back propagation), δηλαδή είχαμε αυτό που ονομάζουμε spark overhead. Έτσι, αποφασίσαμε να χρησιμοποιήσουμε κάποια μοντέλα με μεγαλύτερο αριθμό υπερπαραμέτρων, των οποίων οι διαδικασίες για την

εκπαίδευση, σίγουρα θα υπερτερούσαν χρονικά αυτών του Spark.

Επομένως, χρησιμοποιήσαμε στη συνέχεια τα μοντέλα MobileNetV2 και ResNet50 με το ίδιο dataset.

Αξίζει να σημειωθεί πως για την εκπαίδευση αυτών ξεκινήσαμε από τα προεκπαιδευμένα βάρη πάνω στο σύνολο δεδομένων 'Imagenet' και στη συνέχεια χρησιμοποιήσαμε τη μέθοδο Fine Tuning επαναεκπαιδεύοντας ένα ποσοστό των τελευταίων layers.

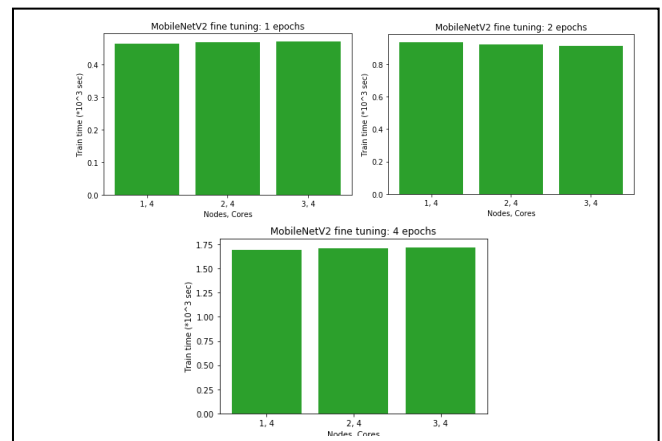


Fig. 25 Αποτελέσματα για το μοντέλο MobileNetV2 για 1,2 και 4 εποχές

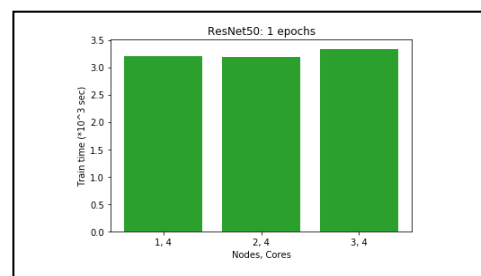


Fig. 26 Αποτελέσματα για το ResNet50 για 1 εποχή

Για τα παραπάνω χρειάστηκαν μεγαλύτεροι χρόνοι εκπαίδευσης ανά εποχή και χαρακτηριστικά για το ResNet50 απαιτήθηκε μία ώρα περίπου για 1 εποχή.

Ωστόσο, έπειτα από αυτά τα πειράματα, τα αποτελέσματα και πάλι δεν έδειξαν μείωση των χρόνων εκπαίδευσης. Συνεπώς, σκεφτήκαμε τη δοκιμή και κάποιου άλλου μεγαλύτερου συνόλου δεδομένων. Αυτό ήταν το Patch Camelyon (σελ.4).

## Πρώτες εκπαιδεύσεις με Patch Camelyon

Επαναλάβουμε τις δοκιμές για το Patch Camelyon στο δικό μας CNN και στο MobileNetV2 αλλά παρατηρήσαμε πως στα αποτελέσματά μας, ενώ είχαμε συνολικά αύξηση του χρόνου λόγω του πολύ μεγαλύτερου συνόλου δεδομένων, δεν είδαμε μείωση του χρόνου με αύξηση των κόμβων και πάλι.

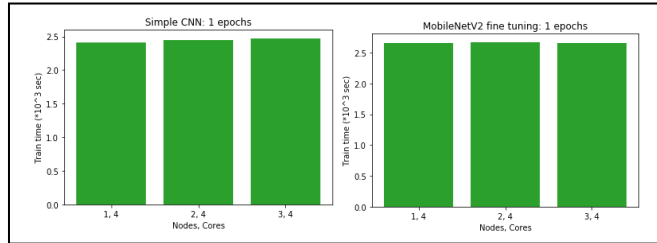


Fig. 27 Αποτελέσματα για την εκπαίδευση του δικού μας CNN και του MobileNetV2 για 1 εποχή στο Patch Camelyon. Παρατηρούμε πως δεν υπάρχει μείωση του χρόνου με την αύξηση των κόμβων.

## Αποτελέσματα με Mini-batch size SGD

Παρατηρώντας ότι στα έως τώρα πειράματα, ο χρόνος εκπαίδευσης των μοντέλων δεν εμφανίζει κάποιο μοτίβο ως προς τον αριθμό των πυρήνων και των κόμβων, αναζητήσαμε την λύση στις υπόλοιπες παραμέτρους της εκπαίδευσης. Μετά από σχετική έρευνα, καταλήξαμε στην μέθοδο της αύξησης του batch size αναλογικά με τον αριθμό των κόμβων, την οποία έχουμε ήδη περιγράψει στο μέρος (C).

Μετά από αυτήν την παραμετροποίηση, παρατηρήσαμε κατευθείαν σημαντική αλλαγή στους χρόνους εκπαίδευσης. Οι χρόνοι αυτοί πλέον, ήταν μικρότεροι συγκριτικά με προηγούμενα πειράματα, και άρχισαν να μειώνονται συναρτήσει του αριθμού των κόμβων. Τα μοτίβα αυτά φαίνονται στους πίνακες και τα διαγράμματα που ακολουθούν.

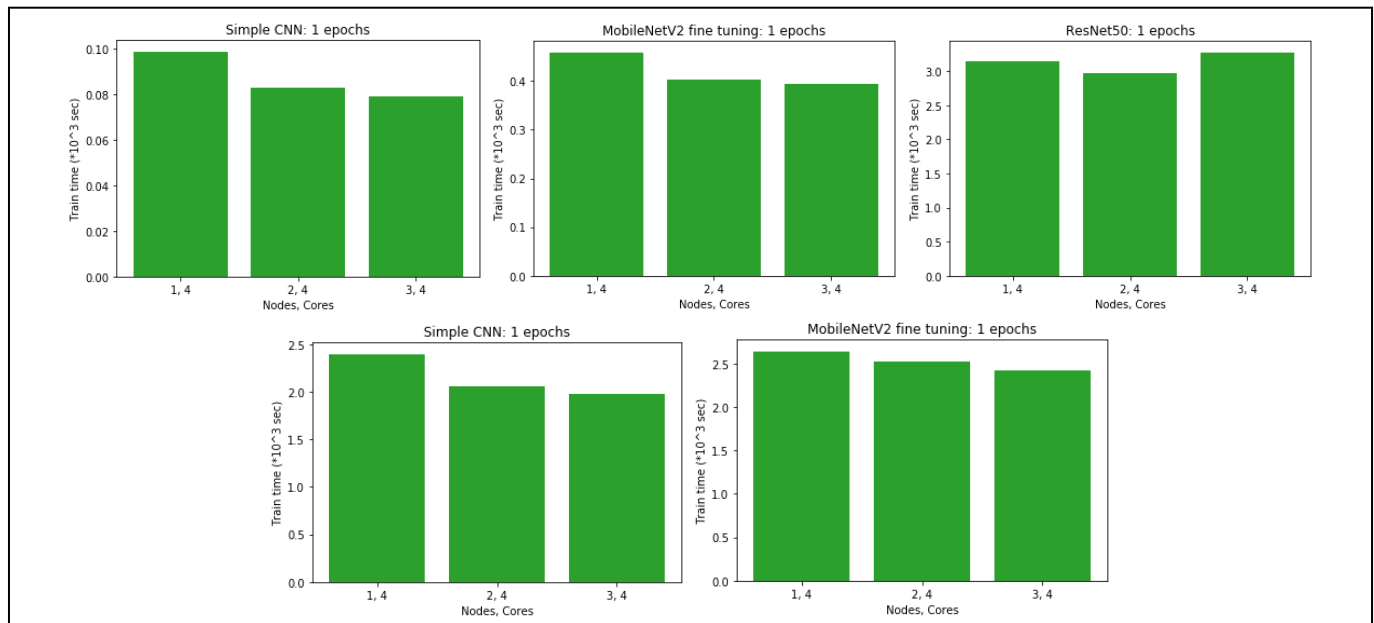


Fig. 28 Αποτελέσματα με γραμμική αύξηση του batch size για 1 epoch, πάνω για το MNIST και κάτω για το Patch Camelyon

Αριθμός κόμβων	Χρόνος εκπαίδευσης (sec)				
	<i>To δικό μας CNN with MNIST</i>	<i>MobileNetV2 with MNIST</i>	<i>ResNet50 with MNIST</i>	<i>To δικό μας CNN with Patch Camelyon</i>	<i>MobileNetV2 with Patch Camelyon</i>
	<i>1 epoch</i>	<i>1 epoch</i>	<i>1 epoch</i>	<i>1 epoch</i>	<i>1 epoch</i>
1	99	459	3142	2398	2640
2	83	403	2972*	2060	2517
3	79*	393*	3274	1982*	2424*

Fig. 29 Συγκεντρωτικά αποτελέσματα για cluster mode με χρήση mini-batch size SGD



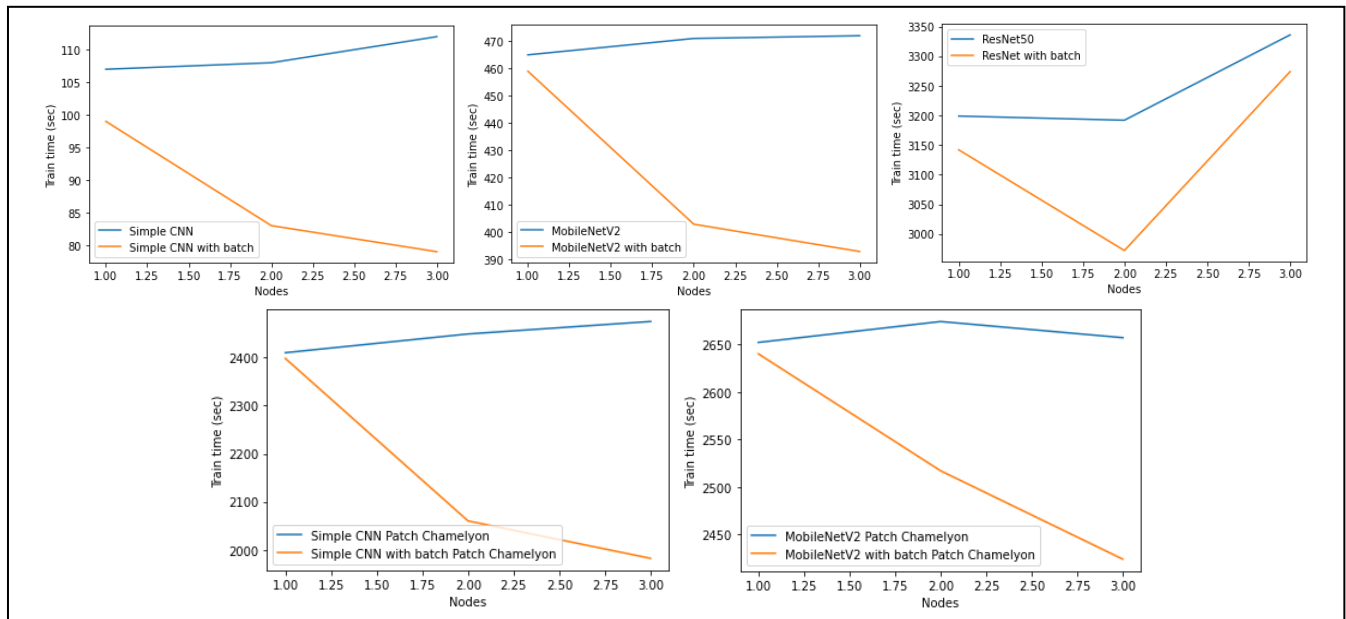


Fig. 30 Διαγράμματα στα οποία απεικονίζουμε την επίδραση της γραμμικής αύξησης του batch size στο χρόνο της εκπαίδευσης

### C. Συμπεράσματα

Η έρευνά μας στο πλαίσιο αυτής της εργασίας εστίασε στα προτερήματα και στα μειονεκτήματα της κατανομής βάθειας μάθησης με χρήση του framework BigDL. Δημιουργώντας ένα Spark cluster στην πλατφόρμα Databricks εκπαιδεύσαμε καταναμημένα ποικίλα μοντέλα πάνω σε διαφορετικά datasets.

Εξερευνήσαμε διαφορετικούς συνδυασμούς πυρήνων, κόμβων, εποχών εκπαίδευσης και batch sizes και μελετήσαμε την επίδρασή τους στο χρόνο εκπαίδευσης των μοντέλων.

Από τα πειράματα σε local mode, δηλαδή σε ένα μόνο μηχάνημα συμπεράναμε πως ο αριθμός των πυρήνων δεν διαδραματίζει καθοριστικό ρόλο στο χρόνο εκπαίδευσης,

τουλάχιστον σε πειράματα της κλίμακας που πραγματοποιήσαμε.

Όσον αφορά τα πειράματα όπου αξιοποιήσαμε πολλούς κόμβους του cluster, συμπεράναμε ότι η παραλληλοποίηση της εκπαίδευσης καθ' αυτή δεν βελτιώνει το χρόνο εκπαίδευσης, παρά μόνο αν συνδυαστεί με γραμμική αύξηση του batch size. Σε αυτήν την περίπτωση, όπως φαίνεται και στα διαγράμματα του Fig. 30 οδηγούμαστε κατά κανόνα σε σημαντική μείωση του χρόνου εκπαίδευσης.

Μελλοντικά, θα ήταν ενδιαφέρον να μία παρόμοια μελέτη σε clusters με ακόμα περισσότερους κόμβους μεγαλύτερης υπολογιστικής δύναμης και ο πειραματισμός με συνδυασμούς μεγαλύτερων μοντέλων και datasets.

### REFERENCES

- [1] BigDL, <https://bigdl.readthedocs.io/en/latest>
- [2] Apache Spark, <https://www.databricks.com/spark/about>
- [3] Yuen, D. (2017, September 17). Retrieved from SlideShare: <https://www.slideshare.net/DesmondYuen/very-large-scale-distributed-deep-learning-on-bigdl>
- [4] Orca: <https://pypi.org/project/orca>
- [5] Bigdl-orca: <https://pypi.org/project/bigdl-orca>
- [6] Tensorflow: <https://www.tensorflow.org>
- [7] Tensorflow-datasets: <https://www.tensorflow.org/datasets>
- [8] Keras models: <https://keras.io/api/applications/>
- [9] Mnist: <https://www.tensorflow.org/datasets/catalog/mnist>
- [10] PatchCamelyon: [https://www.tensorflow.org/datasets/catalog/patch\\_camelyon](https://www.tensorflow.org/datasets/catalog/patch_camelyon)
- [11] MobileNet: <https://arxiv.org/abs/1704.04861>
- [12] MobileNetV2: <https://arxiv.org/abs/1801.04381>
- [13] Tensorflow-Datasets and Google-Storage: <https://www.tensorflow.org/datasets/gcs>
- [14] Google storage buckets configuration: <https://cloud.google.com/storage/docs/access-control/making-data-public>
- [15] <https://builtin.com/data-science/gradient-descent>
- [16] [https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)
- [17] <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>
- [18] BigDL: A Distributed Deep Learning Framework for Big Data (arxiv.org)
- [19] Data parallelism vs. model parallelism - How do they differ in distributed training? (analyticsindiamag.com)