ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Νευρωνικά Δίκτυα και Ευφυή Υπολογιστικά Συστήματα

9ο Εξαμηνο ΗΜΜΥ

Συνεργάτες: Γιαννιός Γεώργιος-Ταξιάρχης, Α.Μ.: 03116156
Μπέτζελος Γιώργος, Α.Μ.: 03117442
Μπέτζελος Χρήστος, Α.Μ.: 03116067

Άσκηση 3: Βαθιά Μάθηση

Στόχος

Στόχος της Άσκησης είναι η βελτιστοποίηση της απόδοσης μοντέλων Βαθιάς Μάθησης στο σύνολο δεδομένων CIFAR-100 χρησιμοποιώντας την βιβλιοθήκη TensorFlow 2. Η ανάλυση ξεκίνησε ορίζοντας διάφορα μοντέλα, είτε from-scratch είτε χρησιμποιώντας transfer learning και προχώρησε με την βελτιστοποίηση του καλύτερου μοντέλου, ως προς το accuracy.

Περιβάλλον Εργασίας

Το περιβάλλον στο οποίο εργαστήκαμε είναι το google colab. Για την επιτάχυνση των αποτελεσματων έγινε χρήση GPU. Όλες οι μετρικές που αναφέρουμε (χρόνος, απαιτήσεις σε μνήμη κλπ) αναφέρονται στο περιβάλλον αυτό.

Σύνολο Δεδομένων

Το συνολο δεδομένων με το οποίο εργαστήκαμε είναι το το CIFAR-100. Το συγκεκριμένο σύνολο χωρίζεται σε 100 κλάσεις και κάθε κλάση διαθέτει 600 εικόνες. Οι εικόνες είναι μεγέθους 32 x 32 και είναι έγχρωμες. Το συγκεκριμένο Dataset, περιλαμβάνει εικόνες οπως ζώα, οχήματα, ανθρώπους κλπ.

Περιγραφή Αρχιτεκτονικής των Μοντέλων

Προκειμένου να πετύχουμε ένα υψηλο accuracy στην ταξινόμηση των εικόνων, ορίσαμε 3 Μοντέλα from-scratch και 10 χρησιμοποιώντας Transfer Learning. Τα μοντέλα που χρησιμοποιήσαμε στη 2η περίπτωση ήταν:

VGG-16: Πρόκειται για ένα βαθύ νευρωνικό δίκτυο μόλις 16 επιπέδων, που αποτελεί βελτίωση του AlexNet.

Xception: Περιέχει συνολικά 71 layers και δημιουργήθηκε ώστε να ταξινομεί επιτυχώς εικόνες του ImageNet .

EfficientNetB7: Το συγκεκριμένο μοντέλο δεν σχεδιάστηκε απο μηχανικούς, αλλά απο ένα νευρωνικό δίκτυο. Στόχος του ήταν η επίτευξη υψηλού accuracy και η ελαχιστοποίηση των υπολογισμών με αριθμούς κινητής υποδιαστολής. Πρόκειται για αρκετά "βαθύ" νευρωνικό καθώς τα layers ξεπερνούν τα 200.

ResNet: Όπως υποδηλώνει και το όνομα πρόκειται για ενα Residual Network. Στην ουσία ένα Layer μπορεί να συνδέεται όχι μόνο με το αμέσως επόμενο αλλά με κάποιο πιο απομακρυσμένο. Το σχήμα αυτό επαναλαμβάνεται περιοδικά κατά blocks επιπέδων. Στην άσκηση χρησιμοποιήθηκε και άλλη μια εκδοχή του: "ResNet152V2".

DenseNet121: Πρόκειται για ένα δίκτυο στο οποίο κάθε layer συνδέεται με όλα τα επόμενα. Στην άσκηση χρησιμοποιήθηκαν και άλλες δύο εκδοχές: "DenseNet169", "DenseNet201".

MobileNetV2: Στο συγκεκριμένο δίκτυο (53 επίπεδα) δεν γίνεται η κανονική συνέλιξη όπως σε άλλα (π.χ Inception) αλλά συνέλιξη κατά βάθος (depthwise convolution).

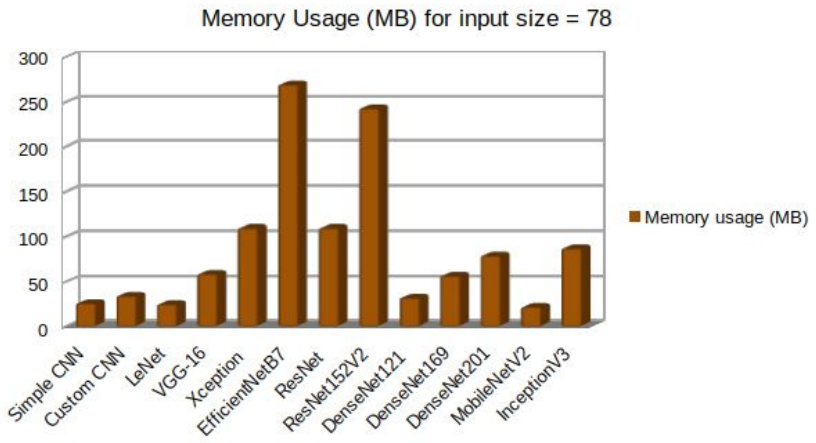
InceptionV3: Η βασική ιδέα που οδήγησε στην υλοποίηση του Inception, είναι ότι δεν αυξάνεται η απόδοση, κάνοντας stack ολοένα και περισσότερα layers (deeper neural networks). Ενώ στα περισσότερα μοντέλα, κάθε φίλτρο έχει ένα προκαθορισμένο μέγεθος (3x3 , 5x5 , κλπ), στο Inception σε ένα επίπεδο μπορεί να έχουμε πολλαπλά φίλτρα διαφόρων μεγεθών. Έτσι το δίκτυο απο "deeper" γίνεται "wider".

Να σημειωθεί ότι σε όλα τα παραπάνω μοντέλα, αντικαταστήσαμε την κεφαλή (τελευταίο layer), με τρία επίπεδα που τα ορίσαμε εμείς (Dropout, pooling, fully connected στο τέλος).

Σύγκριση των Μοντέλων

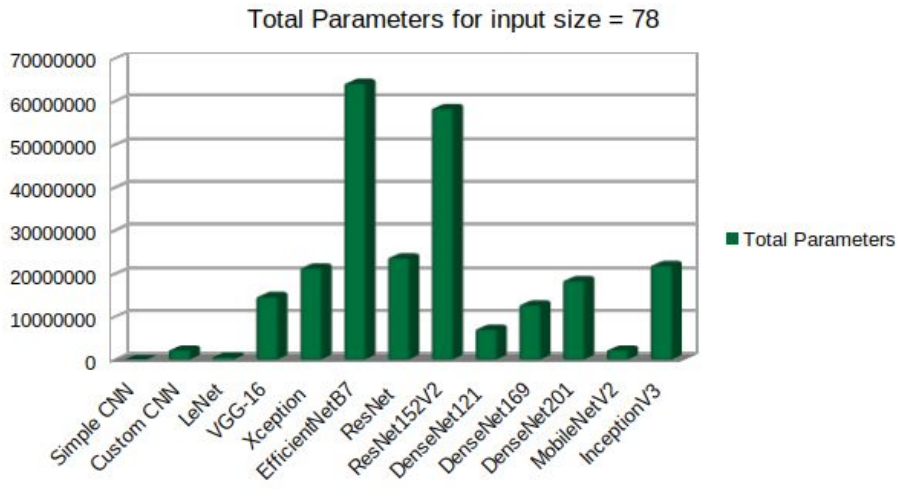
Προκειμένου να επιλέξουμε με ποιο Μοντέλο θα προχωρήσουμε στη βελτιστοποίηση, εξετάσαμε τα παραπάνω μοντέλα με βάση κάποιες μετρικές.

Α.Μνήμη



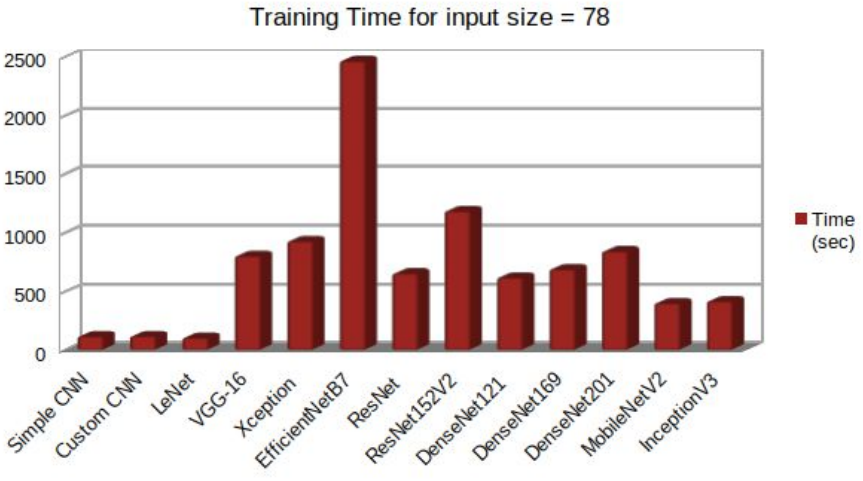
Χρησιμοποιώντας μια συνάρτηση, υπολογίζουμε τις απαιτήσεις ενός μοντέλου σε μνήμη. Ο υπολογισμός εξαρτάται προφανώς απο το batch size αλλά και απο τον συνολικό αριθμό παραμέτρων. Απο το παραπάνω διάγραμμα παρατηρούμε ότι το EfficientNetB7 έχει το μεγαλύτερο memory usage, ενώ το MobileNetV2 το μικροτερο. Αναμενόμενο αφού όπως σημειώθηκε και κατά την περιγραφή των επιμέρους αρχιτεκτονικών, το EfficientNetB7 έχει πάνω απο 200 επίπεδα ενώ το MobileNet μολις 53.

Β. Παράμετροι



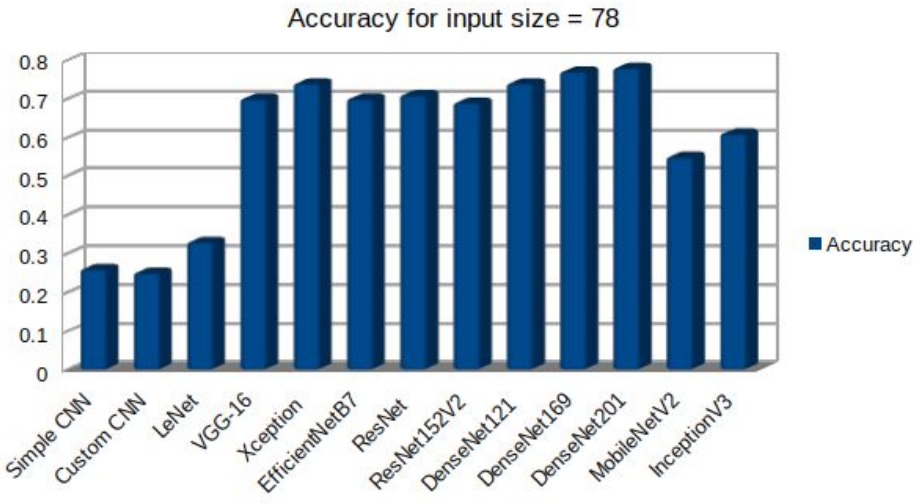
Απαριθμώντας τις παραμέτρους (trainable - non trainable) παρατηρεί κανείς την αντιστοιχία μεταξύ μνήμης (στο προηγούμενο διάγραμμα) και αριθμού παραμέτρων.

Γ. Χρόνος Εκπαίδευσης



Ως προς το χρόνο εκπαίδευσης, το EfficientNetB7 φαίνεται να καθυστερεί περισσότερο από όλα. Όλα τα υπόλοιπα μοντέλα (transfer learning) χρειάστηκαν περίπου τον ίδιο χρόνο, Και εδώ παρατηρείται μια σχέση μεταξύ του αριθμού των παραμέτρων και του χρόνου εκπαίδευσης.

Δ. Accuracy

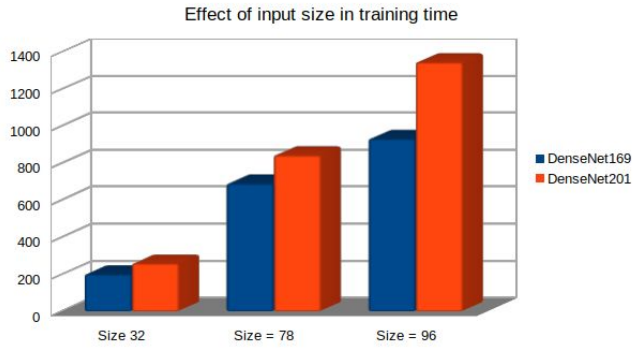


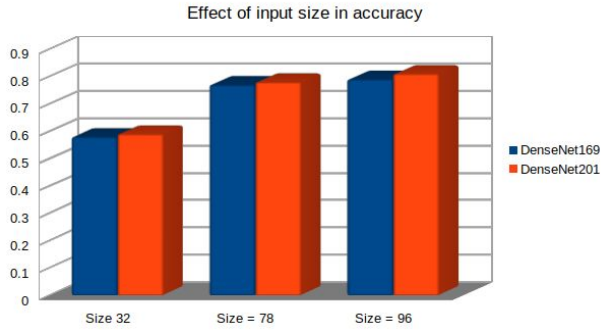
Τέλος το υψηλότερο accuracy πέτυχε το DenseNet201 ενώ ακολουθεί το DenseNet169. Εξετάζοντας και τα παραπάνω διαγράμματα, παρατηρούμε ότι τα συγκεκριμένα μοντέλα αποδίδουν αρκετά καλά, τόσο ως προς το χρόνο εκπαίδευσης, όσο και ως προς τη χρήση της μνήμης. Για τους λόγους αυτούς (υψηλό acc, χαμηλός χρόνος εκπαίδευσης - memory usage), συνιστούν μια καλή επιλογή (μεταξύ των άλλων) για βελτιστοποίηση.

Βελτιστοποίηση του Μοντέλου

Στο σημείο αυτό επιλέξαμε να βελτιστοποιήσουμε το DenseNet201 (*acc = 0.78*) και το DenseNet169 (*acc = 0.77*). Όλη η διαδικασία που περιγράφεται παρακάτω, αναφέρεται σε ***80 κλάσεις*** του CIFAR-100.

Α. Πειραματισμός με Μέγεθος Εικόνας Εισόδου

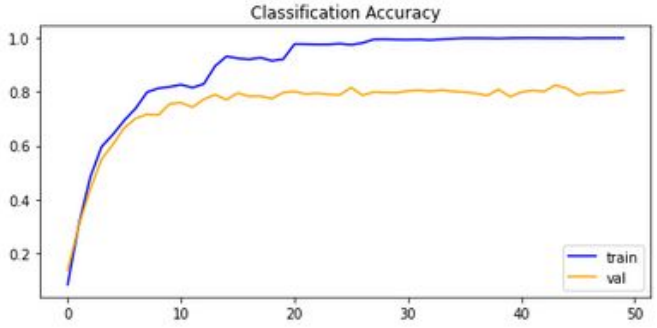




Ο πρώτος πειραματισμός, αφορούσε το μέγεθος[1] της εικόνας εισόδου. Γίνεται αντιληπτό απο τα παρακάτω διαγράμματα ότι για μεγαλύτερο μέγεθος εικόνας εισόδου, μπορεί να απαιτείται περισσότερος χρόνος εκπαίδευσης, ωστόσο όμως σημειώνεται υψηλότερο accuracy.

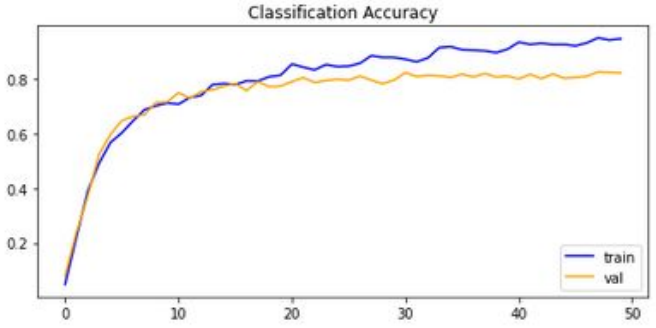
[1] Μέγεθος εικόνας = 128 εμφανίζει στο περιβάλλον colab το μήνυμα “Your session crashed after using all available RAM”

Αντιπαραβάλλοντας τα δυο υποψήφια μοντέλα, είναι προφανές ότι το DenseNet201 υπερτερεί έναντι του DenseNet169, οπότε η ανάλυση συνεχίζει με το πρώτο. Το μέγεθος εικόνας κλειδώνει στην τιμή 96, καθώς με τον τρόπο αυτό παρατηρείται το μεγαλύτερο accuracy. Να σημειωθεί ότι το accuracy που επιτύχαμε στην περίπτωση αυτή, ήταν *acc = 0. 81*. Η γραφική αναπαράσταση του *train_acc, val_acc* είναι η εξής:



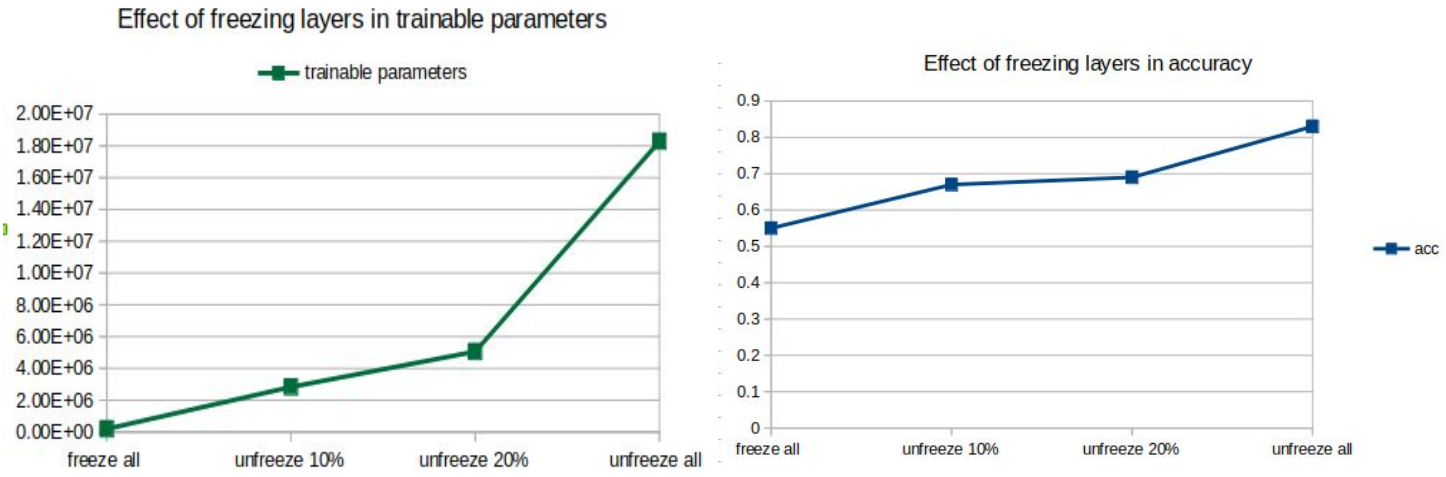
Β. Επαύξηση Δεδομένων

Όπως παρατηρούμε στο προηγούμενο διάγραμμα, η ακρίβεια της εκπαίδευσης αυξάνεται (μετά τις 10 εποχές) σχεδόν γραμμικά με το χρόνο, ενώ η ακρίβεια επικύρωσης καθυστερεί περίπου το 80% στη διαδικασία εκπαίδευσης. Όταν υπάρχει ένας μικρός αριθμός δειγμάτων εκπαίδευσης, το μοντέλο μερικές φορές μαθαίνει από θορύβους ή ανεπιθύμητες λεπτομέρειες από αυτά τα δείγματα, σε βαθμό που επηρεάζει αρνητικά την απόδοση του μοντέλου σε νέα παραδείγματα. Αυτό το φαινόμενο είναι γνωστό ως overfitting. Ένας τρόπος για να διορθώσουμε αυτό το πρόβλημα είναι να αυξήσουμε το σύνολο δεδομένων εκπαίδευσης, χρησιμοποιώντας τυχαίους μετασχηματισμούς (περιστροφές, μετατοπίσεις κ.τ.λ.) των αρχικών εικόνων. Να σημειωθεί ότι το accuracy που επιτύχαμε στην περίπτωση αυτή, ήταν *acc = 0. 83*. Πράγματι, το data augmentation συνέβαλε στη μείωση του κινδύνου overfit. Αυτό αποτυπώνεται και στην καμπύλη εκμάθησης, όπου το val_acc φαίνεται να προσεγγίζει καλύτερα το train_acc (σε σχέση με πριν).



Γ. Fine-Tuning

Ενώ στα προηγούμενα βήματα, εκπαιδεύαμε εξαρχής όλα τα βάρη του DenseNet201, στο σημείο αυτό δοκιμάσαμε να εκπαιδεύσουμε μόνο τα τελευταία, παγώνοντας έτσι τα βάρη των αρχικών layers. Παρατηρήσαμε πως όσο πιο πολλά επίπεδα εκπαιδεύαμε, τόσο πιο πολύ αυξάνεται το accuracy. Βέβαια αυξάνεται σημαντικά και ο χρόνος εκπαίδευσης



Απο τα παραπάνω διαγράμματα, είναι ξεκάθαρο ότι στην περίπτωση unfreeze all, επιτυγχάνεται το υψηλότερο accuracy, οπότε δεν χρειάζεται να παγώσουμε τα αρχικά layers. Αυτο θα συνέβαινε αν είχαμε στη διάθεσή μας ένα μικρότερο Dataset.

Δ. Πειραματισμός με Dropout

Στη συνέχεια εισάγουμε στο τέλος του δικτύο ένα *dropout layer* και πειραματιζόμαστε με την τιμή *dropout*, ώστε να ακυρωθεί τυχαία ένα ποσοστό των εισόδων του. Κάτι τέτοιο ήταν αναγκαίο αφού ορισμένα "μονοπάτια" στο δίκτυό μας δεν αντιπροσωπεύουν πάντα κάποια κλάση αλλά πιθανώς ανεπιθύμητο θόρυβο. Οι πειραματισμοί μας συνοψίζονται στον ακόλουθο πίνακα:

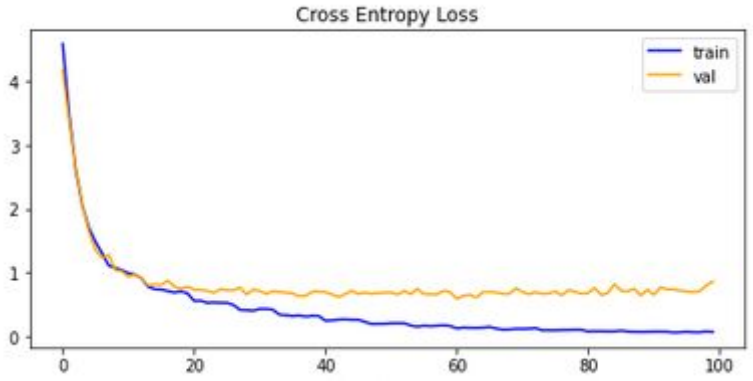
Dropout	Time	Accuracy
0.15	1565.7	0.82
0.25	1506.8	0.83
0.3	1357.63	0.82
0.5	974.81	0.83
0.7	1535.81	0.82

Απο τον παραπάνω πίνακα, φαίνεται ότι δεν μπορούμε να εξαγάγουμε άμεσα κάποιο συμπέρασμα ως προς το dropout. Αυτο οφείλεται στο γεγονός ότι κατά το dropout ακυρώνεται **τυχαία** ένα ποσοστό νευρώνων και όχι με βάση κάποιο κριτήριο. Επιλέγουμε ωστόσο να διατηρήσουμε την αρχική τιμή (*dropout = 0.5*) καθώς οι υπόλοιπες τιμές δεν βοηθούν αισθητά στην καταπολέμηση overfit (όπως αποτυπώνεται και στο test accuracy).

Ε. Πειραματισμός με Early Stopping

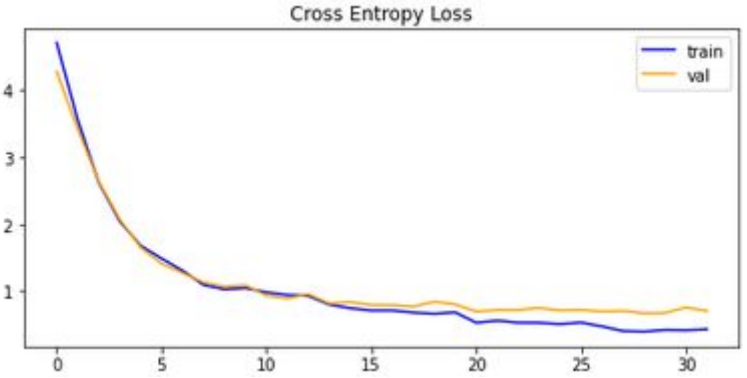
Απο τα διαγράμματα Cross Entropy Loss, μπορεί να ανιχνευτεί ο κίνδυνος overfit. Πιο ειδικά όταν η καμπύλη train loss μειώνεται συνεχώς, αλλά η val loss αυξάνεται ή μένει σταθερή σημαίνει ότι το μοντέλο μας δυσκολεύεται πλέον να γενικεύσει. Στη περίπτωση αυτή, καλό είναι να διακόπτεται η διαδικασία της εκπαίδευσης. Αυτό επιτυγχάνεται με τη χρήση της callback *Early Stopping*. Το μέγεθος που καταγράφουμε είναι το *vall loss* και ορίζουμε μια ανοχή (μέσω της παραμέτρου patience). Για παράδειγμα, patience = 10 σημαίνει ότι αν για 10 εποχές το *val loss* έχει σημειώσει μεγαλύτερη τιμή απο την καλύτερη (που έχει δει), καλό είναι να διακοπεί η εκπαίδευση:

patience = 10



παρατηρούμε στη περίπτωση αυτή, ότι θα πρέπει να μειώσουμε την ανοχή καθώς το *val loss* τείνει να απομακρυνεται αισθητά απο το *train loss*. Με μία μικρότερη ανοχή ωστόσο, υπάρχει ο κίνδυνος να διακοπεί η εκπαίδευση, ενώ δεν ελλοχεύει ο κίνδυνος overfit. Αυτό φαίνεται και στο ακόλουθο διάγραμμα:

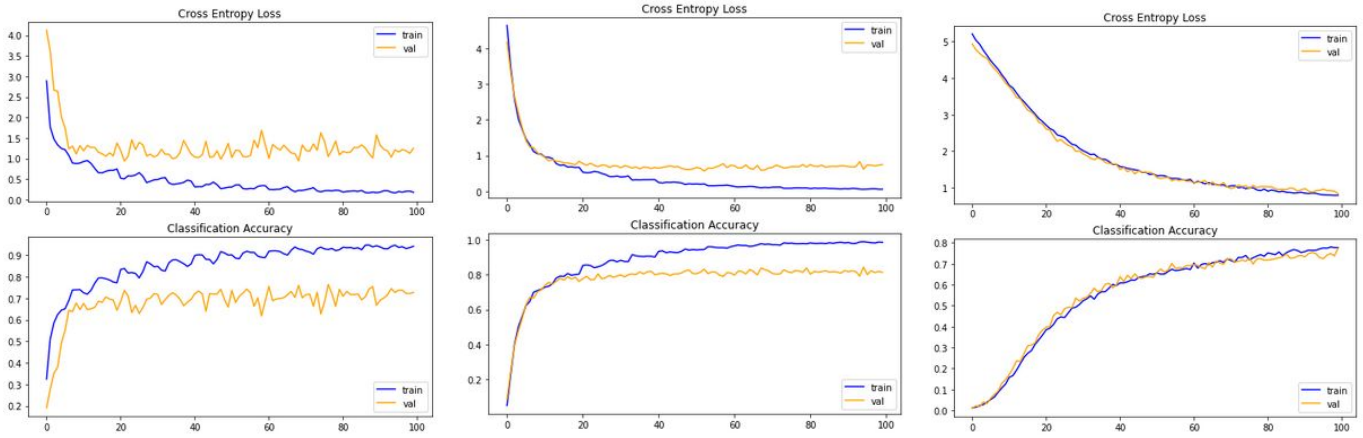
patience = 3



Η ιδανική τιμή που επιλέγουμε είναι patience = 7, ώστε να “προλάβει” να εκπαιδευτεί επαρκώς το μοντέλο μας και να μπορεί στη συνέχεια να γενικεύσει σε νέα δεδομένα.

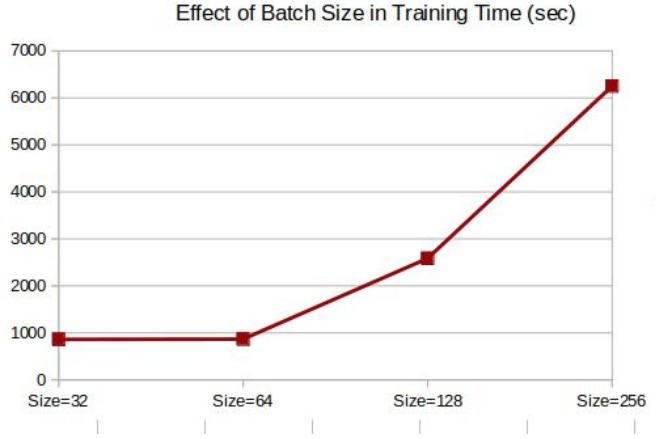
ΣΤ. Πειραματισμός με learning rate

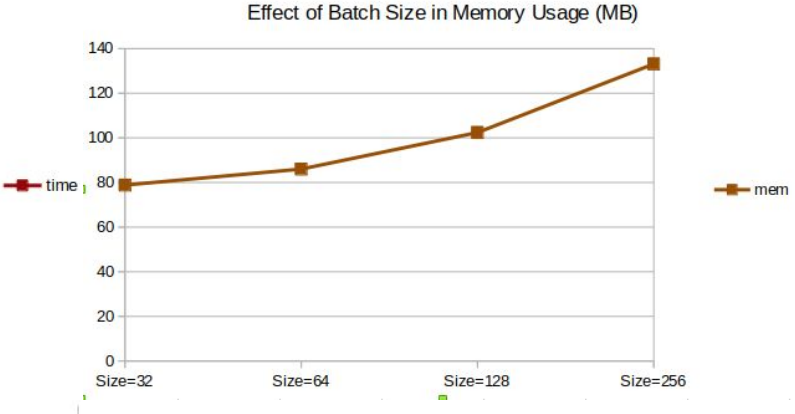
$lr = 5 * 10^{-4}$ $lr = 2.5 * 10^{-5}$ $lr = 5 * 10^{-6}$

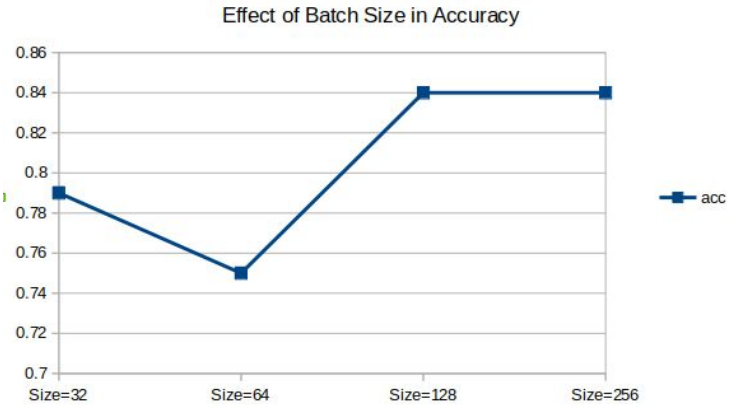


Ανάμεσα στις τιμες του learning rate που εξετάσαμε, ενδιαφέρον παρουσιάζουν οι τιμές : $5 * 10^{-4}$, $2.5 * 10^{-5}$, $5 * 10^{-6}$. Στην πρώτη περίπτωση βλέπουμε ότι για σχετικά μεγάλο learning rate, το νευρωνικό εκπαιδεύεται γρήγορα (ήδη απο τις 50 εποχές το training loss είναι κοντά στο 1). Ωστόσο για πολύ μικρό learning rate (*τρίτο διάγραμμα*), το νευρωνικό αργεί να εκπαιδευτεί (ακόμα και στις 100 εποχές, το training loss είναι στο 0.8). Η ιδανική τιμή που επιλέγουμε είναι $2.5 * 10^{-5}$, καθώς στην περίπτωση αυτή δεν παρατηρούνται ταλαντώσεις γύρω από βέλτιστες τιμές βαρών (όπως στη περίπτωση 1), αλλά ούτε υπάρχει ο κίνδυνος να παγιδευτούμε σε κάποιο τοπικό ακρότατο (όπως στην περίπτωση 3).

Ζ. Επίδραση Batch Size





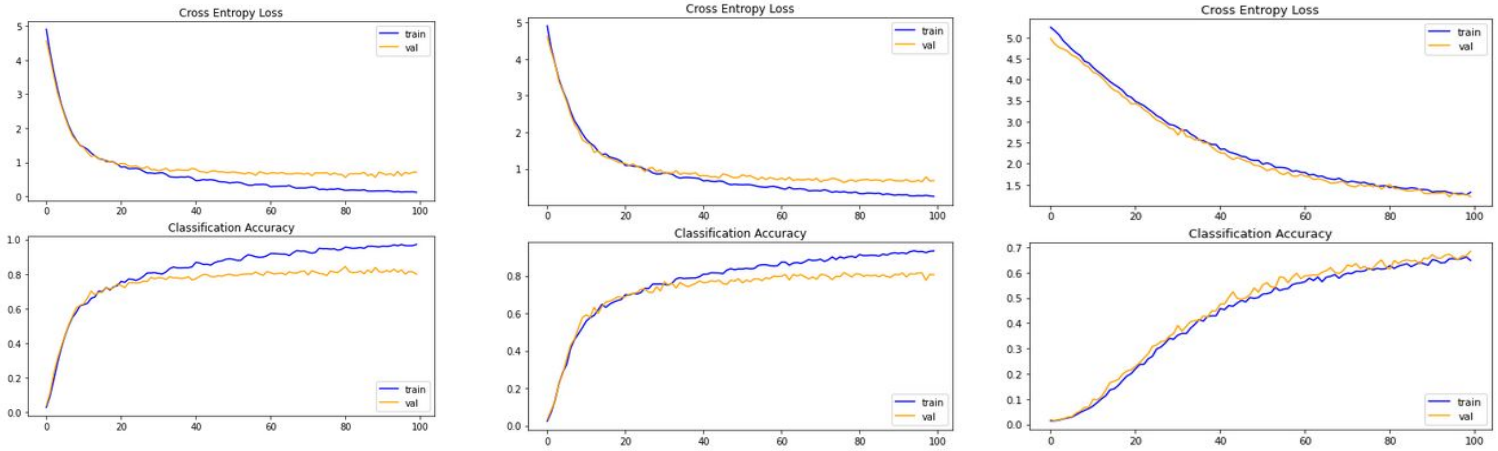


Οφείλουμε να παραδεχτούμε ότι όσο μικρότερο είναι το batch_size, τόσο μικρότερες είναι και οι απαιτήσεις σε μνήμη. Ταυτόχρονα με μικρά batch sizes, πετυχαίνουμε υψηλές ταχύτητες εκπαίδευσης. Απο την άλλη, μεγάλα batches, εγγυώνται και μεγαλύτερο accuracy, καθώς σε ένα iteration το νευρωνικό μας τροφοδοτείται με περισσότερα δεδομένα. Κάτι τέτοιο φαίνεται και απο το τελευταίο διάγραμμα, όπου με batch size ίσο με 128 ή 256, πετυχαίνουμε το υψηλότερο accuracy. Επειδή όμως με batch size = 256 , ο χρόνος εκπαίδευσης είναι μεγαλύτερος και οι απαιτήσεις σε μνήμη υψηλότερες, επιλέγουμε την τιμή 128.

Η. Επίδραση διαφόρων optimizers

Όσον αφορά τους optimizers, επιλέξαμε να μελετήσουμε τους Adam (*αριστερά*) , Adamax (*κέντρο*) και SGD (*δεξιά*):

<u>Adam</u> <u>Adamax</u> <u>SGD</u>



Θα τους συγκρίνουμε χρησιμοποιώντας δύο μετρικές. 1) *Ταχύτητα Εκπαίδευσης*, 2) *Ικανότητα Γενίκευσης*. Όπως παρατηρεί κανείς από την πρώτη στήλη των subplots, ο Adam πετυχαίνει σχεδόν το ίδιο val accuracy με τον *Adamax*. Η μόνη διαφορά που μπορούμε να παρατηρήσουμε είναι ότι στον Adamax η γραφική δεν παρουσιάζει απότομες αλλαγές και είναι πιο smooth. Απο την άλλη, παρατηρώντας την τρίτη στήλη των γραφικών, διαπιστώνουμε ότι ο *SGD* αργεί περισσότερο να συγκλίνει. Ωστόσο στη περίπτωση αυτή, φαίνεται να μην ελλοχεύει ο κίνδυνος overfit (η συνάρτηση val accuracy, σχεδόν ταυτίζεται με τη *val_train*). Οπότε επιλέγουμε σαν optimizer, τον *Adam*.

Τελικά Αποτελέσματα

Το τελικό μοντέλο που επιλέξαμε προέκυψε απο την παραμετροποίηση:

- model: *DenseNet 201*
- input size: *96*
- data_augmentation: *HorizontalFlip, RandomRotation, RandomZoom ,Random Contrast (10% σε όλα).*
- dropout: *0.5*
- Early_Stopping: *monitor = val_loss, patience = 7*
- learning_rate: *0.000025*
- batch_size: 128
- optimizer: *Adam*

Το accuracy που προέκυψε στο test set ήταν 0.84.

Πειραματισμός σε διαφορετικό αριθμό κλάσεων

Τέλος για το μοντέλο αυτό, εξετάστηκε και η απόδοση στις 20 κλάσεις (*acc = 0.92*) στις 40 (*acc = 0.88*) και στις 60 κλάσεις (*acc = 0.86*)

Σχόλια-Παρατηρήσεις

Με την παραπάνω ανάλυση, ξεκινώντας απο accuracy ισο με 0.78, καταφέραμε μια αύξηση κατά 0.6, με το τελικό accuracy να διαμορφώνεται στο 0.84 για τις 80 κλάσεις. Απο την άλλη παρατηρώντας και τις εικόνες του CIFAR - 100, διαπιστώνουμε ότι πολλές κατηγορίες έχουν αρκετά κοινά χαρακτηριστικά. Αυτό έχει ως πιθανό αποτέλεσμα, τα περισσότερα λάθη να αφορούν τις κατηγορίες αυτές. Για παράδειγμα υπάρχει κατηγορία “boy” και ξεχωριστή κατηγορία “man” (αντ. “girl”, “woman”), motorcycle-bicycle hamster - mouse κλπ. Πρόκειται λοιπόν για προβλήματα διαχωρισμού, για τα οποία κατασκευάζεται στοχευμένα και αποκλειστικά ένα μοντέλο και μελετώνται ξεχωριστά.

Παράρτημα

Προσθέτουμε για λόγους πληρότητας τους πίνακες με όλα τα αποτελέσματα που εξήχθησαν από τον κώδικα, και χρησιμοποιήθηκαν στα Διαγράμματα

1.Πειραματισμοί με διάφορες αρχιτεκτονικές - μοντέλα transfer learning:

	Memory usage (MB)	Total Parameters	Time (sec)	Accuracy
Simple CNN	25.6	128,420	116.47	0.26
Custom CNN	33.792	2,333,468	118.9	0.25
LeNet	24.576	723,180	103.86	0.33
VGG-16	58.368	14,765,988	803.34	0.70
Xception	109.568	21,419,916	930.64	0.74
EfficientNetB7	269.312	64,353,787	2467.44	0.70
ResNet	109.568	23,792,612	654.38	0.71
ResNet152V2	242.688	58,536,548	1186.09	0.69
DenseNet121	31.744	7,140,004	617.35	0.74
DenseNet169	56.32	12,809,380	690.92	0.77
DenseNet201	78.848	18,514,084	843.26	0.78
MobileNetV2	21.504	2,386,084	400.66	0.55
InceptionV3	87.04	22,007,684	416.18	0.61

2.Πειραματισμοί με input_size σε DenseNet169 - DenseNet201

memory

size =	32	78	96
DenseNet169	51.2	56.32	64.512
DenseNet201	73.728	78.848	88.064

parameters

size=	32	78	96
DenseNet169	12,809,380	12,809,380	12,809,380
DenseNet201	18,514,084	18,514,084	18,514,084

time

size =	32	78	96
DenseNet169	200.92	690.92	932.38
DenseNet201	259.08	843.26	1345.88

acc

size =	32	78	96
DenseNet169	0.58	0.77	0.79
DenseNet201	0.59	0.78	0.81

3. Fine Tuning

	mem	trainable parameters	time	acc
freeze all	102.4	192,100	983.34	0.55
unfreeze 10%	102.4	2,831,140	965.3	0.67
unfreeze 20%	102.4	5,056,740	982.75	0.69

4. Πειραματισμός με Dropout

dropout	time	acc
0.15	1565.7	0.82
0.25	1506.8	0.83
0.3	1357.63	0.82
0.5	974.81	0.83
0.7	1535.81	0.82

5. Πειραματισμός με Batch Size

Batch Size	epochs	mem	time	acc
32	71	78.848	860.83	0.79
64	37	86.016	865.93	0.75
128	100	102.4	2584.82	0.84
256	100	133.12	6251.78	0.84

6. Πειραματισμός με learning rate

learning rate	epochs	time	acc
$5 * 10^{-6}$	100	1764.08	0.76
10^{-5}	100	1756.85	0.79
$2.5 * 10^{-5}$	100	2584.82	0.84
$5 * 10^{-5}$	100	1761.64	0.82
10^{-4}	89	1574.21	0.81
$5 * 10^{-4}$	100	1768.82	0.72
$5 * 10^{-3}$	100	1767.84	0.35