****

**Νευρωνικά Δίκτυα και Ευφυή Υπολογιστικά Συστήματα**

Αναφορά 3ης Εργασίας

*Ομάδα 43*

Μέλος 1: Μέλος 2:

Ον/μο: Κορακοβούνης Δημήτριος Αναγνωστόπουλος Θεόδωρος

Α.Μ.: 03116692 03116066

*Μέρος 1 – Εισαγωγή μοντέλων*

Στην άσκηση αυτή σκοπός μας είναι να δοκιμάσουμε τεχνικές βελτιστοποίησης επάνω σε νευρωνικά δίκτυα τα οποία είτε ορίζουμε εμείς την αρχιτεκτονική τους είτε παίρνουμε εκπαιδευμένα δίκτυα χρησιμοποιώντας την τεχνική του transfer learning.

Για τον σκοπό αυτό δημιουργήσαμε 3 νευρωνικά δίκτυα (CNNs) αυξανόμενης πολυπλοκότητας τα οποία 2 εξ αυτών αποτελούνται από 3 στρώματα convolution και 2 dense connected στρώματα. Το τρίτο αποτελείται από 5 convolution layers και 4 dense connected layers ενώ περιλαμβάνονται στο ενδιάμεσο και στρώματα που εκτελούν Pooling, Normalization, και Dropouts.

Ακόμη, με την μέθοδο του transfer learning υλοποιήσαμε συνάρτηση η οποία δίνει την δυνατότητα μεταφοράς μάθησης των παρακάτω δικτύων:

* DenseNet 201
* Inception-ResNet V2
* Inception V3
* MobileNet v1
* MobileNet v2
* NASNet-A
* ResNet
* ResNet v2
* VGG16
* VGG19
* Xception

Ενώ για το υπόλοιπο της άσκησης (και για μείωση της υπολογιστικής πολυπλοκότητας) θα χρησιμοποιήσουμε τα δίκτυα:

* DenseNet 201
* Inception-ResNet V2
* Inception V3
* MobileNet v2
* ResNet v2
* VGG16
* VGG19
* Xception

Τα τελικά αποτελέσματα και γραφήματα εμφανίζονται στο αρχείο *final\_report.ipynb*

Τρέξαμε τα μοντέλα για παραμέτρους BATCH\_SIZE = 128, 512 για πλήθος κλάσεων ίσο με 20, και BATCH\_SIZE = 128 για πλήθος κλάσεων 80.

*Μέρος 2 – Εκπαίδευση χωρίς βελτιστοποίηση (num\_of\_classes=20)*

Για την εκπαίδευση των δικτύων υλοποιούμε την συνάρτηση *train\_and\_print(model\_name)* η οποία δέχεται σαν όρισμα το όνομα ενός δικτύου, το εκπαιδεύει, αποθηκεύει το δίκτυο και εκτυπώνει τα αποτελέσματα της μάθησης.

Επίσης η συνάρτηση χειρίζεται την μετατροπή των δεδομένων εισόδου στα δίκτυα καθώς τα δίκτυα Inception-ResNet V2,Inception V3 απαιτούν είσοδο με διαστάσεις (75, 75, 3) και το Xception διαστάσεις (71, 71, 3) ενώ το σύνολο των δεδομένων που μας παρέχεται έχει διαστάσεις (32, 32, 3).

Η μέθοδος fit είχε σαν παραμέτρους 50 εποχές και 30 βήματα για κάθε εποχή ενώ σαν συνάρτηση του loss είχαμε για όλα τα μοντέλα την sparce categorical crossentropy.

Τα αποτελέσματα των εκπαιδεύσεων παρατίθεται στους παρακάτω πίνακες:

BATCH\_SIZE = 128

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Train\_acc | Train\_loss | Test\_acc | Test\_loss | Time(s) |
|  |  |  |  |  |  |
| Scratch\_1 | 0.4731 | 1.7308 | 0.46 | 1.83 | 17.358 |
| Scratch\_2 | 0.7334 | 0.8283 | 0.58 | 1.5 | 15.645 |
| Scratch\_3 | 0.8078 | 0.6609 | 0.6 | 1.49 | 25.817 |
| DenseNet | 0.9978 | 0.0154 | 0.75 | 0.98 | 168.052 |
| Inception-ResNet | 0.9976 | 0.0097 | 0.84 | 2.22 | 406.358 |
| Inception V3 | 0.9981 | 0.0076 | 0.78 | 0.9 | 212.768 |
| MobileNet v2 | 0.8894 | 0.3914 | 0.36 | 3.32 | 43.174 |
| ResNet v2 | 0.9950 | 0.0345 | 0.62 | 1.6 | 253.642 |
| VGG16 | 1.0000 | 0.0001 | 0.8 | 1.3 | 96.688 |
| VGG19 | 0.9957 | 0.0148 | 0.78 | 1.28 | 116.694 |
| Xception | 1.0000 | 0.0013 | 0.86 | 0.56 | 505.147 |

Οι παραπάνω εκπαιδεύσεις δεν είχαν καμία βελτιστοποίηση σε εφαρμογή. Έγινε ξεκάθαρη κατά την εκπαίδευση η αυξημέχη χρήση μνήμης RAM (το πρόγραμμα έτρεξε στην GPU του colab της Google) καθώς και η υπερεκπαίδευση αφού βλέπουμε σε όλα τα δίκτυα μεγάλες αποκλείσεις μεταξύ test accuracy και train accuracy (με πιθανή εξαίρεση στα δικά μας μοντέλα).

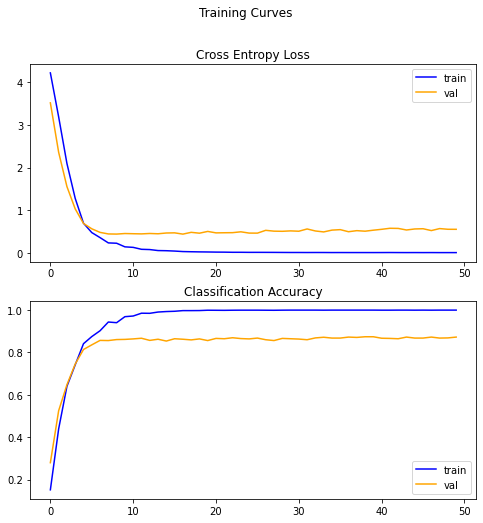
Πιο συγκεκριμένα το MobileNet (και λιγότερο το ResNet) εμφάνισε καλά αποτελέσματα στο train set ενώ είχε πολύ χαμηλή απόδοση 0.36 στο test set αποδίδοντας χειρότερα και από το πιο απλό μοντέλο που υλοποιήσαμε εμείς.

Αντίστοιχα τα υπόλοιπα δίκτυα που υλοποιήθηκαν με transfer learning (όλα πλιν των MobileNet, ResNet) είχαν accuracy μεγαλύτερη από 0.99 στο train set αλλά μικρότερη από 0.85 στα test που δηλώνει έντονη υπερεκπαίδευση.

Ακόμη παρατηρόντας τα γραφήματα βλέπουμε πως τα περισσότερα δίκτυα μετά την 20η ή την 30η εποχή δεν βελτιώνουν την απόδοσή τους και άρα εκτελούνται και καταναλώνουν άσκοπα πόρους για ένα μεγάλο διάστημα.

Ενδεικτικά:

Xception – BATCH\_SIZE = 128



Όλα τα παραπάνω προβλήματα που αναφέρθηκαν προσπαθούν να επιλυθούν με τις βελτιστοποιήσεις που ακολουθούν.

*Μέρος 3 – Βελτιστοποίηση μνήμης και χρόνου (num\_of\_classes=20)*

Τα παραπάνω μοντέλα κατά την εκτέλεσή τους εμφάνισαν σχετικά προβλήματα στην διαχείριση της μνήμης με μερικά από αυτά να καταναλώνουν περισσότερο από 6GB Ram ενώ χρειάστηκαν και αρκετό χρόνο για την εκπαίδευσή τους.

Για αυτό τον λόγο δώσαμε βάση στην βελτιστοποίηση αρχικά αυτών των τομέων οι οποίοι αφορούν την διαχείρηση κυριώς των δεδομένων και της μνήμης και δεν επηρεάζουν την υλοποίηση των μοντέλων (aka Hardware Optimization – Colab).

TFRecord:

Για την βελτιστοποίηση της μνήμης χρησιμοποιήθηκε η τεχνική του tfrecord, κατά την οποία τα δεδομένα αντί να αποθηκεύονται στην μνήμη και να «περιμένουν» να δεχτούν επεξεργασία (να χρησιμοποιηθούν σαν train/val/test), αποθηκεύονται σε εξωτερική μνήμη σαν tfrecord, μια δομή δεδομένων για την εύκολη διαχείρησή τους από την βιβλιοθήκη tensorflow. Κατ’ αυτό τον τρόπο τα δεδομένα αποθηκεύονται σειριακά στον δίσκο ενώ στην μνήμη υπάρχει ένας buffer δεδομένου μεγέθους ο οποίος τραβάει σιγά σιγά τα δεδομένα και τα επεξεργάζεται παράλληλα με το μοντέλο (όσο αυτό χρησιμοποιεί άλλα δεδομένα) για να είναι κατάλληλα προς χρήση. Έτσι αντί να έχουμε όλο το σύνολο δεδομένων στην μνήμη κρατάμε τον buffer ο οποίος παίρνει τα δεδομένα όποτε δημιουργείται χώρος.

Prefetching:

Η εφαρμογή του prefetching επιτρέπει στο pipeline της εισόδου των δεδομένων εκπαίδευσης να είναι πάντα γεμάτο και το μοντέλο να έχει έτοιμα preprocessed δεδομένα προς χρήση σε κάθε βήμα χωρίς να χρειάζεται να «ζητάει» δεδομένα και να δαπανείται χρόνος σε αυτή την αναμονή.

Mapping:

Με την χρήση του mapping επιτρέπουμε στα δεδομένα μας να προ-επεξεργάζονται παράλληλα. Η χρήση του είναι η ακόλουθη. Δηλώνουμε στο σύνολο δεδομένων ότι θέλουμε να εκτελέσουμε μία συνάρτηση (έστω η resize). Κάνοντας map αυτή την συνάρτηση επάνω στα δεδομένα, αυτή εφαρμόζεται παράλληλα στο σύνολο αυτών, γλιτώνοτνας τον χρόνο εφαρμογής αυτού του transformation στα δεδομένα σειριακά.

Caching:

Η χρήση του cache δίνει την δυνατότητα στο σύστημα να αποθηκεύσει ορισμένες εκτελέσεις του dataset ώστε αυτές να μην χρειάζεται να εκτελούνται σε κάθε νέα εποχή.

Χρησιμοποιώντας τις παραπάνω βελτιστοποιήσεις, εκπαιδεύσαμε ξανά τα μοντέλα και τα αποτελέσματα που πήραμε αναγράφονται στους παρακάτω πίνακες:

BATCH\_SIZE = 128

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Train\_acc | Train\_loss | Test\_acc | Test\_loss | Time(s) |
|  |  |  |  |  |  |
| Scratch\_1 | 0.6955 | 0.9718 | 0.51 | 1.74 | 11.237 |
| Scratch\_2 | 0.6815 | 0.9034 | 0.5 | 1.96 | 10.5673 |
| Scratch\_3 | 0.7107 | 0.9623 | 0.6 | 1.35 | 21.4848 |
| DenseNet | 0.9978 | 1.0902 | 0.76 | 1.06 | 163.2927 |
| Inception-ResNet | 0.9935 | 0.0382 | 0.82 | 0.83 | 390.003 |
| Inception V3 | 0.9935 | 0.0375 | 0.8 | 0.95 | 185.784 |
| MobileNet v2 | 0.7042 | 1.0006 | 0.22 | 3.11 | 39.989 |
| ResNet v2 | 0.8111 | 0.7263 | 0.61 | 1.61 | 247.809 |
| VGG16 | 0.9949 | 0.0308 | 0.79 | 1.08 | 92.677 |
| VGG19 | 0.9946 | 0.0296 | 0.79 | 1.05 | 112.914 |
| Xception | 1.0000 | 0.0047 | 0.84 | 0.72 | 492.061 |

Παρατηρούμε πως πετυχαίνουμε μείωση του χρόνου εκπαίδευσης για όλα τα μοντέλα περίπου όμως σταθερό κοντά στα 5 με 15 δευτερόλεπτα σύνολο (της τάξης 5-25% ανάλογα το μέγεθος του μοντέλου). Το γεγονός ότι η μείωση είναι περίπου σταθερή και δεν εξαρτάται από το μοντέλο είναι ολίγον τι αναμενόμενη και έγγυται στο γεγονός ότι οι παραπάνω βελτιστοποιήσεις αφορούσαν την είσοδο-τροφοδοσία των δεδομένων στο μοντέλο. Συνεπώς για σταθερό πλήθος εποχών και βημάτων ανά εποχή αλλά και δεδομένων σε κάθε μοντέλο είναι αναμενόμενο να έχουμε περίπου σταθερή μείωση χρόνου εκπαίδευσης.

Ακόμη πρέπει να επισημάνουμε ότι παρατηρήθηκε σημαντική μείωση στην χρήση της μνήμης της gpu που χρησιμοποιήθηκε με ποσοστό της χρήσης να μένει πρακτικά σταθερό και μονίμως κάτω απο 30% καθόλη την διάρκεια εκπαίδευσης όλων των μοντέλων.

Τέλος παρατηρούμε ότι, όπως ήταν αναμενόμενο δεν είχαμε βελτίωση στην απόδοση των μοντέλων για τις δεδομένες μετρικές, γεγονός που οφείλεται στο ότι δεν αλλάξαμε καθόλου το κομμάτι μοντέλα-δεδομένα-μάθηση ενώ οι μικρές διακυμάνσεις που παρατηρούνται οφείλονται στην τυχαιότητα των train set και validation set που αλλάζουν σε κάθε iteration και εκπαιδεύουν τελικά διαφορετικά το μοντέλο.

Για την βελτιστοποίηση λοιπόν της εκπαίδευσης των μοντέκων ως προς την μετρική και την αντιμετόπιση του overfitting θα χρησιμοποιήσουν επιπλέον μεθόδους, οι οποίες αναγράφονται παραπάτω.

*Μέρος 4 – Βελτιστοποίηση μοντέλου και υπερεκπαίδευσης*

*(num\_of\_classes=20)*

Με τις προηγούμενες βελτιστοποιήσεις, όπως προείπαμε, καταφέραμε να μειώσουμε τον χρόνο εκπαίδευσης με σωστότερη διαχείρηση των δεδομένων αλλά δεν είχαμε κάποια αλλαγή στα αποτελέσματα των μετρικών των μοντέλων. Για αυτό τον λόγο θα χρησιμοποιήσουμε τις ακόλουθες τεχνικές που καλούνται να αντιμετοπίσουν το πρόβλημα της υπερεκπαίδευσης.

Early stopping:

Με την μέθοδο αυτή εμποδίζουμε ένα μοντέλο να συνεχίζει να εκπαιδεύεται ενώ δεν υπάρχει βελτίωση στη ζητούμενη μετρική. Πιο συγκεκριμένα, ελέγχεται η μεταβολή της τιμής της επιλεγμένης μετρικής και εάν αυτή δεν μεταβληθεί σημαντικά για ένα δεδομένο πλήθος αριθμών εποχών, τότε η εκπαίδευση του μοντέλου σταματάει. Στην περίπτωσή μας θέσαμε σαν μετρική το loss και σαν έλεγχο εάν αυτή δεν μεταβληθεί κατά 0.004 σε διάστημα 4 εποχών, να σταματάει η εκπαίδευση.

Dropout:

Για την υλοποίηση του dropout αυτό που κάνουμε είναι να προσθέτουμε ένα layer dropout στο δίκτυό μας το οποίο μηδενίζει τυχαία ένα ποσοστό των βαρών προηγούμενου layer αποφεύγοντας έτσι την υπερεκπαίδευση. Στα μοντέλα from Scratch προσθέσαμε ένα dropout layer με παράμετρο 0.25 (μηδενίζει το 25% των βαρών του προηγούμενου layer), ενώ στα μοντέλα με transfer learning προσθέσαμε 2 dropout layers και ένα dense layer ενδιάμεσα με παραμέτρους 0.25, 256, 0.4 αντίστοιχα.

Data Augmentation:

Η τεχνική του data augmentation είναι αυτή κατά την οποία μετασχηματίζουμε τα δεδομένα με τυχαίους μετασχηματισμούς (όπως flip, zoom, rotate) έτσι ώστε το μοντέλο να μην υπερεκπαιδευτεί στα αρχικά δεδομένα. Στην άσκηση φτιάξαμε το καινούργιο dataset το οποίο αποτελείται από την συνένωση των αρχικών δεδομένων και των μετασχηματισμένων.

Εκπαιδεύσαμε τα μοντέλα με την χρήση όλων των βελτιστοποιήσεν που έχουν προαναφερθεί (και αυτών της μνήμης και χρόνου) και τα αποτελέσματα παρατίθενται στους παρακάτω πίνακες.

Ακόμη, αλλάξαμε το πλήθος των εποχών σε 70 για να δούμε και την αποτελεσματικότητα του Early Stopping

BATCH\_SIZE = 128

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Train\_acc | Train\_loss | Test\_acc | Test\_loss | Time(s) |
|  |  |  |  |  |  |
| Scratch\_1 | 0.7957 | 0.6593 | 0.51 | 1.87 | 16.047 |
| Scratch\_2 | 0.8643 | 0.4243 | 0.56 | 2.51 | 14.523 |
| Scratch\_3 | 0.6944 | 0.9865 | 0.62 | 1.32 | 20.041 |
| DenseNet | 0.9913 | 0.0419 | 0.75 | 1.02 | 170.293 |
| Inception-ResNet | 0.9922 | 0.0383 | 0.82 | 3.47 | 360.893 |
| Inception V3 | 0.9911 | 0.0479 | 0.76 | 0.99 | 168.839 |
| MobileNet v2 | 0.8135 | 0.6092 | 0.27 | 2.66 | 54.717 |
| ResNet v2 | 0.7675 | 0.8333 | 0.57 | 1.64 | 237.388 |
| VGG16 | 0.9946 | 0.0351 | 0.8 | 0.84 | 54.473 |
| VGG19 | 0.9854 | 0.0605 | 0.78 | 1.08 | 83.767 |
| Xception | 0.9982 | 0.136 | 0.84 | 0.7 | 329.434 |

Οι συγκεκριμένες αλλαγές στα μοντέλα και στα δεδομένα εισόδου έφεραν ανάμεικτα αποτελέσματα. Σε πολλές περιπτώσεις όπως αυτή των ResNet και Inception-ResNet είχαμε την εφαρμογή του Early Stopping πολύ νωρίς και σαν αποτέλεσμα τα μοντέλα αυτά δεν έμαθαν με τα augmented inputs. Σε άλλες περιπτώσεις βλέπουμε ότι η μέθοδος του Early Stopping να εφαρμόζεται πλήρως σωστά και να έχουμε παύση της εκπαίδευσης σε κατάλληλο σημείο ώστε να γλιτώσουμε πόρους.

Με την χρήση των παραπάνω βελτιστοποιήσεων πετύχαμε να αποφύγουμε την υπερεκπαίδευση των μοντέλων και την εξοικονόμηση πόρων αφού σταματήσαμε την διαδικασία μάθησης σε κατάλληλο σημείο.

*Μέρος 5 – Μεταφορά συναρτήσεων βελτιστοποίησης σε πρόβλημα 80 κλάσεων*

*(num\_of\_classes=80)*

Στο σημείο αυτό θέσαμε το πλήθος κλάσεων 80 και φτιάξαμε ένα καινούργιο dataset. Επάνω σε αυτό εκπαιδεύσαμε τα μοντέλα μας με BATCH\_SIZE = 128 την ίδια ακολουθία

1. Αρχικά χωρίς κάποια βελτιστοποίηση
2. Βελτιστοποίηση χρόνου και μνήμης
3. Βελτιστοποίηση μοντέλου και υπερεκπαίδευσης

*Εκπαίδευση χωρίς βελτιστοποίηση:*

Εκτελώντας την προαναφερθήσα εκπαίδευση μπορούμε να συγκρίνουμε τις αποδόσεις των μοντέλων. Δεδομένου ότι έχουμε διαφορετικό και πιο δύσκολο task περιμένουμε και χαμηλότερη απόδοση τόσο στην ταχύτητα όσο και στο accuracy.

Συγκρίνοντας βλέπουμε ότι όντως έχουμε σημαντική αύξηση στον χρόνο με κάποια μοντέλα (όπως το model 1) να χρειάζονται διπλάσιο χρόνο εκπαίδευσης ενώ πετυχαίνουν και δραματικά χαμηλότερο (υποτριπλάσιο) accuracy.

Ακόμη, παρατηρούμε ότι τα μοντέλα τα οποία είχαν καλή απόδοση στο προηγούμενο task φαίνεται να παρουσιάζουν μια «ανθεκτικότητα» με την λογική ότι τα υψηλά τους σκορ έπεσαν μεν αλλά κρατήθηκαν σε ένα ικανοποιητικό επίπεδο.

Αυτό που πρέπει να σημειωθεί είναι η πολύ κακή διαχείρηση της μνήμης η οποία σε αρκετές περιπτώσεις ξεπέρασε τα 12.9GB που επιτρέπει το colab με αποτέλεσμα το πρόγραμμα να κρασάρει και να απαιτεί επανέναρξη του πυρήνα.

*Βελτιστοποίηση χρόνου και μνήμης:*

Το πιο σημαντικό γεγονός το οποίο πρέπει να αναφερθεί σε αυτή την παράγραφο είναι η πολύ σωστή διαχείρηση της μνήμης. Σε αντίθεση με το πρώτο κομμάτι -της εκπαίδευσης χωρίς τη δεδομένη βελτιστοποίηση- η μνήμη παρέμενε σταθερά κάτω από τα 4GB σε όλο το κομμάτι εκπαίδευσης αποφεύγοντας τα προβλήματα που αναφέρθηκαν προηγουμένως.

Οι πτώσεις του χρόνου ήταν επίσης εκπληκτικές. Παρατηρούμε, σε αντίθεση με το πρώτο task πτώσεις της τάξης 50% όπως για παράδειγμα στο model\_1 (από 30 σε 15 δευτερόλεπτα), ενώ σε άλλα μοντέλα όμως το Xception και Inception-ResNet είχαμε μείωση κατά περίπου 150 δευτερόλεπτα της τάξης 25% σε αυτά τα «βαριά» μοντέλα.

Οι μειώσεις στον χρόνο που παρατηρούνται είναι της (εκπληκτικής) τάξης 15-50%!

Παρ’ όλα αυτά και όπως είναι αναμενόμενο δεν έχουμε κάποια βελτίωση στα accuracy και loss.

*Βελτιστοποίηση μοντέλου και υπερεκπαίδευσης:*

Οι συγκεκριμένες αλλαγές στα μοντέλα και στα δεδομένα εισόδου έφεραν ανάμεικτα αποτελέσματα. Σε πολλές περιπτώσεις όπως αυτή των model 3 και MobileMet είχαμε την εφαρμογή του Early Stopping πολύ νωρίς και σαν αποτέλεσμα τα μοντέλα αυτά δεν έμαθαν με τα augmented inputs. Σε άλλες περιπτώσεις βλέπουμε το η μέθοδος του Early Stopping να εφαρμόζεται πλήρως σωστά και να έχουμε παύση της εκπαίδευσης σε κατάλληλο σημείο ώστε να γλιτώσουμε πόρους.

Όμως σε περιπτώσεις όπως αυτή του Xception δεν έχουμε εφαρμογή του Early Stopping στις 70 εποχές που δηλώνει ότι το μοντέλο μπορούσε ακόμα να εκπαιδευτεί στο augmented dataset και να δώσει καλύτερα αποτελέσματα. Στην περίπτωση αυτή όπως είναι λογικό ο χρόνος εκπαίδευσης είναι αυξημένος λόγο και του αυξημένου πλήθους εποχών.

*Μέρος 6 – Βελτιστοποίηση με Keras\_tuner*

Το εργαλείο Keras tuner υπολογίζει τις βέλτιστες τιμές για παραμέτρους από layers του δικτύοθ, οριζόμενες από τον χρήστη.

Σε αυτή την άσκηση, για να συγκρίνουμε τα αποτελέσματα με τα προηγούμενα μοντέλα προσθέσαμε και πάλι 3 layers με παραμέτρους, 2 Dropout layers με εκταση τιμών rate = [0.1, 0.5] με βήμα 0.1, και ένα Dense layer με units = [128, 512] με βήμα 128 και πιθανές συναρτήσεις ενεργοποίησης τις tanh και relu.

Ακόμη θέσαμε σαν παράμετρο προς επιλογή το learning rate με πιθανές τιμές τις 10-2, 10-3 και 10-4.

Λόγο της μεγάλης χρονικής πολυπλοκότητας του tuning και της περιορισμένης παροχής υπηρεσιών (GPU-runtime session) από το google colab δεν καταφέραμε να κάνουμε tuning σε όλα τα μοντέλα.

Για BATCH\_SIZE = 128 και το dataset των 80 κλάσεων εκτελέσαμε tuning στο μοντέλο VGG16 και MobileNet.

Για το VGG16 έχουμε το accuracy 0.45 για τις επιλογές του tuner σε αντίθεση με το 0.61 που είχαμε με τις δικές μας βελτιστοποιήσεις το οποίο δείχνει πως ο tuner έκανε overfitting (αφού έχει και train\_acc = 1)

Για το MobileNet έχουμε το accuracy 0.55, εκπληκτικά ανώτερο του 0.11 που έδωσε η εφαρμογή των προηγούμενων βελτιστοποιήσεων.

Η εκτέλεση του keras\_tuner υλοποιείται και παρουσιάζεται στο αρχείο *Neural Exc-3 Models' File - 80 classes.ipynb .*