Νευρωνικά Δίκτυα και Ευφυή Υπολογιστικά Συστήματα

3η Εργαστηριακή Άσκηση

Βαθιά μάθηση στο CIFAR-100

Ομάδα 52

Θωμάς Δούκας	Φοίβος-Ευστράτιος Καλεμκερής	Ιωάννης Ψαρράς
03116081	03116010	03116033

1 Εισαγωγή

Στο πλαίσιο της 3ης εργαστηριακής άσκησης κληθήκαμε να βελτιστοποιήσουμε την απόδοση μοντέλων Βαθιάς Μάθησης στο σύνολο δεδομένων CIFAR-100, χρησιμοποιώντας την βιβλιοθήκη TensorFlow 2. Για το σκοπό αυτό χρησιμοποιήθηκε το notebook της εκφώνησης, στο οποίο πραγματοποιήθηκε η εκπαίδευση των μοντέλων προκειμένου να αξιοποιηθεί η δυνατότητα επιτάχυνσης με GPU που παρέχεται στο cloud από το Google Colab. Στην παρούσα αναφορά θα επιχειρήσουμε να περιγράψουμε τη διαδικασία βελτιστοποίησης που ακολουθήσαμε για κάθε μοντέλο, παραθέτοντας παράλληλα τα αποτελέσματα της εκπαίδευσης και καταλήγοντας σε μια συγκριτική παρουσιάση της συμπεριφοράς των καλύτερων μοντέλων μας για διαφορετικές παραμέτρους εκπαίδευσης (πχ πλήθος κλάσεων, optimizer, learning rate, batch size).

2 Προεπεξεργασία Δεδομένων

Όπως ήδη αναφέραμε, για τις ανάγκες της άσκησης θα χρησιμοποιηθεί το dataset CIFAR-100. Αυτό αποτελείται από 60000 έγχρωμες 32x32 εικόνες, οι οποίες χωρίζονται σε 100 κλάσεις (που κατηγοριοποιούνται περαιτέρω σε 20 υπερκλάσεις) με 600 εικόνες η κάθε μία. Κάθε κλάση περιλαμβάνει 500 εικόνες για training και 100 για testing. Οι εικόνες περιλαμβάνουν ένα label για τον προσδιορισμό της κλάσης στην οποία ανήκουν (”fine”) και ένα για την αντίστοιχη υπερκλάση (”coarse”).
Για την εκπαίδευση και αξιολόγηση των μοντέλων θα χρησιμοποιηθεί ένα υποσύνολο του αρχικού dataset, το οποίο φορτώνεται στην αρχή του notebook σύμφωνα με τον αριθμό της ομάδας μας (52). Ταυτόχρονα, δίνεται η δυνατότητα για επιλογή του αριθμού των κλάσεων. Προκειμένου να ελαττώσουμε τον απαιτούμενο χρόνο εκπαίδευσης θα αναπτύξουμε τα μοντέλα μας για 20 κλάσεις και θα αξιολογήσουμε την απόδοσή τους μετά τη βελτιστοποίηση και για μεγαλύτερα μεγέθη του προβλήματος (80 κλάσεις).

Για την επιτάχυνση της επεξεργασίας των δεδομένων εισόδου έχει χρησιμοποιηθεί η τεχνική data prefetching. Αυτή βασίζεται στην επικάλυψη ενός σταδίου εκπαίδευσης του μοντέλου με τη διαδικασία προφόρτωσης των δεδομένων που απαιτούνται για την εκτέλεση του επόμενου βήματος, με σκοπό την εξοικονόμηση χρόνου. Στην προκειμένη περίπτωση, ο αριθμός των στοιχείων που προφορτώνονται σε κάθε βήμα ρυθμίζεται δυναμικά από το tf.data κατά το runtime (AUTOTUNE). Η παραπάνω μέθοδος, αν και μειώνει το χρόνο εκπαίδευσης, προκαλεί αύξηση της απαιτούμενης μνήμης. Δεδομένου του ότι χρησιμοποιήσαμε υποσύνολο 20 κλάσεων για όλη τη διαδικασία βελτιστοποίησης, ωστόσο, δεν αντιμετωπίσαμε πρόβλημα.
Προτού προβούμε στην ανάπτυξη των μοντέλων, επιλέγουμε να εφαρμόσουμε την τεχνική data augmentation στα δεδομένα εισόδου, προκειμένου να δημιουργήσουμε μεγαλύτερη ποικιλία στα δεδομένα εφαρμόζοντας τυχαίους αλλά ρεαλιστικούς μετασχηματισμούς στις εικόνες, στοχεύοντας στη βελτίωση, τελικά, της ποιότητας του συνόλου δεδομένων. Συγκεκριμένα, θα εφαρμόσουμε μετασχηματισμούς που παρέχονται από το ImageDataGenerator και, ειδικότερα, width shift, height shift, horizontal flip, rotation και zoom.

Στη συνέχεια, θα παρουσιάσουμε τη διαδικασία βελτιστοποίησης 3 μοντέλων, ενός φτιαγμένου from scratch και δύο με την τεχνική μεταφοράς μάθησης. Θα ξεκινήσουμε κάθε φορά από το απλό μοντέλο και, διατηρώντας σταθερά τον αριθμό κλάσεων(20) και το BATCH_SIZE (128) θα επιχειρήσουμε να το βελτιστοποιήσουμε ως προς τη μετρική απόδοσης (ακρίβεια πρόβλεψης - accuracy). Ως μετρική απωλειών θα χρησιμοποιηθεί το Sparse Categorical Crossentropy, ενώ ο αριθμός των epochs παραμένει σταθερός (500) και ο αριθμός των steps σταθερός για συγκεκριμένο μέγεθος dataset. Θα χρησιμοποιηθεί, τέλος, η μέθοδος early stopping, ώστε να διακόπτεται η διαδικασία εκπαίδευσης όταν δεν υπάρχει βελτίωση για έναν αριθμό epochs (20). Ύστερα, θα μελετήσουμε τη συμπεριφορά του βέλτιστου μοντέλου για διαφορετικές τιμές των παραπάνω παραμέτρων.

3 From Scratch

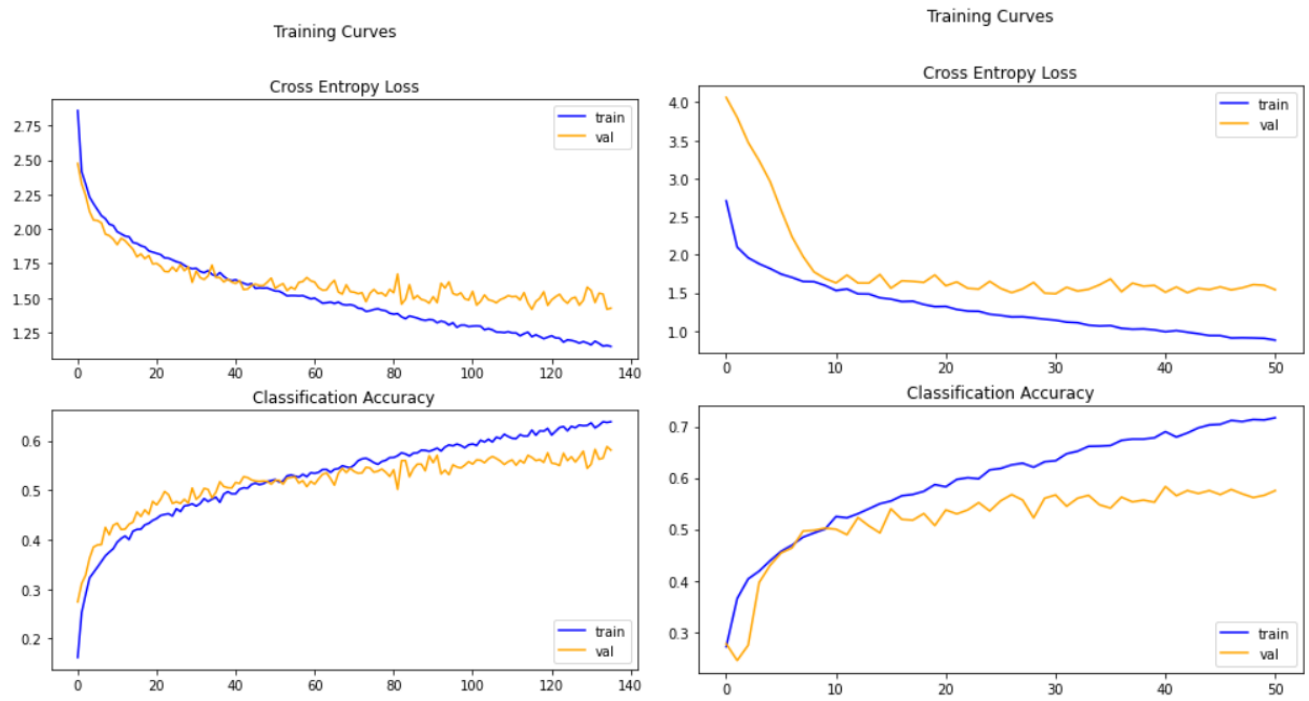
Θα μελετήσουμε πρώτα ένα απλό συνελικτικό μοντέλο γραμμένο from scratch, βασισμένοι σε παραδείγματα του tensorflow documentation, καθώς και σε συνήθεις πρακτικές κατά τη μελέτη του CIFAR. Ξεκινάμε από την απλούστερη μορφή του μοντέλου, με δύο μόνο συνελικτικά επίπεδα και 2 πυκνά στην έξοδο, όπως φαίνεται παρακάτω:

1. 2D Convolutional layer με 32 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
2. 2D Convolutional layer με 256 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
3. 2D MaxPooling layer (πυρήνας 2x2)
4. Flattening layer
5. Dense layer με 1024 εξόδους και συνάρτηση ενεργοποίησης ReLU
6. Dense layer με 100 εξόδους και συνάρτηση Softmax για εξαγωγή των πιθανοτήτων ανά κλάση

Στο παραπάνω μοντέλο έχει χρησιμοποιηθεί max pooling layer αμέσως μετά το 2ο συνελικτικό προκειμένου να περιοριστεί το over-fitting, κρατώντας μόνο τις παραμέτρους με τη μεγαλύτερη επίδραση.

3.1 Βελτιστοποίηση

Πραγματοποιήσαμε στη συνέχεια μια σειρά πειραμάτων διατηρώντας αρχικά σταθερές τις τιμές των optimizer (Adam), learning rate (0.0001) και BATCH_SIZE (128) για 20 κλάσεις. Τα αρχικά αποτελέσματα φαίνονται στο ακόλουθο διάγραμμα.

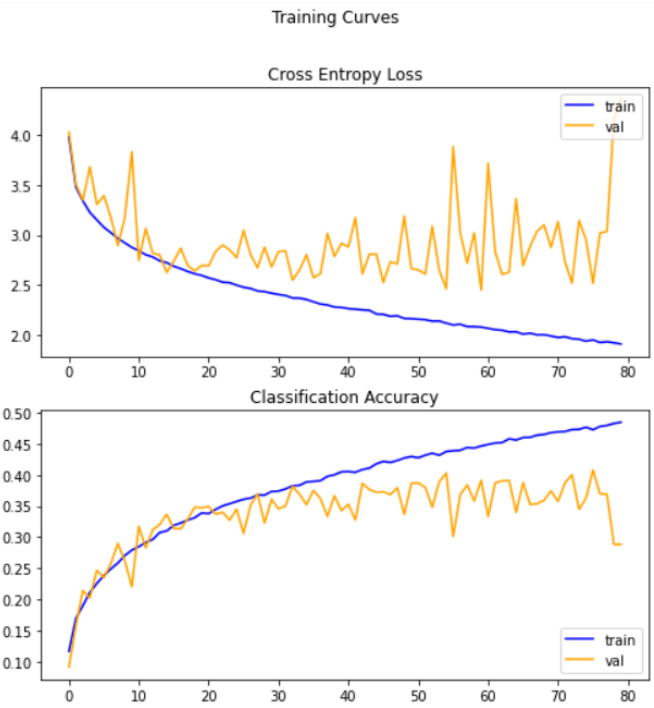


Σχήμα 1: Loss - Accuracy: συνελικτικό μοντέλο χωρίς dropout/normalization (αριστερά) και με (δεξιά)

Συγκεκριμένα, για το παραπάνω μοντέλο μετρήσαμε $loss = 1.38$ και $accuracy = 0.58$. Τα αποτελέσματα αυτά, φυσικά, δεν ήταν καθόλου άσχημα για πρώτη προσπάθεια. Παρατηρώντας το διάγραμμα (Σχήμα 1, εικόνα αριστερά) αντιλαμβανόμαστε ότι με την εφαρμογή του data augmentation στα δεδομένα εισόδου, τη χρήση του MaxPooling layer, αλλά και του early stopping στη φάση της εκπαίδευσης, αποτρέψαμε σε μεγάλο βαθμό την εμφάνιση over-fitting. Υπάρχει, βέβαια, απόκλιση μεταξύ των τιμών στο validation και το test set, αλλά αυτό είναι αναμενόμενο και όχι ιδιαίτερα ανησυχητικό, μιας και μέχρι τη διακοπή της εκπαίδευσης τα validation loss και accuracy εμφανίζουν φθίνουσα και αύξουσα τάση, αντίστοιχα.
Παρατηρούμε από την άλλη έντονη ταλάντωση στην τιμή του accuracy, η οποία μας οδηγεί σε τροποποίηση του αρχικού δικτύου. Η προσθήκη ενός Dropout layer (0.3) αμέσως μετά το MaxPooling, ειδικότερα, δεν επέφερε σημαντικές αλλαγές στην απόδοση, με την ακρίβεια πρόβλεψης να παραμένει αναλλοίωτη και την ταλάντωση να μην εξαλείφεται. Ο συνδυασμός του τελευταίου, ωστόσο, με ένα Batch Normalization layer συνέβαλε στον περιορισμό της ταλάντωσης, χωρίς να παρατηρηθεί μείωση της απόδοσης (Σχήμα 1, εικόνα δεξιά). Μετά τις προηγούμενες προσθήκες, το αρχικό μοντέλο έχει διαμορφωθεί ως εξής:

1. 2D Convolutional layer με 32 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
2. 2D Convolutional layer με 256 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
3. Batch Normalization layer
4. 2D MaxPooling layer (πυρήνας 2x2)
5. Dropout layer (0.3)
6. Flattening layer
7. Dense layer με 1024 εξόδους και συνάρτηση ενεργοποίησης ReLU
8. Dense layer με 100 εξόδους και συνάρτηση Softmax για εξαγωγή των πιθανοτήτων ανά κλάση

Θεωρώντας αρκετά ικανοποιητική την απόδοση του μοντέλου, επιχειρήσαμε να το δοκιμάσουμε για 80 κλάσεις. Ωστόσο, όπως φαίνεται και ακολούθως, τα αποτελέσματα ήταν απογοητευτικά.



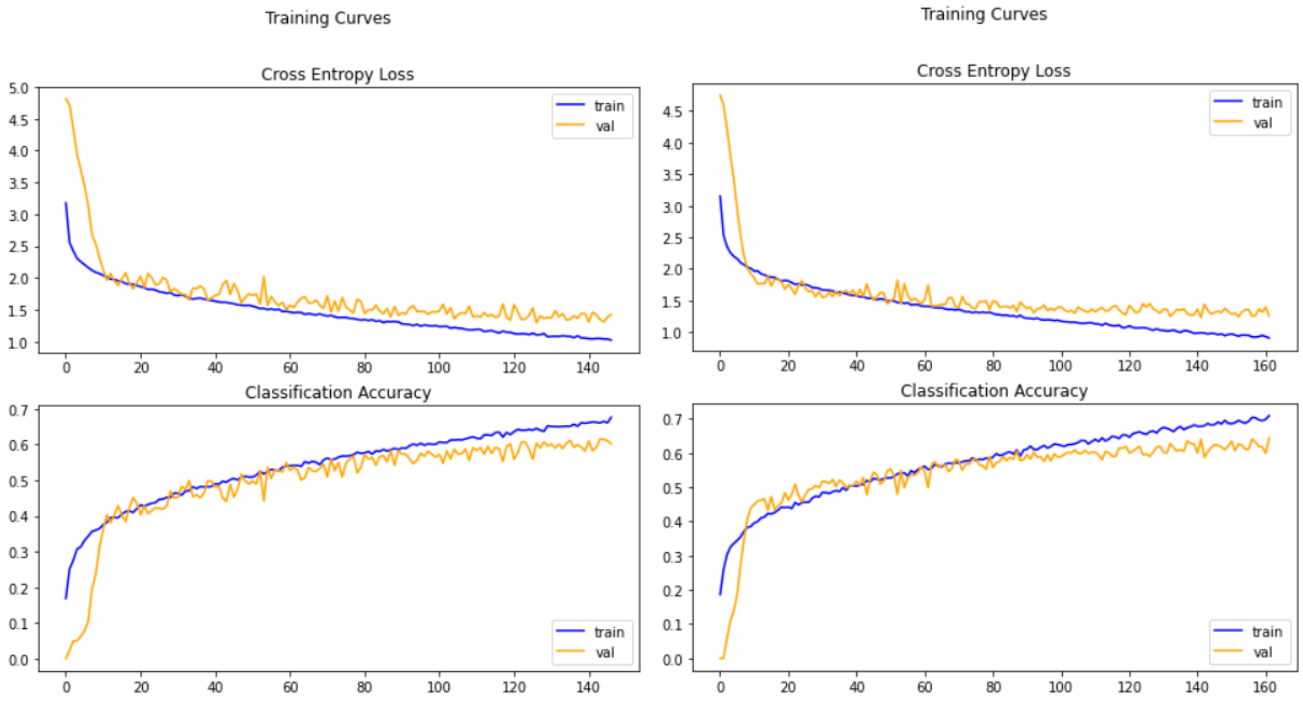
Σχήμα 2: Loss - Accuracy: πιο σύνθετο συνελικτικό μοντέλο με dropout/normalization, 80 κλάσεις

Η δοκιμή του αρχικού μοντέλου για 80 κλάσεις εμφάνισε σαφώς μειωμένη απόδοση. Ειδικότερα, μετρήθηκε $loss = 2.41$ και $accuracy = 0.40$. Ταυτόχρονα, όπως φαίνεται στο παραπάνω διάγραμμα, η ταλάντωση είναι πολύ μεγάλη. Συμπεραίνουμε ως εκ τούτου ότι το απλό μοντέλο, ενώ καταφέρνει να αποδώσει ικανοποιητικά για μικρό αριθμό κλάσεων, αποτυγχάνει για μεγαλύτερα προβλήματα, αφού τα χρησιμοποιούμενα συνελικτικά επίπεδα δεν επαρκούν για την εξαγωγή των χαρακτηριστικών των εικόνων.

Με δεδομένα τα παραπάνω αποφασίζουμε να προσθέσουμε συνελικτικά επίπεδα στο αρχικό μοντέλο με σκοπό αυτό να γενικεύει καλύτερα, ακόμα και για μεγαλύτερο αριθμό κλάσεων. Για τη μείωση του χρόνου εκπαίδευσης χρησιμοποιήθηκαν και πάλι 20 κλάσεις στη φάση της διερεύνησης. Τα πρώτα πειράματα με ένα επιπλέον συνελικτικό επίπεδο εμφανίστηκαν αποθαρρυντικά, ρίχνοντας ελαφρώς την ακρίβεια πρόβλεψης (0.58). Η προσθήκη dropout και batch normalization επιπέδων- όπως και πριν- είχε ήπια θετική επίδραση. Ειδικότερα, χωρίς να επηρεάζει την ακρίβεια ή το loss σημαντικά, φάνηκε να περιορίζει την απόκλιση της απόδοσης μεταξύ validation και test set και να περιορίζει την ταλάντωση. Η πρώτη σημαντική αύξηση της ακρίβειας παρατηρήθηκε με την προσθήκη ενός ακόμα συνελικτικού επιπέδου (μαζί με τα αντίστοιχα dropout, normalization και max pooling). Η νέα αρχιτεκτονική του μοντέλου παρατίθεται στη συνέχεια:

1. 2D Convolutional layer με 64 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
2. 2D Convolutional layer με 64 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
3. Batch Normalization layer
4. 2D MaxPooling layer (πυρήνας 2x2)
5. Dropout layer (0.3)
6. 2D Convolutional layer με 128 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
7. Batch Normalization layer
8. 2D MaxPooling layer (πυρήνας 2x2)
9. Dropout layer (0.3)
10. 2D Convolutional layer με 256 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ReLU και πυρήνα 3x3
11. Batch Normalization layer
12. 2D MaxPooling layer (πυρήνας 2x2)
13. Dropout layer (0.3)
14. Flattening layer
15. Dense layer με 1024 εξόδους και συνάρτηση ενεργοποίησης ReLU
16. Dense layer με 100 εξόδους και συνάρτηση Softmax για εξαγωγή των πιθανοτήτων ανά κλάση

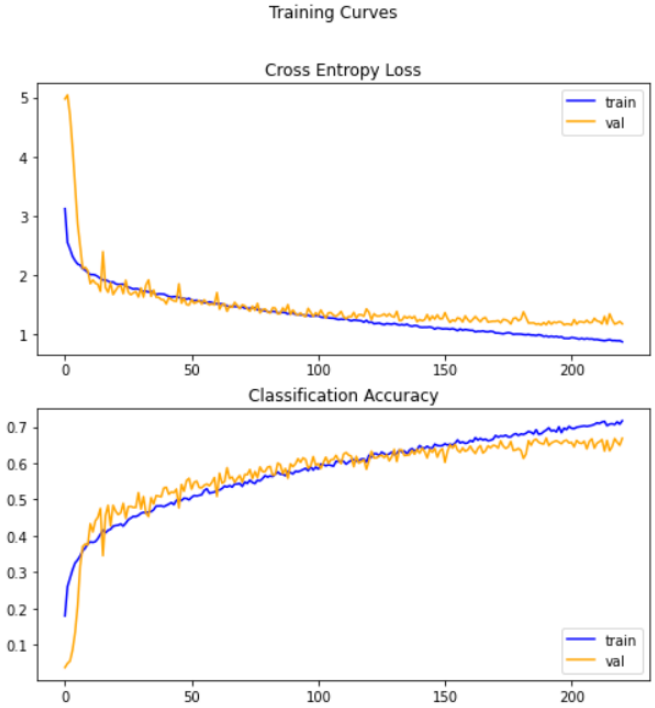
Αυτή τη φορά μετρήθηκαν $loss = 1.25$ και $accuracy = 0.62$, ενώ ικανοποιητικές θεωρούνται και οι καμπύλες εκπαίδευσης:



Σχήμα 3: Loss - Accuracy: πιο σύνθετο συνελικτικό μοντέλο, 20 κλάσεις

Παρατηρούμε ότι η απόκλιση μεταξύ του validation και του test set έχει μειωθεί πολύ, ενώ παρά την ταλάντωση, η ακρίβεια εμφανίζει αύξουσα τάση μέχρι τη διακοπή της εκπαίδευσης. Στο σημείο αυτό επιχειρήθηκαν μικρές τροποποιήσεις σε παραμέτρους του μοντέλου, με μέτρια ομολογουμένως αποτελέσματα.

Πιο συγκεκριμένα, η αλλαγή της τιμής του dropout στα ενδότερα συνελικτικά επίπεδα φάνηκε να μειώνει την ακρίβεια. Από την άλλη, η μείωση του στο 1ο εσωτερικό επίπεδο παρουσίασε βελτίωση που μας οδήγησε, τελικά, στην παντελή αφαίρεση του πρώτου Dropout layer, με τα αποτελέσματα να φαίνονται στα διαγράμματα της δεξιάς εικόνας του παραπάνω σχήματος (Σχήμα 3) με $loss = 1.18$ και $accuracy = 0.64$.
Τέλος, επιχειρήσαμε την αντικατάσταση της ReLU ως activation function από την ELU σε όλα τα επίπεδα, αλλαγή που συνεισέφερε σε μικρή- αλλά υπολογίσιμη- βελτίωση της απόδοσης με ($loss = 1.12$ και $accuracy = 0.66$).
Έχοντας φτάσει σε ένα καλό σημείο ως προς την αναμενόμενη απόδοση, δοκιμάζουμε και πάλι το μοντέλο μας στις 80 κλάσεις. Αυτή τη φορά, τα αποτελέσματα είναι πολύ καλύτερα. Ειδικότερα, η πτώση της ακρίβειας είναι πολύ μικρότερη, γεγονός που επιβεβαιώνει ότι το νέο μοντέλο είναι πολύ πιο ποιοτικό από το αρχικό. Τα διαγράμματα για την απόδοση του μοντέλου στις 80 κλάσεις φαίνονται παρακάτω.



Σχήμα 4: Loss - Accuracy: τελικό συνελικτικό μοντέλο, 80 κλάσεις

Και στις 80 κλάσεις έχουν διατηρηθεί τα καλά χαρακτηριστικά στην απόδοση του μοντέλου, με την απόκλιση μεταξύ validation και test set να είναι περιορισμένη και το over-fitting να έχει αποφευχθεί. Από την άλλη, αξίζει να σχολιάσουμε ότι το τελικό μοντέλο χρειάστηκε πολλή περισσότερη ώρα από τα αρχικά για την εκπαίδευσή του. Πιο συγκεκριμένα, στην περίπτωση των 80 κλάσεων χρειάστηκαν περίπου 230 epochs μέχρι να σταθεροποιηθεί η τιμή των απωλειών, ενώ στο αρχικό μοντέλο η αντίστοιχη σύγκλιση είχε επιτευχθεί σε μόλις 80 epochs. Στον ακόλουθους πίνακες παρουσιάζονται οι τιμές της απόδοσης του τελικού μοντέλου για 20 και 80 κλάσεις, ενώ προτού συνεχίσουμε με διερεύνηση της επίδρασης των υπόλοιπων παραμέτρων εκπαίδευσης παραθέτουμε και την αρχιτεκτονική του βέλτιστου μέχρι τώρα μοντέλου.

Loss	**Accuracy**
1.12	0.66

Πίνακας 1: Loss - Accuracy για το βέλτιστο μοντέλο, 20 κλάσεις

Loss	**Accuracy**
1.73	0.55

Πίνακας 2: Loss - Accuracy για το βέλτιστο μοντέλο, 80 κλάσεις

1. 2D Convolutional layer με 64 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ELU και πυρήνα 3x3
2. 2D Convolutional layer με 64 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ELU και πυρήνα 3x3
3. Batch Normalization layer
4. 2D MaxPooling layer (πυρήνας 2x2)
5. 2D Convolutional layer με 128 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ELU και πυρήνα 3x3
6. Batch Normalization layer
7. 2D MaxPooling layer (πυρήνας 2x2)
8. Dropout layer (0.3)
9. 2D Convolutional layer με 256 φίλτρα εξόδου, συνάρτηση ενεργοποίησης ELU και πυρήνα 3x3
10. Batch Normalization layer
11. 2D MaxPooling layer (πυρήνας 2x2)
12. Dropout layer (0.3)
13. Flattening layer
14. Dense layer με 1024 εξόδους και συνάρτηση ενεργοποίησης ELU
15. Dense layer με 100 εξόδους και συνάρτηση Softmax για εξαγωγή των πιθανοτήτων ανά κλάση

Βασιζόμενοι στο τελευταίο μοντέλο επιχειρήσαμε, ακολούθως, να πειραματιστούμε με διαφορετικές τιμές παραμέτρων εκπαίδευσης. Προτού αλλάξουμε τον Optimizer, δοκιμάσαμε αλλαγές στο learning rate για τον Adam. Πιο συγκεκριμένα έγιναν οι ακόλουθες μετρήσεις:

Learning Rate	**Loss**	**Accuracy**
0.01	3.12	0.23
0.001	1.99	0.51
0.0001	1.73	0.55
0.00001	2.05	0.46

Πίνακας 3: Adam Learning Rate - Loss - Accuracy, 80 κλάσεις

Όπως καθίσταται σαφές από τον παραπάνω πίνακα, τα καλύτερα αποτελέσματα για τον Adam λαμβάνονται για learning rate = 0.0001, βάσει του οποίου πραγματοποιήθηκε η βελτιστοποίηση, άλλωστε. Περαιτέρω μείωση του learning rate όχι μόνο δε βελτιώνει την απόδοση του μοντέλου, αλλά εκτοξεύει και το χρόνο εκπαίδευσης. Χαρακτηριστικά σημειώνουμε ότι για learning rate = 0.00001 και 80 κλάσεις η εκπαίδευση του μοντέλου μέχρι τη σταθεροποίηση της απόδοσης χρειάστηκε περίπου μιάμιση ώρα.

Έπειτα, ακολουθήσαμε την ίδια διαδικασία για διαφορετικούς optimizers. Ειδικότερα, δοκιμάσαμε διαφορετικές τιμές του learning rate για 2 συνηθισμένους optimizers, τους Stochastic Gradient Descent (SGD) και Root Mean Square propagation (RMSprop).

Learning Rate	**Loss**	**Accuracy**
0.1	1.74	0.55
0.01	1.71	0.54
0.001	2.37	0.38

Πίνακας 4: SGD Learning Rate - Loss - Accuracy, 80 κλάσεις

Φαίνεται ότι για το SGD χρειάστηκαν λιγότερα πειράματα. Ειδικότερα, η μετάβαση από learning rate 0.1 σε 0.01 παρουσίασε θετικά αποτελέσματα, μειώνοντας ελάχιστα το validation accuracy (0.54), αλλά μειώνοντας και τις απώλειες. Η τάση αυτή δε συνεχίστηκε και για learning rate = 0.001, για το οποίο παρατηρήθηκε απότομη πτώση της ακρίβειας και αντίστοιχη αύξηση των απωλειών, με αποτέλεσμα να μη δοκιμάσουμε μικρότερες τιμές.

Learning Rate	**Loss**	**Accuracy**
0.01	2.47	0.35
0.001	1.82	0.53
0.0001	1.74	0.55
0.00001	1.90	0.50

Πίνακας 5: RMSprop Learning Rate - Loss - Accuracy, 80 κλάσεις

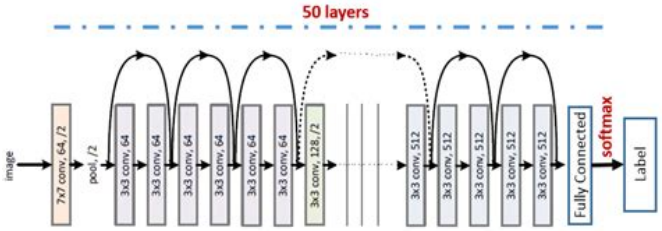
Όσον αφορά τον RMSprop από την άλλη, μείωση του learning rate φάνηκε να έχει θετικά αποτελέσματα ως προς την απόδοση. Βέλτιστη ακρίβεια λάβαμε για learning rate = 0.0001. Επιχειρήσαμε να χρησιμοποιήσουμε μικρότερο κατά μία τάξη μεγέθους rate, αλλά φάνηκε ότι τα 500 epochs δεν επαρκούσαν προκειμένου να επέλθει σύγκλιση, με αποτέλεσμα η εκπαίδευση να σταματήσει.
Τα καλύτερα αποτελέσματα για κάθε Optimizer συνοψίζονται στον ακόλουθο πίνακα. Αν και τα αποτελέσματα είναι παραπλήσια, στο notebook έχουμε κρατήσει ως βέλτιστο το μοντέλο με τον Adam.

Optimizer	**Learning Rate**	**Loss**	**Accuracy**
Adam	0.0001	1.74	0.55
SGD	0.01	1.71	0.54
RMSprop	0.0001	1.74	0.55

Πίνακας 6: Optimizer - Optimal Learning Rate - Loss - Accuracy, 80 κλάσεις

4 ResNet

Το μοντέλο ResNet (Residual Network) είναι ένα ισχυρό μοντέλο που αποδεικνύεται ιδιαίτερα αποτελεσματικό στον τομέα την όρασης υπολογιστών. Κεντρική ιδέα του μοντέλου ResNet είναι η υλοποίηση της λειτουργίας "identity shortcut connection". Αυτή δίνει τη δυνατότητα στο μοντέλο να προσπεράσει ένα ή περισσότερα επίπεδα, συσχετίζοντας τα αποτελέσματα ενός layer με αυτά επακόλουθων layers. Στην συγκεκριμένη περίπτωση, θα χρησιμοποιήσουμε το ResNet50, ένα ResNet μοντέλο που έχει βάθος 50 επιπέδων, όπως αυτό παρέχεται από το keras.



Σχήμα 5: ResNet

4.1 Βελτιστοποίηση

Ξεκινήσαμε την διερεύνηση για την εκπαίδευση του μοντέλου ResNet με μεταφορά μάθησης, εκπαιδεύοντας το μοντέλο που δίνεται από το keras και προσθέτοντας μόνο ένα Dense layer, με τα αποτελέσματα να είναι, αναμενόμενα, μη ικανοποιητικά. Επομένως, προσθέσαμε, αρχικά, ένα Dropout layer με rate = 0.5 ενώ παράλληλα επιχειρήσαμε να εμπλουτίσουμε το δοθέν dataset με χρήση του ImageDataGenerator, με στόχο το Data Augmentation. Παρόλ' αυτά, τα αποτελέσματα που λάβαμε δεν ήταν ενθαρρυντικά, καθώς η τιμή του **validation accuracy** έμεινε σταθερή στο **0.05**, όπως προηγουμένως.

Η πρώτη σημαντική βελτίωση σημειώθηκε όταν προσθέσαμε ένα Global Average Pooling layer, αμέσως μετά το Dropout, αφού από το 0.05, το validation accuracy έφτασε στο **0.48**.
Σε αυτό το σημείο, επιχειρήσαμε να κάνουμε freeze το Dropout layer, ή ακόμη και να το αφαιρέσουμε εντελώς. Παρόλο που η απουσία του δεν ήταν τόσο αισθητή όσο η απουσία του Global Average Pooling, η τιμή της ακρίβειας και πάλι μειώθηκε ελαφρώς. Επιπλέον, δοκιμάσαμε να αντικαταστήσουμε το Global Average Pooling με Global Max Pooling, πρακτική που επίσης οδήγησε σε μείωση του validation accuracy.

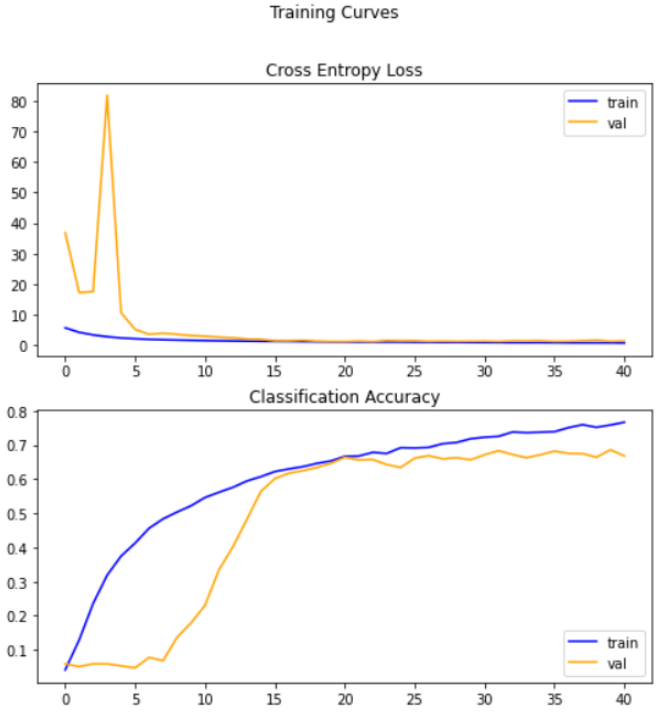
Σε αυτό το σημείο, αποφασίσαμε να προχωρήσουμε στη βελτιστοποίηση του μοντέλου που μας είχε δώσει τα καλύτερα αποτελέσματα μέχρι στιγμής. Το μοντέλο αυτό αποτελείται από το μοντέλο του ResNet, ακολουθούμενο από ένα Dropout και ένα Global Average Pooling layer και στο τέλος ένα Dense layer, όπως φαίνεται και παρακάτω.

1. ResNet50 (trainable)
2. 2D GlobalAveragePooling layer
3. Dropout layer (0.5)
4. Dense layer με εξόδους ίσες με τον αριθμό κλάσεων και συνάρτηση Softmax για εξαγωγή των πιθανοτήτων ανά κλάση

Αρχικά, επιχειρήσαμε να προσδιορίσουμε την βέλτιστη τιμή της παραμέτρου rate του Dropout. Κινούμενοι σε ένα φάσμα τιμών από 0.2 έως και 0.8, τα καλύτερα αποτελέσματα τα λάβαμε, αδιαμφισβήτητα, για rate = 0.7, όπου πήραμε **validation accuracy = 0.51**. Επόμενο βήμα, ήταν η επιλογή κατάλληλου optimizer. Μέχρι στιγμής, ο optimizer που είχε χρησιμοποιηθεί ήταν ο Adam με learning rate = 0.0001. Αποφασίσαμε να συγκρίνουμε τις τιμές του Adam για learning rates = [0.001, 0.0001, 0.00001] με τις τιμές του RMSprop για τα ίδια learning rates. Διαπιστώσαμε, λοιπόν, πως ο optimizer RMSprop δίνει κατά πολύ καλύτερα αποτελέσματα από τον Adam, για όλα τα παραπάνω learning rates. Πιο συγκεκριμένα, ενώ η μέγιστη τιμή validation accuracy με optimizer τον Adam ήταν 0.51 για learning rate = 0.0001, η μέγιστη τιμή που λάβαμε χρησιμοποιώντας τον RMSprop, ήταν **0.64**.

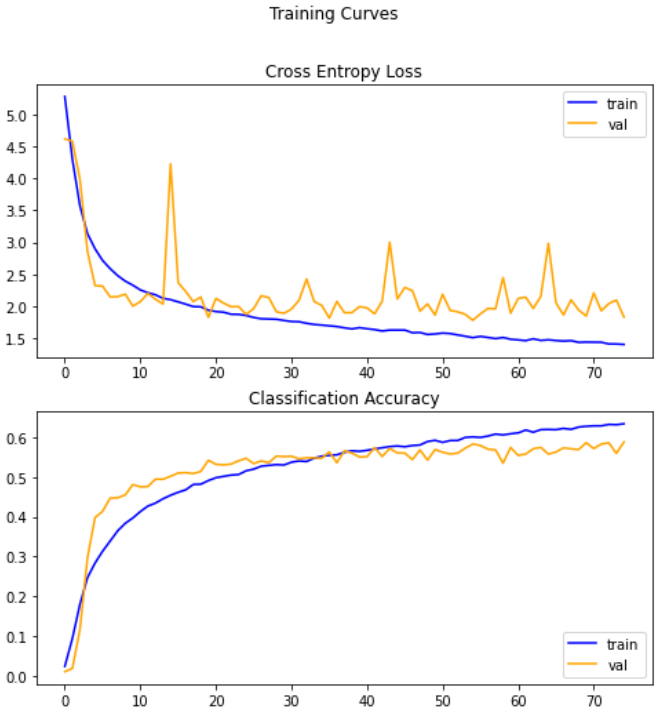
Έχοντας σχεδόν ολοκληρώσει την διαδικασία βελτιστοποίησης, ενσωματώσαμε την τεχνική early stopping, για να αποφύγουμε την περίπτωση του over-fitting. Πράγματι, η πρακτική αυτή ήταν ωφέλιμη, καθώς σημειώθηκε επιπλέον αύξηση του validation accuracy στο **0.67**, ενώ και το validation loss σημείωσε την ελάχιστη τιμή του, το **1.25**. Συγκεκριμένα, το early stopping τερμάτισε την εκπαίδευση του μοντέλου μας στο 41ο epoch.

Παρακάτω φαίνονται τα αποτελέσματα της εκτέλεσης του βελτιστοποιημένου μοντέλου για την περίπτωση των 20 κλάσεων.



Σχήμα 6: Loss - Accuracy: ResNet, 20 κλάσεις

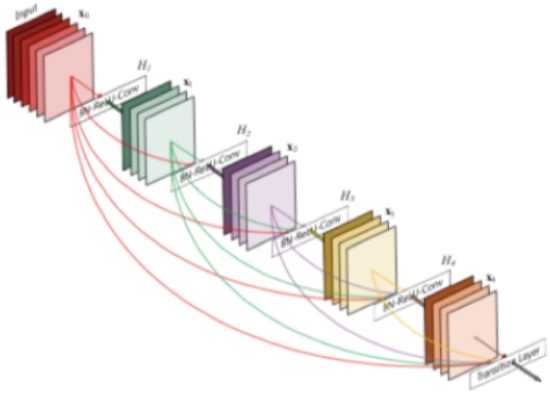
Τελικά, εξετάσαμε το βελτιστοποιημένο μοντέλο και με την χρήση 80 κλάσεων αντί για 20, λαμβάνοντας validation $loss = 1.75$ και $accuracy = 0.60$. Παρακάτω, απεικονίζεται γραφικά η διαδικασία της εκπαίδευσης του μοντέλου για τις μετρικές loss και accuracy.



Σχήμα 7: Loss - Accuracy: ResNet, 80 κλάσεις

5 DenseNet

Τέλος, θα μελετήσουμε τη συμπεριφορά του μοντέλου DenseNet στο πρόβλημα κατηγοριοποίησης που καλούμαστε να επιλύσουμε. Το DenseNet αποτελεί συνελικτικό νευρωνικό δίκτυο το οποίο χρησιμοποιεί πυκνές συνδέσεις μεταξύ των επιπέδων του. Συγκεκριμένα, κάθε επίπεδο δέχεται πολλαπλές εισόδους από όλα τα προηγούμενα επίπεδα και τροφοδοτεί με τα δικά του δεδομένα όλα τα επόμενα, με σκοπό να διατηρηθεί η τροφοδοσία προς τα εμπρός.

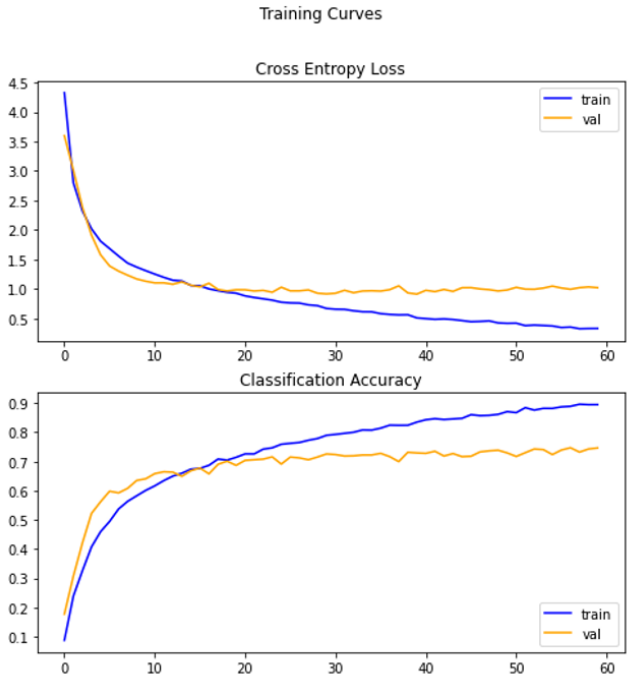


Σχήμα 8: DenseNet Structure

Στην προσπάθεια βελτιστοποίησης των αποτελεσμάτων, θα ακολουθήσουμε παρόμοια στρατηγική με τις προηγούμενες περιπτώσεις.
Εκτός από την αρχιτεκτονική του DenseNet προσθέτουμε στο δίκτυο μας ένα επίπεδο Dropout με σκοπό την μείωση της υπερεκπαίδευσης, ένα GlobalAveragePooling2D και τέλος ένα επίπεδο Dense ώστε να προσαρμόσουμε την προκαθορισμένη αρχιτεκτονική στο task ταξινόμησης που επιτελούμε. Επιπρόσθετα, χρησιμοποιούμε τις τεχνικές data augmentation και early stopping οι οποίες αποδείχθηκαν ιδιαίτερα χρήσιμες στις προηγούμενες ενότητες. Σχετικά με την εισάγωγή μοντέλου με μεταφορά μάθησης αναφέρουμε ότι στα πρώτα στάδια της διερεύνησης μας επιλέγουμε να συνεχίσουμε να εκπαιδεύουμε όλα τα επίπεδα του δικτύου (trainable = True).
Ξεκινήσαμε την μελέτη μας διατηρώντας σταθερές τις τιμές των BATCH_SIZE (128) και Optimizer (Adam). Ακολούθως, θα προσπαθήσουμε να βελτιστοποιήσουμε κατά το δυνατόν το μοντέλο για την περίπτωση των 20 κλάσεων, ενώ στη συνέχεια θα παρουσιαστούν οι επιδόσεις για το σύνολο των 80 κλάσεων.

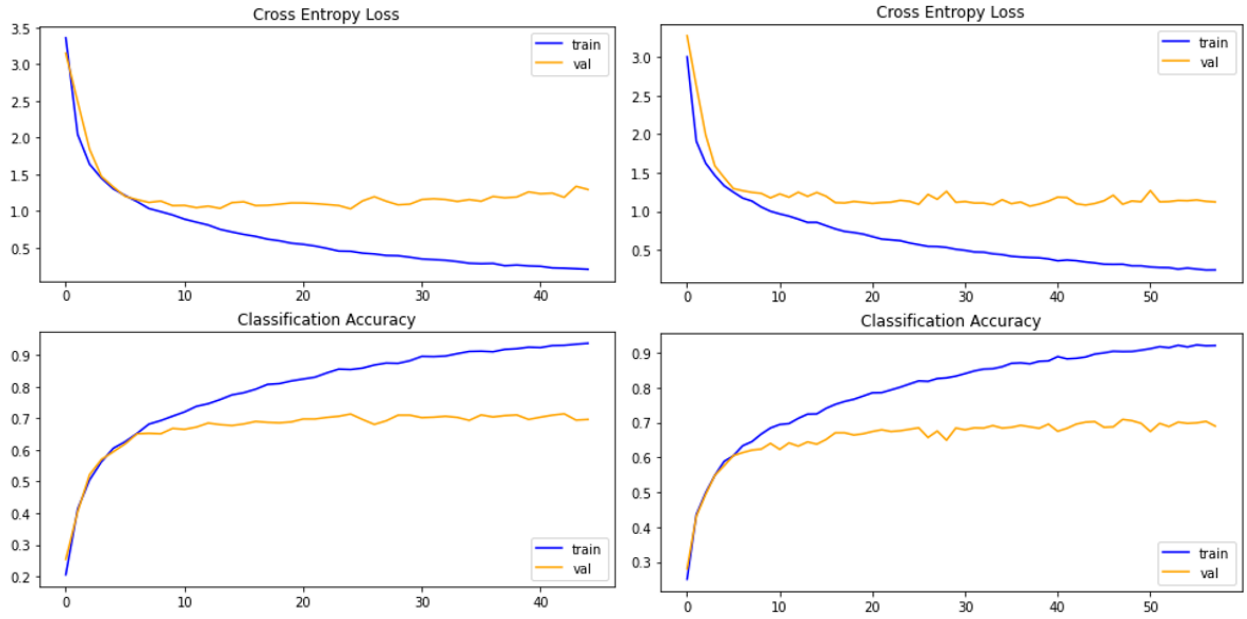
5.1 Βελτιστοποίηση

Σε μια πρώτη προσέγγιση προσπαθήσαμε να μελετήσουμε τον τρόπο με τον οποίο επηρεάζεται η συμπεριφορά του μοντέλου τόσο από το rate του Dropout Layer, όσο και από το learning rate του Adam Optimizer. Διατηρώντας σταθερό learning rate = 0.00001 και για διάφορες τιμές του dropout rate παρατηρήσαμε αρκετά ικανοποιητικά αποτελέσματα, με το accuracy να λαμβάνει τιμές 0.73 και το loss 0.92. Ακολούθως, στα πλαίσια διερεύνησης της απόδοσης δοκιμάσαμε διαφορετικούς συνδυασμούς ανάμεσα στις μετρικές που αναφέρονται παραπάνω, καταλήγοντας στις τιμές learning rate = 0.0001 και dropout layer = 0.5, οι οποίες επιφέρουν τα καλύτερα αποτελέσματα. Όπως παρουσιάζεται στο σχήμα που ακολουθεί, οι μετρικές λαμβάνουν τιμές **accuracy = 0.75** και **loss = 0.85**.



Σχήμα 9: Loss - Accuracy: DenseNet, 20 κλάσεις

Ως επόμενο βήμα στην βελτιστοποίηση του μοντέλου, εξετάσαμε τη χρήση διαφορετικού optimizer. Συγκεκριμένα, λάβαμε τις απαραίτητες μετρήσεις για τις περιπτώσεις των Stochastic Gradient Descent (SGD) και Root Mean Square Propagation (RMSprop) χρησιμοποιώντας διαφορετικές τάξεις μεγέθους της παραμέτρου learning rate. Εξετάζοντας και πάλι την επίδραση του dropout rate στους optimizers που αναφέρονται, επιλέγουμε τον συνδυασμό learning rate και dropout rate ο οποίος παρουσιάζει την μεγαλύτερη ακρίβεια σε κάθε περίπτωση. Ωστόσο, τονίζουμε ότι κατά τη διαδικασία αναζήτησης της βέλτιστης λύσης δεν παρουσιάστηκε σημαντική απόκλιση ανάμεσα στις διαφορετικές εκτελέσεις. Προκειμένου να καταλήξουμε στην βέλτιστη επιλογή λαμβάνουμε υπόψη μας και τα αποτελέσματα των απωλειών, validation loss.
Πιο συγκεκριμένα, στο στάδιο αυτό παρατηρούμε πως και οι 3 optimizers παρουσιάζουν παρόμοια, ικανοποιητική συμπεριφορά. Ωστόσο, ο RMSprop, αν και προσεγγίζει σημαντικά σε ακρίβεια τα αποτελέσματα του Adam επιτυγχάνοντας $acc = 0.74$, εμφανίζει αρκετά μεγαλύτερα ποσοστά απωλειών $loss = 0.92$. Ο optimizer SGD ακολουθεί με χαμηλότερη επίδοση $acc = 0.72$ και ακόμα μεγαλύτερες απώλειες $loss = 0.98$. Για καλύτερη εποπτεία παραθέτουμε συγκεντρωτικά τα αποτελέσματα και τις γραφικές τους παραστάσεις.

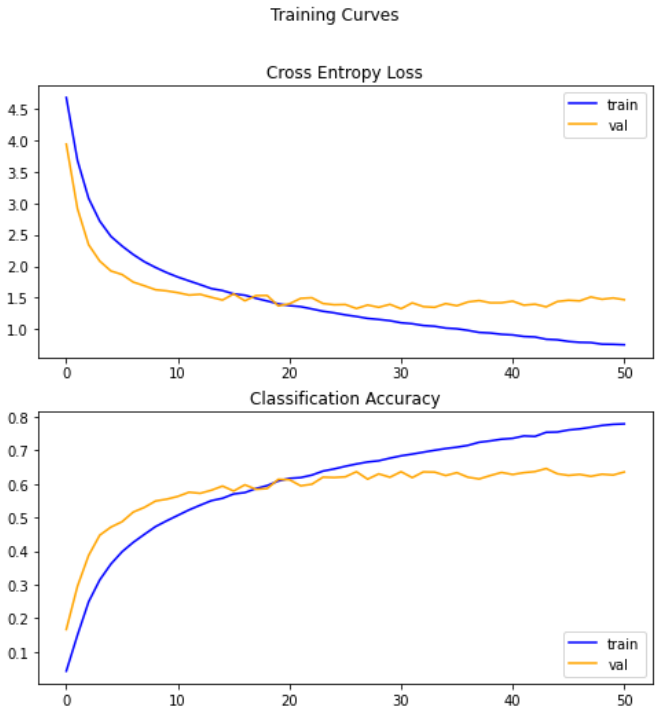


Σχήμα 10: Loss - Accuracy: RMSprop, SGD

Optimizer	Learning Rate	Dropout rate	Loss	Accuracy
Adam	0.0001	0.5	0.85	0.75
RMSprop	0.0001	0.5	0.92	0.74
SGD	0.01	0.5	0.98	0.72

Πίνακας 7: Optimizer - Learning Rate - Loss - Accuracy, 20 κλάσεις

Στο σημείο αυτό μπορούμε να επιλέξουμε τον optimizer Adam ως βέλτιστη λύση χρησιμοποιώντας τους κατάλληλους συνδυασμούς τιμών. Για λόγους πληρότητας οφείλουμε να εξετάσουμε και την πρακτική χρήσης της συνελικτικής βάσης για την εξαγωγή χαρακτηριστικών διατηρώντας την σημαία trainable = False στο DenseNet. Τα αποτελέσματα της συγκεκριμένης προσέγγισης είναι μάλλον αποθαρρυντικά καθώς, ακόμα και στην περίπτωση των 20 κλάσεων, όχι μόνο εμφανίζεται χαμηλότερη ακρίβεια $acc = 0.52$, αλλά ταυτόχρονα αυξάνονται σημαντικά οι απώλειες $loss = 1.63$. Για τους λόγους αυτούς, διατηρούμε την αρχική μας στρατηγική που παρουσιάστηκε σε προηγούμενο διάγραμμα.
Καταλήγοντας, εκτελούμε το μοντέλο DenseNet στο σύνολο των 80 κλάσεων προκειμένου να αξιολογήσουμε την απόδοση του. Χρησιμοποιώντας Adam Optimizer με learning rate 0.0001 και Dense rate 0.5 επιτυγχάνεται ακρίβεια $acc = 0.64$ και απώλειες $loss = 1.33$. Αν και οι μετρικές επίδοσης χαρακτηρίζονται από μικρότερες τιμές σε σχέση με την περίπτωση των 20 κλάσεων, τα validation accuracy και validation loss εξακολουθούν να είναι ικανοποιητικά. Τα αποτελέσματα παρατίθενται αναλυτικότερα στα διαγράμματα που ακολουθούν.

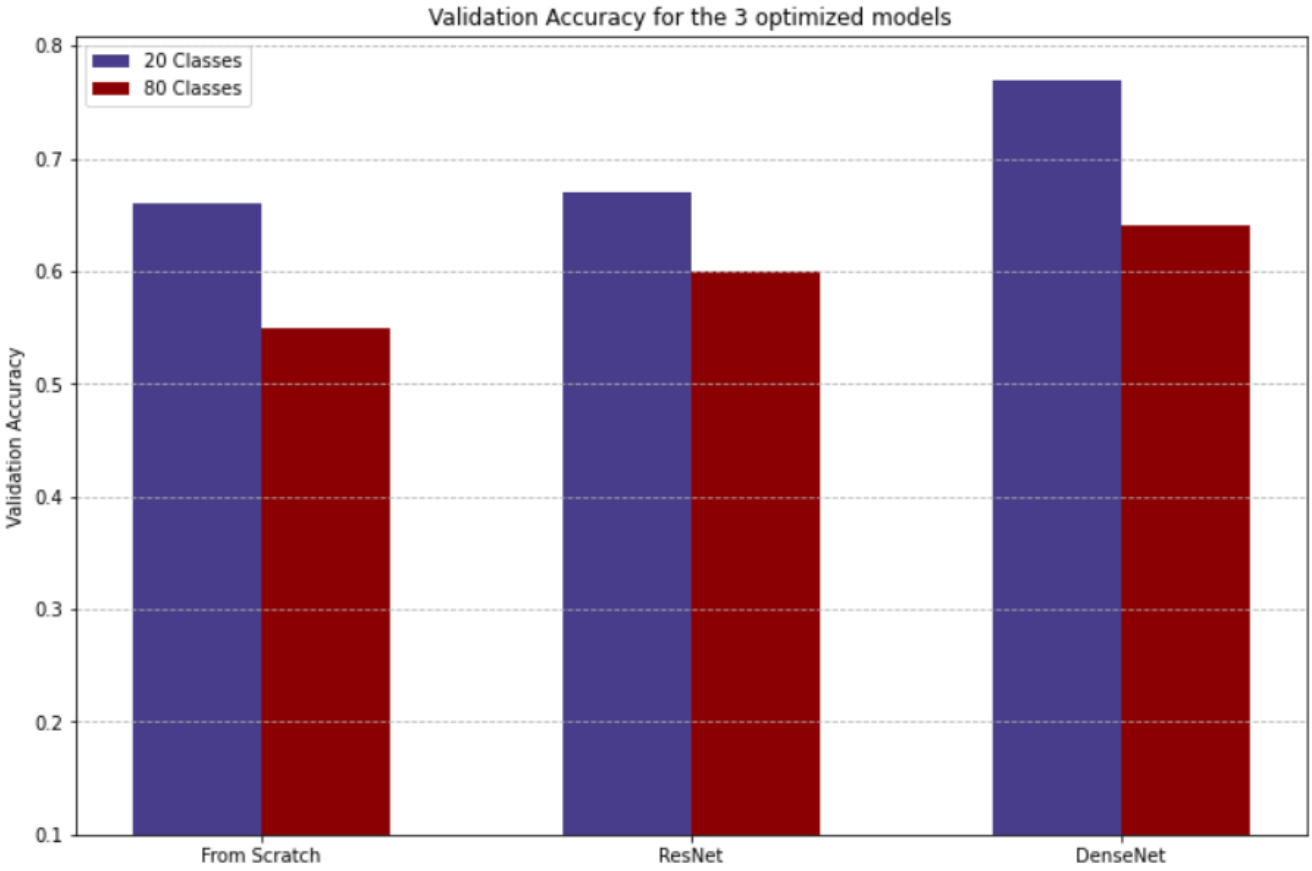


Σχήμα 11: Adam - 80 κλάσεις

6 Σύγκριση μοντέλων

Στην παρούσα εργαστηριακή άσκηση κληθήκαμε να δοκιμάσουμε διαφορετικά μοντέλα συνελικτικών δικτύων, προκειμένου να επιλύσουμε το πρόβλημα της ταξινόμησης στο σύνολο δεδομένων CIFAR-100, και να επιχειρήσουμε να τα βελτιστοποιήσουμε με στόχο την αύξηση της ακρίβειας πρόβλεψης. Ξεκινώντας από ένα εξαιρετικά απλό μοντέλο, ακολουθήσαμε μια διαδικασία εμπλουτισμού του, ώστε να μειώσουμε τις απώλειες, να περιορίσουμε τις επιπτώσεις της υπερεκπαίδευσης και να καταλήξουμε, τελικά, σε μια ικανοποιητική ακρίβεια πρόβλεψης για το πρόβλημα των 80 κλάσεων. Η διαδικασία αυτή, του σχεδιασμού ενός μοντέλου- ακόμα και σχετικά απλού- "from scratch" αποδείχτηκε δύσκολη και, αν μη τι άλλο, εξαιρετικά χρονοβόρα.

Από την άλλη, είδαμε ότι με τη χρήση έτοιμων αρχιτεκτονικών ή ακόμα και προεκπαιδευμένων μοντέλων με την τεχνική της μεταφοράς μάθησης, μπορούμε να ελαττώσουμε το χρόνο σχεδιασμού, εστιάζοντας στο συγκεκριμένο πρόβλημα και στις ιδιαιτερότητές του, πετυχαίνοντας, ταυτόχρονα, αποτελέσματα τουλάχιστον εφάμιλλα (ResNet) ή ακόμα και σαφώς ανώτερα (DenseNet) από την custom προσέγγιση. Παρόλο που η απευθείας σύγκριση της ακρίβειας τόσο διαφορετικών μοντέλων δεν έχει ουσιαστικό νόημα, αποφασίσαμε να παρουσιάσουμε τα συγκριτικά αποτελέσματα του validation accuracy σε ένα γράφημα για λόγους πληρότητας.



Σχήμα 12: Validation Accuracy των τριών μοντέλων για 20 και 80 κλάσεις

Επιπλέον της αναφοράς, ο φάκελος της εργασίας περιλαμβάνει το notebook, όπου παρουσιάζονται τα αποτελέσματα των παραπάνω βελτιστοποιημένων δικτύων για 80 κλάσεις.