



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΤΠΟΛΟΓΙΣΤΩΝ

Νευρωνικά Δίκτυα και Ευφυή¹ Υπολογιστικά Συστήματα

3η Εργαστηριακή Άσκηση

Αχλάτης Στέφανος-Σταμάτης (03116149)

<el16149@mail.ntua.gr>

Ηλιακοπούλου Νικολέτα-Μαρκέλα (03116111)

<el16111@mail.ntua.gr>

Σταυροπούλου Γεωργία (03116162)

<el16162@mail.ntua.gr>

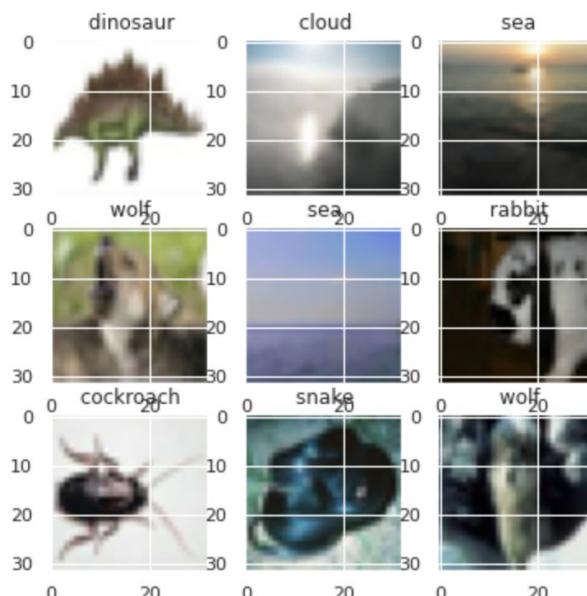
Φεβρουάριος 2021

Εισαγωγή - Μελέτη Dataset

Στόχος της δεύτερης εργαστηριακής Άσκησης είναι να βελτιστοποίηση της απόδοση μοντέλων Βαθιάς Μάθησης στο σύνολο δεδομένων CIFAR-100 χρησιμοποιώντας το TensorFlow. Το CIFAR-100 είναι ένα dataset που περιέχει εικόνες μεγέθους 32x32 pixels από 100 κλάσεις αντικειμένων και η κάθε κλάση έχει 600 εικόνες, εκ των οποίων οι 500 εικόνες είναι για το training του νευρωνικού και οι 100 για το testing του νευρωνικού. Αξίζει να σημειωθεί ότι αυτές οι 100 κλάσεις χωρίζονται σε 20 υποκλάσεις και έτσι κάθε εικόνα έρχεται με ένα επισημείωμα τόσο για το ποια κλάση ανήκει όσο και για ποια υποκλάση ανηκει. Παρακάτω φαίνονται αναλυτικά οι κλάσεις και οι υποκλάσεις των εικόνων του CIFAR-100:

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Ωστόσο στη παρούσα εργασία δουλέψαμε μόνο με 20 εκ των κλάσεων και άρα με πολύ μικρότερο dataset. Αυτές οι 20 κλάσεις ορίζονται με βάση τον αριθμό της ομάδας μας και πιο συγκεκριμένα για την ομάδα 80. Έτσι η παρούσα αναφορά μελετά τις εξείς 20 κλάσεις αντικειμένων από το CIFAR-100: 'beaver', 'can', 'cloud', 'cockroach', 'dinosaur', 'dolphin', 'forest', 'house', 'lamp', 'lawn_mower', 'man', 'motorcycle', 'mouse', 'pickup_truck', 'rabbit', 'ray', 'sea', 'snake', 'sweet_pepper', 'wolf'

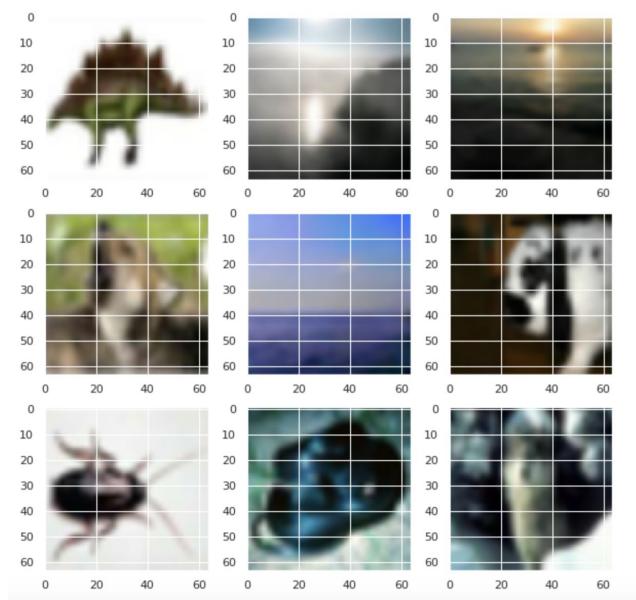


Όπως είχαμε δει και στο πρώτο εργαστήριο οι κλάσεις στο dataset που μελετάμε πρέπει να είναι ισορροπημένες δηλαδή να έχουμε περίπου ίσο πλήθος δεδομένων για κάθε κλάση. Αυτό συμβαίνει έτσι ώστε να μην εισάγουμε bias στην εκπαίδευση και ο ταξινομητής να έχει κάνει overfit ως προς τις πιο διαδεδομένες κατηγορείς στο dataset και να μην έχει μάθει τις πιο εξεζητημένες. Αν δεν ήταν ισορροπημένο θα έπρεπε να κάνουμε oversampling ή undersampling για να ισορροπήσουμε το τελικό dataset που θα εφαρμόσουμε για εκπαίδευση.

Από την ακόλουθη γραφική βλέπουμε ότι τα δεδομένα μας είναι εξισορροπημένα και έχουμε περίπου 500 δείγματα για κάθε κλάση.



Στην συνέχεια της παρούσας εργασίας θα δούμε ότι αυτά τα δεδομένα των εικόνων μπορούμε να τα τροποποιήσουμε, δηλαδή μπορούμε να περιστρεψουμε τις εικόνες, να τις καταμήσουμε να τις τμηματίσουμε να τις μεγαλώσουμε σε διάσταση κλπ. Έτσι παρακάτω μπορείτε να δείτε τις ακριβώς ίδιες εικόνες που παρουσιάστηκαν πρίν σε διπλάσια διάσταση:



Τέλος η εργασία έγινε εξ ολοκλήρου σε TensorFlow. Ωστόσο για λόγους συγκριτικής μελέτης των δυο πιο διαδεδομένων frameworks τμήμα της εργασίας έγινε και σε PyTorch με μελέτη διαφορετικού dataset.

Η εργασία έγινε σε **Google Colab**.

Προσοχή: Στο notebook υπάρχουν μόνο τα χαρακτηριστικά πειράματα και όχι όλο το πλήθος πειραμάτων που κάναμε. Τα μοντέλα και όλο το πλήθος πειραμάτων δεν “χωράνε” στα 16gb του eclass, αν επιθυμείτε να τα λάβετε παρακαλώ επικοινωνήστε στα emails μας.

Δίκτυα from Scratch

Αρχικά παρουσιάζονται η αρχιτεκτονική των δικτύων που κατασκευάσαμε και μελετάμε.

Δίκτυο 1ο:

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_22 (MaxPooling)	(None, 15, 15, 32)	0
batch_normalization_7 (Batch Normalization)	(None, 15, 15, 32)	96
conv2d_27 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_23 (MaxPooling)	(None, 6, 6, 64)	0
conv2d_28 (Conv2D)	(None, 4, 4, 128)	73856
max_pooling2d_24 (MaxPooling)	(None, 2, 2, 128)	0
flatten_8 (Flatten)	(None, 512)	0
dense_23 (Dense)	(None, 256)	131328
dense_24 (Dense)	(None, 64)	16448
dense_25 (Dense)	(None, 100)	6500

Δεύτερο Δίκτυο:

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 30, 30, 64)	1792
max_pooling2d_25 (MaxPooling)	(None, 15, 15, 64)	0
batch_normalization_8 (Batch Normalization)	(None, 15, 15, 64)	192
conv2d_30 (Conv2D)	(None, 14, 14, 128)	32896
max_pooling2d_26 (MaxPooling)	(None, 7, 7, 128)	0
dropout_7 (Dropout)	(None, 7, 7, 128)	0
conv2d_31 (Conv2D)	(None, 6, 6, 256)	131328
max_pooling2d_27 (MaxPooling)	(None, 3, 3, 256)	0
dropout_8 (Dropout)	(None, 3, 3, 256)	0
conv2d_32 (Conv2D)	(None, 2, 2, 512)	524800
max_pooling2d_28 (MaxPooling)	(None, 1, 1, 512)	0
dropout_9 (Dropout)	(None, 1, 1, 512)	0
flatten_9 (Flatten)	(None, 512)	0
dense_26 (Dense)	(None, 1024)	525312
dense_27 (Dense)	(None, 512)	524800
dense_28 (Dense)	(None, 64)	32832
dense_29 (Dense)	(None, 100)	6500

Τρίτο Δίκτυο:

Layer (type)	Output Shape	Param #
conv2d_33 (Conv2D)	(None, 30, 30, 128)	3584
max_pooling2d_29 (MaxPooling)	(None, 15, 15, 128)	0
dropout_10 (Dropout)	(None, 15, 15, 128)	0
batch_normalization_9 (Batch Normalization)	(None, 15, 15, 128)	384
conv2d_34 (Conv2D)	(None, 14, 14, 512)	262656
max_pooling2d_30 (MaxPooling)	(None, 7, 7, 512)	0
dropout_11 (Dropout)	(None, 7, 7, 512)	0
batch_normalization_10 (Batch Normalization)	(None, 7, 7, 512)	1536
conv2d_35 (Conv2D)	(None, 6, 6, 1024)	2098176
max_pooling2d_31 (MaxPooling)	(None, 3, 3, 1024)	0
dropout_12 (Dropout)	(None, 3, 3, 1024)	0
batch_normalization_11 (Batch Normalization)	(None, 3, 3, 1024)	3072
conv2d_36 (Conv2D)	(None, 2, 2, 512)	2097664
max_pooling2d_32 (MaxPooling)	(None, 1, 1, 512)	0
dropout_13 (Dropout)	(None, 1, 1, 512)	0
batch_normalization_12 (Batch Normalization)	(None, 1, 1, 512)	1536
flatten_10 (Flatten)	(None, 512)	0
dense_30 (Dense)	(None, 2048)	1050624
dense_31 (Dense)	(None, 512)	1049088
dense_32 (Dense)	(None, 1024)	525312
dense_33 (Dense)	(None, 512)	524800
dense_34 (Dense)	(None, 100)	51300

Τέταρτο Δίκτυο:

Layer (type)	Output Shape	Param #
conv2d_37 (Conv2D)	(None, 28, 28, 32)	2432
conv2d_38 (Conv2D)	(None, 24, 24, 64)	51264
max_pooling2d_33 (MaxPooling)	(None, 12, 12, 64)	0
dropout_14 (Dropout)	(None, 12, 12, 64)	0
conv2d_39 (Conv2D)	(None, 10, 10, 64)	36928
conv2d_40 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_34 (MaxPooling)	(None, 4, 4, 64)	0
dropout_15 (Dropout)	(None, 4, 4, 64)	0
flatten_11 (Flatten)	(None, 1024)	0
dense_35 (Dense)	(None, 1024)	1049600
dropout_16 (Dropout)	(None, 1024)	0
dense_36 (Dense)	(None, 512)	524800
dropout_17 (Dropout)	(None, 512)	0
dense_37 (Dense)	(None, 100)	51300

Στην συνέχεια έπρεπε να πειραματιστούμε με τις παραμέτρους των δεικτών έτσι ώστε να καταλήξουμε στην επιλογή των βέλτιστων παραμέτρων για κάθε δίκτυο.

Το πόσο καλά πάει η εκπαίδευση σε ένα δίκτυο το βλέπουμε από το validation loss και το validation

accuracy, πιο αναλυτικά θέλουμε να έχουμε το καλύτερο δυνατό validation accuracy με το χαμηλότερο δυνατό validation loss. Το validation accuracy δείχνει το κατά πόσο το νευρωνικό

βρίσκει την σωστή απάντηση ενώ το validation loss δείχνει το πόσο σίγουρο είναι για την απάντηση

που έδωσε. Όταν βλέπουμε πως το validation loss αυξάνεται είμαστε σε περιοχή overfitting ενώ όταν

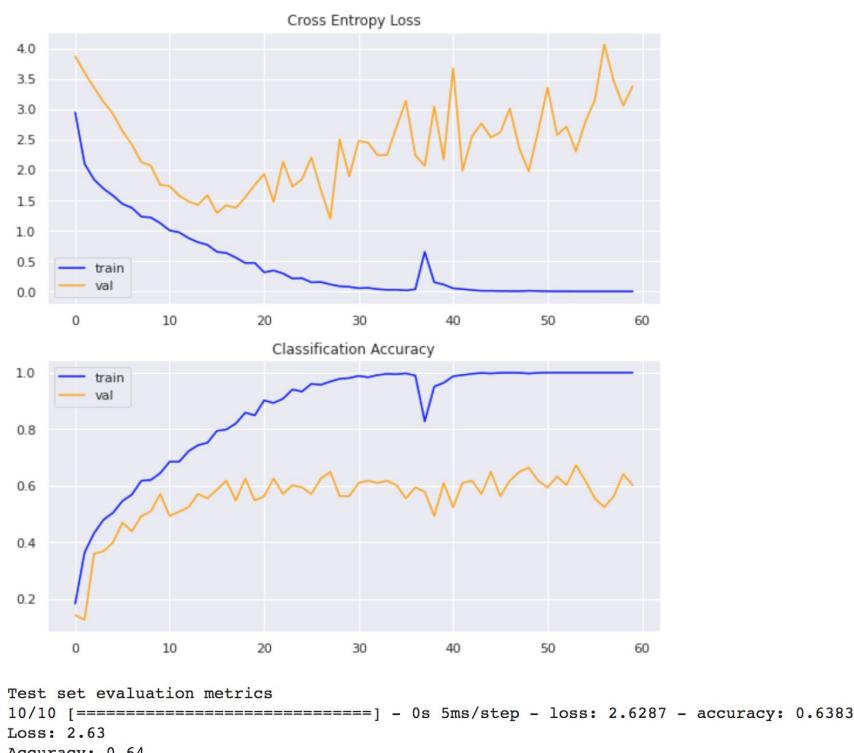
μειωθεί και το validation accuracy τότε έχουμε σίγουρα overfit.

Μερικοί από τους παραμέτρους που πειραματιστήκαμε είναι οι εξείς:

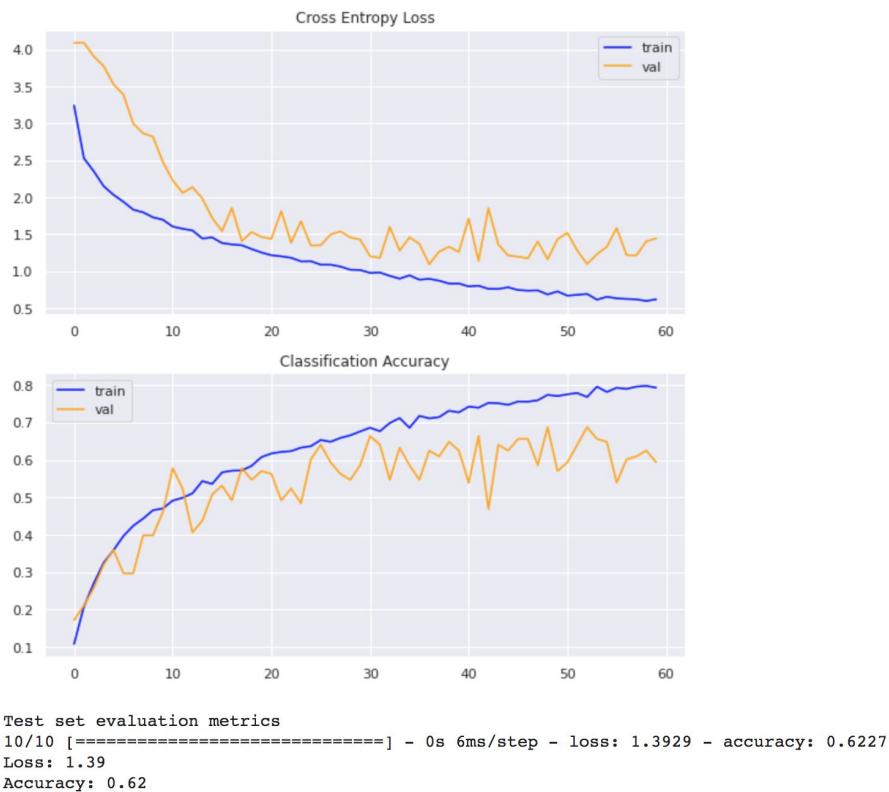
- Batch size, το οποίο μετά από πειραματισμό επιλέξαμε να το ορίσουμε ίσο με 128.
- Optimizers και οι παράμετροι τους, ορισμένοι εκ των όσον χρησιμοποιήσαμε είναι οι εξείς: Adam, AdaBound, SGD, Nadam, Adamax.
- Criterion και οι παράμετροι τους, ορισμένοι εκ των όσον χρησιμοποιήσαμε είναι οι εξείς: CrossEntropyLoss, MultiMarginLoss. Όπου είδαμε ότι το CrossEntropyLoss ήταν το καλύτερο κριτήριο.
- Epochs δηλαδή εποχές εκπαίδευσης του μοντέλου, όπου παρουσιάζεται το εξείς trade off με λίγες εποχές το μοντέλο δεν έχει εκπαιδευτεί επαρκός ενώ με πολλές εποχές το μοντέλο κάνει overfitting στα δεδομένα εκπαίδευσης.
- Metrics, όπου καταλήξαμε στο συμπέρασμα ότι για το εν λόγω πρόβλημα το accuracy έδινε το προσδοκώμενο αποτέλεσμα. Άλλες μετρικές που μελετήσαμε είναι το Precision και το Recall με handwritten κώδικα όπως φαίνεται στο notebook.

Έτσι, παρουσιαζουμε τα καλύτερα μοντέλα για κάθε δίκτυο.

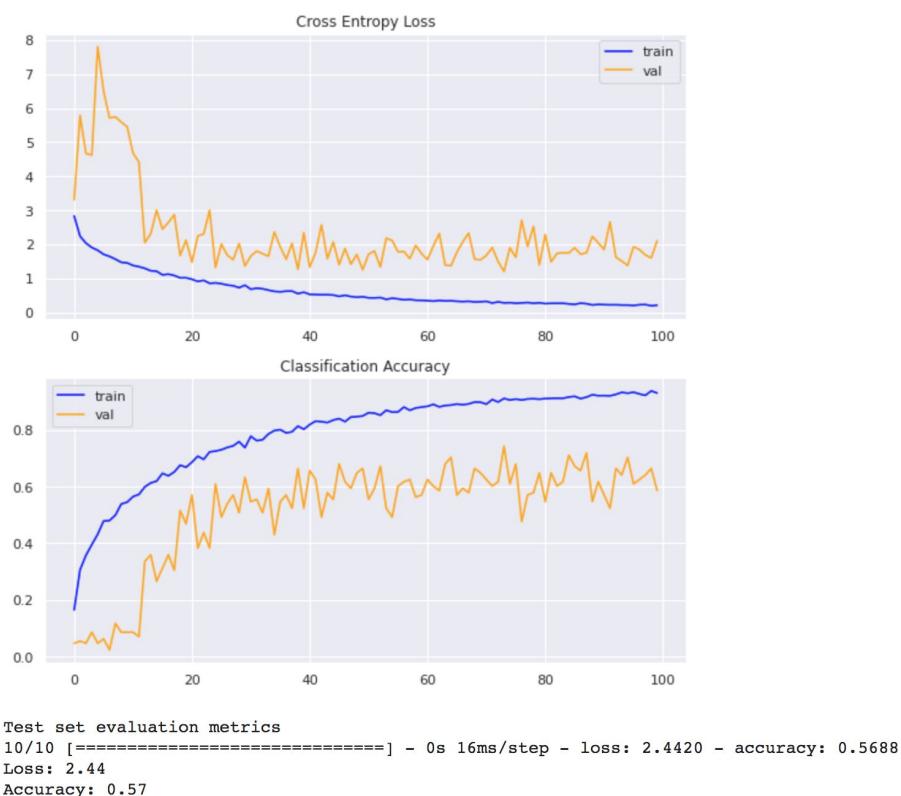
Για το Πρώτο Μοντέλο:



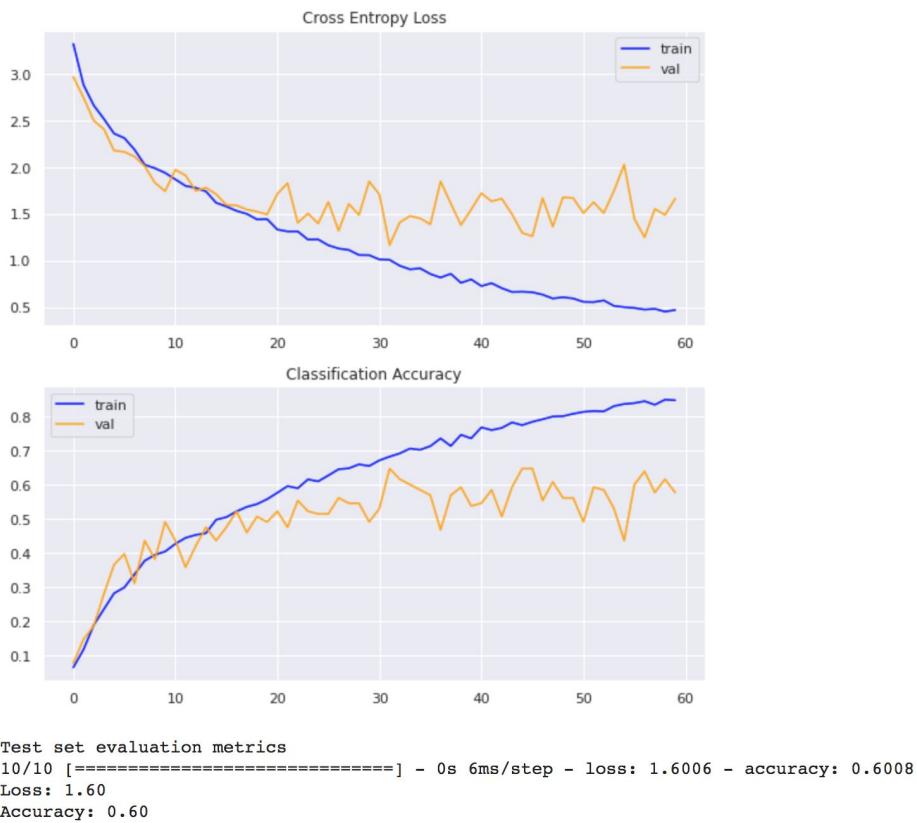
Για το Δεύτερο Μοντέλο:



Για το Τρίτο Μοντέλο:



Για το Τέταρτο Μοντέλο:



Οπότε γενικά μπορούμε να βγάλουμε τα εξείς συμπεράσματα:

- Ο AdaBound δουλεύει συγκριτικά καλύτερα σε CNN δίκτυα όπως τα τρία τελευταία ενώ ο Adam δουλεύει καλύτερα στα δύο πρώτα
- Όσο πιο βαθύ είναι ένα δίκτυο τόσες περισσότερες εποχές χρειάζεται να εκπαιδευτεί.
- To MultiMarginLoss loss δεν αποδίδει τόσο καλά αφού δίνει σχετικά χαμηλότερο accuracy ωστόσο δίνει αξιοθαυμαστά χαμηλό loss.
- Το καλύτερο μοντέλο γενικά είναι το καλύτερο μοντέλο του Πρώτου Δικτύου

Φυσικά, αν πειραματιστούμε με απλά νευρωνικά μοντέλα είναι αναμενόμενη η χαμηλή τους επιδόση αφού δεν είναι “εξειδικευμένα” για task που αφορούν επεξεργασία εικόνας.

Αντίθετα, βλεπουμε ότι τα CNN έχουν σαφώς καλύτερες επιδόσεις.

Στην Συνέχεια προχωράμε με μια αναλυτικότερη διερεύνηση των επιδόσεων του καλύτερου μοντέλου για διαφορετικές τιμές των υπερ παραμέτρων ως εξής:

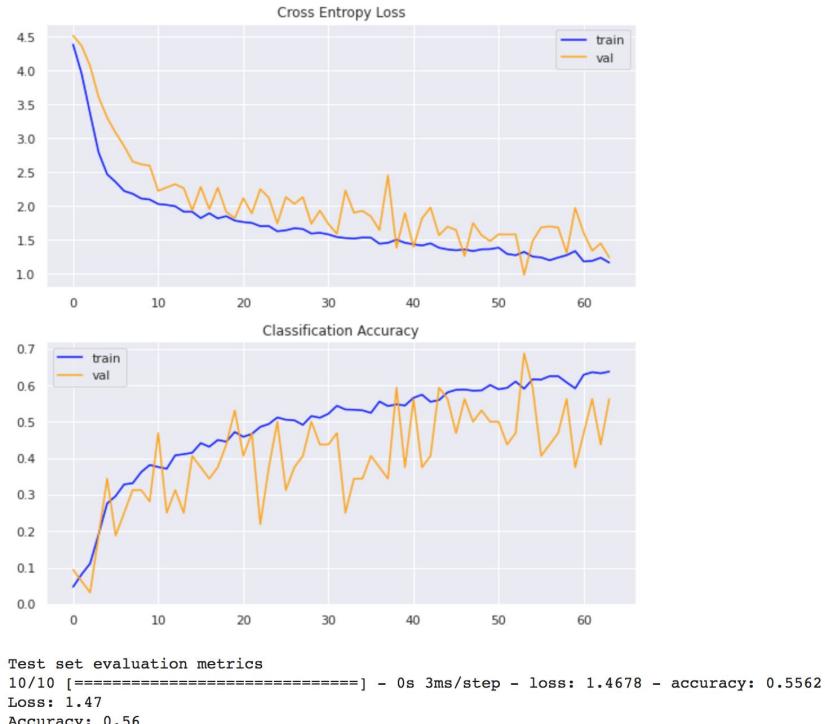
	Accuracy	Time		Accuracy	Time
Batch Size = 32	0.56	23.3 s		Learning Rate = 0.01	0.41
Batch Size = 64	0.47	13.8 s		Learning Rate = 0.001	0.56
Batch Size = 128	0.56	20.8 s		Learning Rate = 0.0001	0.54
Batch Size = 256	0.56	21.6 s		Learning Rate = 0.00001	0.35
optimizer = SGD	0.35	32 s		Accuracy	Time
optimizer = Adadelta	0.2	1min 2s		epochs = 50	0.61
optimizer = Adagrad	0.4	22.1 s		epochs = 100	0.62
optimizer = Nadam	0.6	16.2 s		epochs = 150	0.61
				epochs = 200	0.6
					1min 19s

Επίδραση του μεγέθους δέσμης (batch size)

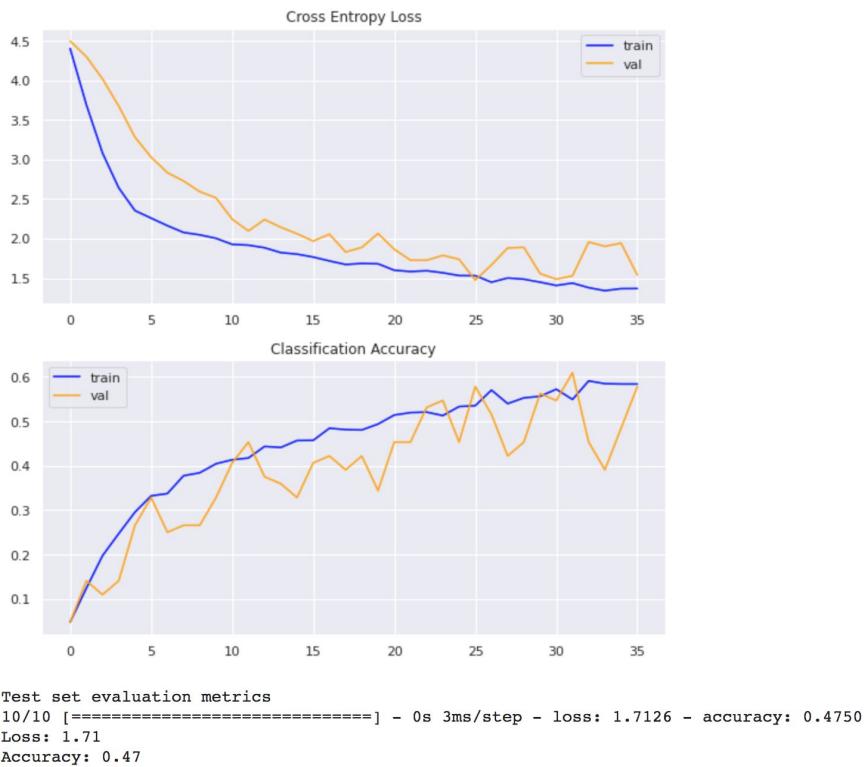
Ομαδοποιούμε τα δεδομένα μας σε Batches για την εκπαίδευση του νευρωνικού χρησιμοποιώντας παραλλαγές του Stochastic Gradient Descent (SGD) και συγκεκριμένα τον optimizer Adam.

Η επιλογή μεγέθους των mini-batches επηρεάζει σημαντικά την εκπαίδευση των μοντέλων μας. Συγκεκριμένα, κατά το SGD ισχύει ότι όσο μεγαλύτερο είναι το mini-batch τόσο πιο κοντά βρισκόμαστε στα gradients του training set και τόσο λιγότερο θόρυβο εισάγουμε στο μοντέλο μας. Γενικότερα, ωστόσο, φαίνεται πως αυξάνοντας το μέγεθος του batch (BATCH_SIZE) ο αλγόριθμος βρίσκει minimum που όμως είναι αρκετά "μυτερά" και όχι "ομαλά". Αυτό οδηγεί σε πολύ μεγάλη υποβάθμιση του μοντέλου μας έχοντας ως μονάδα μέτρησης το πόσο καλά κάνει generalize σε νεαδεδομένα (<https://arxiv.org/abs/1609.04836>). Ακόμα, μεγαλύτερο BATCH_SIZE οδηγεί σε περισσότερες ανάγκες σε πόρους με πολύ μικρή βελτίωση στην αβεβαιότητα του gradient ($O(\sqrt{\text{BATCH_SIZE}})$) (Ian Goodfellow). Συνεπώς αφήσαμε το BATCH_SIZE ως έχει σε 128 που είναι μια καλή default τιμή. Παρακάτω φαίνεται αυτό το πειραματικό αποτέλεσμα για batch size 32, 64, 128 και 256 στο καλύτερο μοντέλο του Δικτύου 2, που είναι το καλύτερο μοντέλο γενικότερα.

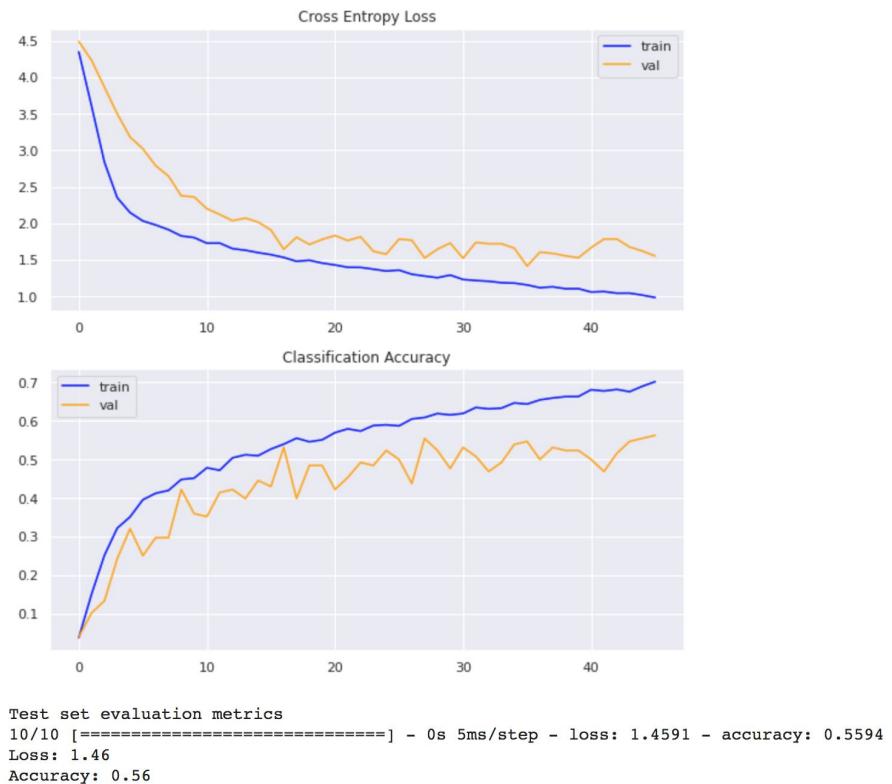
Για batch size = 32:



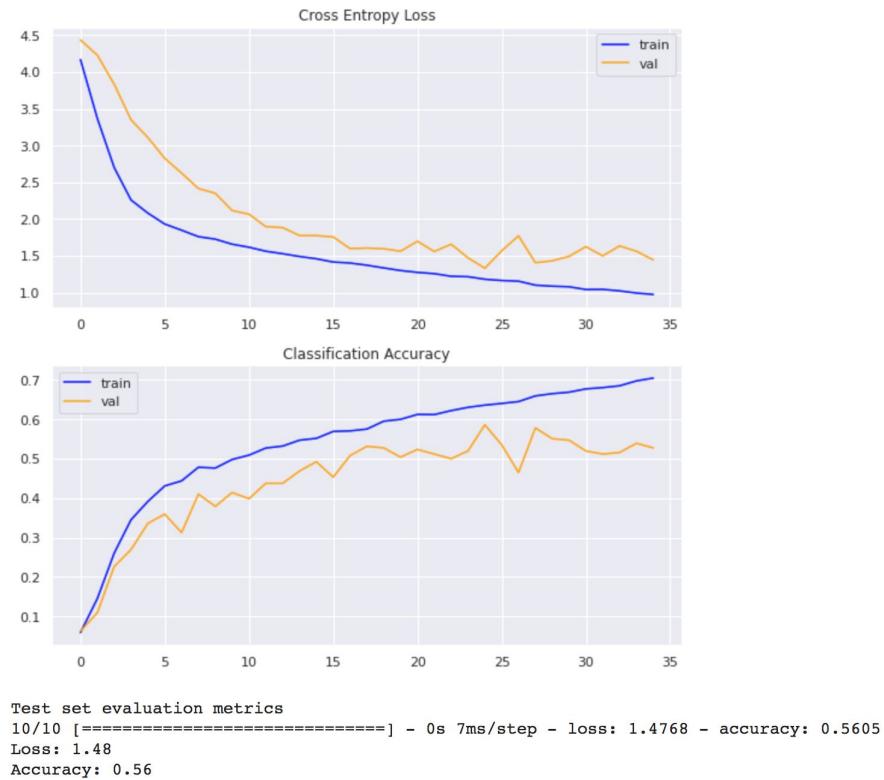
Για batch size = 64:



Για batch size = 128:



Για batch size = 256:



Επίδραση του αλγορίθμου βελτιστοποίησης (optimizer)

Ο αλγόριθμος βελτιστοποίησης παίζει καθοριστικό ρόλο. Είναι ο αλγόριθμος με βάση τον οποίο μεταβάλλονται τα χαρακτηριστικά του μοντέλου όπως τα βάρη των νευρώνων. Στην άσκηση αυτή μελετήσαμε 3 optimizers τους: Adam, AdaBound, Nadam, Adamax και τον κλασικότερο SGD. Σχετικά με τον adabound πρόκειται για μια “παραλλαγή” του Adam optimizer όπου οι ερευνητές που τον πρότειναν στηρίχθηκαν στην ιδέα πως το να υπάρχουν πολύ μεγάλα learning rates κατά την εκπαίδευση κάνουν κακό στο νευρωνικό οπότε τα κάνουν δυναμικά bound, εξού και adabound (adam+bound). Και όντως, βλέπουμε τα νευρωνικά μας να αποδίδουν καλύτερα με τη χρήση αυτού του optimizer από ότι με τον απλό Adam. Άλλα συνολικά βλέπουμε ότι τα νευρωνικά μας αποδίδουν καλύτερα με τον Nadam (<https://openreview.net/forum?id=HJfpZq1DM>).

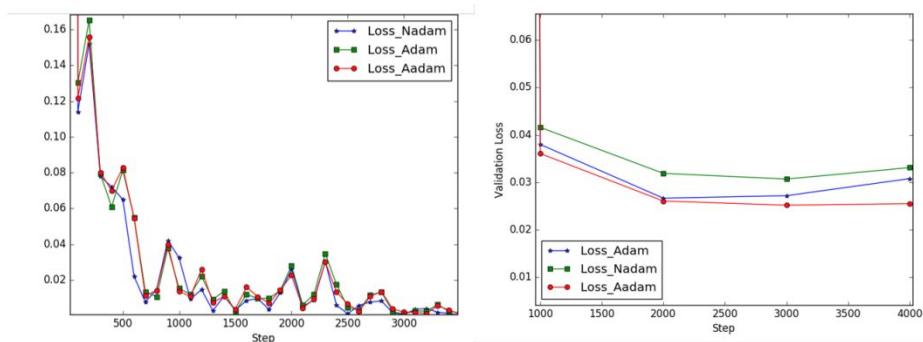
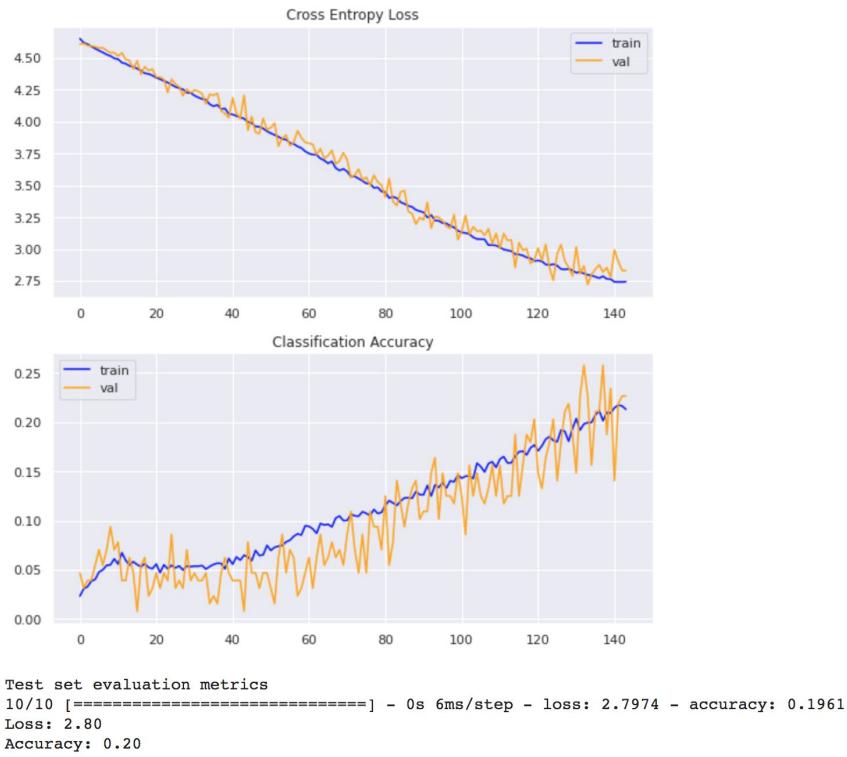
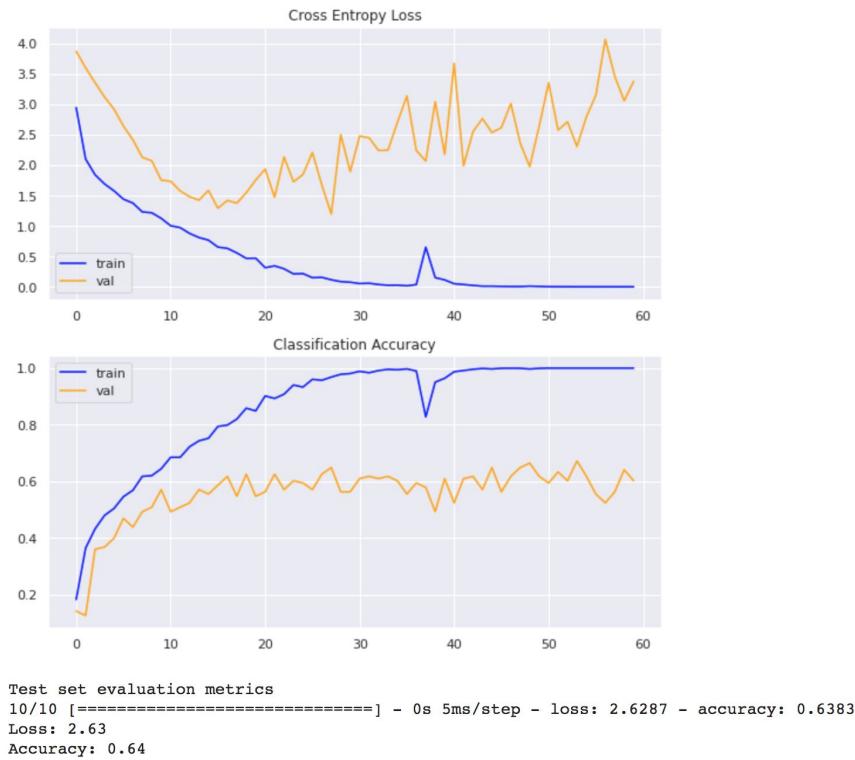


Figure 1: 1) The variation of the loss value in the training data. AAdam is between Adam and NAdam most of the time. 2) The variation of the loss value in the test data. AAdam outperforms Adam and NAdam with same settings. The validation data consist of 10000 images.

Αξίζει να σημειωθεί ότι ο Adadelta πέτυχε το χειρότερο score παρατηρώντας μια πταλινδρόμηση στην επίδοση πράμα που οδήγησε στο να έχει και την πιο χρονοβόρα εκπαίδευση περίπου 1 λεπτού:

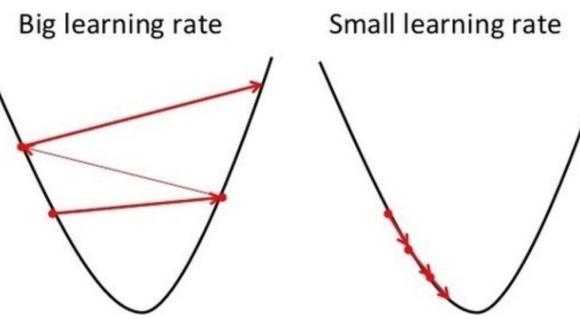


Ενώ ο καλύτερος optimizer είναι ο Nadam:

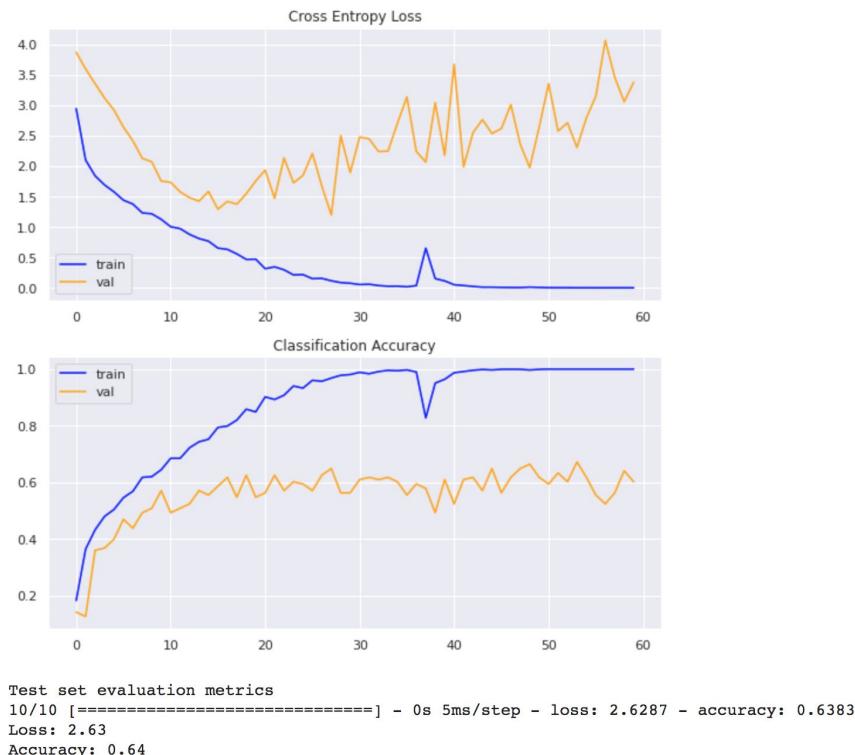


Επίδραση του Learning Rate

Το trade-off του ορισμού του ρυθμού μάθησης είναι γνωστό: Με μεγάλο ρυθμό μάθησης ο gradient descent τρέχει πιο γρήγορα και έτσι μπορεί να ξεπεράσει κάποιο ελάχιστο ενώ με μικρό ρυθμό μάθησης μπορεί να πηγαίνει πιο αργά και να μην προλάβει να εξερευνήσει όλον τον χώρο ικανοποιητικά ενώ προφανώς είναι πιο αργός. Η παρακάτω εικόνα είναι χαρακτηριστική αν και υπεραπλουστευμένη όπου δείχνει έναν απλό χώρο με ένα μόνο ελάχιστο.



Ένα καλό learning rate είναι το 0.001 όπου έχουμε την εξής διαδικασία μάθησης:



Επίδραση των Εποχών

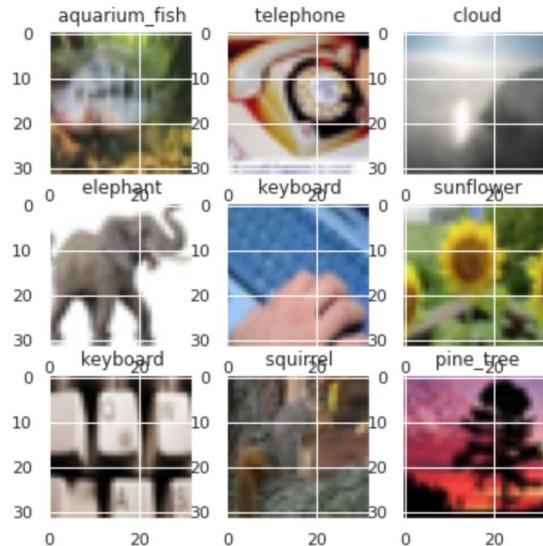
Το πρόβλημα με το πλήθος των εποχών, που θα αναλυθεί αναλυτικότερα στην επόμενη παράγραφο, εκφράζεται μέσω του overfitting. Με λίγες εποχές το μοντέλο δεν μαθαίνει από τα δεδομένα ενώ με πολλές εποχές μαθαίνει πολύ καλά τα δεδομένα και δεν έχει δυνατότητα γενίκευσης.

Παρατηρήσαμε ότι στα περισσότερα μοντέλα μας το ιδανικό πλήθος εποχών είναι 100-150

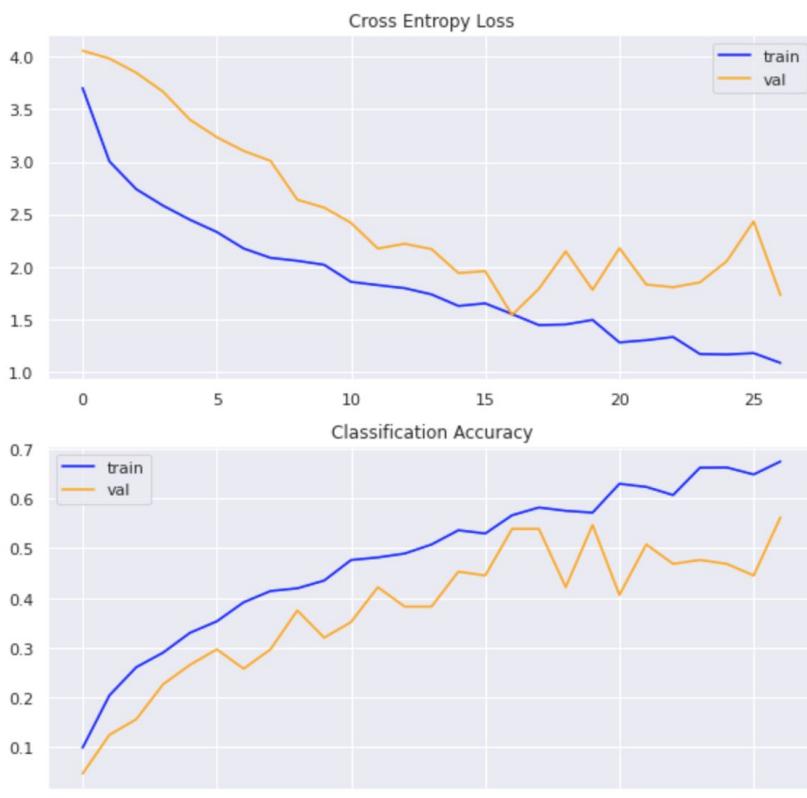
Επίδραση του πλήθους των δεδομένων/κλάσεων στην απόδοση του μοντέλου

Για την μελέτη του πλήθους των δεδομένων/κλάσεων στο μοντέλο μας επιλέγουμε κατάλληλο dataset με τη βοήθεια της συνάρτησης που μας δόθηκε και το εφαρμόζουμε στο καλύτερο μοντέλο του Πρώτου Μοντέλου, που είναι το καλύτερο μοντέλο γενικότερα.

Έτσι για 40 κλάσεις έχουμε:

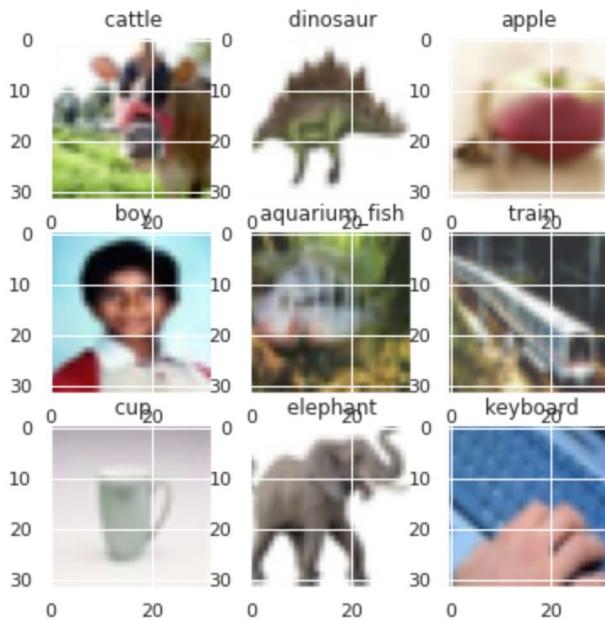


Με τα εξείς αποτελεσμα:

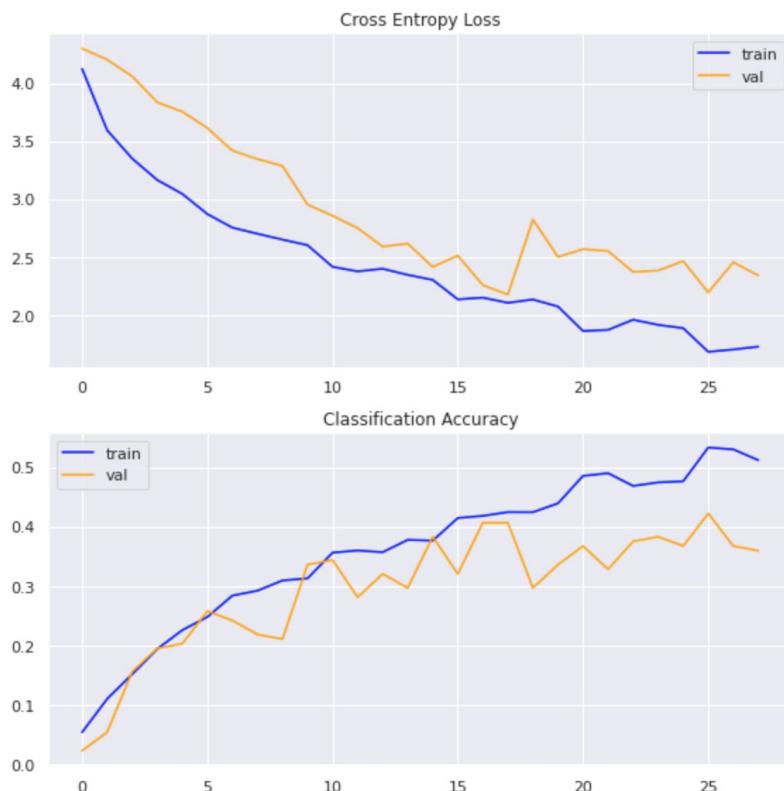


```
Test set evaluation metrics
10/10 [=====] - 0s 7ms/step - loss: 1.8522 - accuracy: 0.4844
Loss: 1.85
Accuracy: 0.48
```

Για 60 κλάσεις:

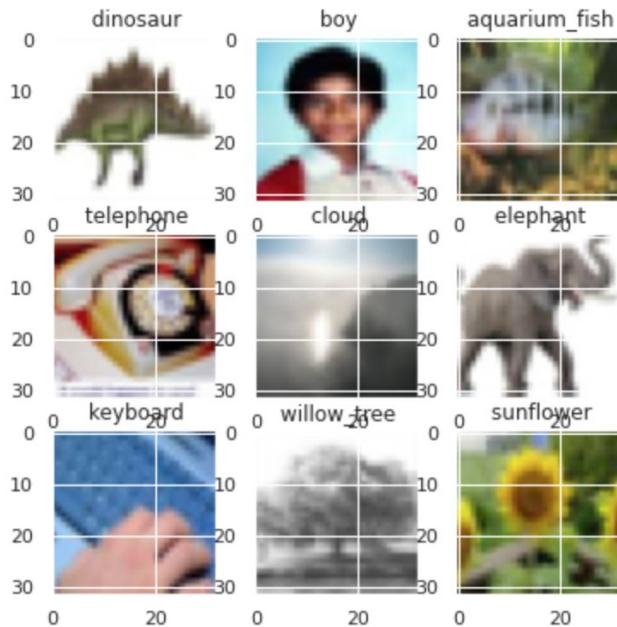


Με τα εξείς αποτελεσμα:

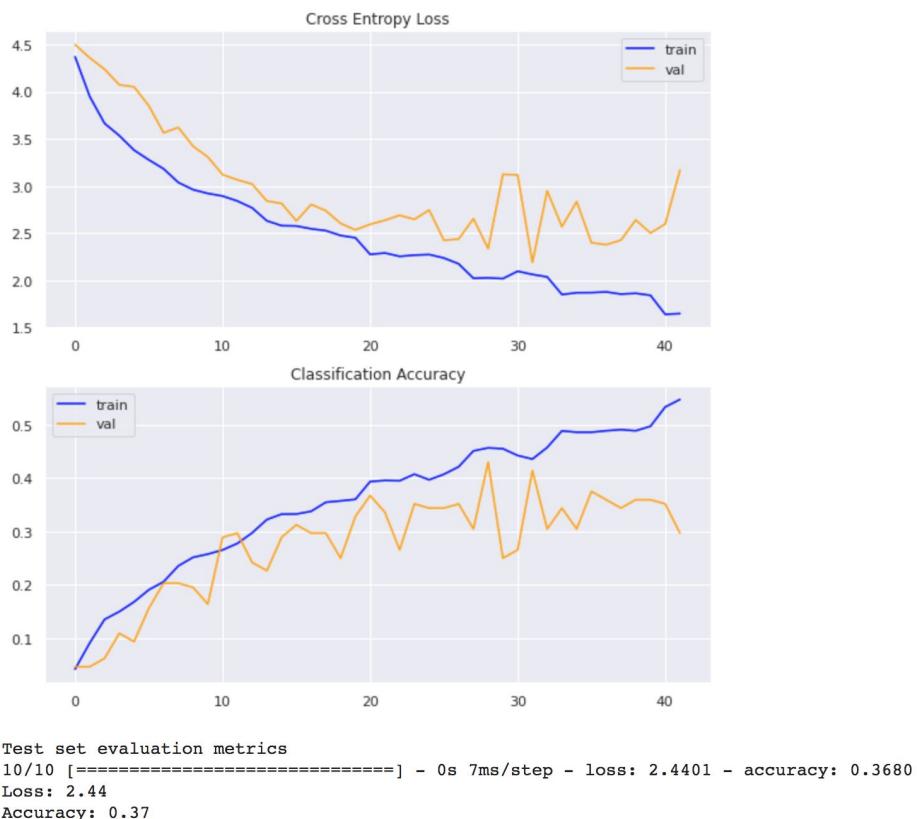


```
Test set evaluation metrics
10/10 [=====] - 0s 7ms/step - loss: 2.3634 - accuracy: 0.3570
Loss: 2.36
Accuracy: 0.36
```

Για 80 κλάσεις:



Με τα εξείς αποτελεσμα:



Έτσι όπως ήταν αναμενόμενο όσο αυξάνεται το πλήθος των κλάσεων, για τα δεδομένα σταθερά μοντέλα, έχουμε συνεχώς χειρότερη επίδοση στο δίκτυο, αφου αν και αυξάνεται ο συνολικός αριθμός των δειγμάτων, αυξάνεται και ο αριθμός των κλάσεων, και έτσι η πολυπλοκότητα του προβλήματος της ταξινόμησης. Θα χρειαζόμασταν πιο βαθιά νευρωνικά δίκτυα με πιο πολλά convolution layers για να πετύχουμε αξιόλογη ταξινόμηση στο μεγαλύτερο ποσοστό του CIFAR100.

Αντιμετώπιση του overfitting

Ένα συχνό πρόβλημα που συναντάται στην εκπαίδευση μοντέλων είναι η υπερεκπαίδευση (overfitting). Η υπερεκπαίδευση συμβαίνει όταν ένα μοντέλο εκπαιδεύεται σε υπερβολικό βαθμό στα δεδομένα εκπαίδευσης που έχει, καθιστώντας το ακατάλληλο να γενικευσει και να ταξινομήσει σωστά άλλα δεδομένα. Ο κίνδυνος της υπερεκπαίδευσης αυξάνεται όταν έχουμε πολλά όμοια δεδομένα καθώς και ένα νευρωνικό με πολύ μεγάλο capacity το οποίο αφήνουμε “ανεξέλεγχτα” να μάθει από τα train data.

Για τον έλεγχο υπερεκπαίδευσης των μοντέλων μελετάμε τις επιδόσεις τους. Συγκεκριμένα, ενώ τρέχει το μοντέλο, σε κάθε εποχή βλέπουμε το loss και το accuracy του μοντέλου στο validation set.

Ταυτόχρονα στο τέλος τυπώνουμε το accuracy του test set. Προφανώς στα πρώτα dataset θα είναι καλύτερα τα αποτελέσματα αφού εκπαιδεύτηκε σε αυτό, αλλά εάν η τελική διαφορά, είτε μεταξύ training και validation/test είναι πολύ μεγάλη, τότε υπάρχει πρόβλημα υπερεκπαίδευσης.

Ένας παράγοντας που επηρεάζει την υπερεκπαίδευση είναι ο αριθμός των εποχών (epochs). Όπως αναφέρθηκε και παραπάνω υπερβολικά μεγάλος αριθμός εποχών μπορεί να “υπερεκπαίδευσε” το μοντέλο στο training dataset και να μη του δίνει τη δυνατότητα για καλό generalization σε νέα δεδομένα. Στο προηγούμενο βήμα που επιλέξαμε αριθμό εποχών προσέξαμε να μην υπερβούμε το όριο και πέσει η απόδοση του μοντέλου.

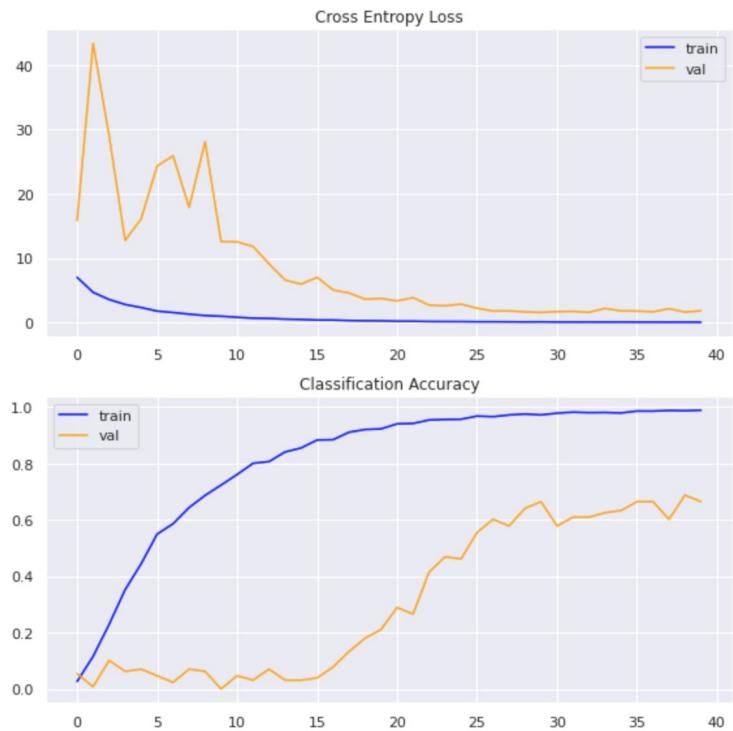
Παρακάτω θα δοκιμάσουμε τους εξεις τρόπους αντιμετώπισης του προβλήματος της υπερεκπαίδευσης:

- Πρόωρος τερματισμός (early stopping). Μια μέθοδος που τερματίζει την εκπαίδευση αν δεν υπάρχει βελτίωση ως προς τη μετρική απόδοσης που παρακολουθούμε.
- Dropout. Μια άλλη τεχνική για τη μείωση της υπερεκπαίδευσης είναι το Dropout. Το dropout απενεργοποιεί τυχαία νευρώνες στο δίκτυο αναγκάζοντάς το να μάθει καλύτερα “την ουσία” των δεδομένων και αποφεύγοντάς την υπερεκπαίδευση. Ουσιαστικά, μπορούμε να πούμε πως κατά μια έννοια προσθέτει θόρυβο στο δίκτυο που το βοηθάει να μάθει καλύτερα χωρίς να κάνει overfit.
- Επαύξηση δεδομένων. Η υπερεκπαίδευση συχνά συμβαίνει όταν έχουμε λίγα ή/και πολύ όμοια δεδομένα εκπαίδευσης ή όταν εκπαιδεύουμε το νευρωνικό μας στα ίδια δεδομένα για πολύ ώρα. Ένας τρόπος να διορθωθεί αυτό το πρόβλημα είναι να αυξήσουμε τα δεδομένα (data augmentation). Το data augmentation δημιουργεί νέα δεδομένα εκπαίδευσης με βάση τα υπάρχοντα εφαρμόζοντάς τυχαίους μετασχηματισμούς ώστε να προκύπτουν αληθοφανείς εικόνες (προσθέτει θόρυβο και βοηθάει το νευρωνικό να κάνει καλύτερο generalization).

Γενικά, σε όλα τα μοντέλα που θα δούμε και στην συνέχεια εφαρμόσαμε early stopping ορίζοντας μια αρκετά ανεκτική απόκλιση πτώσης του accuracy έτσι ώστε να μην είμαστε αυστηρή με το σύστημα, της τάξης του 10%. Ενώ σε όλα τα μοντέλα μας εφαρμόσαμε dropout, καθώς όπως είδαμε και σε αντίστοιχη μελέτη του Μαθήματος “Τεχνολογίες Εικόνας και Βίντεο” αυτές οι δύο τεχνικές βοηθούν πολύ στην ικανότητα γενικευσης του μοντέλου.

Επίσης γενικά το drop out δίνει spikes στις γραφικές της μάθησης.

Για την Επαύξηση δεδομένων βλέπουμε ότι έχει βελτιωθεί αρκετά η επίδοση του μοντέλου δηλαδή έχει αυξηθεί το accuracy και έχει μειωθεί το loss. Σαφώς επειδή αυξάνουμε το πλήθος των δεδομένων υπάρχει καθυστέρηση της εκπαίδευσης, και γενικά μπορούμε να πούμε ότι το μοντέλο εκπαιδεύεται σε θορυβώδη δεδομένα και για αυτό αντιμετωπίζει το overfitting, πράγμα που μπορούμε να δούμε και στην παρακάτω εικόνα από τα πειράματα:



Test set evaluation metrics

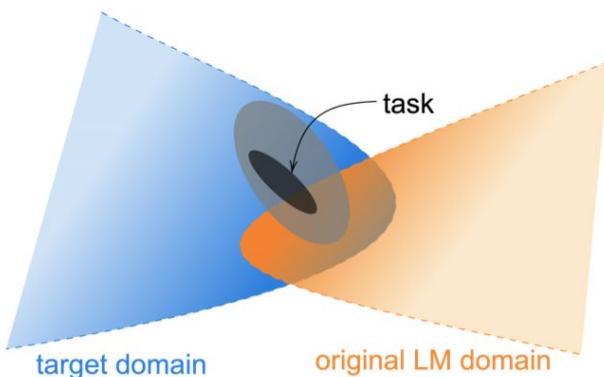
```
10/10 [=====] - 0s 28ms/step - loss: 1.7414 - accuracy: 0.6297
Loss: 1.74
Accuracy: 0.63
```

Μεταφορά Μάθησης

Η μεταφορά μάθησης από ένα είδη εκπαιδευμένο νευρωνικό δίκτυο σε κάποιο διαφορετικό dataset, στο δικό μας πρόβλημα. Γενικά για την μεταφορά μάθησης έχουμε χτίσει μοντέλα που έχουν εκπαιδευτεί σε πολλές εικόνες πχ VGG16, Xception, EfficientNetB7 κλπ ενώ υπάρχουν μοντέλα που έχουν εκπαιδευτεί πάνω σε πολλά κείμενα (natural language) πχ Bert, RoBERT κλπ. Γενικά στον χώρο της μεταφοράς μάθησης προκύπτουν πολλά όπως το domain adaptation και το task adaptation (<https://arxiv.org/abs/2004.10964>). Το πόσα layers θα κάνουμε freeze ή αντίστοιχα τα πόσα layers θα αφήσουμε ελεύθερα προς εκπαίδευση μπορεί να συσχετιστεί με αυτά τα ανοικτά ερευνητικά θέματα.

Κατά την διαδικασία Μεταφοράς Γνώσης (Transfer Learning) εκπαιδεύουμε ένα στατιστικό μοντέλο σε ένα πρόβλημα μηχανική μάθησης και το εφαρμόζουμε (κατόπιν κατάλληλης επανεκπαίδευσης) σε μια άλλη εφαρμογή. Στα πλαίσια των νευρωνικών δικτύων, παρατηρήθηκε ότι τα βάρη μοντέλων εκπαιδευμένων σε τεράστια σύνολα δεδομένων για αναγνώρισης εικόνων (ImageNet), μπορούσαν να χρησιμοποιηθούν για την αρχικοποίηση δικτύων με στόχο την επίλυση άλλων προβλημάτων ανάλυσης εικόνας. Τα δίκτυα αυτά είχαν μάθει να αναγνωρίζουν πολύ γενικά χαρακτηριστικά στα αρχικά επίπεδα τους, τα οποία ήταν χρήσιμα σε πολλά διαφορετικά προβλήματα. Στην απλούστερη περίπτωση μπορεί κανείς να χρησιμοποιήσει ένα προεκπαίδευμένο δίκτυο σε ένα πρόβλημα μηχανική μάθησης με πολλά δεδομένα, και να το προσαρμόσει εκπαίδευοντάς το για λίγες εποχές μια άλλη εφαρμογή, στο οποίο συνήθως δεν υπάρχουν αρκετά δεδομένα εκπαίδευσης.

Η ίδια προσέγγιση μπορεί να εφαρμοστεί και σε προβλήματα επεξεργασίας φυσικής γλώσσας όταν έχουμε λίγα δεδομένα εκπαίδευσης. Η χρήση προεκπαίδευμένων word embeddings είναι μία μορφή Μεταφοράς Γνώσης. Όμως μπορούμε να χρησιμοποιήσουμε και μεταφορά ολόκληρου του δικτύου [In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 245–255.], το οποίο έχει μάθει να κωδικοποιεί και τα συμφραζόμενα των λέξεων (context).



VGG (Visual Geometry Group)

Models

The “16” and “19” stand for the number of weight layers in the network (columns D and E in Figure 2 below):

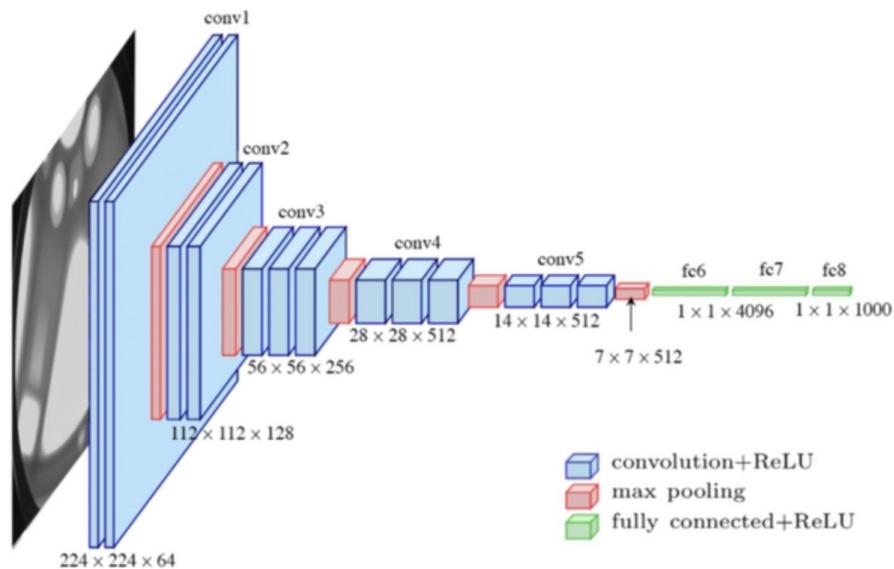
- VGG16
- VGG19

Explain Architecture

In the first part of the network, we see five convolutional blocks (conv1 to 5) which consist in stacked convolutional layers followed by a max pooling layer (you can find an explanation about these layers here). So we'll call this part the convolutional part.

This part produces a tensor with $7 \times 7 \times 512$ values for each image. The first two dimensions, (7,7), are aligned with the dimensions of the original image, and we can think of this as a very coarse version of the image, with only $7 \times 7 = 49$ large pixels. But for each pixel, instead of having 3 color channels, we have 512 features that describe what the network is seeing in this pixel (and also around it).

Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).



Property

This architecture is from VGG group, Oxford. It makes the improvement over AlexNet by replacing large kernel-sized filters(11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. With a given receptive field(the effective area size of input image on which output depends), multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increases the depth of the network which enables it to learn more complex features, and that too at a lower cost.

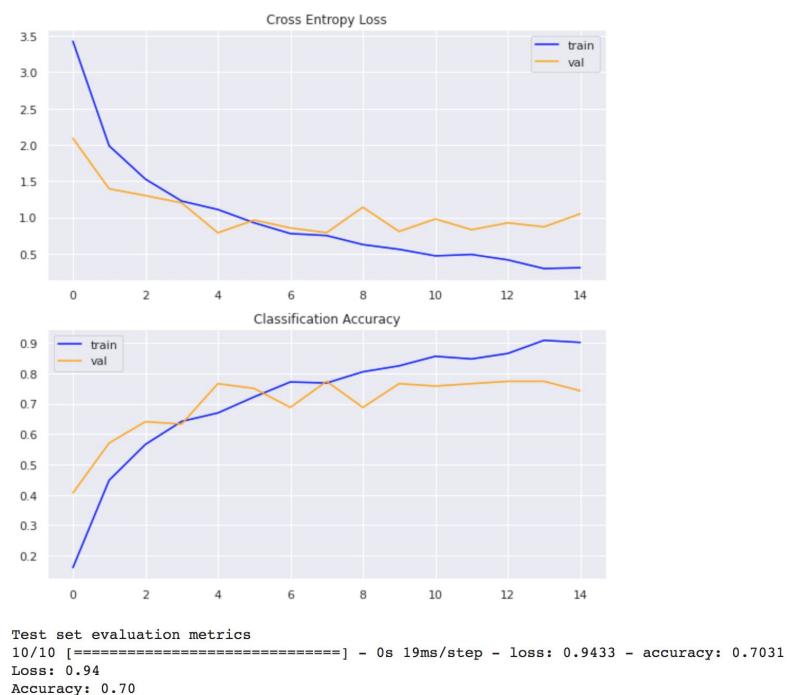
- Simplicity

Drawbacks

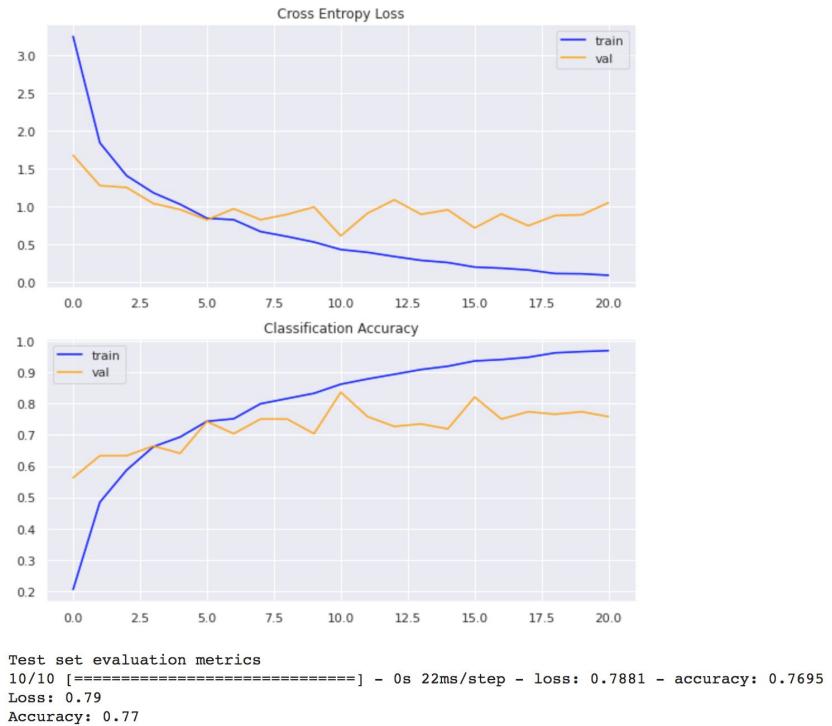
Unfortunately, there are two major drawbacks with VGGNet: It is painfully slow to train. The network architecture weights themselves are quite large (in terms of disk/bandwidth). Due to its depth and number of fully-connected nodes, VGG is over 533MB for VGG16 and 574MB for VGG19. This makes deploying VGG a tiresome task.

VGG16

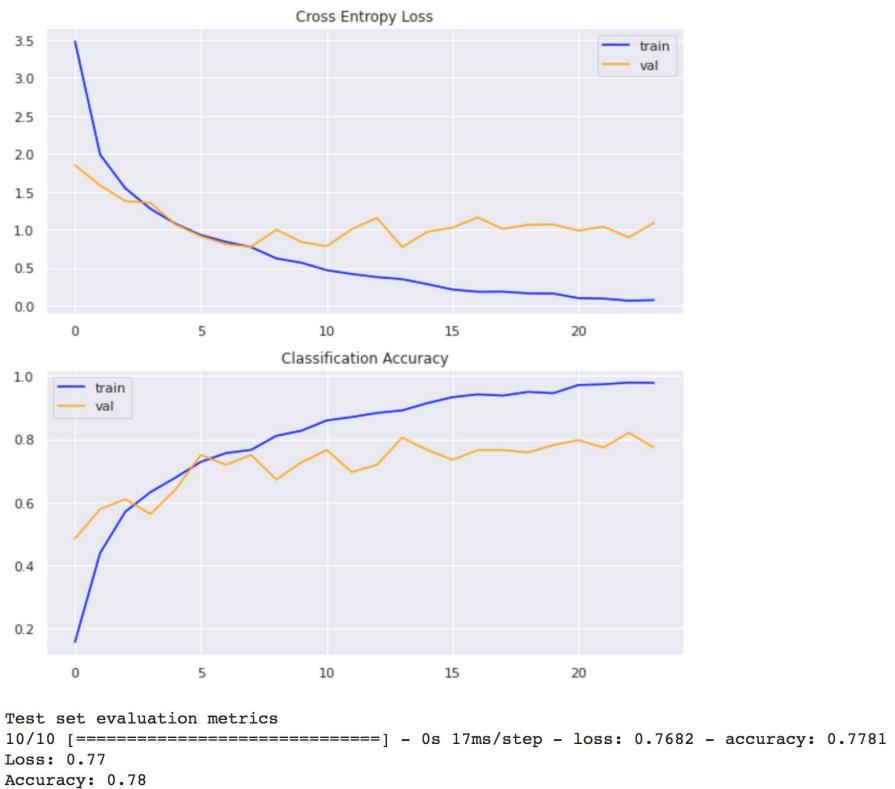
This is our result when we set all the parameters to be trainable:



This is our result when we set the 5% of the parameters to be trainable:

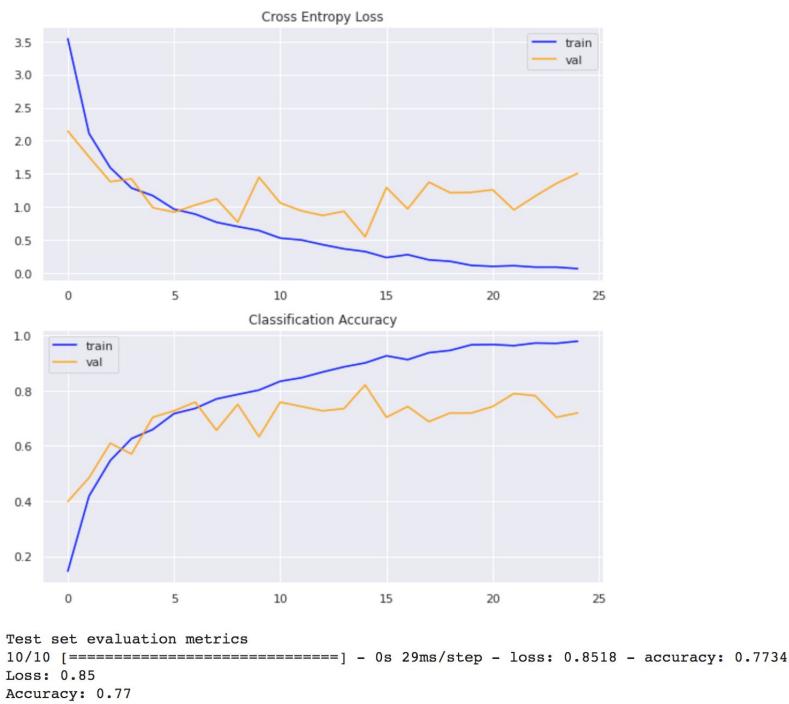


This is our result when we set the 20% of the parameters to be trainable:

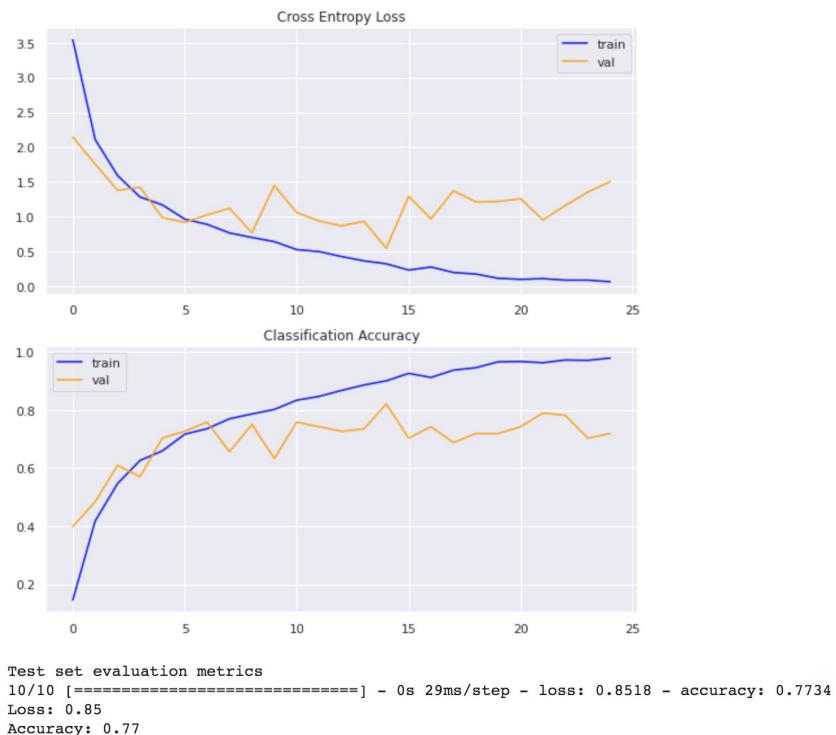


VGG19

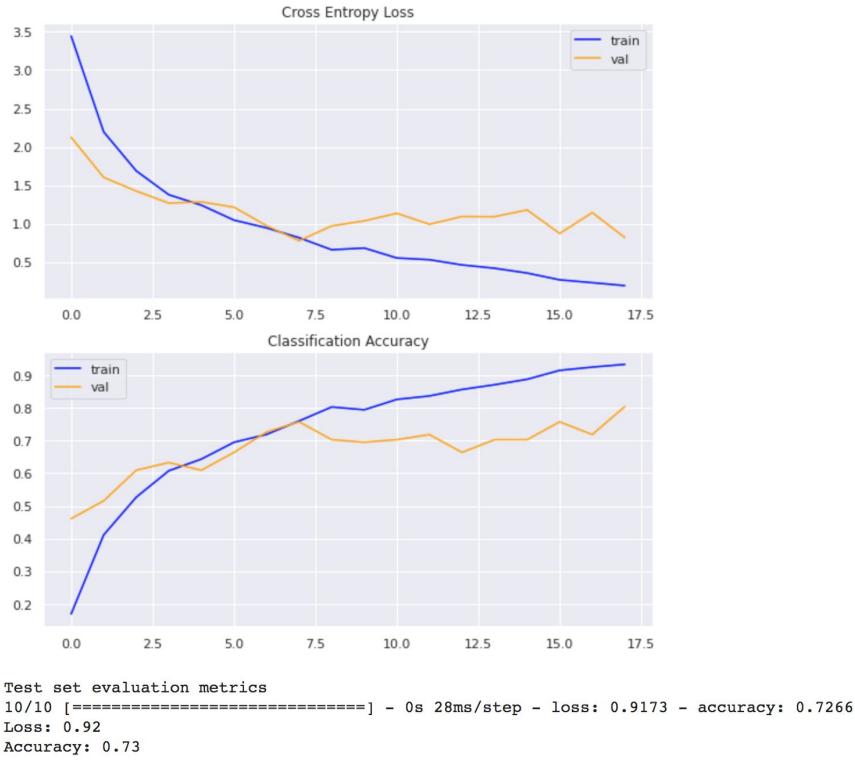
This is our result when we set all the parameters to be trainable:



This is our result when we set 5% of parameters to be trainable:



This is our result when we set 20% of parameters to be trainable:



ResNet (Residual Network)

Explain Architecture Unlike traditional sequential network architectures such as AlexNet, OverFeat, and VGG, ResNet is instead a form of “exotic architecture” that relies on micro-architecture modules (also called “network-in-network architectures”).

The term micro-architecture refers to the set of “building blocks” used to construct the network. A collection of micro-architecture building blocks (along with your standard CONV, POOL, etc. layers) leads to the macro-architecture (i.e., the end network itself). Each ResNet block is either 2 layer deep (Used in small networks like ResNet 18, 34) or 3 layer deep (ResNet 50, 101, 152).

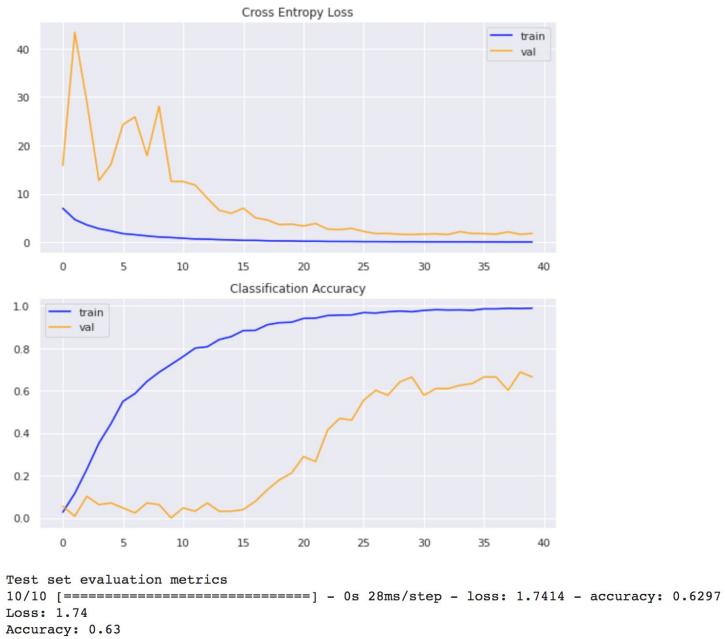
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Property

Even though ResNet is much deeper than VGG16 and VGG19, the model size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers — this reduces the model size down to 102MB for ResNet50. Before ResNet, there

had been several ways to deal the vanishing gradient issue, e.g. adding an auxiliary loss in a middle layer as extra supervision, but none seemed to really tackle the problem once and for all. The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers,

This is our result when we set all the parameters to be trainable:

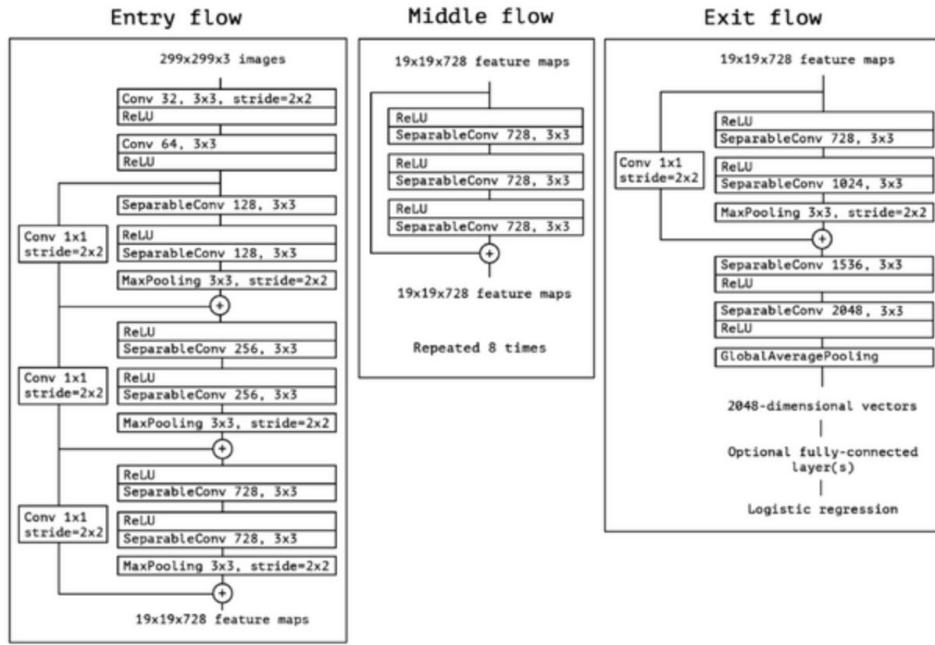


Xception

Explain Architecture

In This story, Xception by Google, stands for Extreme version of Inception, is reviewed. With a modified depthwise separable convolution, it is even better than Inception-v3 (also by Google, 1st Runner Up in ILSVRC 2015) for both ImageNet ILSVRC and JFT datasets.

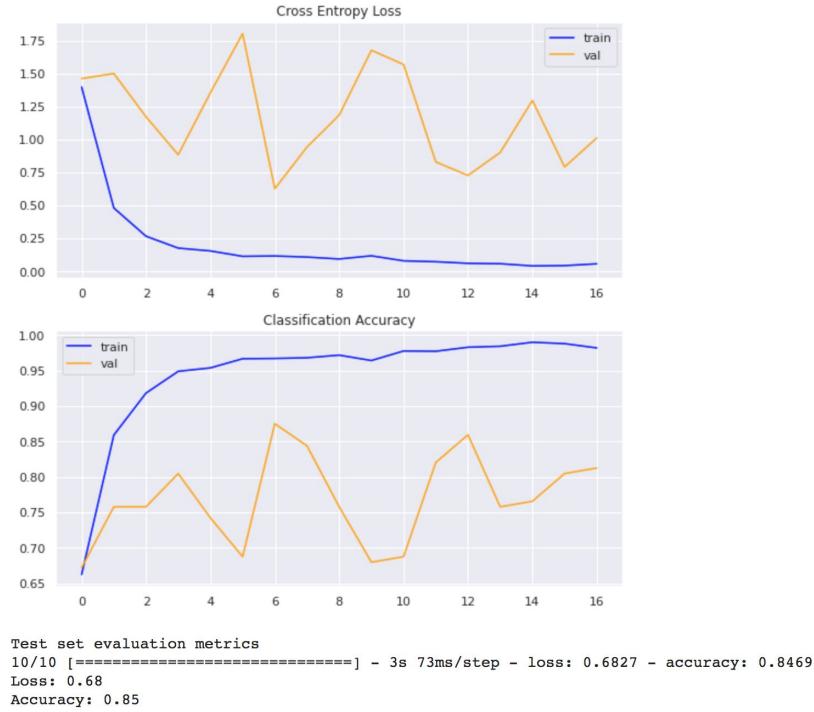
- The modified depthwise separable convolution is the pointwise convolution followed by a depthwise convolution. This modification is motivated by the inception module in Inception-v3 that 1×1 convolution is done first before any $n \times n$ spatial convolutions. Thus, it is a bit different from the original one. ($n=3$ here since 3×3 spatial convolutions are used in Inception-v3.)
- The modified depthwise separable convolution with different activation units are tested. As from the above figure, the Xception without any intermediate activation has the highest accuracy compared with the ones using either ELU or ReLU.
- As seen in the architecture, there are residual connections. Here, it tests for Xception using non-residual version



Property

A convolutional neural network architecture based entirely on depthwise separable convolution layers. The mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled. This hypothesis is a stronger version of the hypothesis underlying the Inception architecture. Xception sports the smallest weight serialization at only 91MB. But if we use non-residual version to compare with Inception-v3, Xception underperforms Inception-v3. Should it be better to have a residual version of Inception-v3 for fair comparison? Anyway, Xception tells us that with both Depthwise Separable Convolution and Residual Connections, it really helps to improve the accuracy.

This is our result when we set all the parameters to be trainable:

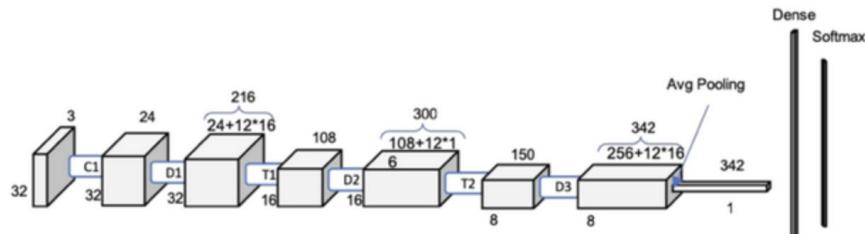


DenseNet

Explain Architecture

We can observe a first single convolutional layer, followed by two pairs of dense block — transition blocks pairs, a third dense block followed by the global average pooling to reduce it to the $1 \times 1 \times 342$ vector that will feed the dense layer.

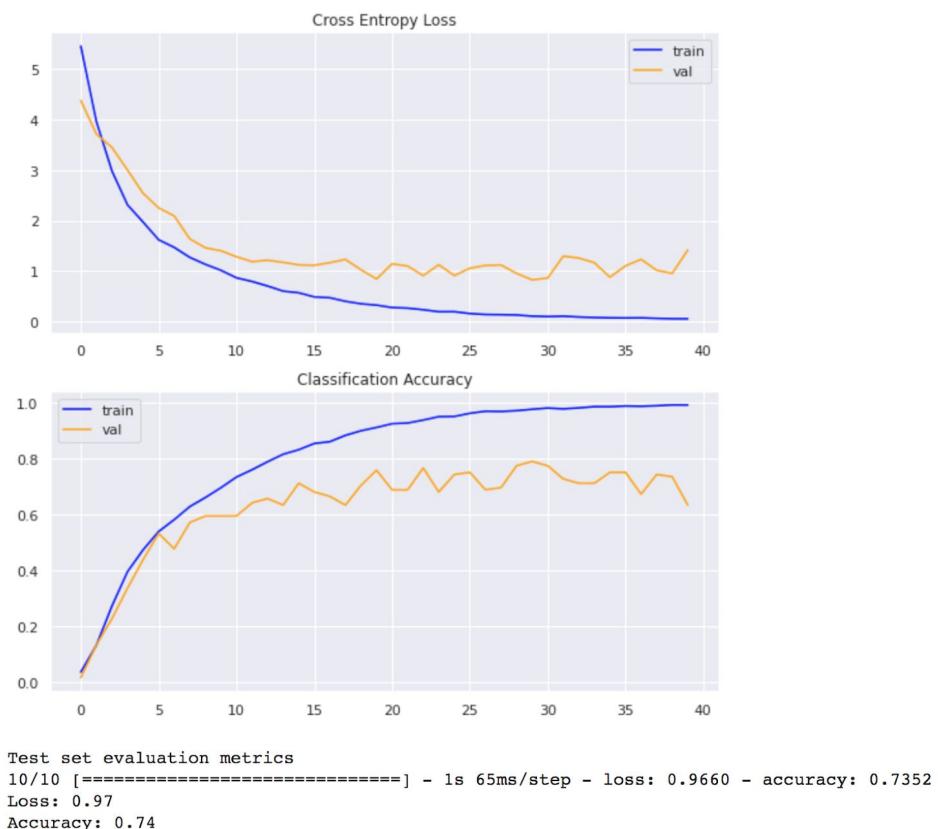
- Convolutional Layer : The first step on the DenseNet before entering into the first Dense Block is a 3×3 convolution with a batch normalization operation. The stride is 1 and there is a padding of 1 to match the output size with the input size. Note how we have already our first big difference with DenseNet for ImageNet, that we have not included here the max pooling operation in this first block to reduce the dimension of the input volume.
- Therefore, and the same way we used for ResNets, DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remain constant within a block, but the number of filters changes between them. These layers between them are called Transition Layers and take care of the downsampling applying a batch normalization, a 1×1 convolution and a 2×2 pooling layers.



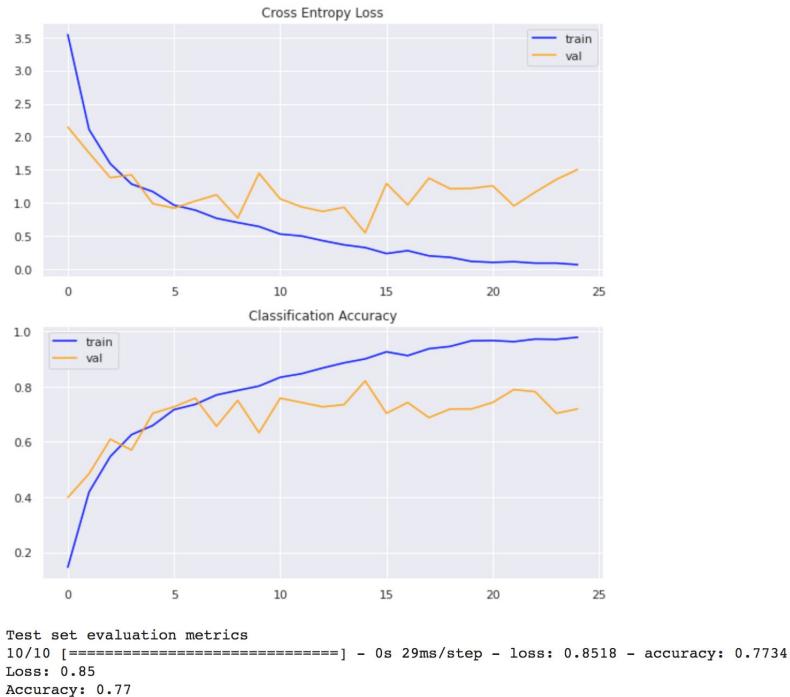
Property

- DenseNets make the first difference with ResNets right here. DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them.
- For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

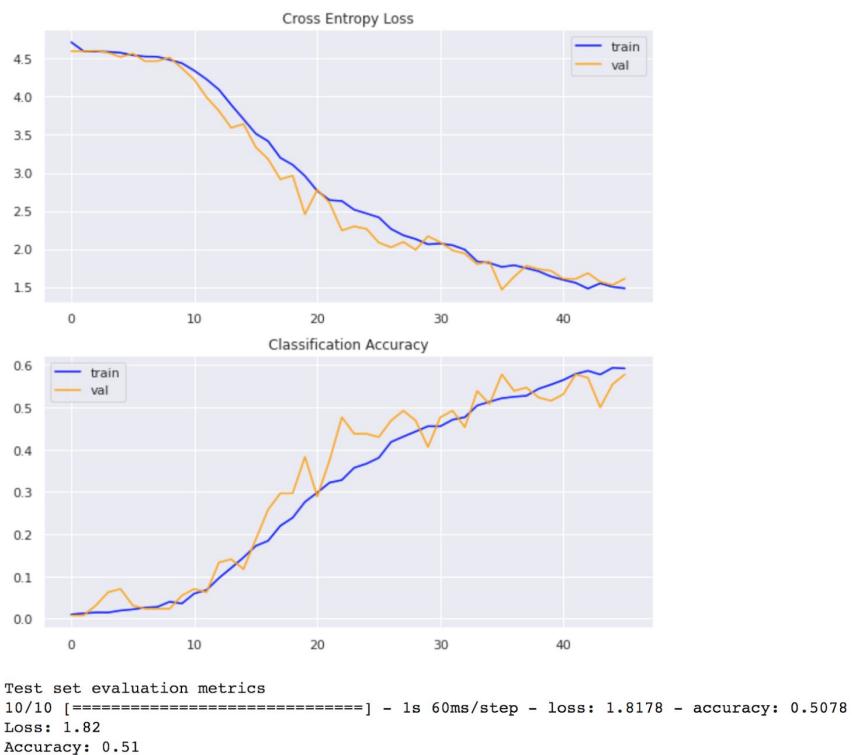
This is our result when we set all the parameters to be trainable:



Για την μελέτη του πλήθους των δεδομένων/κλάσεων στο μοντέλο μας επιλέγουμε κατάλληλο dataset με τη βοήθεια της συνάρτησης που μας δόθηκε και το εφαρμόζουμε στο καλύτερο μοντέλο του Πρώτου Μοντέλου, που είναι το καλύτερο μοντέλο γενικότερα.
Έτσι για 20 κλάσεις με όλες τις παραμετρους trainable στο VGG19 έχουμε:



Έτσι για 80 κλάσεις με όλες τις παραμετρους trainable στο VGG19 έχουμε:



Έτσι όπως ήταν αναμενόμενο όσο αυξάνεται το πλήθος των κλάσεων, για τα δεδομένα σταθερά μοντέλα, έχουμε συνεχώς χειρότερη επίδοση στο δίκτυο, αφου αν και αυξάνεται ο συνολικός αριθμός των δειγμάτων, αυξάνεται και ο αριθμός των κλάσεων, και έτσι η πολυπλοκότητα του προβλήματος της ταξινόμησης.

Μνήμη και Χρόνοι

Σχετικά με την Μνήμη μπορούμε να δούμε ότι για το βέλτιστο σθόνιασμό υπερπαματέρων έχουμε τα εξείς αποτελέσματα:

Example CNN	1.6 MB
CNN1	1.3 MB
CNN2	3.5 MB
CNN3	35.2 MB
CNN4	19.2 MB

Άρα όσο πιο deep είναι ένα μοντέλο τόσο περισσότερο αργεί να εκπαιδευτεί και τόσο περισσότερο χρόνο απαιτεί.

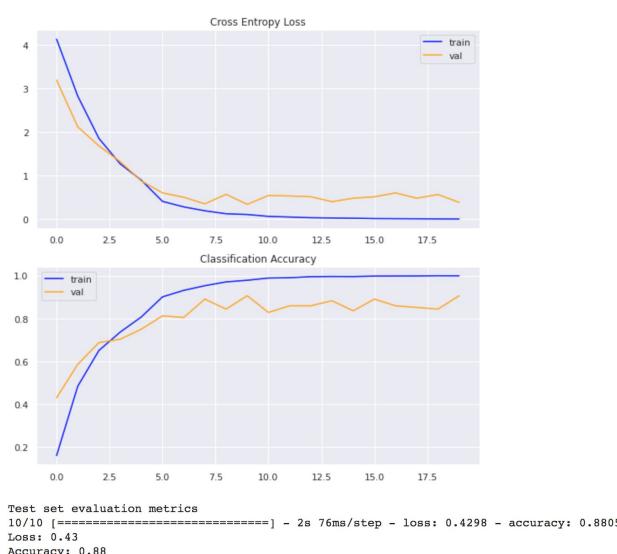
Ενώ τα δίκτυα transfer Learning όπως περιμέναμε είναι πιο βαριά της τάξεως των 100-250MB και έχουμε να αντιμετωπίσουμε αυτό το tradeoff.

Αξίζει να σημειωθεί ότι το πρόβλημα του χώρου προσπαθεί να αντιμετωπίσει το MobileNet. Οι χρόνοι υπάρχουν αναλυτικά στο notebook και σχολιαστηκαν και παραπάνω. Αξίζει να σημειωθεί ότι οι χρόνοι εκπαίδευσης των transfer Learning μοντέλων είναι πολύ μεγαλύτεροι.

Βελτιστοποίηση Καλύτερου Μοντέλου

Το μοντέλο Xception με όλες τις παραμέτρους εκπαίδευσιμες πετυχαίνει αξιοσημείωτα υψηλή επίδοση. Έχουμε ήδη εφαρμόσει early stopping και dropout και τώρα θα βρούμε τον ιδανικό optimizer για να πετύχουμε καλύτερο accuracy. Χρησιμοποιούμε όλες τις τεχνικές που χρησιμοποιούμε στο προηγούμενο αντίστοιχο παράθεμα. Η μοναδική που πέτυχε αύξηση της απόδοσης είναι η παρακάτω:

Με τον Adam και learning_rate=0.00005 πετυχαίνουμε ακόμη υψηλότερο accuracy στο all parameters trainable μοντέλο



Επομένως, πετύχαμε το καλύτερο μοντέλο με 88% accuracy ταξινόμησης